

HTML5 LENGOAIA ETA JAVASCRIPT APIAK

Juanan Pereira Varela

Udako Euskal **Unibertsitatea**
Bilbo, 2021

© Udako Euskal Unibertsitatea

© Juanan Pereira Varela

ISBN: 978-84-8438-806-7

Lege-gordailua: BI-01981-2021

Inprimategia: IMPRIME_ ABZ Impresión Digital

Azalaren diseinua: Igor Markaida

Hizkuntza-zuzenketen arduraduna: Ander Altuna Gabiola

Banatzaileak: UEU. Erribera 14, 1. D BILBO telf. 946790546 Faxe. 944793039

Helbide elektronikoa: argitalpenak@ueu.eus

www.ueu.eus

Elkar Banaketa: Igerabide, 88 DONOSTIA

Galarazita dago liburu honen kopia egitea, osoa nahiz zatikakoa, edozein modutara delarik ere, edizio honen Copyright-jabeen baimenik gabe.

Liburu honek UEUren argitalpen gintzako ebaluazio-prozesua gainditu du.

Kamen, Jon eta Oihaneri

Gaien aurkibidea

Sarrera	7
1. Zer da HTML5?	9
1.1. Historia apur bat	10
1.2. HTML5 etiketa berriak	11
1.2.1. HTML5 elementu berriak	11
1.2.2. Etiketa semantiko berriak	12
1.2.3. META etiketak	12
1.3. Ariketak	13
2. HTML5 egitura-osagaiak	15
2.1. Ariketak	17
3. JavaScript bost minututan	23
3.1. Aldagaien erazagupena	23
3.2. <i>While</i> eta <i>for</i> begiztak	25
3.3. Baldintzazko adierazpenak	25
3.4. Arrayak	26
3.5. Ariketak	29
4. JavaScript eta DOM	33
4.1. DOM aldatzen JS bidez	33
4.2. Nola txertatu JS script bat zure orrian	35
4.2.1. onload gertaera-kudeatzailea	36
4.3. Ariketak	37
5. Objektuetara zuzendutako programazioa JavaScript-en	39
5.1. Klaseak JavaScript-en	39
5.1.1. Klaseen erazagupena	39
5.1.2. Get metodoak	40
5.1.3. Herentzia	41

5.2. Ariketak	41
6. Gertaerak eta gertaera-kudeatzaileak	45
6.1. Gertaera motak	45
6.1.1. Erabiltzaileak sortutako gertaerak	45
6.1.2. Sistemak sortutakoak	46
6.2. Nola kudeatu gertaerak JavaScript-en	46
6.3. Adibide praktikoa	47
6.4. Ariketak	51
7. Datuen komunikazio asinkronoa: JSON, Ajax eta Promesak	53
7.1. Promesak	55
7.2. Fetch APIa	57
7.2.1. Fetch APIa GET eskaerak egiteko	57
7.2.2. Fetch APIa POST eskaerak egiteko	58
7.3. Ariketak	60
8. Canvas, pantailan marrazten	63
8.1. <canvas> osagaia	64
8.2. Arkuak	67
8.3. Irudiak margotzen	68
8.4. drawImage() metodoaren parametroak	69
8.5. Ariketak	69
9. Audioa eta bideoa	73
9.1. <i>Kontainer</i> eta <i>codec</i> -ak	74
9.2. <video> etiketa	74
9.3. Audio-etiketa	78
9.4. Ariketak	78
10. Biltegitratze lokala, localStorage APIa	83
10.1. Web Storage APIa	84
10.1.1. Nola aztertu localStorage biltegian dagoen edukia	85
10.2. Ariketak	86
11. Geokapen APIa	89
11.1. Posizioa lortzen. getCurrentPosition metodoa	90
11.2. Geokapena Google Maps-en marrazten	93
11.2.1. Markatzaileak	94
11.3. Ariketak	95

12. Web Workers APIa	97
12.1. Zer da Web Worker bat?	98
12.2. Zer egin dezakegu Web Workers APIarekin?	98
12.3. Nola erabili Web Worker APIa	98
12.4. Oinarrizko adibidea	99
12.5. Ariketak	101
13. Inprimakiak HTML5en	103
13.1. Eremu eta atributu berriak HTML5en	104
13.1.1. Placeholder atributua	104
13.1.2. Autofocus atributua	104
13.1.3. Email eremu mota	105
13.1.4. URL eremu mota	106
13.1.5. Zenbakiak sartzeko eremuak: <i>range</i> , <i>slider</i> eta <i>spinbox</i> eremu motak	106
13.1.6. Datak sartzeko eremuak	106
13.1.7. Kolore bat hautatzeko eremu mota	107
13.1.8. Bilaketak egiteko <i>search</i> eremuak	107
13.1.9. Derrigorrezko eremuak (<i>required</i>)	108
13.1.10. Inprimakien baliozkotzea	108
13.2. Ariketak	111
14. WebSocket-ak	113
14.1. WebSocket APIa nola erabili	114
14.1.1. Socket berri bat sortu (bezeroan)	114
14.1.2. Konexioa ezarri	114
14.1.3. Mezuak jaso eta bidali	115
14.2. Ariketak	119
15. Zerbitzu-langileak (Service Worker-ak)	121
15.1. Service Worker-ak nola erabili	121
15.2. Konexiorik gabeko aplikazioak	122
15.3. Service Worker-a instalatu	123
15.4. ServiceWorker-a programatzen	124
15.5. <i>Cache</i> -memoria betetzen	124
15.6. Ariketa	128
16. NodeJS	131
16.1. Sarrera	131
16.2. Ariketa	134

17. ExpressJS	135
17.1. Baliabide estatikoak	136
17.2. POST parametroak	137
17.3. GET parametroak URLan bidaltzen	138
17.4. GET parametroak eskaeran bidaltzen	138
17.5. Nodemon	139
17.6. Txantiloia: EJS	140
17.6.1. Txantiloia betetzen	142
17.7. Ariketak	144
18. Nola testatu Express aplikazioak	145
19. Datuen iraunkortasuna eta MongoDB	153
19.1. MongoDB instalatu eta jaurti	153
19.2. Datu-base berri bat ireki edo sortu	154
19.3. Datuak txertatzen	155
19.4. Dokumentuak bilatzen	156
19.5. MongoDB eta NodeJS lotzen	157
19.6. Datuen txertaketak mongo-n	158
19.7. Ariketa	159
20. Araztailea	161
20.1. Firefox-en Web Kontsola	161
20.2. Araztailea erabiltzen	162
20.2.1. Baldintzazko eten-puntuak	166
21. Garapenerako ingurune bateratuak - IDEak	169
21.1. IntelliJ	169
21.1.1. Hobespenak	169
21.2. VSCode	171
21.2.1. Nondik jaitsi	172
21.2.2. Oinarrizko adibidea	173
22. Kontzeptuen aurkibidea	177

Sarrera

Liburu honen jatorria Bilboko Ingeniaritza Eskolan kokatzen da, duela hamar urte, webguneen garapenaren inguruko eskolak ematen hasi nintzenean. Bertan, HTML5 eta JavaScript APIak ikasteko apunteak eta ariketak prestatzen hasi nintzen, unibertsitateko eskoletan erabiltzeko eta etorkizunean, denbora lagun, liburu batean biltzeko asmoarekin.

Esku artean duzu lan horren emaitza. Adibide praktikoz jositako liburu bat da, HTML5 lengoaia eta APIak ikas-irakasteko aproposa. Bertan, JavaScript-en inguruko ezagutzak freskatu ondoren (objektuetara zuzendutako programazioa, promesak, gertaerak, JSON formatua...), honakoak lantzen dira: Canvas (pantailan margotzeko), WebSocket-ak (sareko komunikazio azkarrak lortzeko, adibidez jokalaria anitzeko web jokoak programatzeko), Audio eta Video APIak (multimedia-animazioak eta bideo-osagaiekin efektuak lortzeko), LocalStorage (nabigatzailean bertan informazioa gordetzeko), Geolocation APIa (uneko erabiltzailearen geokapena lortzeko) eta beste hamaika APIren inguruko xehetasunak. Horiek guztiak *front-end* edo bezeroaren aldeko programak egiteko erabiliko ditugu. Dena den, liburua idazten hasi nintzenean *front-end* aldea programatzeko ezagutzak biltzea helburua bazen ere, berehala konturatu nintzen *back-end* edo zerbitzariaren aldeko oinarritzko *scriptak* programatzeko oinarritzko kontzeptuak ere landu beharko nituzkeela: bezeroan sortzen diren datuak jaso eta datu-base batean gordetzeko, erabiltzaileak kautotzeko, sareko konexioak kudeatzeko, etab. Hori dela eta, NodeJS —JavaScript zerbitzarian—, Express —web aplikazioak programatzen laguntzeko framework bat— eta MongoDB, NoSQL motako datu-base kudeatzaile baten inguruko oinarritzko kontzeptuak ere jorratuko dira.

Kapitulu bakoitzean, teoria apur bat landu ondoren, ariketa praktikoak proposatzen dira, gehienak soluzioarekin edo eskatzen den kodearen eskema orokorrarekin. Ariketetan lantzen diren kode-zatiak zure ordenagailuan edo online probatzea oso gomendagarria da. Kodea aztertuz, aldatuz, apurtuz, konponduz eta gauza berriak probatuz ikasten baita.

Kodea probatzeko, hainbat tresna ditugu eskuragarri. Alde batetik, GitHuben argitaratu da ariketa eta adibideen kodea. Bestetik, zenbait kapitulutan agertzen diren aplikazioen kode-zatiak modu azkarrean probatu nahi izanez gero, ordenagailuan ezer instalatu gabe, codesandbox.io webgunean egitea proposatzen da.

Liburu honen bizitza ez da orri hauetan amaitzen. Web garatzaileen komunitateak ariketa berriak proposatuko ditu, HTML estandarraren bertsio berriak aterako dira, API eta funtzio berriekin, framework batzuk desagertu egingo dira, beste berri batzuei lekua egiteko... Hori guztia jaso eta dokumentatzeko webgune bat sortu da: <https://ikasten.io/html5/>. Bertan argitaratuko dira ekarpen eta proiektu honen inguruko berri guztiak. Baita zuk bidalitakoak ere. Anima zaitetz parte hartzera!

Donostian, 2021eko azaroan

1. Zer da HTML5?

HTML5 zer den ulertzeko, lehenengo HTML zer den gogoratu behar dugu. HTML HyperText Markup Language edo Hiper Testua Markatzeko Lengoaia da. Webeko edukiek izan behar duten formatua definitzen duen estandarra, hain zuzen ere. Sir Tim Berners-Leek (webaren asmatzaileak) sortua 1991n, gaur egun W3C (World Wide Web Consortium) erakundeak mantentzen du. HTML estandar internazionala bilakatu zen 2000n (ISO/IEC 15445:2000).

HTML lengoaia, CSS (Kaskadako Estilo Orriak edo Cascading Style Sheets) eta JavaScript lengoaia edozein webgune osatzeko erabiltzen diren oinarritzko teknologiak dira.

Web arakatzailleek (edo nabigatzaileek) HTML dokumentuak web zerbitzaritik edo disko gogorretik irakurri, interpretatu eta pantailan bistaratzen dituzte, testua, irudiak, bideoa eta audioa uztartuz.

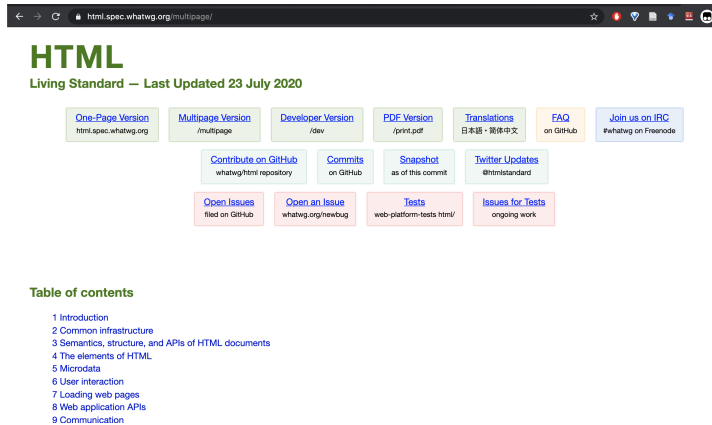
HTML lengoiaren historia interesgarria da, baina erakunde eta teknizismo asko uztartzen dituen bezala, beheko azpiatalean sartu dugu, hautazko irakurketa gisa. Datu bakar batekin geratuko gara: 2014a izan zen HTML5en lehenengo bertsio estandarra argitaratu zuten urtea.

Ez gara luzatuko HTML lengoiaren aurreko bertsioen (HTML4 eta aurrekoen) etiketak zehazten. Liburu hau irakurtzen ari bazara, ziur asko `<h1>`, `<h2>`, `<p>`, `<font>`, `<i>`, `<a>`, `<img>`, `<form>`, `<input>`, `<textarea>` eta horrelakoak primeran menderatuko dituzu.

Gaur egungo webguneak, berriz, ez dira lehengoak bezain errazak. Izan ere, gaur egungo web aplikazioak konplexuak, multimediadunak, mugikorrek eta *offline* lan egin ahal izateko diseinatzen dira. Hori guztia lortzeko, ez dute inolako plugin edo gehigarririk erabiltzen (ez *Flash*, ez *JavaFX*, ez *Silverlight*, ezer ere ez). HTML5, JavaScript eta CSS, besterik ez. Nola demontre lortu dute horrelako web aplikazioak egitea hiru osagai horiekin? HTML5 hitza da gakoa, eta ez bakarrik etiketa-lengoaia...

HTML5 etiketa-lengoaia, JavaScript-ek eskaintzen dituen zenbait API berri eta

CSS3 lengoia kutxa bakar batean sartzen baditugu, HTML5 etiketaz bataiatutako kutxa lortuko dugu. Ikusten denez, izen bakar batek zenbait kontzeptu izendatzen ditu.



1.1. irudia: HTML5 estandarra bizirik dago eta eguneraketak jasotzen jarraitzen du WHATWG webgunean: <https://html.spec.whatwg.org/multipage/>.

1.1 Historia apur bat



Hautazkoa

Atal batzuk nahiko teknikoak bilaka daitezke. Hau bezalako “Hautazko” ohartxoak jarri ditugu horrelako azpialetan, abisu gisa edo :)

HTML lengoia hainbat bertsioetatik igaro da. Tim Berners Leek 1991n proposatu zuen HTMLren lehenengo zirriborroa, 18 etiketaz osatutakoa. Horietatik, 11 etiketa oraindik ere aurki daitezke HTML5en. Zirriborro horretan, IETF (Internet Engineering Task Force) erakundeak 1993ko erdialdean HTML zehaztapienak argitaratu zituen. Ondoren, IETFk HTML espezifikazioa hobetzeko eta kudeatzeko “HTML Working Group” izeneko taldea antolatu zuen. Talde horren lanaren lehenengo emaitza —beste enpresa eta erakunde batzuen ekarpena jaso ondoren, adibidez, Mosaic nabigatzaile aitzindariaren IMG etiketa— 1995ean plazaratu zen, HTML2 estandarra.

IETFk 1995ean HTML3ren proposamena jaso zuen, baina proposamena 6 hilabete geroago iraungi egin zen, inolako hobekuntzarik gabe. Bitartean, World Wide Web Consortium (W3C) erakundea nabigatzaile propio bat garatzen hasia

zen, *Arena* izenekoa, HTML3k proposatzen zituen berrikuntzak probatzeko asmoz (baita CSS estilo-orriak probatzeko ere). Garai hartan Netscape eta Internet Explorer sortu ziren, sekulako arrakastarekin. Nabigatzaileek eskatzen zituzten berrikuntzen kudeaketak eta HTMLk berak zuen ospeak IETFk zituen baliabideak gainezkatu zituzten. Hori dela eta, hurrengo urtetik hasita (1996), HTMLren espezifikazioa World Wide Web Consortium (W3C) erakundearen esku utzi zen.

W3C, IETFk egin zuen legez, kanpoko enpresen ekarpenak ere aztertu eta estandarra eguneratzen saiatu zen. Horren lehenengo fruitua 1997an izan zen, HTML4 estandarren argitalpenarekin.

HTML 4.01 1999an argitaratu zen, nahiz eta akatsak zuzentzeko hainbat bertsio gehiago kaleratu behar izan zituzten 2001 arte.

2004an WHATWG (*Web Hypertext Application Technology Working Group*) lantaldea sortu zen, W3Ck HTMLren espezifikazioarekin zerman erritmo motelari aurre egiteko. Baita W3C erakundeak HTML bertan behera utzi eta horren ordez XML lengoia oinarritutako teknologiak sustatzeko hartutako erabakiari aurre egiteko ere.

WHATWG taldearen posta-zerrenda 2004ko ekainaren 4an sortu zen. Bertan Apple, Mozilla eta Opera nabigatzaileen ingeniariak HTML5 garatzen hasi ziren. WHATWGk Microsofteko ingeniari bati ere bertan parte hartzeko gonbita luzatu zion, baina ezezkoa jaso zuen.

2007an, WHATWGk bere HTML5en espezifikazioa W3Cren HTML garatzeko zuen lantaldeari eskaini zion, hortik abiatzeko haien lana. Izan ere, bertan adostu zuten HTML5 izena emango ziotela lantalde horien entregagaiari. Urte horretan bertan W3Ck eskaintza onartu zuen eta bi taldeen artean hainbat urtetan zehar lan egin ondoren, 2014ko urriaren 28an HTML5en lehenengo bertsio estandarra amaitu zuten.

1.2 HTML5 etiketa berriak

1.2.1 HTML5 elementu berriak

HTML5 bertsioak web dokumentuak eta aplikazioak sortzeko lengoaiaren bertsio zaharra hobetzen du, etiketa eta API (*Application Programming Interfaces*) berriak eskainiz.

Zehazki, osagai grafiko berriak (*<canvas>*, SVG —*Scalable Vector Graphics*, grafiko bektorial-eskalagarriak—) eta multimedia (*<audio>*, *<video>*) eskaintzen ditu, *plugin* edo API osagarriak erabili gabe.

Horretaz aparte, web dokumentuen edukiaren semantika hobetzeko elementuak gehitzen ditu: *<section>*, *<article>*, *<header>* eta *<nav>*, besteak beste.

HTML5en sorkuntzan hasiera-hasieratik dago errotuta gailu mugikorretarako web aplikazioak garatzeko lengoia aproposa izan behar zuela, *smartphone* eta *tablet*-en ezaugarriak aprobetxatuz.

1.2.2 Etiketa semantiko berriak

Etiketa berriak edo HTML4 bertsioarekiko eguneratuak landuko ditugu hemen.

DOCTYPE hitzaurrea

`<!DOCTYPE html>` hitzaurrearekin hasi behar dute HTML5en idatzitako web orriek. Horrekin, nabigatzaileak jakin ahal du estandar berrienei jarraituz eraiki behar duela orria. DOCTYPE ingelesezko *Document Type Declaration* (Dokumentu Motaren Erazagupena) hitzen akronimoa da.

Web orri baten HTML5 egitura zuzena dela egiaztatzeko online balidatzaileak erabil ditzakegu. Balidatzaile horiek egiten duten lehenengo gauza dokumentuaren sarrerari begiratzea da, jakin ahal izateko zer arauten kontra balioztatu behar duten uneko dokumentua. HTML5 lengoian garatutako dokumentu zuzenean `<!DOCTYPE html>` sarrerarekin hasi behar dute.

Quirks mode



HTML5 diseinatzean ordura arte erabiltzen zen dokumentuen hitzaurrea erraztu nahi izan zen. Izan ere, nork gogoratzten du garai hartan erabiltzen zen hurrengo “sarreratxo”?

```
<!DOCTYPE html PUBLIC "-//W3C/DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

Sarreratxo horri DOCTYPE switch deritzo. Agertuz gero, dokumentuak estandarrak betetzen dituela (estandarrei jarraituta idatzi dela) esan nahi du. Ez bada agertzen, estandarrei erdipurdi jarraituta edo estandarrak kontuan izan gabe idatzi dela esan nahi du (normalean webgune zaharretan gertatzen dena). Azken modu anarkiko horri quirks mode deritzo. HTML5en, DOCTYPE sarrera asko laburbildu da. Orain nahikoa da honela idaztea: `<!DOCTYPE html>`

1.2.3 META etiketak

META etiketak bilatzaileek erabiltzen dituzte batez ere jakin ahal izateko, automatikoki, zeintzuk diren dokumentuaren gako-hitzak (*keywords*), hizkuntza, do-

kumentu mota edo erabilitako karaktere-jokoa. HTML5 sortu arte, hau zen maiz erabilitako META etiketa bat:

```
<meta http-equiv="content-type" content="text/html;
↳ charset=UTF-8">
```

Bertan, dokumentua HTMLz osatutako dokumentua zela zehazten zen. Baita UTF-8 karaktere-jokoa erabiltzen zela ere. Berriro, maiz erabilitako etiketa iza-teko, oso luzea, ezta? Orain, horrela laburbil daiteke, guztiz zuzena izanik:

```
<meta charset="utf-8">
```

Gauza bera gertatzen da estilo-orriak zehazteko garaian. Lehen honela idazten zena:

```
<link type="text/css" rel="stylesheet" href="estiloak.css">
```

HTML5en beste era honetara idazten da:

```
<link rel="stylesheet" href="estiloak.css">
```

Edo JavaScript-en egindako *skriptak* estekatzeari lehen honela egiten zena:

```
<script type="text/javascript" src="kodigoa.js"></script>
```

orain honela laburbildu da:

```
<script src="kodigoa.js"></script>
```

Gauza bera orriaren hizkuntza adierazteko. Lehen:

```
<html xmlns="http://www.w3.org/1999/xhtml" lang="en"
↳ xml:lang="en">
```

orain:

```
<html lang="en">
```

Etiketa semantiko berriak (*<header>*, *<footer>*, *<nav>*, *<article>*, *<section>*, *<aside>*) hurrengo gaian jorratuko ditugu. Osagai grafiko berriak aztertze-ko ere atal bereziak izango ditugu liburu honetan. Zehazki, *<canvas>* eta *<svg>* etiketak 8. atalean zehazten dira eta multimediarekin lotutako etiketak, *<audio>* eta *<video>*, 9. atalean.

1.3 Ariketak

1. Bihurtu HTML5 formatura HTML4n programatutako lerro hauek:

```
<meta http-equiv="content-type" content="text/html;
↳ charset=UTF-8">
```

```
<link type="text/css" rel="stylesheet"  
  ↪ href="estiloak.css">
```

```
<script type="text/javascript"  
  ↪ src="kodigoa.js"></script>
```

```
<html xmlns="http://www.w3.org/1999/xhtml"  
  ↪ lang="eu_ES" xml:lang="eu_ES">
```

2. Bihurtu HTML5 formatura HTML4n programatutako adibide hau:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0  
  ↪ Strict//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">  
<html xmlns="http://www.w3.org/1999/xhtml"  
  lang="eu_ES"  
  xml:lang="eu_ES">  
  <head>  
    <meta charset="utf-8" />  
    <title>1. Ariketa </title>  
    <link href="css/style.css" rel="stylesheet" />  
  </head>  
  <body>  
  </body>  
</html>
```

2. HTML5 egitura-osagaiak

Hainbat etiketa semantiko berri aurki ditzakegu HTML5en [espezifikazioan](#)¹. Horietatik, garrantzitsuenak eta askotan erabiltzen direnak, *header*, *footer*, *nav*, *article*, *section* eta *aside* dira.

Deskribapen eta eztabaida sakon eta aspergarri bat eman ordez, hobe iruditu zait adibideekin aztertzea. Dena den, etiketa garrantzitsu horien deskribapen oso labur batekin hasiko gara.

<header> Webgunearen goiburua irudikatzeko etiketa.

<footer> Webgunearen orri-oina irudikatzeko etiketa. Adi, ez da beti orriaren beheko aldean zertan agertu.

<nav> Webgunetik nabigatzeko menua (edo antzeko funtzionalitatea) gordeko du.

<article> Bere baitan zentzua daukan eduki puska gordeko du. Etiketa honen barruan gordetzen dugun edukiak zentzua izan beharko luke RSS jario baten elementu gisa (adibidez, egunkari baten berriak emateko RSS baten barruan, berri-laburpen bat <article> bat izan daiteke)

<section> Artikuluak (<article> osagaiak) gaiaren arabera multzokatzeko erabil daitezke, baina baita artikulu baten barruan dagozkion atalak zehazteko ere.

<aside> Alboko edukiak eduki bloke bat zehazten du, bere inguruan dagoen eduki nagusiarekin zerikusia duen zerbait gordetzeko (baina eduki nagusiaren jarioarekin guztiz bat ez datorren zerbait).

Goazen adibideetara. Lehenengoa, nola ez, [StackOverflow webgunetik](#)² hartua da eta [2.1](#). irudian ikus dezakegu. Bertan orri baten goiburua adierazteko *header* etiketa erabiltzen dela adierazten dugu. Ildo beretik, askotan nabigatzeko barra bat ikusten da, *nav* etiketekin adierazita irudian. Orri baten atal nagusiak, edukia bera eta dokumentuaren azpiataletik nabigatzeko esteka zuzenak, *section* etiketen

¹<https://html.spec.whatwg.org/multipage/sections.html>

²<http://stackoverflow.com/a/24765186/243532>



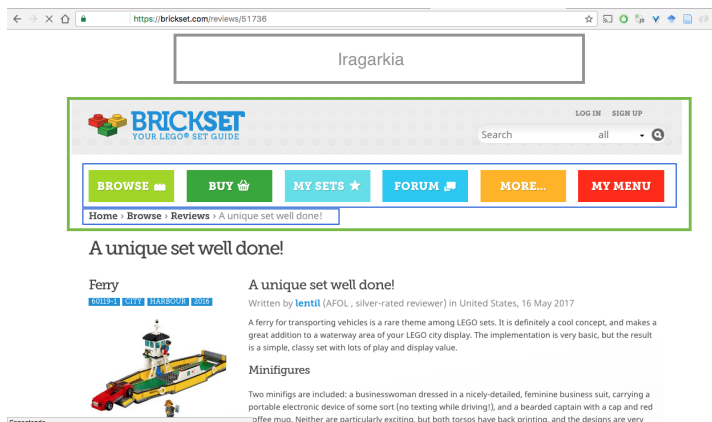
2.1. irudia: HTML5 egitura-osagaiak. Iturria: <http://stackoverflow.com/a/24765186/243532>.

barruan sartuko ditugu. Uneko dokumentuarekin lotura zuzena ez baina zeharkako lotura duen beste eduki bat kodean antolatzeko, *aside* etiketa erabil dezakegu. Bukatzeko, orri-oinak *footer* gisa markatuko ditugu. Aipatutakoa, antolaketa modu bat da, W3Ck gomendatzen duena hain zuzen ere, baina kontuz, zenbait xehetasun daude. Alde batetik, *footer* etiketaren edukiak ez du beti dokumentuaren beheko aldean agertu behar, baliteke beste oin batzuk izatea. *Section* baten barruan artikulua (*article*) batzuk izan ditzakegu... eta alderantziz! Ikus ditzagun adibide zehatz batzuk hau guztia ondo barneratzeko.

Adibidez, har dezagun **Lego webgunearen orri bat**³, 2.2. irudian ikus dezakeguna. Orriari dagokion HTML kodea 2.5 irudian daukagu. Bertan 1. puntuan (grisez), *section* bat aurkitzen dugu, orriaren iragarkia edo *banner-a* adierazteko. Jarraian, 2. puntuan, *header* etiketa, Lego webgunetik nabigatu ahal izateko botoiak, bilatzailea eta ogi-papurren bidea adierazteko. Goiburuaren (*header*-en) barruan, bi *nav* etiketa, nabigatzeko lehenengo botoi-barra eta ogi-papurren bidea markatzeko.

Lego bilduma-bilatzailean sartuz gero, goiburua (*header*) berdina bada ere, edukia noski, aldatzen da (ikus 2.4. irudia). Eduki horri dagokion kodea 2.5. irudian aztertuko dugu. Bertan, bi *aside* daude, figura-bildumetatik nabigatzeko

³<https://brickset.com/reviews/51736>



2.2. irudia: HTML5 egitura-osagaiak.



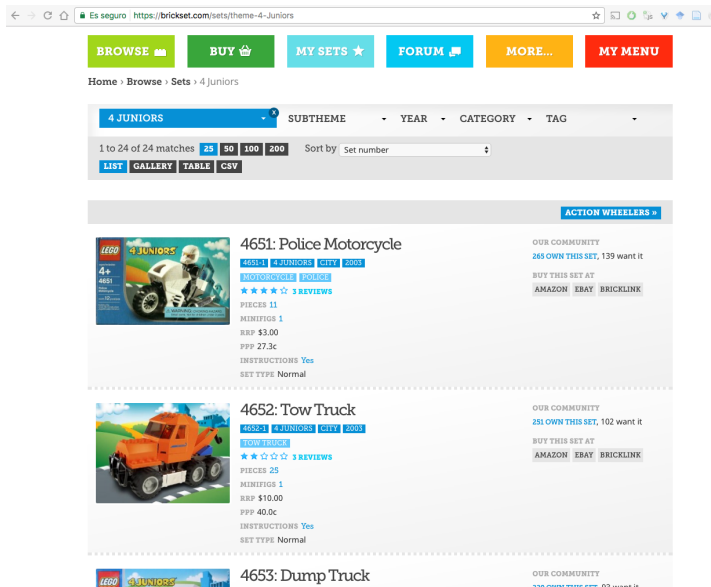
2.3. irudia: HTML5 etiketa semantikoak: Lego orriaren adibidea. Kodea.

eta iragazketa egiteko inprimakiek. Jarraian, *section* etiketaren barruan, figura-bilduma guztiak, banan-banan, bakoitza *article* etiketa baten barruan.

2.1 Ariketak

1. Ordezkatu <div> etiketa bakoitza dagokion HTML5 etiketa semantiko egoziaz:

```
<div id="header">
<h1> Goi mailako goiburu bat naiz</h1>
</div>
<div id="footer">
<h3> UPV/EHU </h3>
<p> 2020/21 ikasturtea </p>
</div>
```



2.4. irudia: HTML5 egitura-osagaiak. Eduki nagusia.

- Ordezkatu <div> etiketa bakoitza dagokion HTML5 etiketa semantiko egokiarekin:

```
<div id="navigation">
<ul>
<li>Euskal Herrian</li>
<li>Datuak munduan</li>
<li>Shock ekonomikoa</li>
<li>Buletin berezia</li>
</ul>
</div>
```

- Ordezkatu <div> etiketa bakoitza dagokion HTML5 etiketa semantiko egokiarekin eta bihurtu goiburu-etiketak h1 eta h2:

```
<div class="section">
<h2>Berrinfekzio kasuen berri eman dute Belgikan eta
Herbehereetan ere</h2>
<h3>OME Osasunaren Mundu Erakundearen iritzia</h3>
<p>Astelehenean jakin zen COVID-19 birusarekin
    ↳ berrinfektutako
    pertsona bat atzeman zutela lehen aldiz...</p>
</div>
```



```

<p class="aside">Pedro Sanchez Espainiako Gobernuko
  ↳ presidenteak ere agerraldia egin zuen
  ↳ atzo...</p>
</article>
</section>

```

6. Bihurtu HTML5 formatura HTML4n programatutako adibide hau:

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  ↳ Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  lang="eu_ES"
  xml:lang="eu_ES">
  <head>
    <meta charset="utf-8" />
    <title>1. Ariketa</title>
    <link href="css/style.css" rel="stylesheet" />
  </head>
  <body>
    <div id="header">
      <h1>Koronabirusa</h1>
      <h2>Euskal Herrian eta Munduan</h2>
    </div>
    <div id="navigation">
      <ul>
        <li>Euskal Herrian</li>
        <li>Datuak munduan</li>
        <li>Shock ekonomikoa</li>
        <li>Buletin berezia</li>
      </ul>
    </div>
    <div id="section">
      <h2>Zelaa aztertzen ari da COVID-19a duten haurren
        ↳ gurasoei
        gaixo agiria ematea</h2>
      <div class="article">
        <h3>Bihar elkartuko dira autonomia erkidegoetako
          ordezkariekin</h3>
        <p>Maskarak, gel hidroalkoholikoak, PCRak,
          ↳ distantzia...
          Funtsean, COVID-19ak baldintzatuko du 2020-2021eko
          ↳ ikasturtea.</p>
      </div>
    </div>
  </body>
</html>

```

```

<p class="aside">Bihar bilera: Zelaak azaldu duenez,
    ↪ bihar
elkartuko dira autonomia erkidegoetako
    ↪ ordezkariekin</p>
</div>
<div class="article">
    <h3>Berrinfekzio kasuen berri eman dute Belgikan eta
Herbehereetan ere</h3>
    <p>
        OME Osasunaren Mundu Erakundeak esan du berrinfekzioen
gaineko berriak oraindik 'anekdotikoak' direla, eta
        ↪ zer ikertua
franko dagoela
    </p>
</div>
</div>
<div id="footer">
    <h3> UPV/EHU </h3>
    <p> 2020/21 ikasturtea </p>
</div>
</body>
</html>

```

7. Bihurtu HTML5 formatura HTML4n programatutako adibide hau:

```

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="content-type" content="text/html;
    ↪ charset=UTF-8">
<title>Taberna</title>
<link type="text/css" rel="stylesheet"
    ↪ href="taberna.css">
<script type="text/javascript"
    ↪ src="taberna.js"></script>
</head>
<body>
<h1>Ongi Etorri Tabernara</h1>
<p>

</p>
<p>
Arratsaldero elkartzen gara xake edo mus partida

```

```
    ↪ batzuk jolastera <a  
href="garagoardoak.html">garagardo batzuk</a> edaten  
    ↪ dugun bitartean,  
solasaldi ederraz gozatuz. WiFi sarea ere badugu,  
    ↪ konexio on batekin.  
</p>  
</body>  
</html>
```

3. JavaScript bost minututan

HTML5en arrakasta ez dago sartu dituen HTML etiketa semantiko berrietan oinarrituta, JavaScript API berrietan baizik. API horietan murgildu baino lehena-go, ezinbestekoa dugu gure JavaScript-en ezagupen herdoilduak apur bat berritu eta garbitzea. Horretarako, atal honetan, JavaScript lengoaiak eskaintzen dituen funtzionalitate nagusiak izango ditugu hizpide. Berrito, jada JavaScript apur bat badakizula onartuko dugu eta soilik HTML5eko APIak ondo erabili ahal izateko funtzionalitate garrantzitsuenetan sakonduko dugu.

3.1 Aldagaien erazagupena

Nola deklaratu aldagai bat JavaScript-en? Zer erabili, *var*, *let* edo *const*? Galdera hauei erantzuten saiatuko gara, labur-labur.

Orain dela urte batzuk (*let* eta *const* existitzen ez zirenean ere), honela erazagutu behar ziren aldagaiak:

```
1 // Iruzkina
2 var zabalera ; // hasieratu gabeko aldagai baten
    ↪ erazagupena
3 var txapeldunak = 2;
4 var txapeldunak = 4; /* ez da batere gomendagarria aldagai
    ↪ bat ber-erazagutzea, baina 'var' erabiliz, posible
    ↪ da */
5 var irakitePuntua = 100;
6 var irakitePuntua = "Ehun"; /* aldagai mota dinamikoki
    ↪ alda daiteke. Berrito, ez da batere gomendagarria,
    ↪ baina egin daiteke */
7 var nahikoAdin = false;
8 var temp = 100, lema = "Kaixo";
9 temp = (temp - 12) * 5 / 9; // Adierazpen matematikoa
10 lema = lema + " mundua!";
```

```
11 | var pos = Math.random();
```

Arau nagusiak hiru dira:

- Aldagaiaren izena hizki batez (edo _ edo \$ karaktereez) hasten da.
- Jarraian, zenbakiak edo hizkiak edo _ edo \$ karaktereak kateatu daitezke, hainbat aldiz.
- Hitz erreserbatuak daude (gako-hitzak), erabili ezin direnak:

abstract, as, boolean, break, byte, case, catch, char, class, continue, const, debugger, default, delete, do, double, else, enum, export, extends, false, final, finally, float, for, function, goto, if, implements, import, in, instanceof, int, interface, is, long, namespace, native, new, null, package, private, protected, public, return, short, static, super, switch, synchronized, this, throw, throws, transient, true, try, typeof, use, var, void, volatile, while, with

Gaur egun, `var` ez erabiltzea gomendatzen da. Horren ordeaz `let` (aldagaiak deklaratzeko) eta `const` (konstanteak erazagutzeko) lehenesten dira. `Let` aldagai bat ezin da birritan erazagutu (saiatuz gero, errore bat jasoko dugu), baina haren balioa, ordea, bai, alda daiteke. `Const` baten kasuan, berriz, ezin da ez berrerezagutu, ezta haren balioa aldatu ere.

```
1 | // Let
2 | let zabalera ; // hasieratu gabeko aldagai baten
   | ↪ erazagupena
3 | let txapeldunak = 2;
4 | let txapeldunak = 4; // ERRORE bat jasoko dugu
5 | const PI = 3.14;
6 | PI = 3.1415; // ERRORE BAT jasoko dugu, jada konstantea
   | ↪ zehaztuta baitago
```

DevTools, let eta const



Adi! Garatzaileei laguntzeko eta JavaScript-en modu azkarean gauzak probatu ahal izateko Google Chrome-k bere Chrome DevTools kontsolan `let` aldagaiak behin baino gehiagotan erazagutzen uzten du. Firefox-en lan egiten baduzu, ez da hori gertatuko. Ikus: <https://umaar.com/dev-tips/214-redeclare-let-console/>.

3.2 *While* eta *for* begiztak

Iterazioak lortzeko (datu-egiturak zeharkatzeko edo baldintza bat betetzen den bitartean kode zati bat exekutatzeko), *while* eta *for* begiztak erabiltzen dira. Noizbait beste programazio-lengoaia batean programatu baduzu, ez duzu inolako arazorik izango kode bloke hauek ulertzeko:

```
1  const kafeKop = 5;
2
3  let i=0;
4  while (i < kafeKop) {
5    console.log("Kafea edan")
6    i = i + 1;
7  }
8
9  for (let i=0; i < kafeKop; i++){
10   console.log("Ebakia edan")
11 }
```

Ohar zaitez let erabili dugula *i* aldagaia erazagutzeko. Hala ere, *for* begizta blokearen barruan erazagutu duguna aldagai lokala da. Blokea amaitzean aldagai lokal hori ezabatuko da (3. lerroan definituta zegoena aldagai globala da, eta bere balioa mantenduko da).

3.3 Baldintzazko adierazpenak

if-then-else baldintzazko adierazpen klasikoa ere beste lengoaia batzuetan bezala erabiltzen da, honako patroiarri jarraituz:

```
if (baldintza) {
    baldintza betetzen denenean exekutatu beharreko kodea
} else {
    baldintza betetzen EZ denean exekutatu beharreko kodea
}
```

Adibidez:

```
1 | let emaitza, nahikoAdin = false;
2 |
3 | if (nahikoAdin) {
4 |   emaitza = "Gidatu ahal duzu!";
5 | } else {
6 |   emaitza = "Tamalez, oraindik ezin duzu gidatu.";
7 | }
8 |
9 | console.log(emaitza)
```

Adibidean, *if* blokearen *else* adarretik sartuko gara.

3.4 Arrayak

Arrayak datu multzo bat memorian gordetzeko datu-egiturak dira. Adibidez, bost zenbaki gorde nahi baditugu taula izeneko array batean:

```
1 | let taula = new Array();
2 | taula[0] = 5.1;
3 | taula[1] = 6.1;
4 | taula[2] = -2.5;
5 | taula[3] = -3;
6 | taula[4] = -4;
```

Edo modu laburrean idatzita (aurrekoaren baliokidea):

```
1 | let taula = [5.1, 6.1, -2.5, -3 -4];
```

Beste lengoia batzuetan array baten tamaina finkoa da bai eta gordetzen dituen datu motak ere. JavaScript-en ez da horrela, alegia, array baten tamaina ez da hasiera-hasieratik zehaztu behar (dinamikoa da) eta barruan gordetzen diren balioen datu motak ere ez du zertan berdina izan.

```
1 // JavaScript-en arrayak dinamikoak dira.
2 // Demagun taula aldagaia erazagutu eta hasieratu dugula
3
4 let taula = [];
5 // Jarraian, hau eginez gero:
6
7 taula[5] = 38;
8
9 // Orain taulan 6 posizio izango ditugu eta 6.ean (0-tik
10 // hasten dira indizeak) 38 zenbakia aurkituko dugu.
11 // Tartean, "undefined" elementuak.
12 // Beraz, orain taularen luzera eskatuz gero, 6 emango digu
13
14 console.log(taula.length); // Eraitza: 6
15
16 // zenbakiak gorde baditugu ere, hurrengo posizioan string
17   ↳ bat gorde dezakegu, arazorik gabe (eta honekin,
18   ↳ berriro, dinamikoki luzatu dugu arrayaren tamaina)
19
20 taula[6] = "Sagarra";
21
22 console.log(taula); // Eraitza: (7) [empty x 5, 38,
23   ↳ "Sagarra"]
```

Jarraian, **Array** klaseak eskaintzen dituen beste metodo interesgarri batzuk aztertuko ditugu:

concat, indexOf, join, lastIndexOf, pop, push, reverse, shift, slice, sort, splice, toString, unshift, valueOf

Elementuak kateatzeko (*concat*):

```
1 let batzuk = ["Carlos", "Lierni"];
2 let besteak = ["Eneritz", "Tadeo", "Linus"];
3 let denak = batzuk.concat(besteak);
4 // denak: ["Carlos", "Lierni", "Eneritz", "Tadeo", "Linus"]
```

Elementu baten indizea lortzeko (*indexOf*):

```
1 let taldeak = ["Athletic", "Barcelona", "Erreala"];
2 let ind = taldeak.indexOf("Erreala"); // ind = 2
```

Elementuak lotzeko (*join*):

```
1 let frutak = ["Banana", "Laranja", "Sagarra", "Mango"];
2 let energia = frutak.join(".");
3 // energia = "Banana.Laranja.Sagarra.Mango"
```

Elementu batek arrayan duen azken posizioaren indizea lortzeko (*lastIndexOf*):

```
1 let lengoaiak = ["Pascal", "Ada", "C++", "Java",
  ↪ "JavaScript", "Ada"];
2 lengoaiak.lastIndexOf("Ada"); // 5
```

Arrayaren azken elementua lortzeko eta arraytik ateratzeko (*pop*):

```
1 let paloak = ["urreak", "kopak", "ezpatak", "bastoiak"];
2 paloak.pop(); // "bastoiak"
3 // paloak: ["urreak", "kopak", "ezpatak"]
```

Arrayaren lehen elementua arraytik atera (*shift*):

```
1 let paloak = ["urreak", "kopak", "ezpatak", "bastoiak"];
2 paloak.shift(); // "urreak"
3 // paloak: ["kopak", "ezpatak", "bastoiak"]
```

Arrayaren azken elementua txertatzeko (*push*):

```
1 // push metodoak elementua azken posizioan txertatu eta
  ↪ hori egin ondoren arrayan zenbat elementu dauden
  ↪ bueltatzen du
2 let paloak = ["urreak", "kopak", "ezpatak"];
3 paloak.push("bastoiak"); // 4
4 // paloak: ["urreak", "kopak", "ezpatak", "bastoiak"]
```

Arrayaren elementuak iraultzeko (*reverse*):

```
1 let paloak = ["urreak", "kopak", "ezpatak", "bastoiak"];
2 paloak.reverse();
3 // paloak : ["bastoiak", "ezpatak", "kopak", "urreak"]
```

Dimentsio anitzeko arrayak ere sor daitezke (alegia, arrayez osatutako arrayak):

```
1
2 let elementuak = [[1,2],[3,4],[5,6]];
3 console.log(elementuak[0][0]); // 1 ematen du
```

10 x 20 elementuz osatutako array bat sortzeko adibidearekin bukatuko dugu:

```
1
2 let x = new Array(10);
3 for (let i = 0; i < 10; i++) {
4   x[i] = new Array(20);
5 }
6
7 x[5][12] = 3.0; // elementu bat txertatu (5,12) gelaxkan
```

3.5 Ariketak

Irakurri `var`, `let` eta `const` klausulen arteko ezberdintasunak lantzen dituen [dokumentu hau](#)¹.

1. Zer lortuko dugu pantailan honakoa idaztean?

```
var tester = "hey hi";  
function newFunction() {  
    var hello = "hello";  
}  
console.log(hello);
```

2. Posible al da hau egitea?

```
var greeter = "hey hi";  
var greeter = "say Hello instead";
```

3. Posible al da hau egitea errorerik jaso gabe? Baiezkoan, nola deitzen zaio JavaScript-eko ezaugarri honi (aldagai baten balioa kodean deklaratu baino lehenago erabiltzeari)?

```
console.log (greeter);  
var greeter = "say hello"
```

4. Zer bistaratuko da kontsolan kode hau exekutatzean?

```
var ind = 0;  
for(var ind=3; ind<10; ind++); // gorputzik gabeko  
    ↪ begizta  
console.log(ind);
```

5. Zer inprimatzen da kode honen amaieran? (adi, DevTools erabiltzen ari bazara, gogoratu fitxa berri bat irekitzeaz testuinguru berri batetik abiatzeko beti)

```
let ind = 0;  
for(let ind=3; ind<10; ind++); // gorputzik gabeko  
    ↪ begizta  
console.log(ind);
```

6. Honakoa egin daiteke errorerik jaso gabe?

¹https://dev.to/sarah_chima/var-let-and-const-whats-the-difference-69e

```
let agurra = "Hola";
let agurra = "Kaixo"
```

7. Eta hau?

```
var agurra = "Hola";
var agurra = "Kaixo";
```

8. Honakoa egin daiteke errorerik jaso gabe?

```
console.log (greeter);
let greeter = "say hello";
```

(Adi, ez da 3. ariketaren berdina)

9. Badago errorerik hemen? Zer lortzen dugu pantailan?

```
const AGUR="agur!"
AGUR="adiós!";
```

10. Badago errorerik hemen? Zer lortzen dugu pantailan?

```
const AGUR="agur!";
if (true) {
  const AGUR="adiós!";
}
console.log(AGUR);
```

11. Badago errorerik hemen? Zer lortzen dugu pantailan?

```
const hiztegia = {
  hola : "kaixo",
  casa : "etxea"
};
hiztegia.hola = "iepa!";
console.log(hiztegia.hola);
```

Bigarren ariketa multzoan arrayak erabiliko ditugu. Lasai hartu, arrayek JavaScripten badute bere zailtasuna eta.

1. Slice metodoa. Hurrengo kodea aztertu. Zer lortzen dugun sliceArr aldagaiari? arr aldagaiari aldatu egin da?

```
const arr = [1, 2, 3, 4, 5];
const slicedArr = arr.slice(1, 4);
```

2. **Splice** metodoa. Hurrengo kodea aztertu. Zer lortzen dugun `splicedOut` aldagaian? arr aldagaia aldatu egin da?

```
const arr = [1, 2, 3, 4, 5];  
const splicedOut = arr.splice(1, 2, 'a', 'b');
```

3. **Unshift** metodoa. Hurrengo kodea aztertu. Zer lortzen dugun `newLength` aldagaian? eta `anotherLength` aldagaian? arr aldagaia aldatu egin da?

```
const arr = [2, 3, 4];  
const newLength = arr.unshift(1);  
const anotherLength = arr.unshift(-2, -1, 0);
```

4. **toString** metodoa. Hurrengo kodea aztertu. Zein da kontsolan lortzen dugun emaitza?

```
let paloak = ["urreak", "kopak", "ezpatak",  
             ↪ "bastoiak"]  
paloak.toString()
```

5. Demagun 3 puntuz osatutako arraya sortu dugula:

```
1 function Point(x,y) {  
2   this.x = x;  
3   this.y = y;  
4 }  
5 let puntuak = [new Point(5,0), new Point(11,1), new  
                 ↪ Point(2,2)]
```

Programa ezazu array horren barruan dauden eta x koordinatua > 10 balioa duten puntuak ezabatzeko *scripta*. Proba ezazu zure soluzioa array hauekin ere:

```
1 let puntuak = [new Point(5,0), new Point(11,1), new  
                 ↪ Point(15,1), new Point(2,2)];  
2 let puntuak = [new Point(5,0), new Point(4,1), new  
                 ↪ Point(5,2), new Point(6,0), new Point(11,1), new  
                 ↪ Point(15,2)];
```

6. Programa ezazu *script* bat puntuen arraya x koordinatuaren arabera ordenatzeko, goranzko hurrenkera jarraituz.

4. JavaScript eta DOM

DOM, *Document Object Model* edo Dokumentuaren Objektu Eredua XML edo HTML dokumentu bat hainbat nodoz osatutako zuhaitz-egitura bat bezala tratatzen duen interfazea da. Nodo bakoitzean dokumentuaren zati bat irudikatzen duen objektu bat dago. Beste era batera esanda, DOM interfazeak dokumentu bat zuhaitz logiko batez irudikatzen du. DOM eredua JavaScript bidez nola atzitu eta nola alda dezakegun ikasiko dugu gai honetan, modu dinamikoan web orri baten edukia aldatu ahal izateko.

Zuhaitz logiko horren barruan dauden objektuak JavaScript erabiliz atzitu, irakurri, aldatu edo ezabatu ditzakegu. Nabigatzaileak automatikoki eta berehala interpretatuko ditu DOM zuhaitzean egindako aldaketak.

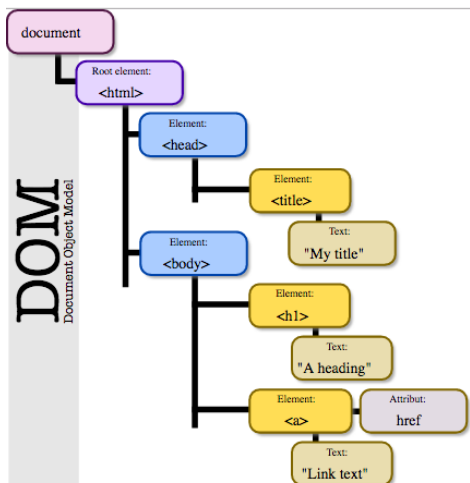
Adibide batekin ikusiko dugu. Demagun HTML orri soil hau kargatu nahi dugula:

```
<!DOCTYPE html>
<html>
<head>
<title>My title</title>
</head>
<body>
<h1>A heading</h1>
<a HREF="">Link text</a>
</body>
</html>
```

Horri dagokion DOM zuhaitza 4.1. irudian ikus dezakegu.

4.1 DOM aldatzen JS bidez

Demagun 4.2. irudiko DOM zuhaitza dugula. Bertan, JS erabiliz, hirugarren paragrafoaren (`<p>`) testua aldatu nahi dugu, "estás" jarri ordez, "zaude?" ager dadin.

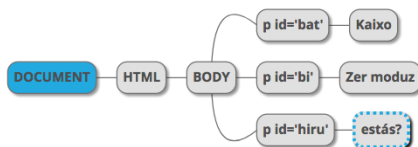


4.1. irudia: DOM Eredua. Iturria:

<http://upload.wikimedia.org/wikipedia/commons/5/5a/DOM-model.svg>.

Nola egin?

Lehenengo *hiru* izeneko identifikatzailea duen objektua atzitu dugu, `document.getElementById()` metodoaz. Metodo horrek DOM zuhaitza zeharkatuko du parametro gisa pasatu diogun identifikatzailea bilatuz. Identifikatzailea bakuna denez, soilik elementu bat aurkituko du. Objektu horrek hainbat metodo eta atributu izango ditu, adibidez "innerHTML" atributua, objektuaren barruko HTMLa atzitu edo editatzeko aukera emango diguna. Beraz, "estás" hitza 4.1. listatuko kodearekin lortu beharko genuke. Probatuko dugu ea egia den...



4.2. irudia: DOM interfazea ulertzeko ariketaren testuingurua.

```
let osagai = document.getElementById("hiru");
osagai.innerHTML = "zaude?";
```

4.1. listatua: `getElementById` erabil dezakegu DOM zuhaitzeko objektu bat atzitzeko.

4.2 Nola txertatu JS script bat zure orrian

Aurreko kodea index.html izeneko orri batean txerta dezakegu. Bi modu daude, *inline* deritzona edo erreferentzia gisa.

```
1 <!DOCTYPE html>
2 <html lang="eu-es">
3   <head>
4     <meta charset="utf-8" />
5     <script>
6       let osagai = document.getElementById("hiru");
7       osagai.innerHTML = "zaude?";
8     </script>
9   </head>
10  <body>
11    <p id="bat">Kaixoooo</p>
12    <p id="bi">Zer moduz</p>
13    <p id="hiru">estás?</p>
14  </body>
15 </html>
```

4.2. listatua: *inline* izeneko moduan kodea zuzenean txertatzen da <script> etiketen artean. Ikus: <https://codesandbox.io/s/4-gaia-dom-k701t?file=/index.html>.

```
1 <!DOCTYPE html>
2 <html lang="eu-es">
3   <head>
4     <meta charset="utf-8" />
5     <script src="scripta.js"></script>
6   </head>
7   <body>
8     <p id="bat">Kaixoooo</p>
9     <p id="bi">Zer moduz</p>
10    <p id="hiru">estás?</p>
11  </body>
12 </html>
```

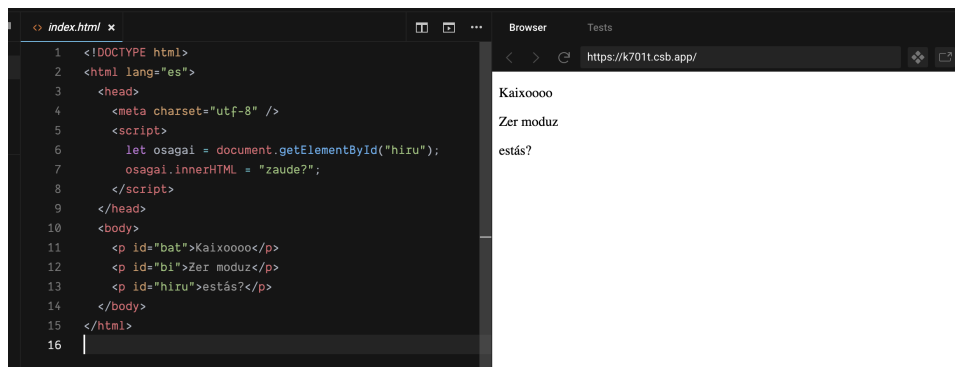
4.3. listatua: JS kodea erreferentziaz ere txerta dezakegu. Horretarako, JS fitxategiaren izena pasatuko diogu <script> etiketari, src atributuan.

Bi kode zati horiek probatuz gero, funtzionatzen ez dutela ikusiko dugu (ikus 4.3. irudia). Zergatik? Nabigatzaileak ikusi eta exekutatu duen lehenengo gauza (kodea irakurritz goitik hasita) <head> blokean dagoena delako. Bertan gure <script>-a exekutatu du, baina oraindik ez du orriaren gorputza (body-a) kargatu! Hori dela eta, ezin du aldaketarik egin. Izan ere, kontsola irekitzen badugu, honako errorea ikusiko dugu:

```
Uncaught TypeError: Cannot set property 'innerHTML' of
↳ null at (index):9
```

Alegia, nabigatzailea saiatu da osagaia aurkitzen `document.getElementById("hiru")` eginez, baina ez du ezer topatu. Beraz, `osagai = null` da, eta, hortaz, `osagai.innerHTML = "zaude?"`; egiten saiatzean, errorea.

Nola konpondu? Nola esan diezaiokegu nabigatzaileari JavaScript kodea orriaren gorputza kargatu eta gero exekutatu behar duela? Horretarako jaio zen `onload` gertaera-kudeatzailea. Azter dezagun hurrengo atalean.



4.3. irudia: DOM edukia aldatzen saiatu gara, baina ez dugu lortu. Zergatik izango ote da... Gai honetan azalduko dugu.

4.2.1 onload gertaera-kudeatzailea

```
1 <!DOCTYPE html>
2 <html lang="eu-es">
3   <head>
4     <meta charset="utf-8" />
5     <script>
6       function edukiaAldatu() {
7         let osagai = document.getElementById("hiru");
8         osagai.innerHTML = "zaude?";
9       }
10      window.onload = edukiaAldatu;
11    </script>
12  </head>
13  <body>
14    <p id="bat">Kaixoooo</p>
15    <p id="bi">Zer moduz</p>
```

```
16     <p id="hiru">estás?</p>
17   </body>
18 </html>
```

4.4. listatua: `window.onload` egitean gertaera-kudeatzaile bat ezartzen ari gara. Kasu honetan kudeatzailea orriaren edukia kargatu ostean exekutatu egin behar dela esaten ari gara. Ikus: <https://codesandbox.io/s/4-gaia-onload-x7wkr?file=/index.html>.

`Window` objektuak `onload` gertaera-kudeatzailea dauka, besteak beste (`window.onload`). Horri esker nabigatzailearen portaera kudea dezakegu. Zehazki, orriaren edukia kargatzen bukatzen denean, *loaded* gertaera altxatuko da. Gertaera hori antzeman eta `edukiaAldatu` izeneko funtzioarekin tratatu nahi dugula esaten ari gara lerro honekin:

```
window.onload = edukiaAldatu;
```

Noski, DOM edukia aldatzeko prestatuta genuen kodea orain `edukiaAldatu` izeneko funtzioan sartu dugu:

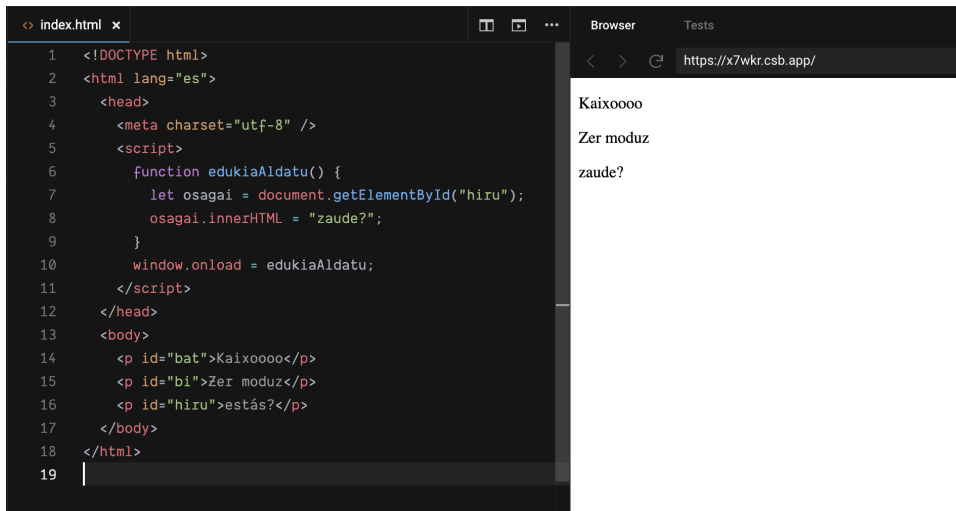
```
1 function edukiaAldatu() {
2     let osagai = document.getElementById("hiru");
3     osagai.innerHTML = "zaude?";
4 }
```

Eta orain bai, dena ondo dabil (ikus 4.5. irudia). Gertaera mota ezberdin interresgarri askoz gehiago daude. Izan ere, horretarako beste gai bat bereziki prestatu dugu liburu honetan (ikus 6. **Gertaerak** kapitulua).

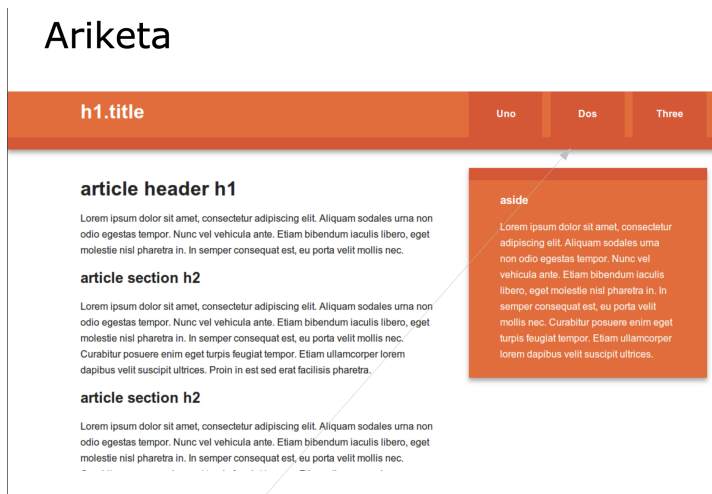
4.3 Ariketak

Esteka honetan <https://www.dropbox.com/s/7kq7nm8j6m3o9zh/initializr.tgz?dl=1> hurrengo ariketa egiteko beharrezkoak diren fitxategiak aurkituko dituzu. Jaitsi, destrinkotu eta zure nabigatzailean `index.html` orria ireki.

Bertan, webgune baten HTML kodea ematen zaizu. Zure eginbeharra: JS script baten bidez orri nagusian dagoen goiburua ordezkatu, "Uno", "Dos", "Three" katearen ordez "Bat", "Bi", "Hiru" katea ager dadin.



4.4. irudia: DOM edukia aldatzeko, lehenengo eta behin, orriaren edukia kargatu behar da. Hori lortzeko *window.onload* gertaera-kudeatzailea prestatu dugu.



4.5. irudia: DOM edukia aldatzeko JS bat prestatu beharko duzu.

5. Objektuetara zuzendutako programazioa JavaScript-en

5.1 Klaseak JavaScript-en

JavaScript ez da berez objektuetara zuzendutako programazio (OZP) lengoaia bat, baizik prototipoetan oinarritutakoa. Azken paradigma hori erabilita badago objektuak, herentzia, klaseak, etab. sortzea. Alegia, posible da objektuetara zuzendutako programazio-paradigmari jarraitzea, JavaScript-en helburua beste bat bada ere. Dena den, OZPa hain arrakastatsua bilakatu da JSn, non ECMAScript 2015¹ bertsioan klaseak modu erraz batean programatzeko sintaxia onartu baitzen. Sintaxi berri horren bidez, klaseak, objektuak eta klaseen arteko herentzia inplementatzeko kodea erraztu egiten da erabat.

5.1.1 Klaseen erazagupena

Lauki izeneko klase bat erazagutzeko honako egiturari jarraituko diogu:

```
class Lauki {  
  constructor(altuera, zabalera) {  
    this.altuera = altuera;  
    this.zabalera = zabalera;  
  }  
}
```

Adibidean, Lauki klaseak eraikitzaile metodo bat dauka, `constructor` izeneko (eraikitzailea beti deituko da *constructor*). Barruan bi atributu deklaratu dira: `altuera` eta `zabalera`. Ohart zaitez atributuak erazagutzeko `this` gako-hitza erabili dela, hots, `this.altuera` eta `this.zabalera`. Atributu hauen balioak esleitzeko, eraikitzaileari parametro gisa pasatu dizkiogun aldagaien balioak erabili ditugu.

¹<https://262.ecma-international.org/6.0/>

Berez, JSn, klase (`class`) bat funtzio berezi bat besterik ez da. Funtzio horri `new` gako-hitza erabiliz deitzen badiogu, funtzio horrek irudikatzen duen klasea instantziatuko dugu, objektua sortuz. Adibidez, Lauki klaseko `laukia` izeneko objektu bat instantziatzeko:

```
let laukia = new Lauki(3,4);
```

Funtzio arruntetan gertatzen den lez, klase-espresioak ere erabil daitezke:

```
let laukiKlasea = class Lauki {
  constructor(altuera, zabalera) {
    this.altuera = altuera;
    this.zabalera = zabalera;
  }
};

let laukia = new laukiKlasea(3,4);
```

5.1.2 Get metodoak

Hurrengo adibidean `zabalera()` metodoa `get` hitzaz apaindu dugu. `get` gako-hitza da, sintaxia errazteko beste teknika bat. Horrela, `laukia.zabaleraKalkulatu()` bezalako deia erabili ordez, zuzenean `laukia.zabalera` erabili ahalko dugu. Gauza bera `set` metodo bereziarekin. `set` metodo bat erazagutuz, posible da balio bat esleitzea edozein atributuri modu laburrean (adib. `laukia.altuera = 10;`)

```
class Lauki {
  constructor(altuera, zabalera) {
    this.altuera = altuera;
    this.zabalera = zabalera;
  }

  set altuera(balioBerria) {
    this.altuera = balioBerria;
  }

  get zabalera() {
    return this.zabaleraKalkulatu();
  }

  zabaleraKalkulatu() {
    return this.altuera * this.zabalera;
  }
}
```

```
const laukia = new Lauki(5, 10);
console.log(laukia.zabalera); // (get metodoari deitu
    ↪ ondoren, emaitza=50)
laukia.altuera = 2; // set
console.log(laukia.zabalera); // (get metodoari deitu
    ↪ ondoren, emaitza=20)
```

5.1.3 Herentzia

Herentzia implementatzeko sintaxia ere erraztu egin du ES2015 estandarrak, **extends** gako-hitzari esker.

```
class Karratu extends Lauki {
  zabaleraKalkulatu() {
    return this.altuera**2;
  }
  get alde() {
    return this.altuera;
  }
}

let karratu = new Karratu(10,10);
console.log(karratu.alde); // 10
console.log(karratu.zabalera); // 100
```

Ohart zaitez *Karratu* klaseko **zabaleraKalkulatu()** metodoan. Gainidatzitako metodo bat da, berez bere gurasoa (*Lauki* klasea) duen metodoa deuseztatu eta *Lauki*ak bere metodo propioa inplementatzen du. *Karratu* klaseak *set zabalera()* metodoa ere badu, *Lauki* klasetik heredatu egin duelako.

5.2 Ariketak

Objektuetara zuzendutako programazioa erabili beharko duzu hainbat ariketa JavaScript-en programatzeko.

1. ES6 sintaxia erabiliz, inplementatu *Puntu* izeneko klase bat, bi dimentsioko espazioan dagoen puntu bat irudikatzen duena. *Puntu* batek bi atributu ditu, **x** eta **y**, metodo eraikitzailetik pasatuko dizkiogunak. *batu()* izeneko metodo bat ere badu, parametro gisa beste puntu bat hartzen duena eta emaitza gisa bi puntuen batura itzultzen duena. Alegia, beste puntu berri bat bueltatzen du, non **x** balioa bi puntuen **x** balioen batura den (eta gauza bera **y** puntuari dagokionez).

Beraz, kode hau exekutatzean:

```
console.log(new Puntu(1, 2).batu(new Puntu(2, 1)))
```

Pantailan honako emaitza jaso behar dugu:

```
Puntu{x: 3, y: 3}
```

2. Gehitu *hurrengoa()* izeneko metodo bat kontagailu objektuari. Metodo horrek kont aldagaiaren uneko balioa itzuli behar du eta unitate batean handitu (erabili `++` eragilea)

```
let kontagailu = {
  kont: 0
  [zure kodea hemen]
}
console.log(kontagailu.hurrengoa())
// -> 0
console.log(kontagailu.hurrengoa())
// -> 1
console.log(kontagailu.hurrengoa())
// -> 2
```

3. `map()`, `filter()` eta `reduce()` programazio funtzionalean asko erabiltzen diren array klaseko hiru metodo dira. Webgune honetan <https://labur.eus/vGIW4> (medium.com) ikus ditzakezu metodo horien oinarritzko erabileraren adibideak. Horren inguruan zenbait ariketa proposatzen dira.

Honako objektu literalak emanik (biltegia izenekoa):

```
const biltegia = [
  {mota: "garbigailua", balioa: 5000},
  {mota: "garbigailua", balioa: 650},
  {mota: "edalonztzia", balioa: 10},
  {mota: "armairua", balioa: 1200},
  {mota: "garbigailua", balioa: 77}
]

let guztiraGarbigailuenBalioa = zure kodea hemen;

console.log ( guztiraGarbigailuenBalioa ); // 5727
↪ erantzuna espero da
```

Erabili `filter()` eta `reduce()` biltegiko garbigailuen balio totala lortzeko.

4. Gezi funtzioa (*arrow function*) $\Rightarrow$ ES6 estandarrak ekarri duen eta oso preziatua den funtzioa bat da. Irakurri gehiago beraren inguruan hemen: <https://labur.eus/1XZv0> (sitepoint.com).

Jarraian, moldatu hurrengo kodea, `arrow` funtzioa erabili dezan:

```
biltegia.forEach( function(item) {  
  console.log(item.balioa);  
});
```

5. Egin hirugarren ariketaren errefaktORIZAZIOA, *arrow* funtzioa erabil dezan.

6. Gertaerak eta gertaera-kudeatzaileak

Web orri baten barruko elementuen egoera aldatzean, gertaerak altxatzen dira. Adibidez, botoi baten gainean klik egitean, botoiaren egoera aldatu dela adierazteko, nabigatzaileak gertaera bat altxatuko du (*onclick* gertaera, hain zuzen ere). Edo orri baten elementu guztiak kargatzen bukatzean, *onload* gertaera altxatuko da. Gertaerak tratatu nahi izanez gero, gertaera-kudeatzaile bat programatu behar-ko dugu. Ezezkoan, ez dugu ezer egin behar, alegia, gertaera altxatuko da, baina ez badu inork tratatzen, ez da ezer gertatuko.

6.1 Gertaera motak

Adibideetan ikusi dugun bezala, bi gertaera mota daude: erabiltzaileak sortutakoak (botoi baten gainean klik egin, sagua mugitu, leihoa handitu, tekla bat sakatu...) eta sistemak automatikoki sortutakoak (orri baten elementuak kargatzen bukatzean, bideo bat bistaratzeko prest dagoenean, errore bat gertatzen denean...). Azter ditzagun horrelako gertaera zehatzen adibide batzuk.

6.1.1 Erabiltzaileak sortutako gertaerak

Maiz erabiltzen diren gertaera mota hauen adibideak:

- onclick (saguarekin elementu baten gainean klik egitean)
- onmouseover (sagua elementu baten gainean jartzean)
- onblur (elementu batek fokua galtzen duenean)
- onfocus (elementu batek fokua hartzen duenean)
- ondrag (elementu bat arrastatzean)
- onkeypress (tekla bat sakatzean)

6.1.2 Sistemak sortutakoak

Maiz erabiltzen diren gertaera mota hauen adibideak:

- `onload` (elementu bat kargatzen bukatzean)
- `oncanplay` (elementu multimedia bat kargatu eta ikusi edo jo dezakegunean)
- `onoffline` (konexioa galtzean)

6.2 Nola kudeatu gertaerak JavaScript-en

Hiru modu ezberdin daude gertaerak JSn kudeatzeko. Kudeatzaileak HTML kodean txertatuz, gertaeraren iturri-osagaiaren gainean kudeatzaile bat zuzenean definituz edo metodo generikoa erabiliz.

JavaScript HTML kodean txertatu

Elementuaren ostean *onclick* atributuan zer funtzio exekutatu nahi dugun zehaztuko dugu, adibidez:

```
<button onclick="alert('Hello world');">
  Sakatu hemen
</button>
```

Osagaiaren gainean kudeatzailea zuzenean definitu

Askotan gomendagarria da gertaera-kudeatzaileak JS fitxategi batean gordetzea eta funtzio horiek osagaiaren izena erabiliz zehaztea, honelako patroiarri jarraituz:

```
osagaia.gertaera = function()
{
  // gertaera kudeatzeko kodea
}
```

Adibidez, aurreko adibide bera honela programatu dezakegu:

```
botoia.onclick = function()
{
  //
  // gertaera kudeatzeko kodea
};
```

addEventListener metodo generikoa erabiliz

addEventListener metodoak 2 parametro hartzen ditu¹, tratatu nahi dugun gertaeraren izena eta funtzio-kudeatzailearen izena, eskema honi jarraituz:

```
osagaia.addEventListener('gertaera', funtzioa);
```

Gauza bera gertaera-kudeatzaile bat elementu batetik ezabatu nahi izanez gero. Kasu honetan, *removeEventListener* metodoa erabiliko dugu.

Adibidez, botoiaren gainean sakatzean funtzio bat exekutatzeko:

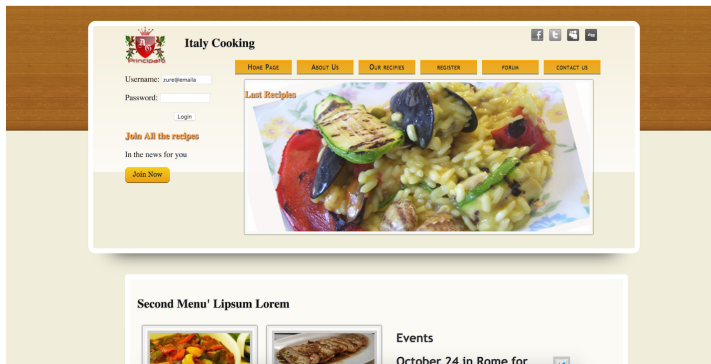
```
botoia.addEventListener('click', funtzioarenizena);
```

Eta momentu batean esleipen hori kendu nahiko bagenu, *removeEventListener* erabiliko genuke:

```
botoia.removeEventListener('click', funtzioarenizena);
```

6.3 Adibide praktikoa

Demagun 6.1. irudian ikusten den bezalako webgune bat dugula. Irudian zenbait errezetaren argazkiak agertuko dira, sekuentzian. Gure helburua errezeta baten gainean klik egitean, kontsolan mezu bat agertzea da.



6.1. irudia: Gertaerak: adibide praktikoa.

```
function kudeatzaileakHasieratu()
{
    let irudia = document.getElementById('irudia');
```

¹*addEventListener* metodoa gainkargatuta dago, alegia, 2 parametro baino gehiagoko bertsioak ere baditu. *addEventListener* eta *removeEventListener* inguruan informazio gehiago MDNn aurkituko duzu (<https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener>).

```
irudia.onclick = function() {  
    console.log("Irudia sakatu duzu");  
}  
}  
  
window.onload = kudeatzaileakHasieratu;
```

Garrantzitsua da *window.onload*-en gertaera-kudeatzailea zehaztea. Horrela egingo ez bagenu, eta zuzenean `kudeatzaileakHasieratu` metodoari deituko bagenio, posible litzateke irudia oraindik kargatu gabe egotea, eta, hortaz, `getElementById()` egitean *null* aurkitzea.

Gertaerak kudeatzen: *onblur*, *onfocus*

Aurreko irudian ikusten dugun inprimakian erabiltzailearen datuak (email-helbide eta pasahitza) eskatzen dira. Helbide elektroniko bat testu-eremu batean idatzi behar duela gogorarazteko trikimailu bat erabil dezakegu *onblur* gertaeraz baliatuz. Erabiltzaileak email-helbidea ez badu oraindik idatzi eta horretarako dagoen testu-eremua hutsik badago, orduan "zure@emaila" bezalako argibidea idatziko dugu bertan testu-eremu horrek fokua galtzean (*onblur*). Berriz, testu-eremu horrek berriro fokua hartzen duenean (*onfocus*), eta oraindik email-helbidea idatzi gabe badu, balioa hustu egingo dugu.

```
1 let erabiltzaile = document.getElementById('erabiltzaile');  
2 erabiltzaile.value = 'zure@emaila';  
3  
4 erabiltzaile.onfocus = function(event) {  
5     let input = event.target  
6     if (input.value == 'zure@emaila') {  
7         erabiltzaile.value = '';  
8     }  
9 }  
10  
11 erabiltzaile.onblur = function(event) {  
12     let input = event.target  
13     if (input.value == '') {  
14         erabiltzaile.value = 'zure@emaila';  
15     }  
16 }
```

Gertaerak kudeatzen: *onchange*

Erabiltzaileak aukera-zerrenda (*combobox*) bateko aukeraren bat hautatzean *onchange* gertaera altxatzen da. Zerrendan hainbat osagai daude <select> etiketaren barruan:

```
<option value="rice">Arroza</option>
<option value="mushrooms">Txanpinoiak</option>
```

Hurrengo adibidean *zerrendaKudeatzaile* funtzio bat prestatu dugu gertaera horri erantzuteko. Zehazki, aukeratu den elementuaren balioa (*value* atributuan dagoena), elementu horren indizea zerrendan (*selectedIndex*) eta aukera horren testua (adibidez, Arroza).

```
1 let item = document.getElementById('combobox');
2 item.addEventListener('change', zerrendaKudeatzaile);
3
4 function zerrendaKudeatzaile(event) {
5     let item = event.target
6     console.log(item.value); // rice
7     console.log(item.selectedIndex); // 2
8     console.log(item.options[item.selectedIndex].text); //
9         ↳ Rice with mushrooms
10 }
11
12 <select name="GoMenu" style="width:159px" id="combobox">
13 <option selected="selected">Select a recipe</option>
14 <option class="_self" value="spaghetti">Spaghetti with
15     ↳ aubergine</option>
16 <option class="_self" value="rice">Rice with
17     ↳ mushrooms</option>
18 <option class="_self" value="rolls">Rolls with
19     ↳ polenta</option>
20 <option class="_self" value="chicken">Chicken
21     ↳ Hunter</option>
22 <option class="_self" value="tortiglioni">Tortiglioni
23     ↳ filled</option>
24 <option class="_self" value="swordfish">Smoked
25     ↳ swordfish</option>
26 <option class="_self" value="pumpkins">Stuffed
27     ↳ pumpkins</option>
28 </select>
```

Gertaerak kudeatzen: *onsubmit*

Inprimaki baten *onsubmit* gertaera inprimakiaren “Bidali” botoian sakatzean altxatzen da. Gertaera hori erabil dezakegu inprimakian sartu diren balioak zuzenak diren edo ez egiaztatzeko (zerbitzarira bidali baino lehen). Horretarako, funtzio bat esleituko diogu *onsubmit* kudeatzaileari. Funtzio horrek parametroak aztertu eta denak zuzenak badira, `true` bueltatu behar du. `False` kontrako kasuan.

```
1 let inprimakia = document.getElementById('inprimakia');
2 inprimakia.onsubmit = function() {
3   console.log("Bidali botoian klikatu duzu");
4   // eremuak baliostatu.
5   // Baten bat bete gabe balego, false itzuli
6   // Bestela, true itzuli
7   return false;
8 }
```

Gertaerak kudeatzen: *onchange range*

Inprimaki baten *range* motako osagaiak *onchange* gertaera altxatzen du erabiltzaileak tiraderatik tira egiten duenean alde batera edo bestera. *Range* osagaiaren balioa une oro ikusi ahal izateko, gertaera horri honako kudeatzailea esleitu diezaiokegu:

```
1 <form>
2   <input type='range' min=1 max=100 step=1 id='tarte'>
3 </form>
4
5 <div id='balioa'></div>
6 <script>
7 let kudeatzaileHasieratu = function()
8 {
9   let tarte = document.getElementById('tarte');
10  tarte.addEventListener('change', balioaBistaratu)
11
12  function balioaBistaratu( event ){
13    let tarte = event.target
14    document.getElementById('balioa').innerHTML =
15      ↳ tarte.value;
16  }
17 }
18 window.onload = kudeatzaileaHasieratu;
19 </script>
```

Gertaerak kudeatzen: *setTimeout*, *setInterval*

Bukatzeko, aldiro-aldiro funtzio bat exekutatu nahiko bagenu (adibidez, 5 segundotan behin funtzio bat exekutatu nahiko bagenu), oso interesgarria litzateke *setInterval* metodoa. Bi parametro hartzen ditu, exekutatu nahi den funtzioaren izena eta zer maiztasunez exekutatu nahi dugun, milisegundotan.

Adibidez, idatzi() izeneko funtzioa 5 segundotan behin exekutatu nahiko bagenu:

```
1 function idatzi() {  
2     console.log("Idazten..." + new Date());  
3 }  
4 let erlojua = setInterval(idatzi, 5000);
```

Beste batzuetan ez dugu aldiro-aldiro exekutatu nahiko, baizik eta behin bakarrik X segundo pasa ostean. Horretarako, *setTimeout* funtzioa erabiliko dugu. Adibidez, idatzi funtzioa 5 segundo barru exekutatzeko (ez aldiro, baizik eta soilik behin, 5 segundo barru):

```
1 function idatzi()  
2 {  
3     console.log("Idazten...." + new Date());  
4 }  
5 let reloj = setTimeout(idatzi, 5000);
```

6.4 Ariketak

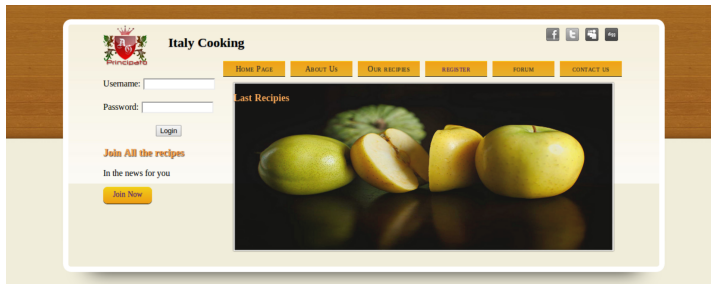
Gai honen inguruan planteatzen diren ariketak webgune baten gainean egin behar dira. Jaitsi beharrezkoak diren txantiloia <https://labur.eus/2DWcm> (ikasten.io) eta irudiak <https://labur.eus/BiAms> (ikasten.io) eta jarrai iezaiezu ariketa egiteko argibideei.

- JavaScript erabiliz dinamikoki alda daiteke <div> etiketa baten barruan zehazten den irudia. Adibidez, 6.2. irudian dugun webgunearen kasuan, errezeta-ren irudia alda dezakegu.

Horretarako, honako kodea erabiliko dugu:

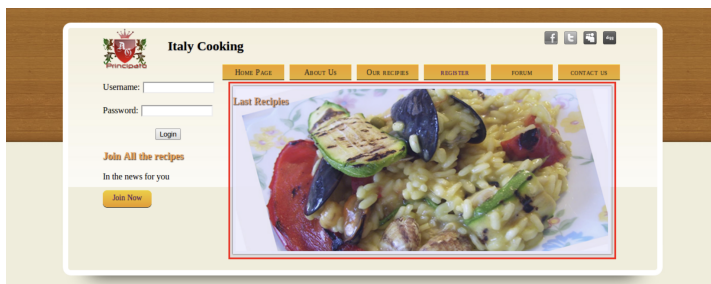
```
let irudia = document.getElementById("irudia")  
irudia.style.backgroundImage =  
    ↪ "url(irudiak/irudiberria.jpg)"
```

Ariketa honetan irudi hori 5 segundotan behin aldatzea eskatzen da (ikus 6.3. irudia). Sei irudiz osatutako array batetik hartuko dugu hurrengo irudia. Alegia, gai honetan ikusi dugun webgunean oinarrituta, beharrezkoa



6.2. irudia: Gertaerak: ariketaren hasierako egoera.

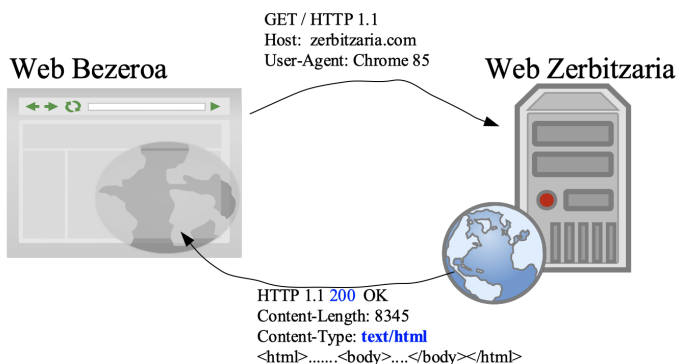
den JavaScript kodea programatu irudi guztiak sekuentzian bistaratzeko (5 segunduan behin). Erabiltzaileak irudiaren gainean sakatzen badu, irudien animazioa eten egin behar da. Irudiaren gainean ez badu sakatzen, 6. irudiaren ondoren berriro lehenengoa bistaratuko da, sekuentziari jarraituz (ikus: <http://www.youtube.com/watch?v=aaoNMmnfGD4>).



6.3. irudia: Gertaerak: ariketaren helburu-egoera.

7. Datuen komunikazio asinkronoa: JSON, Ajax eta Promesak

Nabigatzaileek kanpoko zerbitzariei web orriak eta gainontzeko elementuak eskatzeko HTTP protokoloa erabiltzen dute. Besterik ezean, nabigatzaileak eskaera sinkronoak egiten ditu: orria eskatu eta HTML kodearen zain geratzen da. Jaso ondoren, URLa aldatu eta beste orri bat eskatu ahalko dugu (aurrekoa gainidatziz).



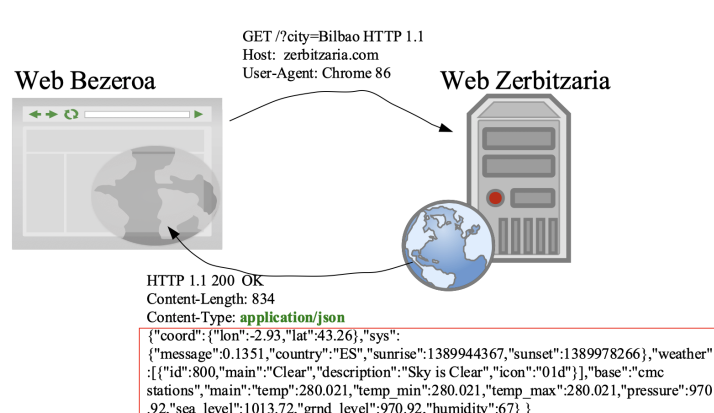
7.1. irudia: HTTP protokoloa eskaera sinkronoak eta asinkronoak egiteko erabil dezakegu.

Baina batzuetan, behar dugun gauza bakarra ez dira web orri osoak izango, datu gordinak baizik. Are gehiago, batzuetan web orri baten barruan soilik datu zehatz bat bistaratu nahiko dugu, eta datua jasotzean ez dugu web orria zapaldu nahi, baizik eta datu hori pantailan ikusten dugun orriarekin integratu.

Adibidez, demagun Bilbo hiriaren GPS geokopakena non dagoen jakin nahi dugula. OpenWeatherMap izeneko zerbitzu bati, munduko hirien latitudea eta longitudea gordetzen dituen web zerbitzu bati, deitu diezaiokegu horretarako. Irudian

ikusten dugun bezala, haren erantzuna ez da HTML formatuan etorriko, JSON formatuan (JavaScript Object Notation) baizik. Horrez gain, eskaera ez dugu HTTP sinkrono bat erabiliz jaso nahi, baizik eta eskaera asinkrono batekin. Alegia, pantailan dugun orriaren edukia ordezkatu gabe, nabigatzaileak eskaera bat egingo dio OpenWeatherMap zerbitzuari eta JSON erantzuna jasotzean, Bilboko koordinatuak orrian bertan txertatu, dagokion lekuan.

HTTP eskaera asinkrono hauek XHR (XMLHttpRequest) edo AJAX izenarekin ezagutzen dira.



7.2. irudia: HTTP protokoloa erabiliz eskaera asinkronoak egiteari AJAX edo XHR deritzo. Adibidean, XHR dei bat egin diogu web zerbitzariari eta hark erantzuna bidali digu JSON formatuan.

Adi! XHR eskaerak egitean ez dugu zertan XML erabili. Hasieran XML Interneteko formatu estandarra bazen ere, gaur egun gehien erabiltzen den formatua JSON da eta, berez, XHR eskaeretan XML edo JSON jaso dezakegu.

Orain dela urte batzuk AJAX edo XHR eskaera bat egiteko kodea nahiko korapilatsua zen:

```

1 // zerbitzaria eta eskaera zehaztu
2 let url = "http://api.openweathermap.org/data/2.5/weather?
    ↪ q=Bilbao,es";
3
4 // Eskaera kudeatzeko XHR objektua sortu
5 let kontsulta = new XMLHttpRequest();
6
7 // URL horri eskaera egiteko GET metodoa erabiliko dugula
    ↪ zehaztu
8 kontsulta.open("GET", url);
9
10 // eskaera asinkronoaren erantzuna kudeatuko duen
    ↪ funtzioa zehaztu
11 kontsulta.onload = function() {
12     if (kontsulta.status == 200) {
13         console.log("Arrakastaz egikaritutako kontsulta");
14         console.log( kontsulta.responseText );
15     }
16 };
17 // bukatzeko, eskaera jaurti besterik ez zaigu falta
18 kontsulta.send();

```

AJAX eskaera baten erantzuna JSON formatuan baldin badator ere (ikus 7.2. irudia), *string* huts bat da. Modu eroso batean tratatu nahi badugu, JSON objektu bihurtu beharko dugu. Adibidez, aurreko kode zatian, *kontsulta.responseText* katea JSON objektu bihurtzeko, *JSON.parse* metodoa erabili beharko dugu.

```
let erantzuna = JSON.parse(kontsulta.responseText);
```

Gaur egun, kode hori guztia asko laburbil daiteke *fetch* APIarekin. Izan ere, aurreko AJAX eskaera egiteko eta JSON formatura bihurtzeko, nahikoa litzateke lerro bakar hau exekutatzea (ikus 7.3. irudia):

```
fetch("http://api.openweathermap.org/data/2.5/weather?
q=Bilbao,es").then( r => r.json())
```

7.1 Promesak

Promes baten funtzionamendua ondo ulertzeko adibide bat ekarriko dugu. Irudi bat kargatu nahi dugu dinamikoki, promes baten bidez. Irudia deskargatu eta prest dagoenean dokumentuari erantsiko diogu DOM erabiliz, honela:

alegia, badakigu noiz hasten den (komandoa exekutatzen hasten denean), baina ez noiz bukatzen den zehazki (denbora gutxiago edo gehiago har dezake irudia deskargatzeko konexio-kalitatearen arabera, adibidez).

Hurrengo irudian (7.3. irudian), exekutatu dugun promesaren emaitza irudikatzen da. Zehazki, `fetch()` metodoaz dei asinkrono bat egin dugu eta promesa bat jaso dugu. Promesa betetzean (`fetch().then( r )` klausularen barruan gaudenean, non `r` jaso dugun erantzuna den) erantzun bat aurkituko dugu. Erantzun hori JSON formatura bihurtu dugu (`r.json()` erabiliz). Orain, erantzuna JSON objektua denez, oso modu erosoan trata dezakegu. Adibidez, latitudearen eta longitudearen koordinatuak lortzeko, objektua.city.coord erabiliko genuke. Ikus hurrengo kode zatia:

```
1 fetch("https://api.openweathermap.org/data/2.5/forecast?
2 q=Bilbao,es&appid=XXXXXXXXXXXXXXX").then( r =>
   ↪ r.json()).then( objektua => {
3   console.log(objektua);
4   console.log(objektua.city.coord);
5 })
```

JavaScript-en edozein objektu JSON formatura bihur dezakegu eta hortik String arrunt batera `JSON.stringify` metodoa erabiliz (serializazioa deitzen zaio prozesu horri). Adibidez:

```
1 let liburu = new Liburu("Dublinés", "Alfonso Zapico", 18);
2 let jsonLiburua = JSON.stringify(liburu);
3 // orain jsonLiburua JSON formatuan dagoen String bat da
4 console.log(jsonLiburua);
5 // {"izenburua": "Dublinés", "egilea": "Alfonso Zapico",
6 // "salneurria": 18}
```

7.2 Fetch APIa

Fetch APIarekin eskaera asinkronoak egin ditzakegu, bai GET nola POST metodoak erabiliz (besteak beste). Ikus ditzagun adibide batzuk.

7.2.1 Fetch APIa GET eskaerak egiteko

OpenLibrary zerbitzuak eskaintzen duen APIa erabiliz liburu baten datuak jasoko ditugu fetch dei batekin:

```
1 fetch(
2 'https://openlibrary.org/api/books?
   ↪ bibkeys=ISBN:0451526538&format=json').
```

```

3 then( r => r.json() ).
4 then( datuak => {
5   console.log(datuak)
6 })

```

fetch egin ondoren, erantzun gordina jasoko dugu r parametroan. Parametro horren edukia JSON objektu bat denez, r.json() erabiliko dugu erantzuna JSON bihurtzeko. Jarraian, datuak izeneko parametroan jasoko dugu JSON objektua eta kontsolatik bistaratuko dugu, honako emaitza jasoz:

```

1 { "ISBN:0451526538":
2   { "bib_key": "ISBN:0451526538",
3     "preview": "noview",
4     "thumbnail_url":
5       ↪ "https://covers.openlibrary.org/b/id/295577-S.jpg",
6     "preview_url": "https://openlibrary.org/books/OL1017798M/
7       ↪ The_adventures_of_Tom_Sawyer",
8     "info_url": "https://openlibrary.org/books/OL1017798M/
9       ↪ The_adventures_of_Tom_Sawyer"
10  } }

```

Erantzun horretan thumbnail_url edo liburu-azalaren irudi txikia eskuragarri izango dugu. Beste `fetch()` dei batekin jaso eta uneko orrian txertatuko dugu (7.4. irudia):

```

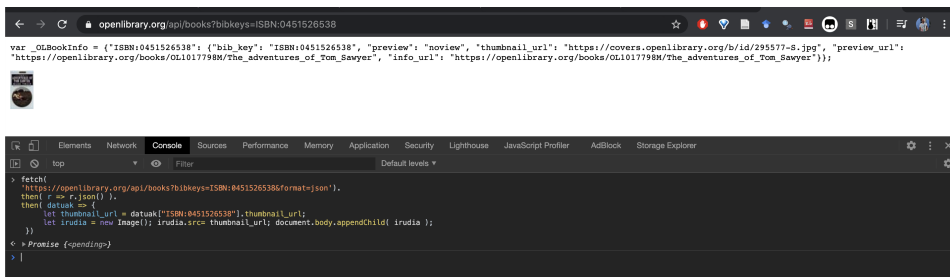
1 fetch(
2   'https://openlibrary.org/api/books?
3     ↪ bibkeys=ISBN:0451526538&format=json' ).
4 then( r => r.json() ).
5 then( datuak => {
6   let thumbnail_url =
7     ↪ datuak["ISBN:0451526538"].thumbnail_url;
8   let irudia = new Image(); irudia.src= thumbnail_url;
9     ↪ document.body.appendChild( irudia );
10 })

```

7.2.2 Fetch APIa POST eskaerak egiteko

Fetch APIa POST eskaerak egiteko ere erabil daiteke. Horretarako, URLaren ondoren JSON objektu gisa parametroak pasa diezazkiokegu, *method* eta *body* atribuetan.

Adibide gisa, `httpbin.org/post` helbidera (bidaltzen diogun guztia erantzun gisa errepikatuko du) POST eskaera bat bidaliko dugu, `aldagai1=balio1` eta `aldagai2=balio2` parametroekin:



7.4. irudia: Fetch APIa erabiliz API bat kontsultatu eta erantzuna *parseatu* ondoren, bertan zegoen irudi bat dokumentuan txertatu dugu. Probak egiteko nabigatzailearen web kontsola erabili dugu (ikus 20 atala).

```
1 fetch("https://httpbin.org/post",
2 {
3   method: 'POST',
4   body: 'aldagail=balio1&aldagai2=balio2'
5 }).then( resp => resp.text()).
6 then( erantzuna => console.log(erantzuna))
```

Eta erantzun gisa, honakoa jasoko dugu:

```
{
  "args": {},
  "data": "aldagail=balio1&aldagai2=balio2",
  "files": {},
  "form": {},
  "headers": {
    "Accept": "*/*",
    "Accept-Encoding": "gzip, deflate, br",
    "Accept-Language": "eu,en-US;q=0.9,en;q=0.8,es;q=0.7",
    "Cache-Control": "no-cache",
    "Content-Length": "31",
    "Content-Type": "text/plain;charset=UTF-8",
    "Host": "httpbin.org",
    "Origin": "chrome-search://local-ntp",
    "Pragma": "no-cache",
    "Sec-Fetch-Dest": "empty",
    "Sec-Fetch-Mode": "cors",
    "Sec-Fetch-Site": "cross-site",
    "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X
      ↳ 10_15_4) AppleWebKit/537.36 (KHTML, like Gecko)
      ↳ Chrome/83.0.4103.97 Safari/537.36",
    "X-Amzn-Trace-Id":
```

```

    ↪ "Root=1-5edfaeca-dbf4fb5585f92c45ae2efda3"
  },
  "json": null,
  "origin": "47.63.89.49",
  "url": "https://httpbin.org/post"
}

```

7.3 Ariketak

Gai honetan zenbait teknologia eta teknika berri landu dira. JSON, AJAX, Fetch APIa eta Promesak. Horiek guztiak ondo ulertzeko ezinbestekoa da praktikatzea.

1. GDAX webguneak kriptotxanponen prezioa ezagutzeko API bat eskaintzen du. Adibidez, URL honi deituz <https://api.gdax.com/products/btc-eur/ticker> Bitcoin baten prezioa (€urotan) emango digu, JSON formatuan (besteak beste, hainbat datu eskaintzen baititu dei horren erantzunak).



```

{"trade_id":29524444,"price":"9062.14","size":"0.02090596","time":"2020-09-15T07:33:40.521697Z","bid":"9062.13","ask":"9062.14","volume":"1670.25100072"}

```

7.5. irudia: JSON erantzun ezberdina jasoko dugu exekutatzen dugun guztietan, kriptotxanpon baten balioa une oro aldatzen baita.

Bitcoin baten prezioa dinamikoa da, egunean zehar hainbat aldiz aldatuko da (berez segundoero!).

fetch APIa erabiliz aurreko URLari deitu eta kontsolan idatzi Bitcoin baten uneko prezioa.

2. loadImage() metodoa txantiloi gisa erabiliz (gogoratu gai honi dagokion kapituluaz azaltzen dela bere funtzionamendua):

```

function loadImage(url) {
  return new Promise(resolve => {
    const image = new Image();
    image.addEventListener('load', () => {
      resolve(image);
    });
    image.src = url;
  });
}

```

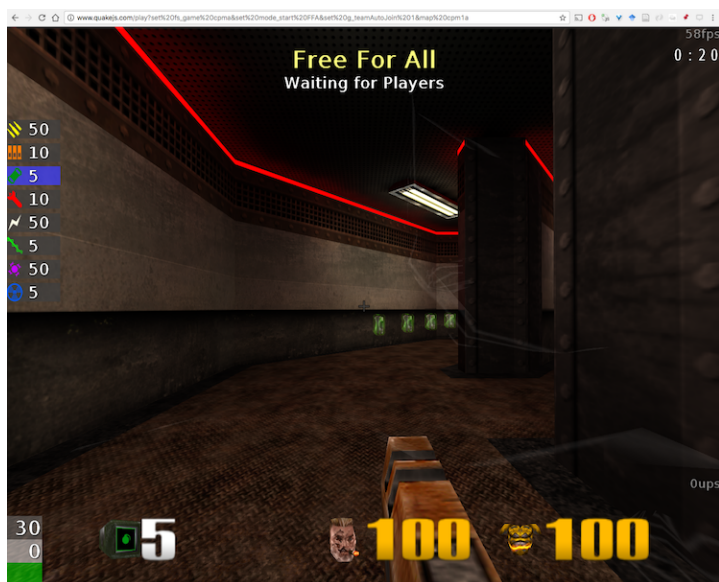
```
loadImage('https://developers.google.com/web/images/  
    ↪ irudia.png').  
then( image => document.body.appendChild(image) )
```

Implementa ezazu loadAudio(url) izeneko beste funtzio bat, non parametro gisa audio-fitxategi baten URLa ematen zaion eta audio hori memorian kargatzen duen. Funtzioa inplementatu ondoren, dei honen bidez audioa entzun beharko litzateke:

```
loadAudio('https://zerbitzaria.com/audioa.mp3').then(  
    ↪ audioa => audioa.play() );
```

8. Canvas, pantailan marrazten

Canvas osagaiak oihal edo mihise bat eskaintzen du web orrian edozein marrazketa egin ahal izateko, pixelak, formak edo testua. Canvas erabiliz 60 *frame/segundoko* abiaduran pantaila osoan marraztu dezakegu, animazioak sortuz. Ospe handiko Doom edo Quake bezalako bideo-jokoak HTML5era eraman dira, besteak beste, *canvas* osagaiari esker.



8.1. irudia: Canvas eta JavaScript APIak erabiliz Quake bideo-jokoaren bertsio bat nabigatzailean.

8.1 <canvas> osagaia

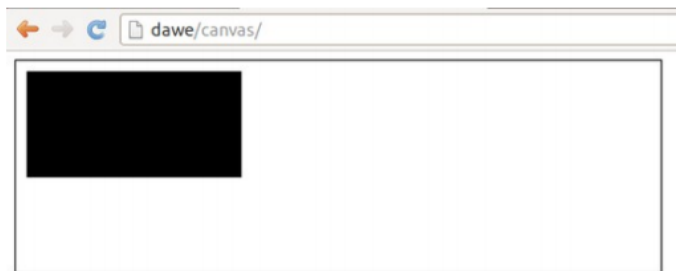
Canvas-en marrazten hasteko, *canvas* (oihala) osagaiaren luzera eta zabalera zehaztu behar ditugu:

```
<canvas id="oihala" width="600" height="200"></canvas>
```

Jarraian, bertan bi dimentsioko grafikoak marraztu nahi ditugula adieraziko dugu, `getContext('2d')` metodoaren bidez:

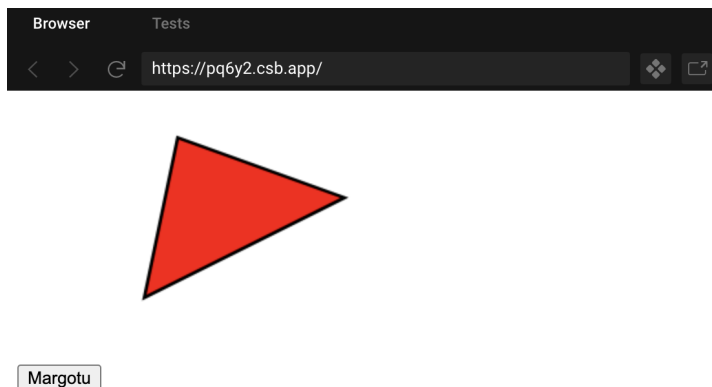
```
let oihala = document.getElementById("oihala");
let context = oihala.getContext("2d");
```

Behin 2D ingurunea prest dugunean, hainbat eragiketa izango ditugu esku-argarri, adibidez, laukizuzen beteak marrazteko `fillRect()` metodoa. Ikus 8.1. listatua.



8.2. irudia: Canvas-en laukizuzen sinplea margotzen.

```
1 <!doctype html>
2 <html lang="es">
3 <head>
4 <title>Canvas Osagaia</title>
5 </head>
6 <meta charset="utf-8">
7 <style>
8 canvas {border: 1px solid black;}
9 </style>
10 <script>
11 window.onload = function(){
12     let oihala = document.getElementById("oihala");
13     let context = oihala.getContext("2d");
14     // defektuz, beltzez margotzen da
15     context.fillRect(10,10,200,100);
16 }
```



8.3. irudia: *Margotu* izeneko botoian sakatzean, canvas-en gorriz betetako hiruki bat margotuko dugu. Adibidearen kode osoa hemen: <https://codesandbox.io/s/canvas1-pq6y2>.

```
17 </script>
18 </head>
19 <body>
20   <canvas id="oihala" width="600" height="200"></canvas>
21 </body>
22 </html>
```

8.1. listatua: Laukizuzen bat margotuko dugu canvas osagaian.

Laukizuzen beteak edo hutsak (strokeRect) margotu ditzakegu. Baita canvas-en dagoen zati bat laukizuzen baten bidez ezabatu (clearRect). Canvas-ekin lan egiteko metodo hauek ezagutzea komenigarria da:

- Laukizuzen hutsa margotu, x, y posizioan, w zabalera eta h altuerarekin: strokeRect(x, y, w, h)
- Laukizuzen betea margotu: fillRect(x, y, w, h)
- Laukizuzen zuri batekin canvas-eko zati bat ezabatu: clearRect(x, y, w, h)

Bideak (marrak, zuzenak edo ez) ere marraztu ahal ditugu canvas osagaian, honako metodo hauen bitartez:

- Bidea hasi: beginPath()
- Mugitu posizio batera: moveTo(x, y)

- Marra egin, kokatua gauden posiziotik hasita eta x, y punturaino: `lineTo(x, y)`
- Arkua marraztu (parametroak ulertzeko, ikus 8.2 azpiatala): `arc(x, y, r, angle_0, angle_1, sense)`
- Bidea amaitu (adibidez, hiruki bat margotzen ari bagara, soilik bi alde margotu ondoren, *closePath* metodoari deitzen badiogu, hirukia itxiz hirugarren aldean automatikoki margotuko da): `closePath()`
- Bide hutsa marraztu (ikus hurrengo informazio-kutxa, "Stroke metodoaren garrantzia"): `stroke()`
- Bide betea marraztu: `fill()`

Stroke metodoaren garrantzia



Bide edo arku bat marrazten baduzu, gogoratu `stroke()` metodoa aplikatzeaz. Adibidez, `arc()` metodoaren ondoren ez baduzu `stroke()` erabiltzen, ez da ezer agertuko pantailan (nahiz eta arkua berez, hor egon). Stroke metodoaz gogoratzeko pentsa ezazu margolari bat zarela. Zerbait tintaz marraztu baino lehen, arkatzez egingo dituzu hasierako zirriborroak, ia-ia ikusezinak diren marrak erabiliz. Ondoren, tinta aplikatuko diozu eta orduan ikusiko da zer egin nahi zenuen. Tinta horren papera jokutzen du stroke metodoak.

Aipatutako metodoak ondo ulertzeko, adibide bat aztertuko dugu jarraian. *Path* edo bideak erabiliz, hiruki gorri bat margotuko dugu botoi baten gainean saketzean:

```

1 window.onload = function() {
2     // orriko elementuak kargatzean, botoiaren
3     //   ↪ erreferentzia lortu
4     let margotuBotoia = document.getElementById("margotu");
5
6     // botoia sakatzean kudeatzaile batek erantzun behar du
7     margotuBotoia.onclick = margotuKudeatzaile;
8 }
9 function margotuKudeatzaile()
10 {
11     let oihala = document.getElementById("oihala");
12     let context = oihala.getContext("2d");

```

```

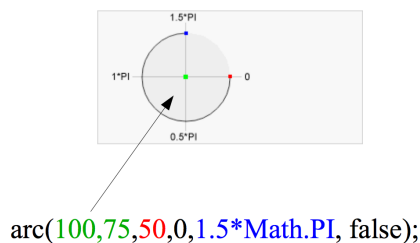
13   hirukiaMargotu(context);
14 }
15
16 function hirukiaMargotu(context)
17 {
18     context.beginPath();
19     context.moveTo(100,150);
20     context.lineTo(250,75);
21     context.lineTo(125,30);
22     context.closePath();
23
24     context.lineWidth = 5;
25     context.stroke();
26     // gorriz bete
27     context.fillStyle = "red";
28     context.fill();
29 }

```

8.2 Arkuak

Arkuak marrazteko `arc()` metodoa erabiliko dugu. Honek 6 parametro hartzen ditu (ikus 8.4 irudia). Lehenengo biek (100,75 adibidean) arkuak osatzen duen zirkunferentziaren zentroa adierazten dute. Hurrengoak (50), erradioa. Hirugarrenak, hasierako angelua (radianetan). Laugarrenak, bukaerako angelua (non bukatu behar dugun arkuak margotzen), eta, azkenak, angelua erloju-orratzen norabidean edo kontrakoan marraztu behar den.

Adibidez, 8.4. irudiko arkuak margotzeko kodea 8.2. listatukoa izango litzateke.



8.4. irudia: Canvas osagaian arkuak margotzeko `arc()` funtzioa erabiliko dugu.

```

1 <!doctype html>
2 <html>

```

```
3 <head>
4   <title>Canvas adibidea</title>
5 </head>
6 <body>
7   <canvas id="oihala" width="150" height="150"></canvas>
8
9 <script>
10 var oihala = document.getElementById("oihala");
11 var ctx = oihala.getContext("2d");
12
13 ctx.arc(100, 75, 50, 0, 1.5*Math.PI, false);
14 ctx.stroke();
15
16 </script>
17 </body>
18 </html>
```

8.2. listatua: Arku bat margotzeko kodea. Hemen eskuragarri: <https://codesandbox.io/s/canvas-arkua-75d7q>.

8.3 Irudiak margotzen

Irudi-fitxategiak ere canvas-en txerta ditzakegu. Oso funtzionalitate interesgarria du, jokoak HTML5en programatu nahi baditugu, adibidez. Honi esker, jokalaria (edo arerioak, edo ingurunea) ez dugu zertan lerroz lerro margotu, baizik eta jada existitzen diren sprite-ak (irudi-fitxategiak) canvas-en kargatu eta horiek mugituko ditugu (animazioak lortuz).

Hasteko, irudi bat canvas-en margotzeko, `drawImage()` metodoa erabiliko dugu. Metodo hori gainkargatuta dago. Ikus ditzagun adibide batzuk.

Lehenengo adibidean, UEUren ikurra kargatu eta canvas-en margotuko dugu.

```
1 <canvas id="oihala" width="120" height="110"></canvas>
2 <script>
3   var oihala = document.getElementById("oihala");
4   var context = oihala.getContext("2d");
5   var logo = new Image();
6   logo.src = "irudiak/ueu.png";
7
8   // irudia memorian kargatzen denean oihalan idatziko dugu
9   //   ↳ (ez lehenago)
10  logo.onload = function() {
11    context.drawImage(logo, 0, 0);
12  };
```

12 | `</script>`

8.3. listatua: UEUren logoa canvas-en margotuko dugu. Kodea hemen eskuragarri: <https://codesandbox.io/s/canvas-irudia-kk55q>.

8.4 drawImage() metodoaren parametroak

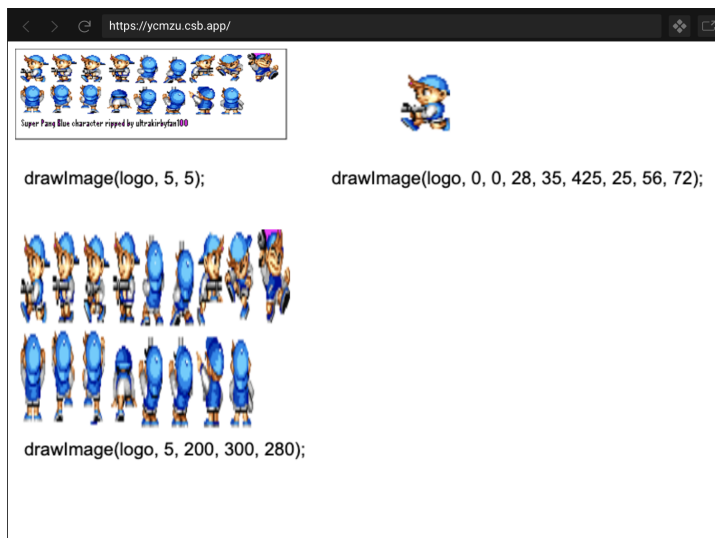
`drawImage()` metodoa gainkargatuta dago, alegia, posible da metodo horri deitzea 3, 5 edo 9 parametroekin. Parametro kopuruaren arabera gauza bat edo bestea egingo du. Adibidez, 3 parametroekin, zer margotu nahi dugun eta non (x,y) zehaztuko dugu. Bost parametro hartzen baditu, lehenengoa margotu nahi dugun irudiaren erreferentzia izango da (adibidean, *logo* izeneko aldagaian kargatu duguna). Jarraian, x,y posizioa canvas-en. Adibidean, goi-ezkerreko pixelaren posizioa (0,0) izango da. Ondoren (eta adibidean agertzen ez direnak, hautazkoak baitira), canvas-en margotuko dugun irudiaren tamaina (zabalera, altuera). Ez badugu ezer zehazten, jatorrizko irudiaren tamaina eta canvas-en margotzen denaren tamaina berdinak izango dira. Interesgarriak dira azken bi parametro horiek animazio sinpleak lortzeko (irudia handituz edo txikituz). Bederatzi parametroren kasuan, lehenengo parametroa margotu nahi dugun irudiaren izena izango da. Jarraian dauden hurrengo 4 parametroek jatorrizko irudiaren zer zati (x, y, zabalera, altuera) hartu nahi dugun adierazten dute, eta, hurrengo laurak, canvas-en zer posiziotan eta zer tamainatan margotu nahi dugun zehazten dute.

`drawImage()` metodoaren 3 forma posibleen (3, 5 eta 9 parametroekin) egingdako adibide baten kodea hemen aurkituko dugu: <https://codesandbox.io/s/canvas-buster-ycmzu> (ikus 8.5. irudia). Adibidean agertzen den testua, `fillText()` metodoa erabiliz margotu dugu. Zehazki:

```
// zer letra mota eta tamaina nahi dugun
context.font = "20px Arial";
// idatzi nahi dugun testua eta non (canvas-eko posizioa)
context.fillText("nahi duzun testua", 10, 150);
```

8.5 Ariketak

Ariketa hauetan (https://ikasten.io/html5/ariketak/08_gaia.pdf) sprite orri bat memorian kargatzen (ikus 8.5. irudia) ikasiko dugu. Baita sprite orri batetik irudi zehatzak ateratzen eta horiek canvas-en margotzen animazioak lortzeko. Zehazki, Buster izeneko pertsonaiaren animazioa lortu nahi dugu.



8.5. irudia: Canvas osagaian irudiak margotzeko drawImage metodoa erabiliko dugu. Adibide honetan irudi bera agertzen da, baina drawImage erabiliz, irudiaren zati bat margotu dezakegu, aukeran zoom eginez, edo irudi osoa, berriro ere, aukeran zoom eginez.

1. Hemengo helbidean <https://codesandbox.io/s/canvas-animazioa-8p7kx>) Buster-en irudi-orritik (sprite orritik) lau posizio dinamikoki hartzeko kodea aurkituko duzu.



8.6. irudia: Buster pertsonaiaren zenbait animazio.

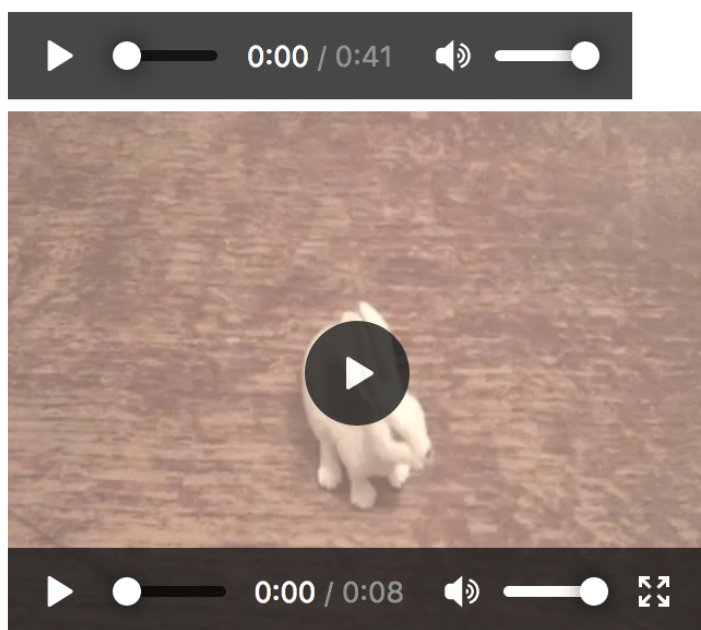
Molda ezazu kodea setInterval() metodoaren bidez Buster-en 4 posizio horiek leku berean, amaigabeko bukle —begizta— batean, bata bestearen atzetik, 150 milisegundoan behin margotzeko, animazioaren efektua lortuz. Dena ondo eginez gero, Buster ibiltzen ari dela emango du.

2. Aurreko ariketa moldatu Buster mugi dadin, pantaila eskuinetik hasita ez-

kerrerantz, ezkerreko pareta ukitu arte (puntu horretan gelditu egin behar da).

9. Audioa eta bideoa

HTML5 estandarrak <video> eta <audio> etiketak sartu zituen, eta horiekin batera, JavaScript bidez audioa eta bideoa kontrolatzeko HTMLMediaElement APIa. Bideoarekin lan egiten hasiko gara, termino batzuk definituz, *kontainer*, *codec* eta formatuak, besteak beste.



9.1. irudia: Audio —goian— eta bideo —behean— kontrol-osagaiak.

9.1 Kontainer eta codec-ak

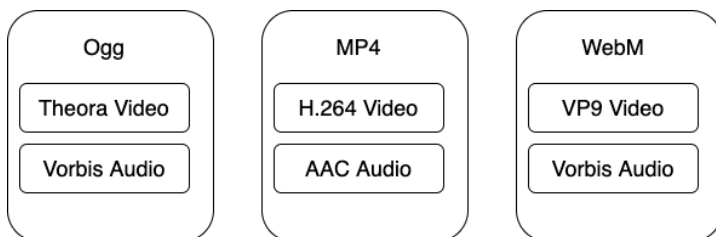
Bideo-fitxategi baten inguruan hitz egitean askotan entzun izan ditugu “AVI fitxategi” edo “MP4 fitxategi” hitzak. Baina AVI edo MP4 ez dira fitxategi mota bat, baizik eta kontainer motak. AVI MP4 edo WebM kontainer batek, bere barnean, audio- eta bideo-kanalak gordetzen ditu, hainbat formatutan kodetuta egon daitezkeenak (*codec* —kodek— ezberdinekin, Theora, H.264, VP9...).

9.2 <video> etiketa

Berez, bideo bat bistaratzeko etiketa sinple bat erabili ahal dugu:

```
<video src="bideoa.webm"></video>
```

Berrir, WebM kontainer mota bat da. Barruan duen bideoaren formatua (erabili den kodeka) zein den ez dakigu (bideoa jaitsi eta aztertu arte). HTML5 estandarrak <video> etiketa eskaintzen badu ere, ez du zehazten bideoak erabili behar duen formatua... edo nabigatzaileak berak onartu behar dituen formatuak. Izan ere, hainbat aukera ditugu (9.2 irudian adibide batzuk zehazten dira).



9.2. irudia: Bideo-kontainer eta *codec*-en artean hainbat aukera ditugu.

Nabigatzaile bakoitzak kodek zehatz bat onartzen duen edo ez jakiteko, Wikipediak eskaintzen duen [taulara joko dugu](https://en.wikipedia.org/wiki/HTML5_video)¹ (ikus 9.3. irudia). Bertan kontainer eta kodeken bateragarritasuna nabigatzailearen arabera zehazten da.

Hori guztia kontuan hartuta, ulertzekoa da <video> etiketaren erabilera aurkeztu dugun adibidea baino konplexuagoa izatea. Hurrengo listatuan <video> etiketaren erabileraren adibide zehatza ikus dezakegu:

```

1 <video width="320" height="240" controls>
2 <source src="clip.mp4" type="video/mp4; codecs=avc1,mp4a">
3 <source src="clip.webm" type="video/webm;
   ↪ codecs=vp8,vorbis">

```

¹https://en.wikipedia.org/wiki/HTML5_video

Status of video format support in each web browser						
Browser	Operating System	Theora (Ogg)	H.264 (MP4)	HEVC (MP4)	VP8 (WebM)	VP9 (WebM)
Android browser	Android	Since 2.3 ^[46]	Since 3.0 ^[46]	Since 5.0 ^[46]	Since 2.3 ^[46]	Since 4.4 ^[46]
Chromium	Unix-like and Windows	Since 1829 ^[47]	Via FFmpeg ^[48]	No ^[49]	Since 147759 ^[51]	Since 172738 ^[52]
Google Chrome	Unix-like, Android, macOS, iOS, and Windows	Since 3.0 ^[53]	Since 3.0 ^[54]	No ^[55]	Since 6.0 ^[57]	Since 29.0 ^[58]
Internet Explorer	Windows	Via OpenCodecs	Since 9.0 ^[60]	No ^[56]	Via OpenCodecs	No
	Windows Phone	No	Since 9.0 ^[61]	No	No	No
	Windows RT	No	Since 10.0 ^[62]	No	No	No
Microsoft Edge	Windows 10	Since 17.0 (with Web Media Extensions) ^[63]	Since 12.0 ^[67]	Needs hardware decoder ^[6]	Since 17.0 (supports <video> tag with Web Media Extensions ^[6] and VP8 Video Extensions ^[68])	Only enabled by default if hardware decoder present ^[69]
	Windows 10 Mobile	No	Since 13.0 ^[72]	No	Since 15.0 (only via MSE) ^[73]	Since 14.0 (only via MSE) ^[74]
Konqueror	Unix-like and Windows	Needs OS-level codec ^[6]				
Mozilla Firefox	Windows 7x	Since 3.5 ^[75]	Since 21.0 ^[64]	No ^[56]	Since 4.0 ^[76]	Since 65.0 ^[82]
	Windows Vista		Since 22.0 ^[83]			
	Windows XP and N editions		Since 46.0 ^[84]			
	Linux		26.0 (via GStreamer) ^[85] 43.0 (via FFmpeg) ^[87]			Since 67
	Android		Since 17.0 ^[86]			In Nightly
Opera Mobile	Android, iOS, Symbian, and Windows Mobile	Since 13.0	Since 11.50	No ^[91]	Since 15.0	Since 16.0
	macOS, Windows, Linux		Since 24.0 ^[92]			Since 57.0 ^[93]
	iOS		No			Since 57.0 ^[93]
Safari	macOS	Via Xiph QuickTime Components (macOS 10.11 and earlier)	Since 3.1 ^[96]	Since 11 ^[97]	Since 12.1 (only supports WebRTC) ^[98]	No

9.3. irudia: Wikipediak badu orri bat nabigatzaileen arteko bideo-koдек eta kon-tainerren bateragarritasuna neurtzeko.

```

4 <source src="clip.ogv" type="video/ogg;
   ↪ codecs=theora,vorbis">
5 </video>

```

Bertan 320 x 240 tamainako bideo bat bistaratu nahi dugula zehaztu dugu. Bideoarekin batera, *controls* atributuarekin, *Play*, *Pause*, *FastForward* eta *Rewind* egiteko aukera izan nahi dugula esaten ari gara.

Horrez gain, `type=video/mp4` atributuarekin, MP4 kontainerra hobesten dugu-la esaten ari gara, barruan `avc1` bideo-kodeka eta `mp4` audio-kodeka izango ditue-na (`codecs=avc1,mp4a`). Hori ezinezkoa balitz (nabigatzaileak onartuko ez balu), orduan WebM kontainerra aukeratuko genuke, `vp8` bideo-koдек eta `vorbis` audioa-rekin. Eta hori ere ezinezkoa balitz, orduan Ogg kontainerra izango litzateke gure aukera, *theora video* eta *vorbis* audioarekin.

Controls izeneko atributua ez da erabil dezakegun bakarra. Badugu *autoplay*, *loop*, *preload* eta *poster* izenekoak ere.

- *autoplay*: bideoa kargatu bezain pronto martxan jarri nahi dugula zehazten du.
- *loop*: bideoa amaitzean berriro hasieratik automatikoki hasiko dela adieraz-teko.

Atributuak		Metodoak	Gertaerak	
width	currentTime	play()	progress	timeupdate
height	paused	pause()	loadeddata	volumechange
loop	duration	load()	error	ended
muted	readyState	canPlayType()	loadedmetadata	play
ended	seeking		pause	abort
error	volume		waiting	

9.1. taula: Bideoak JavaScript-ez kontrola ditzakegu, hainbat metodo, atributu eta gertaeraren bidez.

- *preload*: `preload=auto` edo `preload=metadata` izan daiteke atributu honen balioa. *Auto*-ren kasuan, nabigatzaileari bideoa ahal bezain pronto kargatzea (orria kargatzen den unean) gomendatzen diogu. Alegia, erabiltzaileak *play* botoian sakatu baino lehen, automatikoki buffer batean bideoa kargatzea nahi dugula zehazten du. Horrela erabiltzaileak *play* botoian sakatzean, bideoa berehala hasiko da bistaratzen, inolako etenik gabe. *Metadata*-ren kasuan gauza bera, baina bideo osoa aurrekargatu beharrean, bideoaren metadatuak (luzera, *track* kopurua, bideoaren lehenengo fotograma, poster gisa erabiltzeko...) aurrekargatuko dira
- *poster*: bideoa hasi baino lehen pantailan bideoari dagokion irudi txiki bat agertuko da.

Bideo bat JS erabiliz kontrolatzeko API bat daukagu eskuragarri HTML5en. Metodo, atributu eta gertaera asko eskaintzen baditu ere, garrantzitsuenak 9.1. taulan aurki daitezke.

Bideoen inguruko APIa hobeto ulertzeko adibide pare bat ekarriko ditugu hona. Lehenengoak hiru bideo kargatzen ditu array batean eta, APIa erabiliz, bata bestearen ondoren joko ditu.

```

1 let position = -1;
2 let playlist;
3 let bideo;
4 window.onload = function() {
5     playlist = ["video/bat.mp4", "video/bi.mp4",
6                 ↪ "video/hiru.mp4"];
7     bideo = document.getElementById("video");
8     bideo.addEventListener("ended", hurrengoBideoa);
9     hurrengoBideoa();
10 }
```

```

10
11 function hurrengoBideoa() {
12     position++;
13     if (position >= playlist.length) position = 0;
14     bideo.src = playlist[position];
15     bideo.load();
16     bideo.play();
17 }

```

Bigarren adibidean gauza bera egiten dugu, baina bideoa martxan jarri baino lehen bideoaren kontainerra nabigatzaileak onartzen duen edo ez begiratu dugu.

```

1
2 window.onload = function() {
3     // ...
4     playlist = ["video/bat", "video/bi", "video/hiru"];
5     // ...
6 }
7 function hurrengoBideoa() {
8     // ...
9     bideo.src = playlist[position] + getFormatExtension();
10    // ...
11 }
12 function getFormatExtension() {
13     if (bideo.canPlayType("video/mp4") != "") {
14         return ".mp4";
15     } else if (bideo.canPlayType("video/ogg") != "") {
16         return ".ogg";
17     } else if (bideo.canPlayType("video/webm") != "") {
18         return ".webm";
19     }
20 }

```

canPlayType() metodoaren balio posibleak



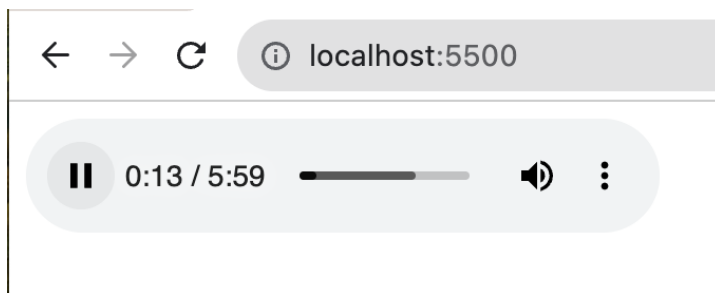
Bideoa baten *canPlayType()* metodoak parametro bat hartzen du, bideoaren kontainer mota (adibidez "video/webm", edo "video/mp4") eta hiru balio posible itzultzen ditu: kate hutsa (""), "maybe" edo "probably". Kate hutsaren kasuan, nabigatzaileak kontainer mota hori ez duela onartzen esan nahi du. *maybe* kasuan, agian bistaratu dezakeela eta *probably* ia ziur onartzen duela. Nabigatzaileak ezin du erabat ziurtatu jo ahal duen edo ez, kontainer baten barruan kodek ezberdinak egon daitezkeelako.

9.3 Audio-etiketa

Audioaren eta bideoaren etiketek oso antzeria funtzionatzen dute. Ikus bestela honako HTML5 <audio> etiketaren adibidea:

```
1 <audio controls>
2 <source src="horse.ogg" type="audio/ogg">
3 <source src="horse.mp3" type="audio/mpeg">
4 Zure nabigatzaileak ez du audio etiketa onartzen.
5 </audio>
```

Ogg motako kontainera ez bada onartzen, orduan MP3 kontainer mota duen audioarekin saiaturiko gara. Horrekin ere ezin badu nabigatzaileak, orduan errore-mezu bat emango dugu. Ohart zaitez *controls* atributua ere erabiltzen ari garela (ikus 9.4. irudia).



9.4. irudia: Audio player-a, hainbat kontrolekin.

Nabigatzaileek audio-formatuekin duten bateragarritasun-maila jakin ahal izateko badugu Wikipedian orri berezi bat ere².

9.4 Ariketak

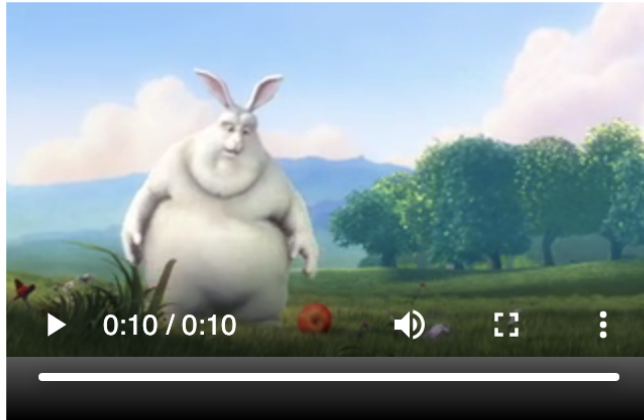
Ariketa multzo honetan canvas eta <video> etiketak uztartuko ditugu, adibidez 9.5. irudian dagoen ariketa egiteko.

Hemengo kode-adibideari jarraituz <https://codesandbox.io/s/bideo-ariketa-1-bh3cg>:

1. Controls atributua ezabatu <video> etiketatik. Jarraian 3 botoi prestatu bideoaren azpian agertzeko: Jo, Eten, Kaptura.

- Jo botoian sakatzean, bideoa martxan jarriko da.

²http://en.wikipedia.org/wiki/HTML5_Audio



9.5. irudia: Bideo batetik fotogramak atera daitezke eta canvas batean utzi baino lehen itxuraz aldatu. Adibidez, irudia zuri-beltz bihurtzeko.

- Eten botoian sakatzean, bideoa eten egingo da. Berriro martxan jartze-ko, Jo botoian sakatu.
- Kaptura botoian sakatzean, bideoaren uneko fotogramaren kaptura egingo da. Kaptura `<canvas>` etiketan bistaratu behar da, dimentsioak aldatuz —txikiagoa ikusi behar da bideoa baino, zuk aukeratu tamaina—. Adibide gisa, ikus 9.7. irudia.

Argibidea: `drawImage()` metodoak lehenengo parametroan bideo baten erreferentzia onartzen du.

Ariketaren soluzioa begiratu gabe egiten saia zaitez, baina argibideren bat behar baduzu, hona hemen ariketaren soluzioa:

<https://codesandbox.io/s/bideoariketak-soluzioa-i3kp8>



Jo Eten Kaptura

9.6. irudia: Kaptura botoian sakatzean uneko fotogramaren kaptura egin behar da eta tamaina txikian margotu, irudian bezala.

2. Aurreko ariketa moldatu bideoaren kapturak automatikoki egin daitezen, alegia *canvas* osagaian bideoaren fotogramak agertuko dira bideoa martxan dagoen bitartean, automatikoki.

Honako lerro hau lagungarria egingo zaizu:

```
bideoa.addEventListener("play", function () { ....
    ↪ zure kodea ... }
```

Bideoa amaitzen denean edo bideoa etetean (“pause” egitean), *canvas* osagaian fotogramak kopiatzeari utzi.

Beste lerro hau ere lagungarria egingo zaizu:

```
bideoa.addEventListener("pause", function () { ...
    ↪ zure kodea hemen ... }
```

Ariketaren soluzioa begiratu gabe egiten saia zaitetz, baina argibideren bat behar baduzu, hona hemen: <https://codesandbox.io/s/bideosoluzioa2-jiqyn>.

3. Hurrengo ariketan `<video>` eta `<canvas>` etiketak uztartuko ditugu. Helburua 9.5. irudian dugun aplikazio lortzea da. Goiko aldean bideo bat dugu. Haren azpian bi *canvas* txiki. Ezkerrekoan orain arte egin duguna. Eskuinekoan zuri-beltzez margotuko dugun beste *canvas* bat. Prozesua honakoa da:

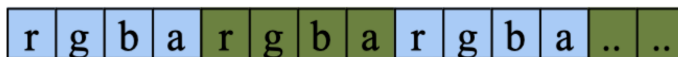
- Bideotik frame bat atera eta ezkerreko canvas osagaian (*buffer* deritzona) utzi:

```
buffer.drawImage(bideoa, 0, 0, 160, 120);
```

Jarraian, *buffer*etik frame baten informazioa erauzi, array gisa:

```
let frame = buffer.getImageData(0, 0, 320, 120);
```

frame arrayak 9.7. irudiko itxura izango du:



9.7. irudia: R = Red-Gorria, G = Green-Berde, B = Blue-Urdina, A = Alpha-Gardentasuna.

Alegia, pixel bakoitzak 4 balio izango ditu: R (*red* edo kolore gorria), G (*green* edo berdea), B (*blue* edo urdina), A (*alpha* edo gardentasun-maila). Adibidez, pixel bat guztiz gorria eta erdi gardena izango bage-nu, orduan haren balioa (255, 0, 0, 100) izango litzateke.

Zenbat pixel ditugun jakiteko, zatiketa sinple bat egin dezakegu:

```
let length = frame.data.length / 4;
```

Ondoren, pixel bakoitzeko, haren posizioa eta (r,g,b,a) balioak aterako ditugu eta horiekin zuri-beltzez() izeneko funtzio bati deituko diogu, kolorez dagoen pixel bat zuri-beltz bihurtzeko:

```
for (let i = 0; i < length; i++) {
  let r = frame.data[i * 4 + 0];
  let g = frame.data[i * 4 + 1];
  let b = frame.data[i * 4 + 2];
  zuri-beltzez(i, r, g, b, frame.data);
}
```

zuri-beltzez() funtzioa erraza da, kolore-osagai bakoitzaren balioak batu eta batezbestekoa lortuko du hasieran (kolore grisa lortzeko). Ondoren pixel horri kolore grisa esleitzen dio:

```
function zuri-beltzez(pos, r, g, b, data) {
  var grisa = (r + g + b) / 3;
  data[pos * 4 + 0] = grisa;
  data[pos * 4 + 1] = grisa;
  data[pos * 4 + 2] = grisa;
}
```

Egitea falta zaigun bakarra zuri-beltzez lortu dugun frame arraya canvas-en margotzea da:

```
zuri-beltza.putImageData(frame, 0, 0);
```

Ariketaren soluzioa begiratu gabe egiten saia zaitez, baina argibideren bat behar baduzu, hona hemen: <https://juananpe.github.io/txuribeltz/>.

4. Audio-ariketa batekin jarraituko dugu. Hiru fitxategiz osatutako array bat dugularik: ['audio1.mp3', 'audio2.mp3', 'audio3.mp3'], programa ezazu script bat hiru audio horiek jotzeko, bata bestearen atzetik. Bukatzean, lehenengoarekin jarraitu, amaigabeko begizta batean.
5. Web orri bat prestatu <audio> elementuarekin, baina *controls* atributurik gabe (ez dira audio kontrolatzeko botoiak agertuko). Sortu 4 botoia: play, pause, bolumena igo eta bolumena jaisteko. JavaScript erabili audioa jotzeko eta kontrolatzeko. Adibidez, erabiltzaileak play botoia sakatzean, audioa martxan jarri behar da (eta play botoia desgaitu behar da). Pause botoian sakatzean audioa eten egin behar da eta play botoia berriro gaitu. Argibidea: aztertu play(), pause() funtzioak, eta audio elementuaren volume atributuak.
6. Audio elementu batean hainbat source jarri daitezke ikusi dugu. Horietatik nabigatzaileak bat aukeratuko du (onartzen duen lehenengo codec-a, adibidez). Baina nola jakin dezakegu zein izan den nabigatzaileak aukeratu duen audio mota? Argibidea: aztertu audio elementuaren loadeddata gertaera eta currentSrc atributua.
7. Fast Forward (FF) botoia gehitu. Botoia sakatzean audioaren uneko posizioa 10 segundu aurreratuko da.
8. Uneko posizioa bistaratzen duen geruza gehitu. Bertan uneko posizioa (minutu:segundu) eta audioaren iraupena uneoro bistaratu behar da. Argibidea: setInterval edo hobe, audioaren timeupdate gertaera kudeatuz lortuko duzu.

10. Biltegitratze lokala, localStorage APIa

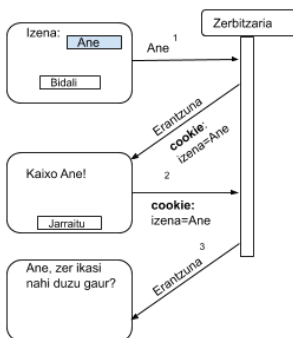
HTTP protokoloak ez du memoriarik. Eskaera baten eta ondorengoaren artean protokoloak ez du ezer gogoratzen, hau da, *stateless* protokoloa dela esaten dugu. Adibide batekin aztertuko dugu hau. Demagun hiru web orri ditugula (ikus 10.1. irudia):



10.1. irudia: HTTP protokoloak ez du memoriarik. Formulario batek jasotzen duen informazioa bistartzeko ez dago arazorik, baina 3. pantaila batera informazio hori bidaltzeko bai, ordea.

Lehenengo orrian, erabiltzaileari izena eskatzen zaio. Kasu honetan “Ane” idatzi, Bidali botoian sakatu eta bigarren orrira goaz. Bertan zerbitzariak “Kaixo Ane!” idatzi behar du eta jarraitzeko botoi bat eskaini. Hau lortzeko ez dugu arazorik, “Ane” izena parametro gisa baitoa formularioan. Baina bigarren orritik hirugarren orrirako jauzian, nola bidali “Ane” izena? Aukera bat izan daiteke *hidden* motako eremu batean bidaltzea. Baina badago beste aukera bat sartu ditugun datuak HTTP zerbitzariari gogorarazteko: cookieak erabiltzea. Honela funtzionatuko du (ikus 10.2 irudia):

Erabiltzaileak Ane izena sartu eta Bidali botoian sakatu ondoren, zerbitzariak “Ane” parametroa jaso eta erabiltzaileari hurrengo orria bidaltzen dio, cookie batekin batera (cookiearen balioa izena=Ane delarik). Erabiltzailearen nabigatzaileak, hemendik aurrera, zerbitzari horri edozein gauza eskatzearekin batera beti bidaliko dio jaso duen cookiea, lekuko bat izango balitz bezala. Bigarren pausoa zerbitza-



10.2. irudia: HTTP protokoloan egoera mantentzeko cookieak erabiltzen ohi dira.

riak cookiea jaso eta izena=Ane ikusita badaki zeinek bidali dion. Gauzak horrela, azken (hirugarren) orrialdea prestatu eta nabigatzaileari bidaliko dio, “Ane, zer ikasi nahi duzu gaur” testuarekin batera. Landu dugun adibide honen inplementazioarekin praktikatzeko, hemengo¹ kodea erabili (martxan ikusi nahiko bazenu, Heroku aplikazio honetan² argitaratu da).

Beraz, cookieek informazioa gordetzeko balio dute, eta, horrekin batera, HTTP protokoloari egoera gogoratzeko bide bat emango diogu.

Informazio pribatua eta cookieak



Adi! Ez gorde informazio pribatua cookie baten barruan (adibidez, ez gorde inoiz pasahitzak cookie baten barruan). Horren ordez, cookiean identifikatzaile bat gorde dezakegu eta zerbitzariaren lana izango da cookie horren identifikatzaileari edukia esleitzea eta gordetzea. Horrela, informazio pribatua zerbitzarian gordeko da beti, eta ez bezeroan.

Cookie baten tamaina maximoa 4 KB da, beraz biltegitratze lokala lortzeko ez du askotarako ematen. Baina HTML5ek beste API bat eskaintzen du datuak bezeroan gorde eta kudeatu ahal izateko, Web Storage API delakoa.

10.1 Web Storage APIa

API honek *izena=balioa* formatuari jarraitzen dioten balioen biltegitratze lokala (*localStorage* delakoa) ahalbidetzen du. Domeinu bakoitzeko 5-10 MB gorde di-

¹<https://gist.github.com/juananpe/824b1d9ad2141d5c64bc8693492863f9>

²<https://mighty-anchorage-92406.herokuapp.com/>

tzake nabigatzailean bertan. Cookieekin ez bezala, behar direnean erabiltzen dira (gogoratu cookieen kasuan nabigatzaileak beti bidali behar zituela zerbitzarira).

LocalStorage biltegian idazteko *setItem* metodoa erabiliko dugu:

```
localStorage.setItem("gakoa", balioa);
```

Eta balio bat irakurtzeko, *getItem*:

```
let balioa = localStorage.getItem("gakoak");
```

LocalStorage array bat izango balitz bezala ere trata daiteke, bai datuak irakurtzeko:

```
let balioa = localStorage["gakoa"];
```

nola gordetzeko:

```
localStorage["gakoa"] = balioa;
```

Balio bat localStorage-tik ezabatzeko, *removeItem* erabiliko dugu:

```
localStorage.removeItem("gakoa");
```

Posible da, halaber, localStorage biltegian dauden aldagai guztiak ezabatzea, *clear* metodoarekin :

```
localStorage.clear();
```

length atributua eta *key()* metodoaren laguntzaz, localStorage biltegiko balioak zeharka ditzakegu :

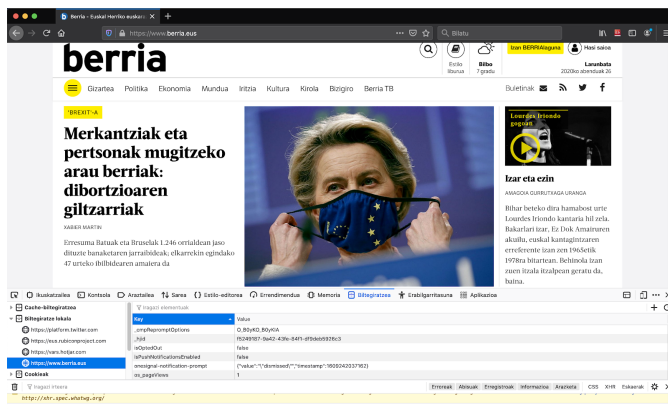
```
for (let i=localStorage.length; i >= 0; i--){
  let gakoa = localStorage.key(i);
  console.log( localStorage.get(gakoa) );
}
```

10.1.1 Nola aztertu localStorage biltegian dagoen edukia

Firefox-en *Biltegitratzea* izeneko erlaitzean, *Biltegitratze lokala* atalean aurkituko dugu localStorage biltegia, domeinuka antolatuta. Adibidez, 10.3 irudian, berria.eus webgunean sartzean gordetzen den informazioa ikus daiteke.

Kontsolan *localStorage.clear()* metodoari deitzean, bertan zeuden datu guztiak ezabatu ahalko ditugu.

LocalStorage biltegia hutsik dagoelarik, berriro kargatzen badugu berria.eus webgunea, Berriak (push) jakinarazpenak jaso nahi ditugun galdetuko digu (10.4 irudia).



10.3. irudia: Berriak localStorage erabiltzen du hainbat informazio nabigatzailean bertan gordetzeko, besteak beste erabiltzailearen baimena (*push*) jakinarazpenak jaso nahi dituen edo ez zehazteko.



10.4. irudia: Berriak push jakinarazpenak jaso nahi ditugun edo ez galdetuko digu lehendabiziko bisitan. Erabiltzailearen erantzuna localStorage biltegian gordetzen da (horrela ez zaio berriro ere galdetuko hurrengo bisitan).

10.2 Ariketak

OpenLibrary APIa erabil dezakegu liburu baten datuak lortzeko bere ISBNa pasatuz. Adibidez:

```
fetch("https://openlibrary.org/api/books?
  ↳ bibkeys=ISBN:9781491906187&
  jscmd=details&format=json").then( r=> r.json()).then( r
  ↳ => console.log( r['ISBN:9781491906187'].details))
```

OpenLibrary APIa erabiliz, egin ezazu programa bat ISBN array honetan dau- den liburu guztien datuak localStorage biltegian gordetzeko:

```
[ 'ISBN:9781491906187', 'ISBN:9781491920497',
  ↳ 'ISBN:1491910399',
  'ISBN:1491946008', 'ISBN:1491978236',
  ↳ 'ISBN:9781491906187']
```

Bukatzean, dena ondo egin badugu, kode hau exekutatzean:

```
let liburua = localStorage.getItem( 'ISBN:9781491906187' );  
console.log( liburua.title );
```

Emitza hau jaso beharko genuke: "Data Algorithms: Recipes for Scaling Up with Hadoop and Spark".

Soluzioa: <https://github.com/juananpe/express/blob/master/public/js/module.mjs>

11. Geokokapen APIa

Geokokapen APIak erabiltzailearen kokapen zehatza aurkitzeko metodoak eskaintzen ditu. API hau bereziki interesgarria da HTML5 euskarria ematen duten gailu mugikorretan erabiltzeko. W3C erakundeak argitaratutako espezifikazioa bada ere, berez geokokapen APIa ez dago HTML5en espezifikazioaren barruan. API honen bidez, munduko puntu baten geokokapena lortu ahalko dugu, adibidez: Bilbo, latitudea: 43° 15' iparra, longitudea: 2° 55' mendebaldea. Ikus dezagun API honen erabileraren adibide bat. Gure helburua uneko geokokapena lortzea izango da.

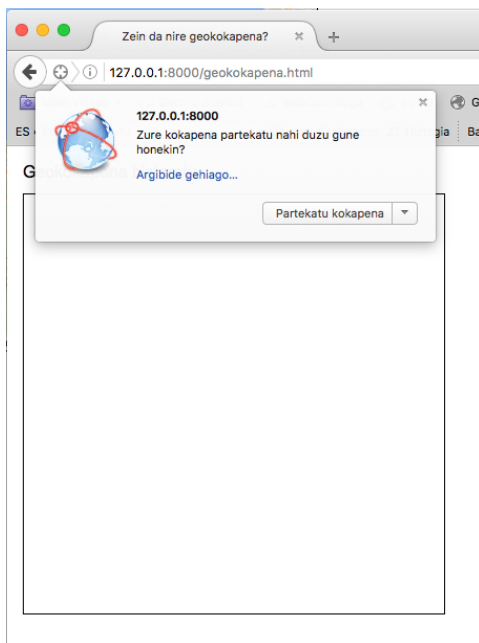
HTTPS protokoloa erabili geokokapena partekatu ahal izateko



Segurtasuna dela eta, ezin da file:// protokoloa erabili geokokapena lortzeko. Alegia, HTTP zerbitzari bat beharko duzu gai honen inguruko ariketak egiteko. Askotan ikasleek file:// protokoloa erabiltzen dute beren probak egiteko. Orain arte horrela egiteak ez zuen inolako oztoporik sortzen, baina orain bai, ordea. Ez baduzu Apache (HTTP zerbitzari ezagunena) instalatu edo konfiguratu nahi, Python erabil dezakezu, modu errazean HTTP zerbitzari bat martxan jarri ahal izateko:

```
$ python -mSimpleHTTPServer
```

Komando horrek Pythonek eskaintzen duen HTTP zerbitzari simple bat martxan jarriko du, 8000 portuan entzunez, uneko karpetaren fitxategiak eskainiz. Orain, nabigatzailean, `http://localhost:8000` edo `http://127.0.0.1:8000` idatziz, HTTP protokoloa erabili ahalko duzu gai honen ariketak modu erraz batean probatu ahal izateko. Pribatutasuna dela eta, HTTPS da erabili beharreko protokoloa localhost domeinua ez badugu erabiltzen.



11.1. irudia: Geokokapena lortzeko nabigatzaileak erabiltzaileari baimena eskatu behar dio (lehendabiziko aldia bada).

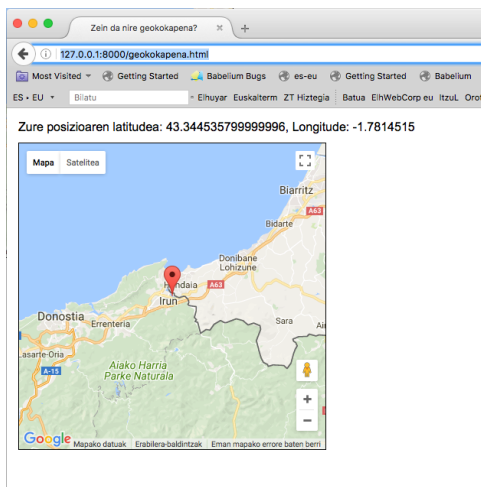
11.1 Posizioa lortzen. `getCurrentPosition` metodoa

`getCurrentPosition()` metodoaren sinadura hau da:

```
getCurrentPosition(callback, errorrea, aukerak)
```

Funtzio horri deitzean, nabigatzaileak egiaztatu beharko du ea erabiltzaileak posizioa ezagutzeko baimena eman duen edo ez. Baiezkoan, *callback* funtzioari deituko dio. Ezezkoan, erabiltzaileari galdetu eta haren erantzunaren arabera *callback* edo errore-funtzioari deituko dio (ikus 11.3. irudia). *Aukerak* izeneko hautazko parametroaren bidez kokapenaren zehaztasun batzuk ezar daitezke.

```
// defektuzko balioak
let aukerak = {
  enableHighAccuracy: false, // zehaztasun maila altua nahi
    ↳ den edo ez
  timeout: Infinity, // asko jota zenbat denbora itxaron
    ↳ behar den geokokapena lortzen errorerik eman gabe
  maximumAge: 0 // cachean dauden geokokapenen denbora
    ↳ maximoa berrerabiliak ahal izateko, milisegundutan.
    ↳ 0 = ez erabili cachea
```



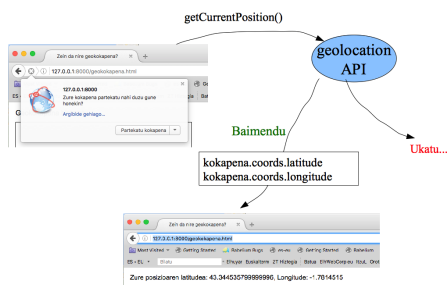
11.2. irudia: Google Maps API eta geokokapen APIa erabiliz, uneko kokapena lortu eta mapa batean margotu dugu.

```
}

```

Ikus dezagun adibide zehatz bat (implementazioa [hemen](#)¹ proba dezakezu).

Geolocation API



11.3. irudia: Kokapena: Nola funtzionatzen duen Geolocation APIak.

```
1 window.onload = lortuGeokokapena;
2
3 function lortuGeokokapena() {
4
5 // nabigatzaileak geolocation APIa implementatzen du?
```

¹<https://github.com/yuchenlin/rebiber>

```

6  if (navigator.geolocation) {
7
8      navigator.geolocation.getCurrentPosition(
9          ↪ bistaratuGeokokapena , bistaratuErrorea);
10 }
11 else {
12     alert("Nabigatzaille honek ez du geokokapen APIa
13         ↪ onartzen");
14 }
15 }
16
17 function bistaratuGeokokapena(posizioa) {
18     let latitude = posizioa.coords.latitude;
19     let longitude = posizioa.coords.longitude;
20
21     let div = document.getElementById("geokokapena");
22     div.innerHTML = "Zure posizioaren latitudea: " +
23         ↪ latitude + ", Longitude: " + longitude;
24
25     // mapaBistaratu(posizioa.coords);
26 }
27
28 function bistaratuErrorea(error) {
29     const errorTypes = {
30         0: "Errore ezezagun",
31         1: "Ez duzu kokapena lortzeko baimenik",
32         2: "Kokapena ez dago eskuragarri",
33         3: "Itxaron denbora agortu da"
34     };
35     let errorMessage = errorTypes[error.code];
36     if (error.code == 0 || error.code == 2) {
37         errorMessage = errorMessage + " " + error.message;
38     }
39     let div = document.getElementById("geokokapena");
40     div.innerHTML = errorMessage;
41 }

```



macOS sistema erabiltzen duzu?

Adi! macOS sistema eragilean bazaude Security&Privacy fitxan Chrome edo Firefox nabigatzailerei geokokapena lortzeko baimena eman beharko diezu Geolocation APIa erabili ahal izateko.

11.2 Geokokapena Google Maps-en marrazten

Aurreko kodean erabiltzailearen geokokapena lortu dugu, baina ez dugu mapa batean marraztu. Horretarako, Google Maps-eko APIaren laguntza beharko dugu. Ohart zaitez aurreko kodean 22. lerroa komentatuta dagoela (*mapaBistaratu* metodoari deia). Lerro hori deskomentatu eta jarraian dugun *mapaBistaratu* metodoa aztertu.

```

1 <head>
2 <script type="text/javascript"
   ↳ src="//maps.googleapis.com/maps/api/js?key=GAKOA">
   ↳ </script>
3 </head>
4 <body>
5 <div id="kokapena">
6 Zure geokokapena hemen bistaratuko da.
7 </div>
8 <div id="mapa">
9 </div>
10
11 </body>
12 </html>

```

Google Maps API key



Adi! Google Maps-eko APIa erabiltzeko API gako bat eskatu beharko duzu hemen

<https://developers.google.com/maps/documentation/javascript/get-api-key>

```

1 let map; // aldagai globala
2
3 function mapaBistaratu(coords) {
4   let googleLatAndLong = new
     ↳ google.maps.LatLng(coords.latitude, coords.longitude);
5
6   const mapOptions = {
7     zoom: 10,
8     center: googleLatAndLong,
9     mapTypeId: google.maps.MapTypeId.ROADMAP
10  };
11
12  let mapDiv = document.getElementById("mapa");
13  map = new google.maps.Map(mapDiv, mapOptions);
14 }

```

Bi metodo erabiltzen ari gara hemen, *google.maps.LatLng* (mapan kokatu nahi dugun puntuaren latitudea eta longitudea parametro gisa espero dituen) eta *google.maps.Map* eraikitzailea. Horrek bi parametro jasotzen ditu: *div* etiketa baten erreferentzia (mapa zer geruzatan marraztu nahi dugun jakiteko) eta aukera-objektu bat, *mapOptions*, zer motatako mapa eta zer zoomekin ikusi nahi dugun zehazteko.

Orain arte landutako adibidea <https://ikasten.io/html5/geokokapena.html> web-gunean aurki dezakezu.

11.2.1 Markatzaileak

Falta zaigun gauza bakarra markatzaile gorri bat jartzea da (*marker* bat). Markatzaile horren gainean klik egitean, gure uneko kokapena agertuko da maparen gainean. Presta dezagun `markatzaileaGehitu` izeneko funtzio bat.

```

1 function markatzaileaGehitu(mapa, latlong, izenburua,
  ↪ edukia){
2     let markerOptions = {
3         position: latlong,
4         map: mapa,
5         title: izenburua,
6         clickable: true
7     };
8     let marker = new google.maps.Marker(markerOptions);
9
10
11     let infoWindowOptions = {
12         content: edukia,
13         position: latlong
14     };
15     let infoWindow = new
16         ↪ google.maps.InfoWindow(infoWindowOptions);
17     google.maps.event.addListener(marker, "click",
18         ↪ function() {
19         infoWindow.open(mapa);
20     });
21 }

```

Orain `mapaBistaratu()` funtziotik, `markatzaileaGehitu` funtzioari deituko diogu:

```

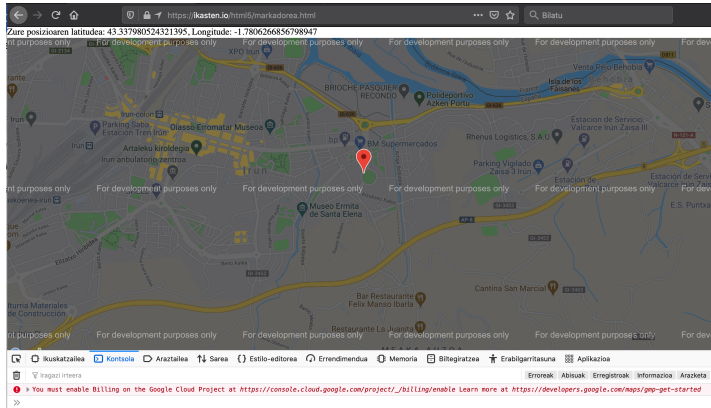
1 let izenburua = "Zure kokapena";
2 let edukia = "Hemen zaude kokatua: " + coords.latitude +
  ↪ ", " + coords.longitude;

```

```
3 markatzaileaGehitu(map, googleLatAndLong, izenburua,
    ↪ edukia);
```

Markatzaileak erabiltzen dituen kodearen adibidea online ikusteko, jo ezazu honako helbide honetara:

<https://ikasten.io/html5/markadorea.html>



11.4. irudia: Markatzaileak (mapan agertzen den ikur gorria) Google Maps APIa erabiliz maparen gainean jar daitezke. Ohart zaitez “*For development purposes only*” mezuan eta kontsolan agertzen den mezu gorrian. Mezu horiek kentzeko (eta mapa hobeto ikusteko) ordainpeko kontu bat ireki beharko dugu Google Maps-en honako helbide honetan: https://console.cloud.google.com/project/_/billing/enable.

11.3 Ariketak

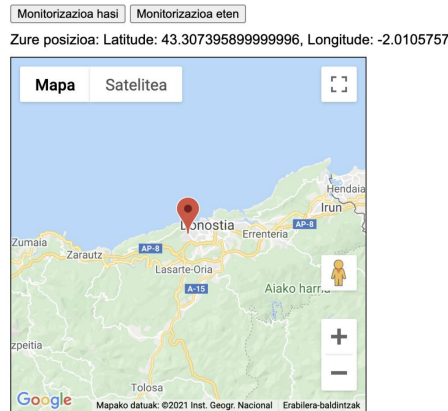
Mugitu ahala zure geokokapena mapa baten gainean bistaratu nahi dugu. Zure geokokapenaren gainean dagoen markatzailean sakatzean, irekitzen den informazio-panelean longitudea eta latitudea bistaratzeaz gain, zauden tokiaren altitudea ere ikusi nahi dugu. Horretarako, Google Maps-ek eskaintzen duen *locationService* APIaren bidez altitudea lortzea eskatzen da.

Geokokapen APIaren *Coordinates* klaseak hautazko lau parametro eskaintzen ditu (ez daude nabigatzaile guztietan inplementatuta edo ezin dira denak erabili gailu guztietan): *altitude*, *altitudeAccuracy*, *heading* eta *speed*.

Txantiloia honetan oinarriturik: <https://jsfiddle.net/juanan/jsexk3rh/12/> bertan dagoen JavaScript kodea aldatu markagailu batean klikatzean pantailan uneko posizioa eta altitudea bistaratu ahal izateko (ikus 11.5. irudia). Altueraren datu hori

zuzenean eskuratzea posible ez balitz, Google Maps-ek eskaintzen duen `elevationService()`² zerbitzuari deitu eta hortik eskuratu ahalko duzu³.

`elevationService()` metodoa Google Maps-eko 3. bertsiotik aurrera dago eskuragarri. Bertsio hori nola kargatzen den ikasteko, metodoaren dokumentazioan eskaintzen den adibidearen HTML kodea aztertzea gomendagarria da.



11.5. irudia: Google Maps-ekin egindako ariketaren emaitza.

²`elevationService()` metodoaren dokumentazioa hemen aurkituko duzu:

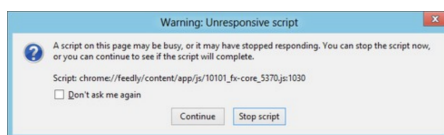
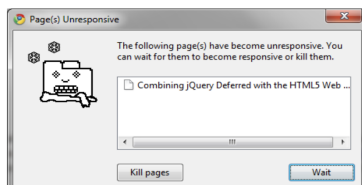
<https://developers.google.com/maps/documentation/javascript/examples/elevation-simple>

³`elevationService()` zerbitzutik jasotzen duzun altuera zenbaki erreal gisa bistaratu, bi dezimalera biribilduz (`parseFloat()`) metodoa lagungarria egingo zaizu)

12. Web Workers APIa

Orain arte, JavaScript erabiliz hari bakarreko programak sortu ditugu. Web Worker APIak JavaScript scriptak aldi berean exekutatzeke ahalmena ematen digu. Hau oso interesgarria da konputazio handiko prozesuak lantzeko. Web Worker-ak existituko ez balira, arazo bat izango genuke, konputazio astun horiek egitean nabigatzailearen interfaze grafikoa blokeatu egingo litzatekeelako: hari bakar batek kalkuluak egin beharko lituzke eta aldi berean ezingo lituzke erabiltzailearen sarrerek kudeatu —saguarekin egindako mugimenduak, botoi-sakatzak, etab.—.

Kasurik okerreanean nabigatzaileak errore-mezu bat pantailaratuko luke, skript batek kontrola hartu duela eta gainontzeko eragiketa guztiak blokeatu egin direla esateko, skripta amaitzeko aukera emanez (ikus [12.1.](#) irudia).



12.1. irudia: Nabigatzaileak exekuzio-denbora luzea behar duen skripta aurkitzen duenean, skript hori amaitu edo itxaroten jarraitu nahi dugun galdetuko digu. Web Worker APIak horrelako errore-mezuak ekiditen lagunduko digu.

Web Workers APIa bereziki garrantzitsua da RIA (*Rich Internet Application*)

aplikazioetan, aldi berean exekutatzen diren eragiketak egin nahi ditugulako, eta ziur asko horietako batzuk (edo asko) paraleloki egikaritu daitezkeelako, interfaze grafikoaren blokeoa saihestuz. API hau, web aplikazioen elkarreragin arina lortzeko, ezinbestekoa bilaka daiteke gure web aplikazioetan konputazio handiko eragiketak egin behar direnean.

Gai honetan ikusiko dugun Web Workers APIa erabiliz egindako adibideaz gain, John Resig ¹ programatzaileak bere blogean proposatzen dituenak ere aztertea komeni da.²

12.1 Zer da Web Worker bat?

Laburbilduz, Web Worker bat bigarren planoan exekutatzen den JavaScript programa bat da.

Erabiltzailearen interfaze grafikoa kudeatzen duen JavaScript haria blokeatu gabe, kalkulu-ahalmen handiko atazak exekutatzeko oso egokiak dira.

12.2 Zer egin dezakegu Web Workers APIarekin?

Web Worker-ak honako atazak sortzen dituzten kalkuluak egiteko aukera ezin hobeak dira, adibidez kasu hauetan:

- Jokoak, grafikoak, kriptografia...
- Sarrera/Irteera: URL bati *polling* egiteko atzeko planoan.
- Web editoreetan erabiltzen den sintaxiaren koloreztatzea lortzeko.
- Bigarren planoan exekutatzen diren zuzentzaile ortografikoak.
- Online kalkulu-orriak programatzeko.

12.3 Nola erabili Web Worker APIa

Web Worker berri bat sortzeko *Worker* klasea instantziatuko dugu:

```
let worker = new Worker("worker_script.js");
```

Web Worker bati mezu bat bidaltzeko *postMessage* metodoa erabiliko dugu :

```
worker.postMessage("Kaixo mundua!");
```

¹ John Resig jQuery liburutegiaren sortzailea da.

²<http://ejohn.org/blog/web-workers/>

Web Worker batek bidaltzen dizkigun erantzunak kudeatu:

```
worker.onmessage = function(event) {
  console.log("Jasotako mezua: " + event.data);
  zerbaitEgin();
}
```

Web Worker bat amaitzeko, berriz, *terminate* metodoaz baliatuko gara:

```
worker.terminate();
```

12.4 Oinarrizko adibidea

Zenbaki lehenak kalkulatzeko lagunduko digun Web Worker bat programatzeko dugu adibide honetan.

Lehenengo, *lehenDa()* funtzioa programatzeko dugu, `worker.js` fitxategian³.

```
// worker.js fitxategia
function lehenDa(n) {
  if (n == 2) return true;
  for (var i = 2; i <= Math.sqrt(n); ++i) {
    if (n % i == 0) return false;
  }
  return true;
}

// amaigabeko begizta batean lehenDa metodoari
// deitu, 1-etik aurrera dauden zenbaki lehenak
// kalkulatu eta programa nagusiari postMessage
// erabiliz bidaliko dizkiona
for (var i = 1; ; i++)
  if (lehenDa(i))
    postMessage(i)
```

Jarraian, *worker*-a instantziatu eta *onmessage* metodoaren bidez *worker*-aren mezuak tratatzeko ditugu:

```
<!doctype html>
<html>
<head>
  <title>Web Worker bakar batekin zenbaki lehenak
    ↪ kalkulatzeko adibidea</title>
</head>
<body>
```

³ Adibidearen kodea hemen aurkituko duzu: <https://ikasten.io/html5/webworkers/index.html>

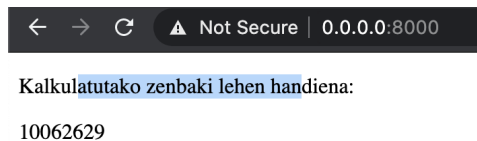
```

<p>Kalkulatutako zenbaki lehen handiena: <div
  ↪ id="result"></div></p>
<script>
let worker = new Worker('worker.js');

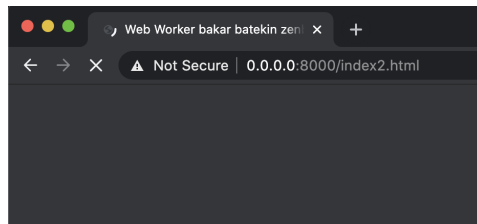
worker.onmessage = function (event) {
    document.getElementById('result').innerHTML =
      ↪ event.data;
};
</script>
</body>
</html>

```

Aplikazioa martxan jarri eta zenbaki lehenak kalkulatzen dituen bitartean, saguarekin testuaren zati bat hauta dezakegu, nabigatzailearen fitxa itxi, edo nahi duguna egin (12.2. irudia). Programa bera probatzen badugu⁴ baina Web Worker-ak erabili gabe, nabigatzailea nola blokeatzen den nabarituko dugu (eta kasurik okerreanean ez digu utziko ezta aplikazioaren emaitza ikusi ere —12.3. irudia—).



12.2. irudia: Web Worker bat erabiliz posible da nabigatzailearekin lan egitea, kalkulu sakonak egiten diren bitartean.



12.3. irudia: Web Worker APIa erabili gabe, orri nagusiak ez du kargatu ere egiten.

⁴Programa bera, baina Web Worker-ak erabili gabe, hemen aurkituko duzu: <https://ikasten.io/html5/webworkers/index2.html>. Adi, workerrak erabiltzen direnez, adibide honek nabigatzailea blokeatuko du!

12.5 Ariketak

Ariketa bakar bat oraingo honetan. Web Worker APIa erabiltzeko *script* baten berridazketa egitea eskatuko zaizu. <https://ikasten.io/html5/webworkers/ww.html> orrian dugun skriptak zenbaki lehenak kalkulatzeko dituen bitartean # karakterea inprimatzen du pantailan, behin eta berriro. Web Worker-ik erabiltzen ez duenez, kalkuluak egiten dituen bitartean nabigatzailea erdi blokeatuta geratzen da. Ariketa honen eginbeharra kodea Web Worker APIa erabiltzeko moldatzea da.

13. Inprimakiak HTML5en

HTML4 bertsioan inprimakiak programatzeko hainbat osagai bagenituen ere (*input*, *text*, *numeric*, *select*, *checkbox*, *radiobutton*...), HTML5ek osagai eta atributu berriak ekarri ditu: *placeholder*, *autofocus*, *email* eta url —URLak sartzeko eremuak—, zenbakiak sartzeko eremuak (*spinbox*, *slider*), datak hautatzeko eremua, kolore bat hautatzeko eremua, bilaketak egiteko kutxak... Berritasun horiek guztiak gai honetan aztertuko dira.

HTMLn, oinarritzko inprimaki baten itxura honakoa da:

```
1 <form name="inprimakia" action="/kudeatzailea">
2   Izena: <input type="text">
3   <br><input type="submit">
4 </form>
```

HTML4 bertsioan, inprimakietan erabil zitezkeen eremu motak 13.1. taulan zehazten dira.

Noski, eremu horiek HTML5en ere erabil daitezke, baina bertsio berri honetan beste eremu mota batzuk sartu ziren, atal honetan aztertuko ditugunak.

Eremu mota	HTML kodea	Iruzkina
Kontrol-laukia	<input type="checkbox">	gaitu eta desgaitu daiteke
Aukera-botoia	<input type="radio">	multzokatu egiten dira
Pasahitza-eremua	<input type="password">	puntuak idatziko ditu edozein tekla sakatzean
Aukera-zerrenda	<select><option>...	<i>combobox</i> izenarekin ere ezagutzen dira
Fitxategi-eremua	<input type="file">	disco gogorretik fitxategi bat hautatzeko eremua
Bidaltzeko botoia	<input type="submit">	submit botoia sakatuko dugu inprimakia bidaltzeko
Testu-eremua	<input type="text">	gehien erabiltzen den eremu mota

13.1. taula: HTML4 bertsioan inprimakietan erabil zitezkeen zenbait eremu mota.

13.1 Eremu eta atributu berriak HTML5en

Banan-banan hurrengo atributu eta funtzionalitate berriak landuko ditugu:

1. placeholder
2. autofocus
3. email eta URLak sartzeko eremuak
4. zenbakiak (*spinbox*, *slider*)
5. data bat hautatzeko eremua
6. kolore bat hautatzeko eremua
7. bilaketak egiteko kutxak
8. derrigorrezko eremuak
9. inprimakien baliozkotzea

13.1.1 Placeholder atributua

Testu-eremu batean sar daitekeen testu adibide bat bistartzeko atributua da *placeholder* (ikus 13.1. irudia). Adibidez:

```
<form>
  <input name="q" placeholder="Zure izena">
  <input type="submit" value="Bidali">
</form>
```



The image shows a rendered HTML form. It consists of a single-line text input field with the placeholder text "Zure izena" (Your name) in a light gray font. To the right of the input field is a rounded rectangular button with the text "Bidali" (Send) in a dark gray font.

13.1. irudia: *Placeholder* atributu motaren adibidea.

13.1.2 Autofocus atributua

Autofocus atributuak duen eremuak fokua hartuko du orria kargatzen denean. Adibidez, google.com kargatzean zuzenean idatz dezakezu bilatu nahi duzuna, inongo eremuan klik egin gabe, bilaketaren testua idazteko eremuak *autofocus* atributua gaituta duelako.

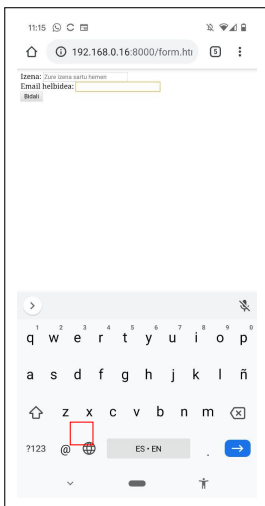
```
<form>
  <input name="q" autofocus>
  <input type="submit" value="Search">
</form>
```

13.1.3 Email eremu mota

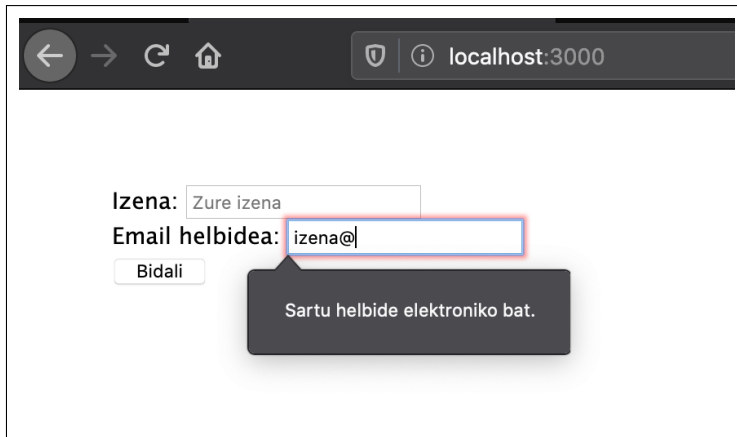
Email eremu mota (13.3. irudia) testu-eremu motakoa bezalakoa da, berezitasun pare batekin:

- Android eta iOS sistema eragileetan teklatu mota aldatzen da bertan klik egitean.
- Nabigatzaile batzuek egiaztapen automatikoak egiten dituzte (helbide elektroniko zuzena dela egiaztatzeko).

```
<form>
  <input type="email">
  <input type="submit" value="Go">
</form>
```



13.2. irudia: Email motako eremu batean sakatzean, mugikorretan irekitzen den teklatuan, @ karaktereak berezko tekla izango du (testu arruntetan *emojiak* idazteko okupatzen duen tekla).



13.3. irudia: Nabigatzaileek automatikoki egiaztatzen dute email zuzen bat sartu dela email motako eremu batean.

13.1.4 URL eremu mota

URL eremu mota URLak sartzeko erabiliko dugu (adibidez, <https://domeinua.eus>). Nabigatzaileak automatikoki egiaztatuko du benetan URL bat dela, hainbat arau betetzen dituela baieztatuz (barra bat / agertzen dela, gutxienez puntu bat, ez duela zuriunerik...)

13.1.5 Zenbakiak sartzeko eremuak: *range*, *slider* eta *spinbox* eremu motak

Zenbakiak `text` edo `range` motako eremuak erabiliz sar daitezke. *Range* eremu motan 13.4. irudian bistaratzen den bezalako osagai bat izango dugu. Osagai horri *slider* ere esaten zaio. Beste eremu mota bat zenbakiak sartzeko *number* izenekoa da. Mota horri, berriz, *spinbox* ere esaten zaio. Bertan soilik zenbakiak sar daitezke eta gezien bidez zenbakia handitu edo gutxitu egin daiteke. Zenbakien arteko jauzia ere kontrola daiteke *step* atributuarekin (ikus 13.5 irudia).

13.1.6 Datak sartzeko eremuak

HTML4 bertsioan ez zegoen eremu zehatzik datak sartzeko inprimaki batean; izan ere, JavaScript liburutegi ezberdinak sortu ziren arazo hori konpontzeko (adibidez, [jQuery DatePicker](https://jqueryui.com/datepicker/)¹). HTML5en, berriz, hainbat osagai ditugu datak eta orduak hautatzeko (13.6. irudia).

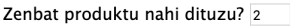
¹<https://jqueryui.com/datepicker/>



```

<form>
  <input name="r" type="range" min="1" max="11" value="9"
  <input type="submit" value="Go">
</form>

```

13.4. irudia: *Range* eremu mota.


```

<input type="number" min="2" max="10" step="2" value="2" required>

```

13.5. irudia: *Number* (edo *spinbox*) eremu mota.

13.1.7 Kolore bat hautatzeko eremu mota

Kolorea hautatzeko eremua ere berria da HTML5en. Eremuaren izena erraza da: *color*. Besterik ezeko balio bat ere eman diezaiokegu. Kolorearen balioa RGB (*red, green, blue*) erabiliz edo zuzenean izen bat esleituz ezar daiteke. Alegia, kolorearen izena gorria bada, *red* izena erabil dezakegu edo haren baliokidea RGB formatuan “#ff0000”.

RGB formatuak hiru osagai zehazten ditu: gorria, berdea eta urdina koloreen balioak, sistema hamaseitarrean. Berdea adierazteko, adibidez, “#00ff00” erabiliko genuke (edo *ff* baino txikiagoa den beste balio bat berde apalago bat hautatzeko). Noski, oinarritzko koloreak nahastu egin daitezke beste kolore batzuk lortzeko. Adibidez, morea lortzeko kolorea honakoa litzateke: #ff40ff. Dena den, sistema eragileak berak *pantone* bezalako kontrol bat eskaintzen du, kolorea grafikoki hautatu ahal izateko, zenbakiak zeintzuk diren jakin gabe (ikus [13.7](#) irudia).

13.1.8 Bilaketak egiteko *search* eremuak

Bilaketak egiteko *search* motako *<input>* osagaiak testu-eremu arruntak bezala-koak dira, erabiltzaileak bertan kontsultak eta bilaketen testua idazteko diseinatuak. Bisualki, berriz, berezitasunak izan ditzake, batez ere mugikorretan bistaratzeko badira. Adibidez, Android sistema eragileak lupa baten ikurra jarriko dio Enter tekleri (ikus [13.9](#) irudia).

Izena:

Email helbidea:

Ordua:

Data:

Bidali

< 2020(e)ko apirila >

al.	ar.	az.	og.	or.	lr.	ig.
30	31	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	1	2	3
4	5	6	7	8	9	10

13.6. irudia: Data eta orduak hautatzeko eremuak.

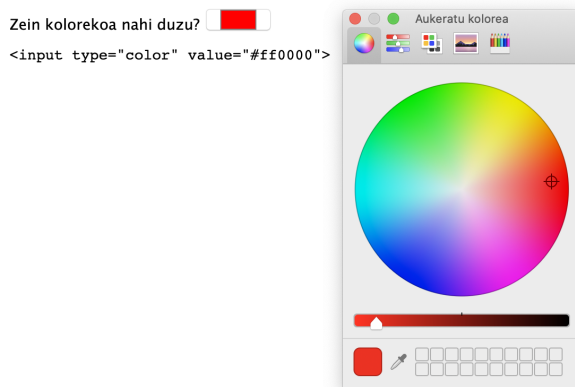
13.1.9 Derrigorrezko eremuak (*required*)

Color eremu motak aztertzean, `<input>` osagaian `required` atributua ikus dezakegu (13.7. irudia). Eremu horretan erabiltzaileak zerbait aukeratu behar duela esan nahi du *required* horrek. Ez badu horrela egiten, nabigatzaileak errore bat jaurtiko du inprimakia bidaltzeko botoian sakatzean eta ez digu bidaltzen utziko arazoa konpondu arte.

13.1.10 Inprimakien baliozkotzea

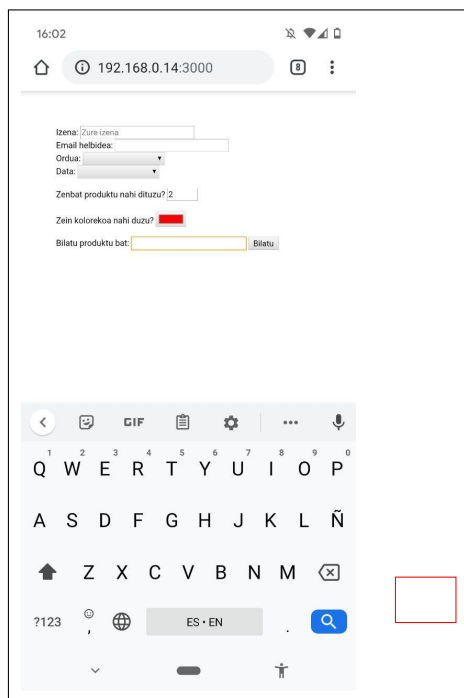
Inprimaki bat bidaltzean, derrigorrezko (*required*) eremuez gain, eremuen balioak egiaztatu behar dira inprimakia zerbitzarira bidali baino lehen. Eremu jakin batzuen balioa automatikoki konprobatuko da (adibidez, email motakoak, 13.3. irudian ikusten den bezala). Beste elementu batzuen balioa, ordea, skript baten bidez egiaztatu beharko ditugu. Adibidez, demagun 13.6. irudian data eta ordu bat hautatzean, erabiltzaileak soilik lanegunak direnak aukeratu ditzakeela, eta soilik 10:00-14:00 ordu-tartean. Aukeraketa hori zuzena ez bada, errore-mezu bat pantaillaratuko dugu, arrazoia azalduz.

Kodea 13.1.10. listatua aurkezten da. Bertan *form* etiketari *onsubmit* kudea-

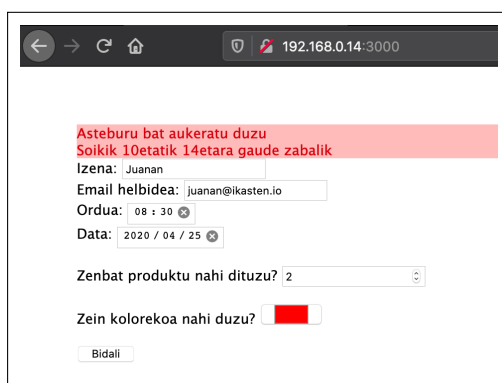


13.7. irudia: Koloreak hautatzeko *color* eremu mota erabiliko dugu.

tzaila bat esleitu zaio. Kudeatzaile horrek boolear bat bueltatu behar du: dena ondo badago inprimakian, *true* (eta inprimakia zerbitzarira bidaliko da, haren *action* atributua zehazten duen helbidera), eta erroreren bat aurkituz gero, *false* (eta, beraz, ez da inprimakia zerbitzarira bidaliko).



13.8. irudia: *Search* eremu motak lupa baten ikurra erabiliko du *Enter* tekla, mugikor batetik bistartzean.



13.9. irudia: Inprimaki baten balioak egiaztatu ondoren errore bat pantailaratu dezakegu beharrezkoa bada.

```

1 <script>
2     function egiaztatu(){
3         const inprimakia =
4             ↳ document.getElementById('inprimakia');
5             // ordua:minutua formatuan dagoen string batetik
6             ↳ ordua lortu
7         const ordua = inprimakia.ordua.value.split(":")[0];
8         // eguna lortu (string bat date datu motan bihurtu)
9         const eguna = new Date(inprimakia.data.value);
10        let erroreak = '';
11        // 6 = larunbata 0 = igandea 1 = astelehena ...
12        if (eguna.getDay() == 6 || eguna.getDay() == 0) {
13            erroreak += 'Asteburu bat aukeratu duzu';
14        }
15        // ordua 10-14 tartean egon behar da
16        if (ordua < 10 || ordua > 14) {
17            erroreak += '<br>Soikik 10etatik 14etara gaude
18                ↳ zabalik';
19        }
20        if (erroreak != ''){
21            document.getElementById('erroreak').innerHTML
22                ↳ = erroreak;
23            return false;
24        }
25        return true;
26    }
27 </script>
28 </head>
29 <body>
30
31 <div id="erroreak"></div>
32 <form id="inprimakia" method="POST" action="/inprimakia"
33     ↳ onsubmit="return egiaztatu();" >

```

13.1. listatua: Inprimaki baten balioak egiaztatzeko kodea.

13.2 Ariketak

Sor ezazu 13.10. irudian dagoen inprimakia HTML5ek eskaintzen dituen eremu berriak erabiliz eta honako arauak kontuan hartuz:

- Izena eremuak orria kargatzean fokua hartu behar du.

Bezeroaren datuak

Izena:

Telefonoa:

E-mail:

Liburuak

Kopurua (Gehienez 5):

13.10. irudia: Erabiltzaileak datuak idatzi ostean *Bidali* botoian sakatuko du. Aka-
tsik balego, jakinaraziko zaio.

- Telefonoa eremua ez da orain arte agertu, baina gai izango zinateke progra-
matzeko? Eta telefono baten patroia automatikoki antzemateko? (Argibi-
dea: ikus *regex* patroiak HTML5en)
- Ohart zaitez eremu batzuetan txantiloit gisa zerbait agertzen dela aurreidatzi-
ta (*placeholder* bat).
- Liburuak izeneko eremuan klik bikoitza egitean 5 libururen izenak agertu
behar dira (ikus *Datalist* HTML5en).

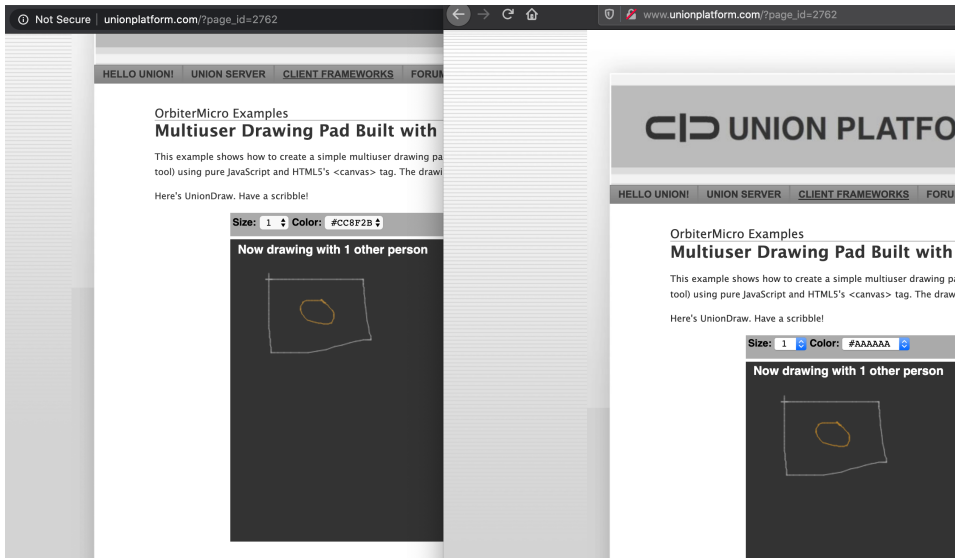
14. WebSocket-ak

WebSocket-ak jaio baino lehen, bezeroak zerbitzaritik datu bat jaso nahi zuenean, AJAX bidez egin behar zuen eskaera (XmlHttpRequest edo XHR objektu bat sortuz edo Fetch APIa erabiliz). Baliabide bakoitzeko XHR konexio bat ireki behar da: irudi bat lortzeko, JavaScript fitxategi bat atzitzeko, HTML orri bat jaisteko... Gauza bera bezeroak zerbitzarian dagoen prozesu baten egoera ezagutu nahiko bala: une oro XHR eskaerak egin beharko lituzke (*polling* delakoa). Prozedura hori ez da batere eraginkorra. Hoberena izango litzateke bezeroa entzuten jartzea eta zerbitzariak, noizbait bezeroari jakinarazi behar dion gertaeraren bat ikustean, bezeroari mezu bat bidaltzea. Hori da hain zuzen ere WebSocket APIaren zeregin nagusia.

WebSocket-a ordenagailuen arteko *full-duplex* edo bi norabideko komunikazioa lortzeko protokolo bat da, TCP konexio bakar baten gainean.

Horretaz aparte, nabarmendu daitekeen beste abantaila bat zera litzateke: WebSocket bat eraiki ondoren, bertatik bidaltzen diren pakete-goiburuen tamaina HTTP protokoloan erabiltzen direnena baino askoz txikiagoa da (ikus [14.1](#). irudia).

WebSocket APIaz, HTTP zerbitzari baten kontra konexio bat ireki eta bertatik datuak jaso edo bidali ahal izango ditugu. Zerbitzariak datu berriak bidali nahi dituenean, zuzenean egingo du, bezeroak ezer eskatu gabe. Horrez gain, WebSocket zerbitzari batek hainbat bezerorekin konexio irekiak izan ditzake aldi berean, bakoitzarekin *socket* bat irekiz, komunikazioa denbora errealean garatuz. Gaur egun, HTML5en programatutako bideo-joko eta aldibereko talde-lana egiteko tresna askotan WebSocket-ak erabiltzen dira komunikazio eraginkorra lortzeko. Adibidez, Google Driven, dokumentu bat hainbat pertsonaren artean aldi berean editatzen dugunean, edo [14.2](#). irudian ikusten dugun arbel digital partekatuan, WebSocket APIa erabiltzen ari gara.



14.2. irudia: WebSocket protokoloa erabiliz arbel digital partekatu bat programatu dezakegu, aldi berean hainbat erabiltzailerri bertan margotzeko aukera eskainiz. Iturria: http://www.unionplatform.com/?page_id=2762.

14.1.3 Mezuak jaso eta bidali

Bezerotik zerbitzarira mezuak bidaltzeko *send* metodoaz baliatuko gara:

```
socket.send("mugimendu zuzena");
```

Bezeroak zerbitzaritik datozen mezuak entzuteko *onmessage* izeneko *callback* bat prestatuko du:

```
socket.onmessage = function(erantzuna) {
    console.log("Jasotako erantzuna: " + erantzuna.data);
}
```

Gaur egun ez da WebSocket APIa zuzenean programatzen, baizik eta horren ordeaz ospetsua egin den socket.io izeneko liburutegia erabiltzen da. Liburutegi hori bezeroan kargatzea erraza da:

```
<script type="text/javascript"
  ↪ src="/socket.io/socket.io.js"></script>
```

Jabetu zaitetz /socket.io/ karpetaren izena dela. Bertan dagoen `socket.io.js` fitxategiak gordetzen ditu erabil ditzakegun funtzioak, *io* objektua barne.

Hurrengo adibidean, *socket.io* liburutegiaz baliatuko gara WebSocket konexio bat irekitzeko (*connect* metodoaz). Jarraian *socket*-etik mezuak bidaltzeko *emit* metodoa erabiliko dugu. Zerbitzaritik mezu bat jasotzean, *socket.on* bidez zer funtzio egikarituko den zehazten da. Funtzioa jaso den mezuaren edukiaren arabera zehazten da. Adibidean, *socket.on('mugi')* dugu, baina baita *socket.on('phone-connect')* mezu mota ere.

```
const serverURL = window.location.host;

// websocket zerbitzarira konektatu
export const socket = io.connect(serverURL);

// socket-atik mezu bat bidali
socket.emit('desktop-connect');

// socket-atik "mugi" mezu mota datorrenean
// mugitu metodoari deitu
socket.on('mugi', mugitu);

// socket-atik "phone-connect" mezu mota datorrenean
// phoneConnected metodoari deitu
socket.on('phone-connect', phoneConnected);
```

Zerbitzariaren aldetik, kodea oso antzekoa da.

```
const io = require('socket.io')(3000);

io.on('connection', socket => {
  // posible da mezu bat zuzenean bidaltzea
  socket.send('Hello!');

  // edo emit() bidez, gertaera zehatz batzuk bidaltzea
  socket.emit('mugi', {'nondik': 12, 'nora': 14});

  // bezeroak socket.send() erabiliz gertaera bat bidali
  // → digu. Hori kudeatzeko:
  socket.on('message', (data) => {
    console.log(data);
  });

  // desktop-connect mezu mota jasotzean, kotsolan mezu
  // → bat bistaratuko dugu
  socket.on('desktop-connect', () => {
    console.log("Parametrorik gabeko deia");
  });
});
```

```
});
```

Zerbitzarian socket.io instalatzeko, `npm` komando hau erabiliko dugu:

```
npm install socket.io
```

Oharra: `npm`, `nodejs` eta orokorrean zerbitzariaren aldeko kodearen inguruan beste liburu bat idatz daiteke. Nolanahi ere, honen inguruan oinarritzko ezagutza beharrezkoa ikusten denez, NodeJS-ri buruzko kapitulu bat eskaini da liburu honetan (ikus 16. kapitulua.)

Zerbitzarian ez dugu soilik WebSocket bat izango, baizik eta hasierako HTTP protokoloa kudeatzeko zerbitzu bat ere eskaini behar dugu. Dena batera egin ahal izateko aurreko kodea apur bat aldatu beharra dago, honela:

```
const content = require('fs').readFileSync(__dirname +
  ⇨ '/index.html', 'utf8');

const httpServer = require('http').createServer((req, res)
  ⇨ => {
  // index.html orria zerbitzatu
  res.setHeader('Content-Type', 'text/html');
  res.setHeader('Content-Length',
    ⇨ Buffer.byteLength(content));
  res.end(content);
});

const io = require('socket.io')(httpServer);

io.on('connection', socket => {
  // posible da mezu bat zuzenean bidaltzea
  socket.send('Hello!');

  // edo emit() bidez, gertaera zehatz batzuk bidaltzea
  socket.emit('mugi', {"nondik": 12, "nora": 14});

  // bezeroak socket.send() erabiliz gertaera bat bidali
  ⇨ digu. Hori kudeatzeko:
  socket.on('desktop-connect', (data) => {
    console.log(data);
  });
});

httpServer.listen(3000, () => {
```

```
console.log('go to http://localhost:3000');  
});
```

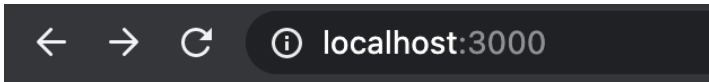
Bezeroaren azken bertsioa, berriz, honela:

```
<!DOCTYPE html>  
<html>  
<head>  
  <script type="text/javascript"  
    ↪ src="/socket.io/socket.io.js"></script>  
</head>  
  
<body>  
  
  <ul id="events"></ul>  
  
  <script>  
  
    // funtzio laguntzailea  
    const $events = document.getElementById('events');  
    const newItem = (content) => {  
      const item = document.createElement('li');  
      item.innerText = content;  
      return item;  
    };  
  
    // zerbitzariaren URLa  
    const serverURL = window.location.host;  
  
    console.log(serverURL);  
  
    // websocket zerbitzarira konektatu  
    const socket = io.connect(serverURL);  
  
    // konexioa lortzean funtzio laguntzaileari deitu eta  
    ↪ pantailan idatzi  
    socket.on('connect', () => {  
      console.log("Connect");  
      $events.appendChild(newItem('connect'));  
    });  
  
    // socket-atik mezu bat bidali  
    socket.emit('desktop-connect', {"datuak": "adibide  
    ↪ bat"});
```

```
// socket-atik "mugi" edo "phone-connect" mezu mota  
  ↪ datorrenean  
// newItem metodoari deitu  
socket.on('mugi', (datuak) => {  
    $events.appendChild(newItem(JSON.stringify(  
        ↪ datuak)));  
});  
</script>  
  
</body>  
</html>
```

Adibidea probatzeko, terminal bat ireki eta, bertatik, `node server.js` komandoa landu.

Dena ondo badoa, `http://localhost:3000` helbidean HTTP eta WS zerbitzaria entzuten egongo dira. Nabigatzailea irekitzean 14.3. irudian ikusten dena ikusi beharko genuke. Bertan, dagoen lehenengo 'connect' lerroa bezeroak zerbitzariaren kontra konexioa ondo irekitzean idatziko da. Hurrengo lerroa, berriz, zerbitzaritik 'mugi' gertaera jasotzean agertuko da (gertaera horren barruan dauden datuak bistaratuz).



- connect
- {"nondik":12,"nora":14}

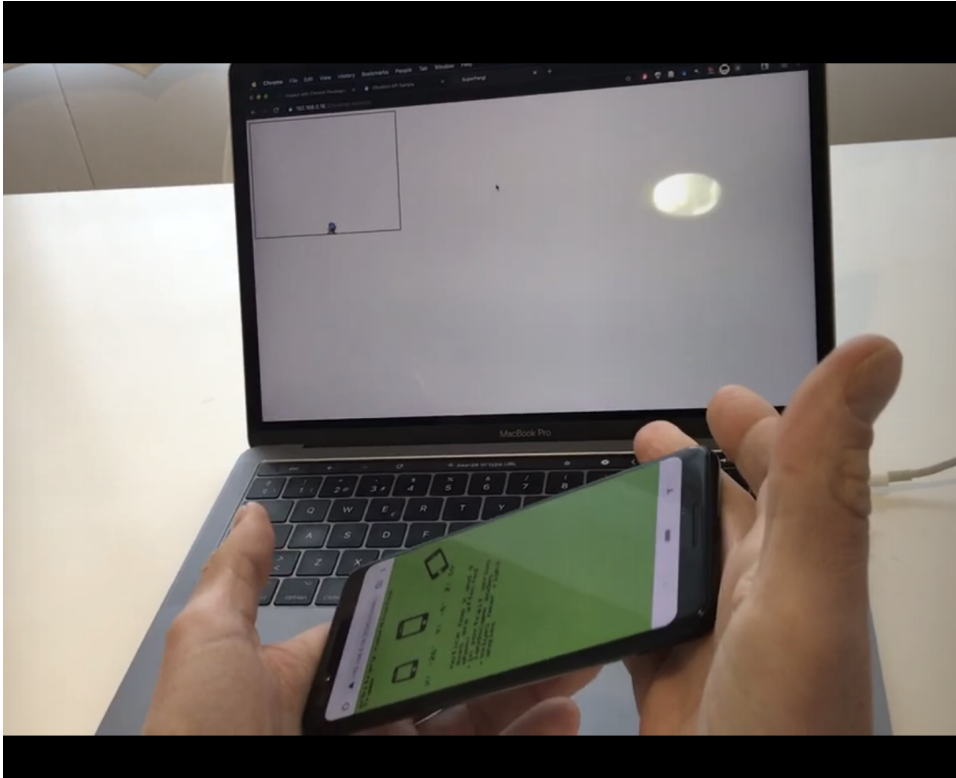
14.3. irudia: WebSocket-en adibidea martxan jartzean horrelako emaitza lortu beharko genuke.

Adibidearen kode osoa hemen aurkituko duzu: <https://labur.eus/5cEjB> (github.com).

14.2 Ariketak

WebSocket-ak eta mugikor bat erabiliko ditugu urruneko aginte bat programatzeko. Zehazki, pantailan grafiko bat bistaratuko da eta mugikorra eskuin aldera biratzean grafikoa eskuinera mugitu behar da. Ezkerrera biratzean, aldiz, grafikoa

ezker aldera mugituko da (ikus 14.4. irudia).



14.4. irudia: WebSocket-ak erabiliz mugikor batetik kontrolatu dezakegu pantailan agertzen den irudi bat, ezker-eskuinera mugituz, mugikorra alde batera edo bestera biratzean.

Mugikorraren biraketak kontrolatzeko kodea hemen aurkituko duzu: <https://labur.eus/evkQ2> (dropbox.com).

Zerbitzariaren kode-txantiloia ere lagungarria aurkituko duzu: <https://labur.eus/J9van> (github.com). NodeJS-n programatuta dago, beraz, ondo ulertzeko liburu honen NodeJS-ri buruzko gaia aztertzea gomendagarria da.

15. Zerbitzu-langileak (Service Worker-ak)

Service Worker-a (SW) nabigatzailean, atzeko planoan eta web orririk gabe exekututzen den script bat da. Script hauek egikaritzeko ez da behar ez erabiltzailearen interakziorik ezta web orririk ere. Ezaugarri horri esker, Service Worker-ak Interneteko konexiorik gabe geratzen garenean erabiltzaileari beste zerbait eskaintzeko erabil ditzakegu. Adibidez, demagun erabiltzailea inprimaki batean mezu bat idazten ari denean konexioa galdu egiten duela. Idazten ari zen mezu hori bidali nahiko balu, inprimakian bidali botoian sakatu eta... mezua galdu egingo du! Ez badugu ezer programatzen egoera hori ekiditeko, mezua galdu egingo da erabiltzailea konexiorik gabe baitago. Hemen sartzen da Service Worker APIa. SW batek konexiorik ez dagoela antzeman dezake eta mezua nabigatzailearen katxean gorde, erabiltzaileari ohartaraziz konexiorik gabe dagoela (baina mezua ez duela galdu). Konexioa berriro lortzean Service Worker-ak egoera berria antzeman eta gordeta zituen mezu guztiak zerbitzarira bidal ditzake (sinkronizazioa lortuz). Gai honetan, hain zuzen ere, deskribatutako konexiorik gabeko (*offline*) egoera hori kudeatzen ikasiko dugu.

15.1 Service Worker-ak nola erabili

Gure adibidea prestatuko dugu. Hasteko, webgune bat kargatzean, konexioa baldin badago, [15.1](#). irudiko egoera bistaratu nahiko dugu, alegia, ikur bat eta konexioa dagoela dioen mezu bat.

Baina konexioa galtzean, erabiltzaileak orri bera kargatzen badu, [15.2](#). irudian agertzen den ikurra eta mezua ikustea nahiko dugu.

Service Worker-a hasierako orria kargatzean nabigatzailearen atzeko planoan geratzen da egoiliar. Hortik, *proxy* baten lanak egiten ditu ([15.3](#). irudia) Hau da, web aplikazioak HTTP(s) eskaera bat egiten duenean SWatik igaroko da eta horrek hortik aurrera konexioak monitorizatuko ditu, *proxy* baten lanak eginez.



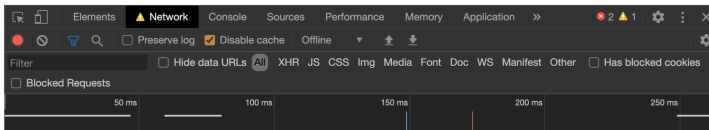
Primeran! Orain online zaude.

15.1. irudia: Service Worker-a konexioa dagoenean kargatzen da eta nabigatzai-learen atzeko planoan exekutatzen geratzen da (egoiliar).



Adi! Orain ez duzu interneteko konexiorik.

Mesedez zure konexioaren konfigurazioa berrikusi eta saiatu zaitez berriro konektatzen.



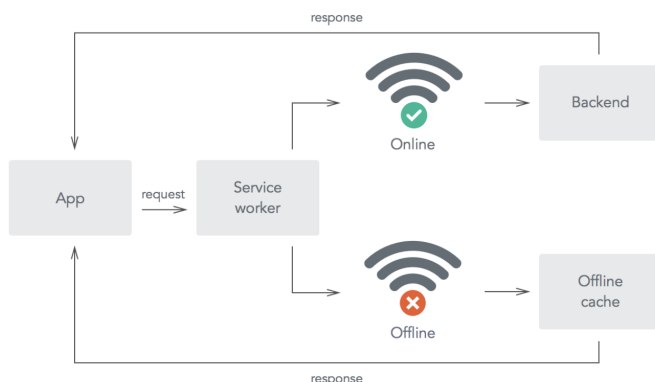
15.2. irudia: Service Worker bat probatzeko konexioa eten egin behar dugu. Horretarako Chrome-k Network fitxan *Offline* egoerara jauzi egiteko aukera ematen du.

Konexioa badago (*online* egoeran bagaude), SWa gardena izango da, dagokion webgunetik baliabidea eskatuko du eta aplikazioari itzuliko dio. Konexioa galtzen bada (*offline* egoera), alternatiba bat *cache* batetik jasoko du eta hori izango da aplikazioan bistaratuko dena. Hurrengo ataletan prozesu hau nola inplementatzen den ikasiko dugu.

15.2 Konexiorik gabeko aplikazioak

Aipatutako konexiorik gabeko aplikazioa lortzeko, hiru fasetan banatuko ditugu eman beharreko pausoak:

- SWa instalatu eta *cachean offline* orria txertatu (*offline* egoeran ikusi behar



15.3. irudia: Service Worker batek proxy-aren lanak egiten ditu *fetch* eskaerak antzematen. Konexioa dagoenean, urruneko zerbitzariari birbidaliko dizkio eskaerak. Ez dagoenean, *cache* batetik jasoko ditu. Iturria: <https://auth0.com/blog/creating-offline-first-web-apps-with-service-workers/>.

den orriaren kopia bat).

- Erabiltzailea konexiorik gabe nabigatzen saiatzen bada, *cachean* dagoen orria itzuli.
- Erabiltzaileak konexioa izanik orrian nabigatzen badu, zerbitzaritik jasoko du orriaren edukia (SWrik egongo ez balitz bezala).

15.3 Service Worker-a instalatu

Service Worker-a instalatzeko honako scripta erabiliko dugu.

```

1 <script>
2   // Service Worker-a erregistratu
3   if ('serviceWorker' in navigator) {
4     navigator.serviceWorker.register('./service-worker.js')
5     .then(function(registration) {
6       // Ondo erregistratu da
7       console.log('ServiceWorker-a irismen honekin
8         ↳ erregistratu da: ', registration.scope);
9     }).catch(function(err) {
10      // Erregistroak kale egin du :(
11      console.log('ServiceWorker-a ezin izan da
12        ↳ erregistratu: ', err);
  
```

```

11     });
12   }
13 </script>

```

Bertan ServiceWorker-a uneko nabigatzailean onartzen den edo ez egiaztatu ondoren (3. lerroa), erregistratu egiten da, promes baten bidez. Erregistratzeko `navigator.serviceWorker.register` metodoa erabiliko dugu. Metodo horrek parametro gisa SWaren kodea duen scriptaren bidea jasoko du. Parametro hori garrantzitsua da. Adibidean, `'/'` (uneko katalogoa); alegia, uneko karpeta barruan egiten diren eskaerak monitorizatuko dituela soilik (haren irismena `'/'` dela esango dugu). Edo teknikoki, *fetch* gertaerak soilik `'/'` karpeta monitorizatuko direla. Horren ordez, bidea `/adibidea/service-worker.js` izango balitz, irismena soilik `/adibidea/` karpeta barrukoa izango litzateke.

SW bat hainbat aldiz erregistratzen saiatuz gero, ez da ezer gertatuko (nabigatzaileak behin bakarrik erregistratuko du).

15.4 ServiceWorker-a programatzen

SWa programatzeko garbi izan behar dugu haren funtzionamendua (ikus 15.4. irudia). ServiceWorker bat irismen baten barruan erregistratu ondoren, irismen horren barruan dagoen eskaera berriak jasotzen hasiko da.

Eskaera horiei erantzuteko SWak `event.respondWith()` metodoa erabiliko du, erantzunaren objektua parametro gisa pasatuz. Erantzun-objektu horren edukia betetzeko hiru aukera ditu: 1) konexioa badago, eskaera dagokion zerbitzariari bidaliko dio eta horren erantzuna bueltatuko du. 2) Konexiorik egon ezean, SWak *cache*a erabil dezake erantzuna eraikitzeke, edo 3) zuzenean, erantzuna eraiki script bat exekutatuz.

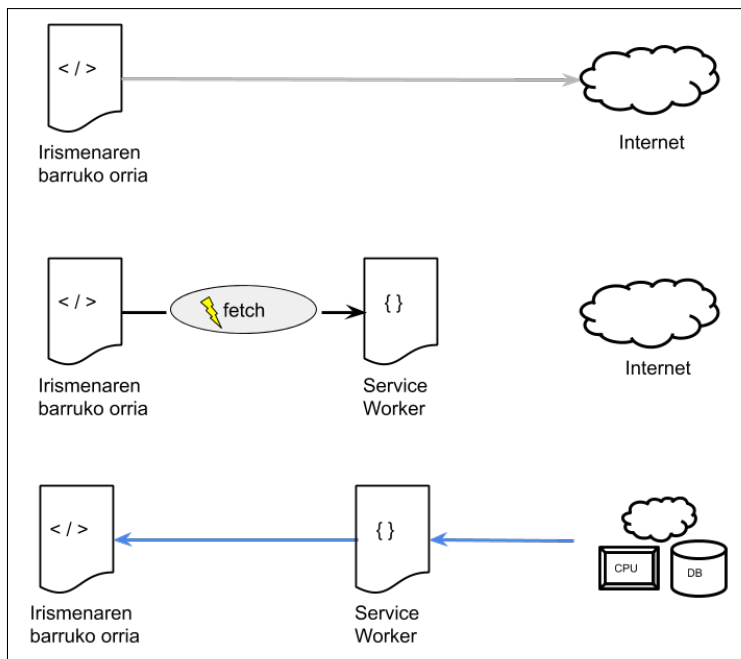
15.5 Cache-memoria betetzen

Azter dezagun SWaren hasierako prozedura *offline* edukia *cache*an gordetzeko. Horretarako, `caches` objektua erabiliko du, alegia, nabigatzaileak iraunkortasuna lortzeko eskaintzen duen mekanismo bat.

```

1 let cacheVersion = 1;
2 let currentCache = {
3   offline: 'offline-cache' + cacheVersion
4 };
5 const offlineUrl = 'offline-page.html';
6
7 this.addEventListener('install', event => {

```



15.4. irudia: *Fetch* gertaerak antzeman eta gero SWak erabaki egin behar du nondik jaso horiei erantzuteko balioak: Internetetik (kone-xioa badago), cache batetik edo zuzenean sortu programa baten bi-dez (konexiorik izan ezean). Iturria: https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API/Using_Service_Workers.

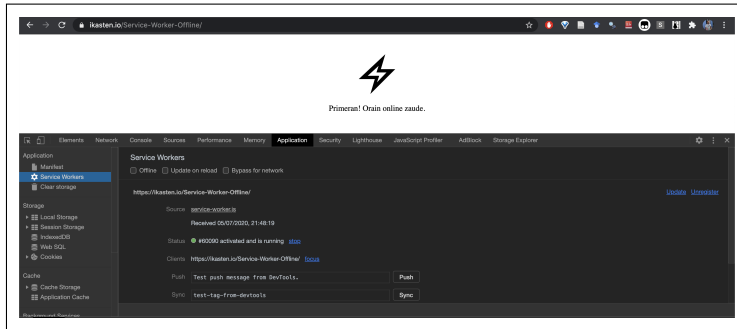
```

8   event.waitUntil(
9     caches.open(currentCache.offline).then(function(cache)
10      ↪ {
11        return cache.addAll([
12          './img/offline.svg',
13          offlineUrl
14        ]);
15      })
16    ));

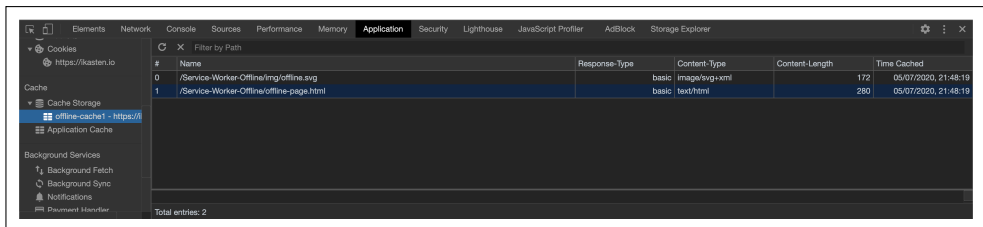
```

Install gertaeran (alegia, SWa exekutatzen den lehendabiziko aldian), 'offline-cache1' izeneko *cachea* ireki (izen hori guk erabaki dugu) eta bertan bi baliabide gordeko ditu (`cache.addAll` metodoari bi elementuren arraya pasatuz): “./img/offline.svg” (ikurra) eta offline-page.html (konexiorik gabe gaudenean bistaratu nahi dugun HTML orria). *Cachean* sartutako fitxategi horiek 15.5. irudian ikus daitezke. `waitUntil` metodoak (8. lerroa) promes bat hartzen du parametro gisa eta

promes hori bete arte itxarongo du. Promesa bi modutara bete daiteke: *cachean* gorde diren fitxategi guztiak ondo instalatzen badira, zuzen beteko da (*resolved*); *cachean* ezin izan bada fitxategiren bat sartu, errore batekin beteko da eta SWa ez da instalatuko (*rejected*).



15.5. irudia: Service Worker-ak kudeatzeko edo arazteko DevTools tresnak atal berezia dauka **Application/Service Workers** fitxan.



15.6. irudia: *Offline Cachea*-ren edukia ikusteko DevTools tresnak atal berezi bat eskaintzen du **Application/Cache/Cache Storage** fitxan.

Behin instalatuta dagoenean, SWak monitorizatu egingo ditu erabiltzailearen mugimenduak (irismenaren barruan dauden mugimenduak). Alegia, beste orri batera joateko estekan sakatzeari *fetch* gertaera altxatuko da eta SWak jasoko du.

Bertan, SWak HTML orri bat eskatu den begiratu du eta horrela baliatuz aipatutako `event.respondWith` metodoaz baliatuko da erantzuna emateko. Hurrengo kode-adibidean, 8. lerroan SWak `fetch()` metodoari deituko dio, urruneko zerbitzaritik eskatutako baliabidea jaitsi nahian. Ezin badu (konexiorik ez dagoelako, adibidez), salbuespen bat altxatuko da, *catch* blokean tratatuko dena. Bertan SWak *cachetik* *offlineUrl*-a jaso eta erabiltzaileari bidaliko dio erantzun gisa.

Ohart gaitezen *offlineUrl*-ak SVG bat erabiltzen duela (ez HTML orri bat), beraz, irudi hori jaisteko *fetch* gertaera datorrenean SWa *if* baldintzaren *else* ada-

rretik sartuko da (12. lerroa). Hor SVGa *cachean* dagoen begiratuko du eta horrela balitz (kasu honetan bai, badago), *cachean* dagoen baliabidea itzuliko du erantzun gisa. Ez balego, *fetch()* metodoari deituko lioke Internetetik atzitu nahian (eta hor ere ez balego, errore bat emango luke).

```

1  this.addEventListener('fetch', event => {
2    // request.mode = navigate ez dago eskuragarri
   ↳ nabigatzaile guztietan
3    // horregatik sartu dugu baldintza alternatibo bat
4    // Accept: text/html goiburua aztertuz
5    if (event.request.mode === 'navigate' ||
   ↳ (event.request.method === 'GET' &&
   ↳ event.request.headers.get('accept')
   ↳ includes('text/html')))) {
6      event.respondWith(
7        fetch(event.request.url).catch(error => {
8          // offline orria bueltatu
9          return caches.match(offlineUrl);
10       })
11     );
12   }
13   else{
14     // erantzun ahal denarekin (cache-tik edo hor ez
   ↳ balego,
15     // internetetik (noski, konexioa badago, bestela
   ↳ errore bat jasoko dugu)
16     event.respondWith(caches.match(event.request)
17                       .then(function (response) {
18                         return response ||
   ↳ fetch(event.request);
19                       })
20     );
21   }
22 });

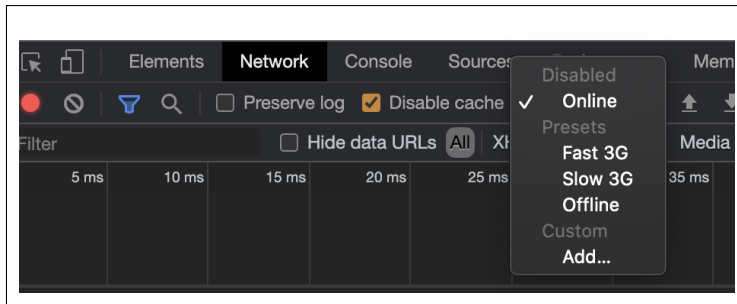
```

Gai honetan zehar landu dugun adibidearen kodea Dean Hume erabiltzailearen GitHubeko biltegi honetan oinarritu da: <https://github.com/deanhume/Service-Worker-Offline>.

Adibidearen inplementazioa hemen proba dezakezu: <https://ikasten.io/Service-Worker-Offline>.

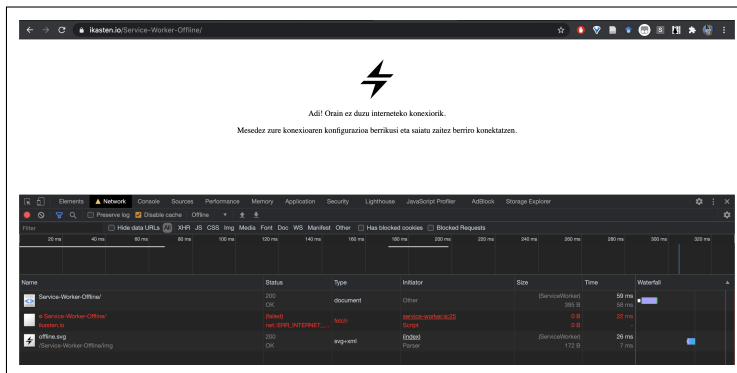
Adibidea probatzeko, Chromen DevTools ireki eta bertan Network fitxan sakatu. Jarraian, aukera-zerrendan *online* egoeratik *offline* egoerara pasa (ikus 15.7. irudia).

Offline gaudelarik orria birkargatzen badugu, SWak egoera antzemango du eta



15.7. irudia: Google Chromek Online egoetatik Offline egoerara jauzia egiteko aukera ematen digu. Service Worker APIarekin probak egiteko ezin egokiagoa.

offline bertsioa erakutsiko digu (ikus 15.8. irudia).

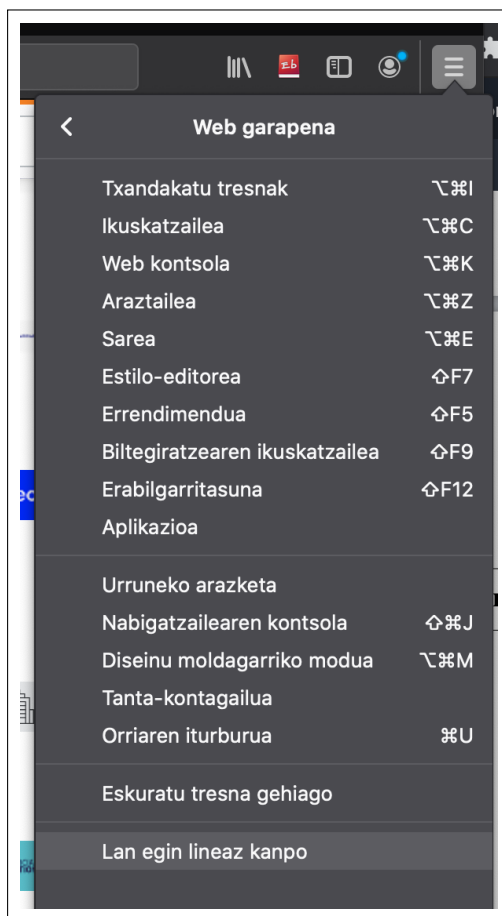


15.8. irudia: Interneteko konexiorik gabe gaudenean Service Worker batek *offline-cache*an dugun edukia bistara dezake, eta horrekin erabiltzaileak lan egiten jarrai dezake, konexioa bueltatu arte.

15.6 Ariketa

Gai honetan azaldu den adibidea implementatu eta probatu, bai Chrome-n nola Firefox-en.

Ohart zaitez Firefox-en, Chrome-n bezala, lineaz kanpo lan egiteko aukera bat ere badugula Web Garapenerako menuaren azken lerroan (ikus 15.9. irudia).



15.9. irudia: Firefox-en eta Chrome-n *throttling* izeneko aukerak ditugu, alegia, konexio-abiaduraren baldintzak simulatzeko aukerak. Firefox-en, berriz, konexio-rik gabe lan egiteko aukera menu honetan isolatuta dago.

16. NodeJS

16.1 Sarrera

Node.js plataformarekin JavaScript aplikazioak zerbitzarian programatu ahal izango ditugu. Orain arte HTML5ek eskaintzen dituen APIekin egin dugun guztia bezero aldean izan da. Baina web aplikazio batek, orokorrean, bezeroa eta zerbitzari bat behar ditu. Zerbitzariaren aldean programatzeko hainbat aukera ditugu: PHP, ASP.NET, Ruby, Python... eta JavaScript. Azken lengoaia horretan zerbitzarian sortuko ditugun aplikazioak programatzeko, NodeJS (askotan node izenaz laburbilduta) erabiliko dugu. Node Google Chrome-k eskaintzen duen JavaScript motorrean (v8 deritzona) dago oinarrituta. Gaur egun milaka aplikazio daude Node-n programatuta, eta haren inguruan sortu den ekosistema sekulakoa da.

Liburu honetan ez dugu asko sakonduko Node-n (azken finean HTML5en inguruko liburu bat da!), baina gutxienez planteatutako ariketa batzuk egin ahal izateko, gainetik ezagutu behar dugu nola sortu eta nola erabili Node-n egindako aplikazio bat.

Adibide sinple batekin hasiko gara: betiko “*Hello World*” edo “*Kaixo mundua*” inprimatzen duen HTTP zerbitzari bat programatzeko dugu, 4 lerrotan. Has-teko, sortu `zerbitzaria.js` izeneko fitxategia, honako edukiarekin:

```
1 const http = require('http');
2 http.createServer (function (request, response) {
3   response.writeHead(200, {'Content-Type': 'text/plain'});
4   response.end('Kaixo Mundua\n');
5 }).listen(8000);
6
7 console.log('Zerbitzaria http://127.0.0.1:8000/
  ↳ eskuragarri duzu');
```

Lehenengo lerroan `http` modulua kargatzen dugu `require` aginduaren bidez. `http` moduluak HTTP zerbitzari bat sortzeko aukera ematen du (2. lerroan), 8000

portuan entzuten jarriko dena. Eskaera (*request*) bat jasotzean HTTP/200 kodearekin erantzungo du, “Kaixo Mundua” edukia bidaliz. Azkenengo lerroak zerbitzariko kontsolan mezu bat pantailaratuko du, informazio gisa.

Martxan jartzeko honako komandoa landuko dugu:

```
node zerbitzaria.js
```



16.1. irudia: Node-n egindako HTTP zerbitzari soil bat.

Prestatu dugun HTTP zerbitzaria gure probatxoak egiteko ondo dago, baina ezin dugu programatu JavaScript lerro bat eskaera mota bakoitzeko! Ziur asko, hasieran jada HTML, JS, CSS eta irudiak prestatuta izango ditugu karpeta batean eta horiek HTTP bidez eskaini nahiko ditugu, ezer programatu gabe. Horretarako (fitxategi estatikoak eskaintzeko), node-k *serve* izeneko HTTP zerbitzari single bat eskaintzen du, *npm* pakete-kudeatzailearekin instalatu behar dena. *npm* (*node package manager*) node-rako paketeak kudeatzeko sistema bat da. Haren erabilera erraza demostratzeko, ikus dezagun nola instala dezakegun *serve* paketea *npm*-aren bidez.



Nondik deskargatu node?

NodeJS bere webgune ofizialetik jaits dezakegu (<https://nodejs.org>), Linux, MacOS edo Windows-erako.

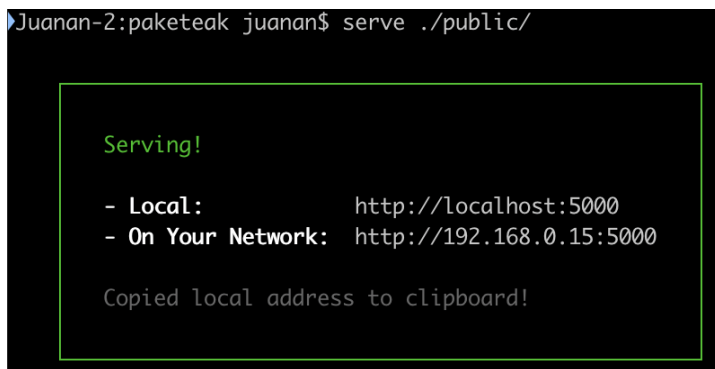
Hasteko, lehenengo komandoa `npm init` izango da. Horrekin, mendekotasunak dituen node proiektu berri bat prestatuko dugu. `npm init` prozesuak galdera asko egiten badu ere, hasieran guztiei *Enter* tekla sakatuz erantzungo diegu. Prozesua bukatzean `package.json` izeneko fitxategi bat sortuko da. Bertan, *npm*-ak gure aplikaziorako beharrezkoak diren mendekotasunak apuntatuko ditu.

Ondoren, *serve* paketea instalatzeko, `npm install serve` idatziko dugu. Ohartzaitez nola sortzen den *node_modules* izeneko karpeta bat. Horren barruan *serve* zerbitzariak dituen mendekotasun guztiak instalatu dira automatikoki.

Jarraian, `public/` izeneko karpeta bat sortuko dugu eta bertan HTML orri bat (adibidez, `index.html`, bertan “Kaixo mundua!” idazten duena, berriro ere) sartuko dugu.

Dena prest *serve* zerbitzaria martxan jartzeko!:

```
serve ./public
```

A terminal window with a black background and green text. The prompt is 'Juanan-2:paketeak juanan\$'. The command 'serve ./public/' has been executed. The output shows 'Serving!' in green, followed by two lines: '- Local: http://localhost:5000' and '- On Your Network: http://192.168.0.15:5000'. At the bottom, it says 'Copied local address to clipboard!' in a lighter green color.

```
Juanan-2:paketeak juanan$ serve ./public/

Serving!

- Local:      http://localhost:5000
- On Your Network: http://192.168.0.15:5000

Copied local address to clipboard!
```

16.2. irudia: *Serve* paketearen laguntzaz web zerbitzari oso bat presta dezakegu, (adibidean) 5000 portuan entzuten, lerro bakar batekin: `serve ./public`.

Zerbitzaria modu erosoan martxan jarri ahal izateko, `npm` komando sinple bat presta dezakegu. Lortu nahi duguna zera da, `npm start` idaztean, automatikoki `serve ./public` exekutatzea. Horretarako, *package.json* fitxategia editatuko dugu, zehazki *scripts* atala, honela gera dadin:

```
{
  "name": "paketeak",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "serve ./public",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "serve": "^11.3.0"
  }
}
```

**npm : node package manager**

npm node-rako packageak instalatzeko eta kudeatzeko aplikazio bat da. **npm install X** egitean, X izeneko paketea —eta haren mendekotasun guztiak— *node_modules* izeneko karpetan gordeko da. Guztira, gaur (2020/05/26), 1.100.000 **pakete** baino gehiago daude^a.

^a<https://twitter.com/juanan/status/1265236302934478851>

Serve zerbitzaria oso egokia da baliabide estatikoak (html, css, js, irudiak...) HTTP bidez zerbitzatzeko, baina batzuetan zerbait dinamikoagoa beharko dugu, adibidez inprimaki baten datuak jaso eta datu-base batean gordetzeko, edo webgune profesional bat zerbitzatzeko. Horrelakoetan, *serve* eskas samar geratuko zaigu, eta beste alternatiba bat beharko dugu, hala nola *express*. Hori da, hain zuzen ere, hurrengo kapituluan aztertuko dugun gaia.

16.2 Ariketa

Probatu gai honetan azaldu diren adibideak. Horretarako, NodeJS instalatu behar duzu. Baita `serve` moduloa ere. Garrantzitsua da hau egitea, hurrengo gaien inguruko ariketak egin nahi badituzu (NodeJS, MongoDB eta ExpressJS uztartzen dituen ariketa bat planteatuko baita hurrengo gaietan).

17. ExpressJS

Express edo ExpressJS kode irekiko Node-rako web aplikazio dinamikoak programatzeko *framework* bat da. Kapitulu honetan adibide batekin egingo dugu lan: bezeroen datuak kudeatzeko aplikazio sinple bat programatuko dugu, hasiera-hasieratik.

Bezeroen izen-abizenak eta helbidea elektronikoa gorde, editatu, ezabatu eta bistartzeko inprimaki bat izango dugu. Lehendabiziko bertsioan datu horiek soilik memorian gordeko ditugu. Bigarren bertsioan, bezeroen datuak *noSQL* motako datu-base batean gordeko ditugu (*mongodb* izeneko datu-base batean, hain zuzen ere).

Has gaitezen, lehenengo eta behin, *express framework*-a instalatzen.

Lehenengo eta behin aplikazioa hasieratu beharko dugu, `npm init` komandoaz.

```
$ npm init
Aplikazioaren izena: bezeroak
Bertsioa: 1.0
Deskribapena: bezeroen kudeaketa
Entry Point (nondik hasten den aplikazioa): app.js
Testak egiteko komandoa: momentuz hutsik
Git reepo: momentuz hutsik
keywords: hitz gakoak, momentuz hutsik
author: zure izena
lizentzia: momentuz hutsik
```

`npm init` komandoa exekutatu ondoren, *package.json* fitxategia sortuko da.

Jarraian, *express* eta haren mendekotasun guztiak instalatuko ditugu:

```
$ npm install express
```

Bukatzeko, HTTP zerbitzari sinple bat programatuko dugu:

```
1 // app.js fitxategia
2
```

```

3 let express = require('express');
4
5 let app = express();
6 app.listen( 3000, function() {
7   console.log("Zerbitzaria 3000 portuan entzuten");
8 })

```

Zerbitzaria martxan jartzeko, `node app.js` komandoa landuko dugu.

Orain `http://localhost:3000/` URLan sartzen bagara, erantzun bat jasoko dugu, baina ziur asko ez da guk espero dugun erantzuna izango:

```
"Cannot GET /"
```

Erroreak esan nahi du zerbitzariak ez dakiela nola erantzun bidali diogun GET motako eskaerari. Alegia, `/` bidea (*route*) ez dugula tratatu zerbitzarian. GET motako bideak tratatzeko, metodo berri bat programatu behar dugu:

```

1 app.get("/", function(req, res) {
2   res.send("Kaixo mundua!");
3 });

```

Orain bai, URL bera eskatzean (`http://localhost:3000/`), “Kaixo mundua!” mezua jasoko dugu erantzun gisa.

17.1 Baliabide estatikoak

`app.get()` edo `app.post()` erabili behar dugu edozein HTML orria zerbitzatzeko? Ezin dugu HTML5en egindako orri estatiko bat zerbitzatu beste modu egokiago batean? Noski, horretarako *express* aplikazioaren bideraketak zehaztu baino lehen (`app.get`, `app.post` zehaztu baino lehen) baliabide estatikoen karpeta non dagoen adierazi beharko dugu (kasu honetan, *public* izeneko karpeta):

```
1 app.use(express.static('public'))
```

`app.use` egiturari *middleware* deitzen zaio. Bideraketak baino lehenago egin behar diren beste ataza eta prozesuak zehaztuko ditugu *middleware* funtzioekin. Hurrengo ataletan gehiago sakonduko dugu *middleware* honetan, baina momentuz horrela utziko dugu.

Adibide gisa, *public* karpetan orri estatiko hau utziko dugu, inprimaki sinple batekin, *post.html* izenarekin:

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>POST metodoa erabiltzen duen inprimakia</title>

```

```

6 </head>
7 <body>
8 <form action="/bezeroa" method="POST">
9 Izena: <input type="text" name="izena"><input
    ↪ type="submit" value="Bidali">
10 </form>
11 </body>
12 </html>

```

Nabigatzailean orri hori atzitzeko hurrengo URLa idatziko dugu:

`http://localhost:3000/post.html`

Ohart zaitez URLan ez dugula `public/post.html` idatzi. *Express*-ek badaki besterik ezean, HTML orriaren fitxategi estatikoak `public/` karpetan egon behar duela.

17.2 POST parametroak

Inprimaki baten datuak orokorrean POST bidez bidaltzen dira, ez GET bidez. Nola jaso POST balio horiek zerbitzarian? Eskaera (*request*) baten *body* atributuan daude parametro hauek. Adibidez, *req.body.izena*. Baina, besterik gabe, hau inplementatzen badugu:

```

1 app.post('/bezeroa', function(req, res) {
2   res.send("Bezeroaren izena:" + req.body.izena;
3 }

```

ez dugu ezer jasoko. Zergatik? Arazoa da POST parametroak bide edo *route* batean sartu baino lehen tratatu egin behar direla. Nabigatzaileak bidaltzen dituen datuak zerbitzarian jaso eta bideratu baino lehen tratatu egin nahi baditugu, *express*-en middleware izeneko funtzioak erabili beharko ditugu.

Middleware bat erabiltzeko, aurreko gaian ikusi dugun lez, *app.use(funtzioa)* erabiliko dugu, bideraketak egin baino lehen. Kasu honetan, *express.urlencoded* izeneko middlewarea. Adibidez, POST bidez datozen parametroak tratatzeko:

```

1 // application/x-www-form-urlencoded
2 app.use(express.urlencoded({ extended: true }));

```

Horrekin aurreko kodeak ondo funtzionatuko du.

```

1 // parseatu application/x-www-form-urlencoded datuak
2 app.use(express.urlencoded({ extended: true }));
3 app.post('/bezeroa', function(req, res) {
4   res.send("Bezeroaren izena:" + req.body.izena;
5 }

```

x-www-form-urlencoded

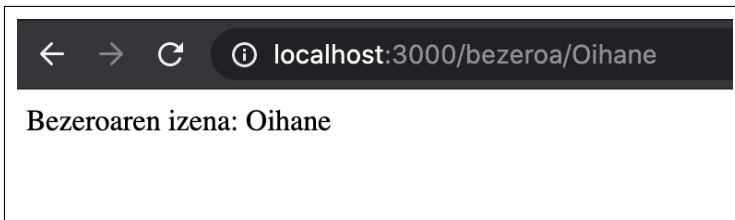
17.2. irudian ikusten dugunez, form baten datuak POST bidez bidaltzean, besterik ezean, HTTP eskaeraren eduki mota (Content-Type) `application/x-www-form-urlencoded` izango da.

Bestalde, urlencoded parametroak tratatzeko bi liburutegi daude: `querystring` eta `qs`. `express.urlencoded( extended: true )` aukerarekin `qs` erabili nahi dugula esaten dugu. Bestalde, `extended: false` erabiliz gero, `querystring` liburutegia erabiliko dugula adierazten dugu. Ikus: <https://github.com/expressjs/body-parser#extended>

17.3 GET parametroak URLan bidaltzen

Nola bidali ahal diogu gure *express* aplikazioari GET parametro bat? Alegia, demagun horrelako URL bat idatzita: `http://localhost:3000/bezeroa/Oihane` zerbitzariak eskaera jaso eta “Kaixo Oihane” mezuarekin erantzun behar duela (ikus 17.1. irudia). URLan agertzen den izena jasotzeko `req.params` objektua erabiliko dugu, honela:

```
app.get('/bezeroa/:izena', function(req, res) {
  res.send("Bezeroaren izena: " + req.params.izena);
});
```



17.1. irudia: URL batean bidaltzen diren parametroak (*URL parameters*) jaso eta tratatu ahal dira `req.params` erabiliz.

17.4 GET parametroak eskaeran bidaltzen

GET metodoan parametroak URLan bi modutara bidali daitezke: URL bidearen parte izanik (ikus dugun bezala) edo URLaren azken zatian, eskaera (*query string*)

izeneko parametroetan, galdera-ikur baten ostean, honela:

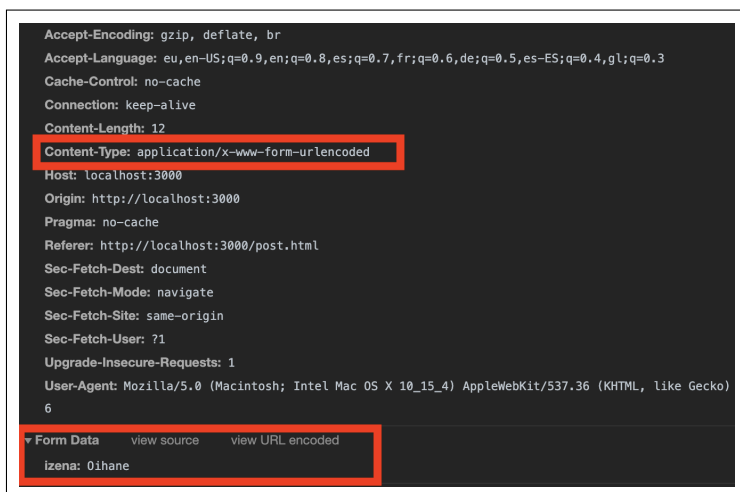
<http://localhost:3000/bezeroa/?izena=Oihane>.

Eskaera-parametro horiek jasotzeko, *req.params* erabili ordeaz, *req.query* erabiliko dugu. Adibidez:

```
app.get('/bezeroa', function(req, res) {  
  res.send("Bezeroaren izena: " + req.query.izena);  
});
```

Parametro bat baino gehiago bidali nahi izanez gero, aldagaiak & karaktereaz banatuko ditugu, honela: <http://localhost:3000/bezeroa/?izena=Oihane&abizena=Agirre>.

```
app.get('/bezeroa', function(req, res) {  
  res.send(`Bezeroaren izena: ${req.query.izena}  
    ↪ ${req.query.abizena}`);  
});
```



17.2. irudia: Form baten datuak bidaltzeko bi metodo daude: POST eta GET. GET kasuan, datuak URLan ikusiko dira. POST kasuan, HTTP eskaeraren edukian joango dira eta eskaeraren eduki mota *application/x-www-form-urlencoded* izango da.

17.5 Nodemon

Orain arte `index.js` edo beste edozein JS fitxategitan aldaketa bat egitean, *express* zerbitzaria itzali eta berriro martxan jarri behar izan dugu, aldaketak har zitzan.

Baina edizio-prozesu hori oso astuna bilaka daiteke. Hoberena izango litzateke modulu bat izatea JS fitxategietan aldaketak automatikoki antzematen dituen eta berak bakarrik zerbitzaria berrabiarazten duena. Hori da, hain zuzen ere, **nodemon** izeneko tresnaren zeregina. **nodemon** instalatzeko **npm** erabiliko dugu:

```
$ npm install -g nodemon
```

-g parametroak instalazio globala egin nahi dugula adierazten du. Hortik aurrera, erabiltzaile guztiek eskuragarri izango dute **nodemon** aplikazioa, ez soilik uneko erabiltzaileak. Behin instalatuta erabili egingo dugu:

```
$ nodemon index.js
```

Orain, edozein aldaketa egiten badugu **index.js** fitxategian, **nodemon**-ek automatikoki aldaketa antzeman eta zerbitzaria automatikoki berrabiaraziko du.

17.6 Txantiloiak: EJS

Web garapenean diharduten enpresetan web programatzaileak eta web diseinatzaileak izaten dituzte. Orokorrean, programatzaileek aplikazioaren kodea programatzen dute eta diseinatzaileek web orrien itxura moldatzen dute, alegia, orrien diseinu grafikoa. Diseinatzaileek orri baten txantiloia prestatzen dute gero programatzaileek aplikazioaren datuekin betetzeko (adibidez, datu-base batetik ekarritako datuekin). **Express**-ek ere **HTML** txantiloia baten gainean datuak bistartzeko aukera ematen du. Izan ere, hainbat modulu daude txantiloiekin lan egiteko (*ejs*, *pug*, *mustache*, *handlebars*, *jade*...). Liburu honetan **ejs** erabiliko dugu, erraza baita berearekin lan egitea.

Lehenengo eta behin, *ejs* instalatu egin behar da:

```
$ npm install ejs
```

Jarraian, txantiloiak kudeatzeko **ejs** erabiliko dugula zehaztuko dugu (9. lehenengoan):

```
const express = require('express');
const path = require('path');

const app = express();
app.use(express.urlencoded({ extended: true }));
app.use(express.static('public'))

app.set('view engine', 'ejs');
app.set('views', path.join(__dirname, 'views'));
```

Azkenengo lerroan txantiloia zer karpetatan gordeko ditugun adierazten da (lerro horrek *path* moduluarekin mendekotasun bat dauka, horregatik inportatu behar dugu 2. lerroan)

Orain, erro-katalogoa eskatzean, ez dugu zuzenean “Kaixo mundua!” katearekin erantzuko:

```
app.get("/", (req, res) => {
  res.send("Kaixo mundua!");
});
```

baizik eta, horren ordez, `index.ejs` txantiloia bidaliko dugu:

```
app.get("/", (req, res) => {
  res.render('index');
});
```

`index.ejs` txantiloiaaren edukian 17.3. irudian daukagun kodea sartuko dugu, hots:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>EJS probatzen</title>
</head>
<body>
  Kaixo EJS txantiloia kudeatzaileretik
</body>
</html>
```

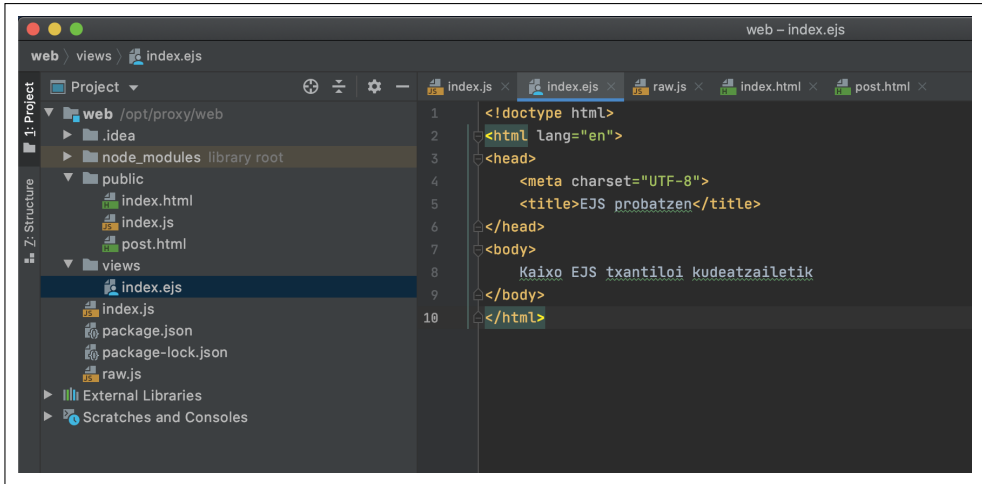
Orain arte ikusi dugunaren arabera, ez dugu inolako aldaketarik somatzen EJS txantiloia baten eta HTML orri estatiko baten aldean (public/ karpetan ditugun orrien artean). Ezberdintasun nagusia da txantiloiek parametroak onartzen dituztela. Adibidez, `app.get('/') kudeatzaileretik` parametro bat pasatu nahi badiogu txantiloia, honakoa idatziko dugu `index.ejs` fitxategiaren *title* atalean, izenburua parametro gisa jasotzeko:

```
1 <title><%=izenburua %></title>
```

Alegia, EJS aldagai bat `<% %>` ikurren artean sartuko dugu. Eta aldagai hori txantiloian bistaratu nahi badugu, `<%=aldagaia%>` idatziko dugu.

Bukatzeko, `app.get('/') kudeatzailea` honela geratuko da:

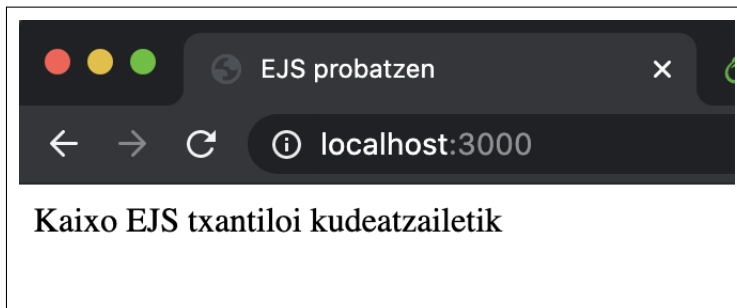
```
1 app.get("/", function(req, res) {
2   res.render('index', {
3     'izenburua': 'EJS probatzen'
4   })
5 });
```



17.3. irudia: EJS txantiloia probatzeko, `views/` izeneko karpeta bat sortu dugu eta bertan `index.ejs` fitxategia sartu.

Txantiloia ondo dabilela probatu baino lehen, `public/` karpetatik `index.html` ezabatu egingo dugu. Ez badugu egiten, *express*-ek hor aurkituko du `index` izeneko orri bat eta hori hobetsiko du txantiloiaaren aurrean.

Dena ondo badoa, `http://localhost:3000` webgunea atzitzean, 17.5. irudian ikusten dugun emaitza lortu beharko genuke.



17.4. irudia: EJS txantiloiek parametroak onartzen dituzte. Izenburuan ikus dezakegu orri honi pasatu diogun parametroa.

17.6.1 Txantiloia betetzen

Demagun hiru bezeroren datuak ditugula:

```
let bezeroak = [
  {
    id: 1,
    izena: 'Ane',
    abizena: 'Uriarte',
    email: 'ane@ni.eus'
  },
  {
    id: 2,
    izena: 'Jon',
    abizena: 'Juanenea',
    email: 'jon@ni.eus'
  },
  {
    id: 3,
    izena: 'Oihane',
    abizena: 'Lete',
    email: 'oihane@ni.eus'
  },
]
```

Orain, datu horiek web orrian bistaratu nahi baditugu, parametro gisa pasa behar dizkiogu `index.ejs` txantiloari:

```
app.get("/", function(req, res) {
  res.render('index', {
    'izenburua': 'EJS probatzen',
    'bezeroak': bezeroak
  })
});
```

Eta `index.ejs` txantiloian datuak zeharkatu, *forEach* egitura erabiliz:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title><%=izenburua%></title>
</head>
<body>
  Kaixo EJS txantiloia kudeatzailetik
  <ul>
    <%= bezeroak.forEach( function(bezeroa) { %>
      <li><%=bezeroa.izena %> <%=bezeroa.abizena%></li>
    <% }) %>
  </ul>
</body></html>
```

Dena ondo badoa, `http://localhost:3000/` helbidean sartzean, 17.5. irudiaren emaitza ikusiko dugu.



17.5. irudia: Bezzeroen datuak JSON array batean EJS txantiloari ondo bidaltzen badizkiogu, honela ikusi beharko litzateke nabigatzailean.

17.7 Ariketak

- Prestatu inprimaki bat `index.ejs` txantiloian, bezero berri baten datuak (ize-na, abizena eta helbide elektronikoa) zerbitzarira POST bidez bidaltzeko. Zerbitzariak datuak jaso eta pantailan bistaratuko ditu.
- Bezero bakoitzaren ondoan [x] ikurra idatzi. [x] horretan klik egitean bezero hori bezero zerrendatik ezabatu eta zerrenda berria pantailaratu.

Espero den emaitza ondo ulertzeko, ikus <https://youtu.be/2zE4M5HSJRY>

18. Nola testatu Express aplikazioak

Kapitulu honetan Express aplikazio bat testatzen ikasiko dugu. Hemen proba unitarioak erabiliko ditugu soilik (proba funtzionalak, integrazio probak, erregresio probak etab. alde batera utziz). Eskarmentua baduzu Javaz landutako proba unitarioekin (*JUnit frameworka* erabiliz), bide luze bat jada aurreratu duzu: express-en egindako aplikazioak berdin testatu daitezke, baina beste framework pare baten laguntzarekin: eta .

1. Gauzak probatzen hasteko, express aplikazio sinple bat sortuko dugu:

```
$ express --view=ejs express-test
$ cd express-test
$ npm install
```

2. jest, supertest eta cross-env paketeak instalatu

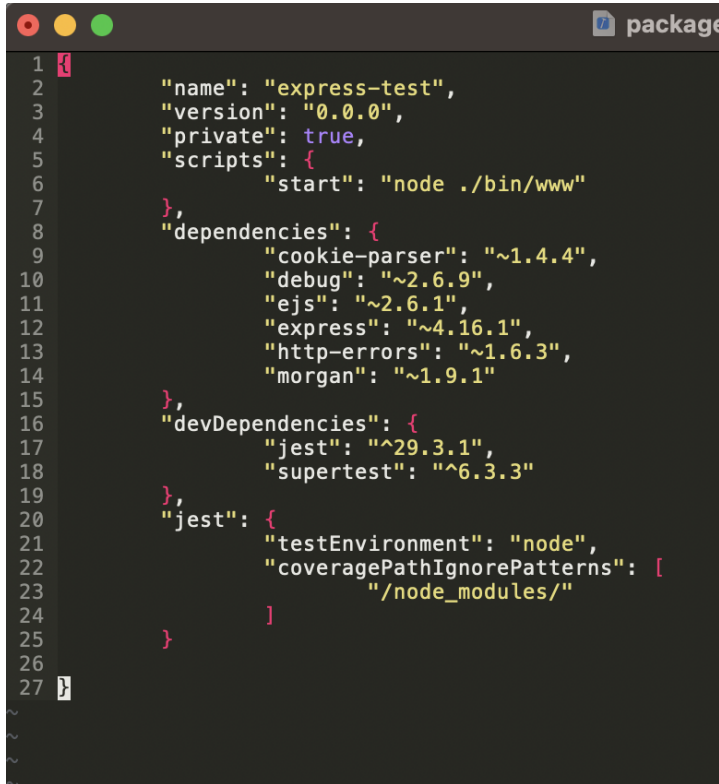
(-D aukerak devDependencies - alegia, garatze-garaiko mendekotasunak instalatzen ditu. Pakete horiek ez dira beharrezkoak aplikazioa martxan jartzeko edota erabiltzeko, soilik testatzeko/garatzeko erabiliko ditugu)

```
$ npm install -D jest supertest
$ npm install -g cross-env
```

3. package.json fixategian hau txertatu (node aplikazioak testatu nahi ditugula esaten ari gara hemen, eta node_modules karpeta baren barruan dagoena ez dugula testatu nahi)

```
....
"jest": {
  "testEnvironment": "node",
  "coveragePathIgnorePatterns": [
    "/node_modules/"
  ]
}...
```

Gutxi gora-behera horrelako package.json bat lortuko duzu:



```

1  {
2      "name": "express-test",
3      "version": "0.0.0",
4      "private": true,
5      "scripts": {
6          "start": "node ./bin/www"
7      },
8      "dependencies": {
9          "cookie-parser": "~1.4.4",
10         "debug": "~2.6.9",
11         "ejs": "~2.6.1",
12         "express": "~4.16.1",
13         "http-errors": "~1.6.3",
14         "morgan": "~1.9.1"
15     },
16     "devDependencies": {
17         "jest": "^29.3.1",
18         "supertest": "^6.3.3"
19     },
20     "jest": {
21         "testEnvironment": "node",
22         "coveragePathIgnorePatterns": [
23             "/node_modules/"
24         ]
25     }
26 }
  
```

18.1. irudia: Testing: package json.

4. Jarraian, routes/users.js ezabatu eta routes/index.js fitxategiaren edukia Gist honen edukiarekin ordezkatu:

<https://gist.github.com/juananpe/8aa0cd571f07df8f278015ec476d3ecf>

Bertan, horrelako objektuak gorde (send), aldatu (update) eta ezabatzeko (destroy) rutak prestatu ditugu:

```

const fakeDB = [
  {
    id: Math.floor(Math.random() * 100),
    email: "test@example.com",
  },
];
  
```

- Oharra: adibidea errazteko, fakeDB objektua sortu dugu, mongoDB database bat simulatzen duena, array baten bidez.
- Oharra: bi lerro hauek ezabatu app.js fitxategitik:

```
var usersRouter = require('./routes/users');
....
app.use('/users', usersRouter);
```

5. package.json fitxategian, scripts atala horrela moldatu:

```
"scripts": {
  "start": "node ./bin/www",
  "test": "cross-env NODE_ENV=test jest"
},
```

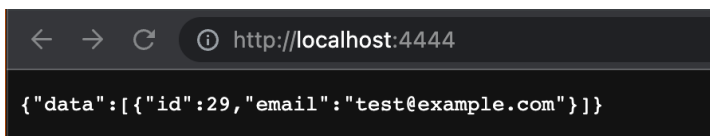
Oharra: NODE_ENV bidez testeatzeko ingurune batean ari garela zehazten ari gara. Besteak beste 404 erroreak baldin badaude eta test ingurunean bagaude, orduan erroreak zergatia bistaratzen duen traza bat agertuko zaiugu (traza horiek ez dira agertu behar production ingurunean, segurtasunaren aldetik informazio gehiegi - eta pribatua- ematen dutelako)

6. Aplikazioa martxan jarri:

```
cross-env PORT=4444 DEBUG="express-test:*" nodemon start
```

eta probatu ondo dabilela (eskuz, testak oraindik ez ditugu prestatu eta)

Zehazki, <http://localhost:4444/> bidean, erantzun hau jaso beharko genuke:



```
{"data":[{"id":29,"email":"test@example.com"}]}
```

Berriz, POST **/send** endpoint-era eginez gero:

```
juananeu033782 > /opt/express-test > curl -X POST http://localhost:4444/send -d '{"email":"ane@euskadi.eus"}'
{"data":[{"id":24,"email":"test@example.com"}, {"id":25,"email":"ane@euskadi.eus"}]}
```

PUT endpoint-a probatzeko:

```
juananeu033782 > /opt/express-test > curl -X PUT http://localhost:4444/update/24 -d '{"email":"jota@pe.com"}'
{"data":[{"id":24,"email":"jota@pe.com"}, {"id":25,"email":"ane@euskadi.eus"}]}
```

eta DELETE endpoint-a probatzeko:

```
juaan@eu033782:~/opt/express-test$ curl -X DELETE http://localhost:4444/destroy/24
{"data":[{"id":25,"email":"ane@euskadi.eus"}]}
```

7. Dena ondo doala probatu ostean, zerbitzaria eten eta berriro martxan jarri
8. Sortu **tests** karpeta eta bertan fitxategi hau:

```
/* server.test.js */

const request = require('supertest');
const PORT = process.env.PORT || 4444;
const url = `http://localhost:${PORT}`

describe('Testing index', () => {
  it('GET /', async () => {
    const res = await request(url).get('/');
    expect(res.statusCode).toBe(200);
    expect(res.type).toBe('application/json');
    expect(res.type).toMatch(/json/);
  });
  // hurrengo it()
});
```

Azalpena:

- (a) lehenengo 3 lerroetan supertest paketea inportatu ondoren, gure aplikazioarekin lotzen dugu.
- (b) jarraian describe() funtzioak test multzo (test suite) bat deskribatzen du. describe baten barruan hainbat it() (test) funtzio izan ditzakegu. Baina momentzu ez dugu ezer probatu. Berez, jarraian datozen funtzioekin egingo ditugu testak.
- (c) get('/') funtzioak GET request bat egingo du / bidera. Erantzuna res (response) objektuan dator.
- (d) behin res (erantzuna) dugula, barruan dagoen objektuen atributuak espero ditugun balioen kontra konparatuko ditugu expect() funtzioekin. Adibidez, GET / egin ondoren HTTP status code = 200 izatea espero dugu (gogoratu HTTP/200 = OK)
- (e) toBe(), toMatch, etab. expect() objektuaren metodoak dira. Gainontzeko metodo guztiak ezagutzeko, botaiozu begirada bat URL honi: <https://jestjs.io/docs/expect>

9. Exekutatu testak (Oharra: jest komandoak *.test.js bilatu eta exekutatuko ditu, automatikoki)

```
$ npm test
```

Dena ondo badoa, hau ikusiko duzu:



```
juanan@Juanans-MacBook-Air > /opt/express-testing % npm test
> express-test@0.0.0 test
> NODE_ENV=test jest --testTimeout=10000

GET / 200 1.395 ms - 47
PASS tests/server.test.js
  Testing index
    ✓ GET / (10 ms)

Test Suites: 1 passed, 1 total
Tests: 1 passed, 1 total
Snapshots: 0 total
Time: 0.183 s, estimated 1 s
Ran all test suites.
```

10. Aurreko kode zatian, `//hurrengo it()` komentarioaren ondoren, hau sartu:

```
it("POST /send", async () => {
  const res = await request(url).post("/send")
    .send({
      email: 'janire@example.com'
    });

  expect(res.type).toMatch(/json/)
  expect(res.statusCode).toBe(201)
  expect(res.body.data.length).toBe(2)
  expect(res.body.data[0].email).toMatch("test@example.com")
  expect(res.body.data[1].email).toMatch("janire@example.com")
})
```

Eta exekutatu berriro test multzoa, horrela:

```
$ npm test
```

Dena ondo badoa hurrengo jaso beharko zenuke:

```

x /opt/express-testing ▶ main ± npm test

> express-test@0.0.0 test
> NODE_ENV=test jest --testTimeout=10000

PASS tests/server.test.js
  Testing index
    ✓ GET / (14 ms)
    ✓ POST /send

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        0.192 s, estimated 1 s
Ran all test suites.
x /opt/express-testing ▶ main ±

```

- Zer gertatuko litzateke janire@example.com ordeztu, maitane@example.com esperoko bagenut?

```
expect(res.body.data[1].email).toMatch("maitane@example.com")
```

Zein da jest exekutatzearren emaitza kasu honetan? (pantaila kaptura bat atera)

11. Orain beste erabiltzaile berri bat sartuko dugu, curl erabiliz:

```
curl -X POST http://localhost:4444/send/ -d
  ↪ "email=jota@gmail.com"
```

Ariketa: Nola aldatuko zenuke 10. ariketan egin dugun post() test-a 11. ariketan sartu dugun elementua ondo txertatu dela probatzeko?

12. Jarraian DELETE ruta probatuko dugu

Demagun hau dela uneko egoera:

```
{ "data":
  [ { "id": 52, "email": "test@example.com" },
    { "id": 86, "email": "janire@example.com" },
    { "id": 28, "email": "jota@gmail.com" } ] }
```

Eta id:28-a duen elementua ezabatu nahi dugula. Lehenengo curl komandoaz probatuko dugu:

```
$ curl -X DELETE http://localhost:4444/destroy/28
{ "data":
  [ { "id": 52, "email": "test@example.com" },
    ↪ { "id": 86, "email": "janire@example.com" } ] }
```

Ondo doala dirudi.

Orain test bat prestatuko dugu. Horretarako, azken ID-a zein den kalkulatu ondoren (get() metodoaz) delete() bat exekutatuko dugu ID horren gainean)

```
it("DELETE /destroy/:id", async () => {
  const res = await request(url).get('/');
  const luzera = res.body.data.length
  // orduan azken elementua luzera -1 posizioan
  ↪ egongo da
  const id = res.body.data[luzera - 1].id
  ...
})
```

Jarraian id hori duen objektuaren ezabaketa ondo dabilela testeatuko dugu:

```
it("DELETE /destroy/:id", async () => {
  const res = await request(url).get('/');
  const luzera = res.body.data.length
  // orduan azken elementua luzera -1 posizioan
  ↪ egongo da
  const id = res.body.data[luzera - 1].id

  const response = await
    ↪ request(url).delete(`/destroy/${id}`);
  expect(response.type).toMatch(/json/);
  expect(res.statusCode).toBe(200);
  expect(response.body.data.length).toBe(luzera - 1)
})
```

Probatzeko:

```
$ npm test
```

```

/opt/express-testing  main  npm test

> express-test@0.0.0 test
> NODE_ENV=test jest --testTimeout=10000

PASS tests/server.test.js
  Testing index
    ✓ GET / (12 ms)
    ✓ POST /send (1 ms)
    ✓ DELETE /destroy/:id (1 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        0.185 s, estimated 1 s
Ran all test suites.
```

13. Ariketa: PUT ruta probatu

```
router.put("/update/:id", ....)
```

Horretarako:

- 13.1) Zein da CURL aplikazioaz probatzeko komandoa? (demagun lehenengo elementuaren email-a aldatu nahi dugula, test@example.com izan ordez, erabiltzaile@example.com izan dadila)
Erabili CURL aplikazioa lehenengo elementuaren emaila berriro test@example.com izan dadin.
- 13.2) Programatu test bat Jest eta SuperTest erabiliz (oinarritu zaitez 12. ariketan, oso-oso antzekoak baitira)

19. Datuen iraunkortasuna eta MongoDB

Orain arte, inprimakiekin maneiatu ditugun datuak ez ditugu inon gorde. Express-en egindako aplikazioan ere ez gara datuen iraunkortasunaz kezkatu. Alegia, zer-bitzaria itzaliz gero, memorian genituen datuak (adibidez, bezeroen izen-abizenak, helbide elektronikoak, etab.) galdu egingo ditugu. Hori ekiditeko, datuen iraunkortasuna bermatu behar dugu. Fitxategietan edo datu-baseetan gorde ditzakegu. Bigarren aukera interesgarriena da gero datu horiek modu errazean kontsultatu, iragazi edo editatu nahi baditugu. Baina datu-baseen artean sistema bat hautatzeko aukera asko ditugu. Horietako bat, MongoDB da.

JavaScript-en programatu nahi baditugu datuen txertaketak, ezabaketak edo edizioak, oso aukera interesgarria bilakatu da MongoDB. Mongo NoSQL motako dokumentuetara oinarritutako datu-base bat da. Mota honetako datu-baseetan ez dugu datuen eskema bat aldeztu aurretik derrigorrez zehaztu behar, eta datuak zuzenean txertatu ahalko ditugu JSON formatuan badaude.

19.1 MongoDB instalatu eta jaurti

MongoDB (*mongo* izenarekin askotan ezagutzen dena) Windows, Linux eta macOS sistema eragileetan instalatu daiteke. Bi bertsiotan banatzen da: komertziala eta doakoa (Community). **Community server**¹ doako bertsioa erabiliko dugu liburu honetan.

Linux eta macOS-en mongo-ren konfigurazio-fitxategia bide honetan aurkituko dugu: `/usr/local/etc/mongod.conf`. Bertan zehaztuko dugu *log* fitxategia non gorde nahi dugun eta zerbitzaria zer IP helbidetan jarri nahi dugun entzuten. Bestetik ezeko balioak hauek dira:

```
1 systemLog:
2   destination: file
```

¹<https://www.mongodb.com/download-center/community>

```

3 path: /usr/local/var/log/mongodb/mongo.log
4 logAppend: true
5 storage:
6   dbPath: /usr/local/var/mongodb
7 net:
8   bindIp: 127.0.0.1

```

Segurtasunaren aldetik interesgarria da *localhost*-en (127.0.0.1 helbidean) jar-tea entzuten mongodb, besterik ezeko konfigurazioan ez baitu irakurtzeko eta idazteko inolako segurtasun-neurririk indarrean, beraz, ez da batere gomendaga-rria *bindIp* parametroan IP publiko bat jartzea.

Mongo datu-basea jaurtitzeko `mongod` komandoa erabiliko dugu:

```
$ mongod --config /usr/local/etc/mongodb.conf
```

Dena ondo badoa, ez dugu inolako errorerik ikusiko. Konektatzeko, `mongo` ko-mandoa landuko dugu (ikus 19.1. irudia).

```

Juanan-Z:tmp juanan$ mongo
MongoDB shell version v4.2.3
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("d8c6625a-f944-4bf9-a662-b640fd2fb593") }
MongoDB server version: 4.2.3
Server has startup warnings:
2020-05-06T18:10:54.765+0200 I CONTROL [initandlisten]
2020-05-06T18:10:54.765+0200 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
2020-05-06T18:10:54.765+0200 I CONTROL [initandlisten] **           Read and write access to data and configuration is unrestricted.
2020-05-06T18:10:54.765+0200 I CONTROL [initandlisten]
2020-05-06T18:10:54.765+0200 I CONTROL [initandlisten]
2020-05-06T18:10:54.765+0200 I CONTROL [initandlisten] ** WARNING: soft rlimits too low. Number of files is 256, should be at least 1000
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
>

```

19.1. irudia: MongoDB NoSQL motako zerbitzaria jaurti ondoren lortu behar du-gun mezua.

MongoDB-ra konektatu ondoren, bertan ditugun datu-baseen izenak lortzeko, `show dbs` komandoa erabiliko dugu.

19.2 Datu-base berri bat ireki edo sortu

Datu-base bat irekitzeko `use` komandoa erabiliko dugu. Adibidez,

```
use bezeroakdb
```

Datu-basea ez bada existitzen, sortu egingo da, hasieran hutsik. Barruan dokumentuak txertatzen hasteko, bilduma bat sortu behar dugu, `db.createCollection` komandoaz.

```
db.createCollection('bezeroak')
```

db objektua



MongoDB-n db objektuak uneko datu-basea irudikatzen du. JavaScript objektu gisa trata dezakegu, beraz, atributuak eta metodoak izango ditu. Adibidez, `db.createCollection()`, `db.auth()`, `db.getName()`, `db.getLastError()` metodoak, edo `db.bezeroak` (uneko datu-basearen bezeroak bildumara apuntatzen dituen), `db.constructor` edo `db.prototype` atributuak (azken bi horiek, db JavaScript-en Object klasetik eratorritako objektu bat delako).

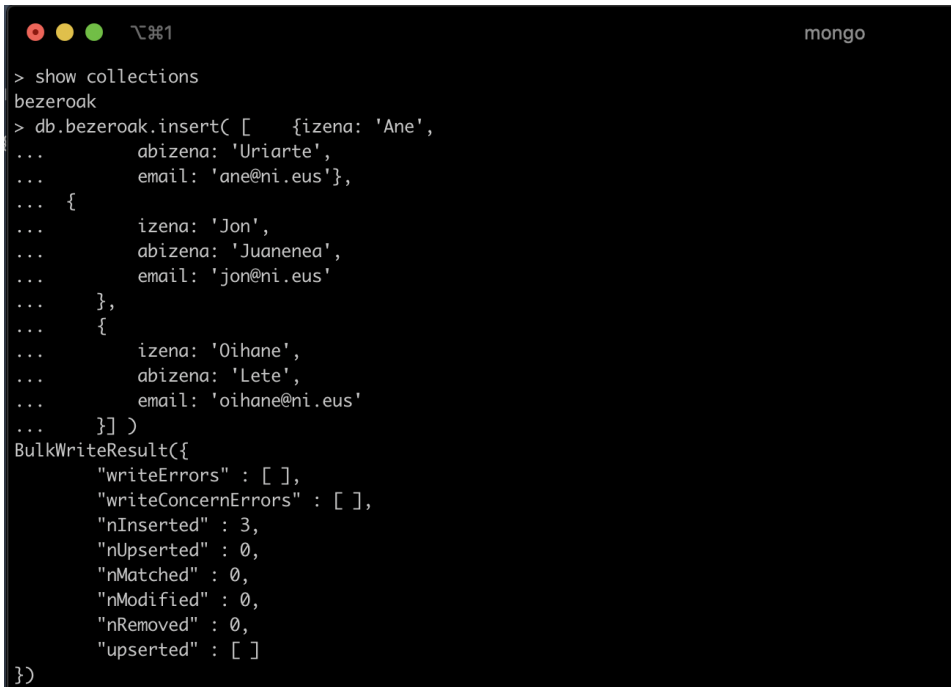
Uneko datu-basearen bildumen izenak ikusteko, `show collections` komandoa erabiliko dugu.

19.3 Datuak txertatzen

Uneko datu-basean datuak txertatzeko, `db.bilduma.insert` erabiliko dugu, eta parametro gisa JSON formatuan txertatu nahi ditugun dokumentuak pasatuko dizkiogu. Adibidez, aurreko ataletan landu ditugun bezeroen datuak txertatzeko:

```
db.bezeroak.insert( [      {izena: 'Ane',
                             abizena: 'Uriarte',
                             email: 'ane@ni.eus'}},
  {
    izena: 'Jon',
    abizena: 'Juanenea',
    email: 'jon@ni.eus'
  },
  {
    izena: 'Oihane',
    abizena: 'Lete',
    email: 'oihane@ni.eus'
  }
] )
```

Dena ondo badoa, *mongo*-k 3 dokumentu berri txertatu ditugula esango digu (ikus 19.2. irudia).



```

> show collections
bezeroak
> db.bezeroak.insert( [ {izena: 'Ane',
...   abizena: 'Uriarte',
...   email: 'ane@ni.eus'},
... {
...   izena: 'Jon',
...   abizena: 'Juanenea',
...   email: 'jon@ni.eus'
... },
... {
...   izena: 'Oihane',
...   abizena: 'Lete',
...   email: 'oihane@ni.eus'
... }
... ] )
BulkWriteResult({
  "writeErrors" : [ ],
  "writeConcernErrors" : [ ],
  "nInserted" : 3,
  "nUpserted" : 0,
  "nMatched" : 0,
  "nModified" : 0,
  "nRemoved" : 0,
  "upserted" : [ ]
})

```

19.2. irudia: Terminalarekin lan egitea ez baduzu gustukoa, Mongo-rekin lan egiteko badaude ingurune grafiko ederrak. Horieta-ko batzuk, Studio3T (<https://studio3t.com/>) eta MongoDB Compass (<https://www.mongodb.com/products/compass>) dira.

19.4 Dokumentuak bilatzen

Bilduma baten barruan dauden objektuak zerrendatu nahi baditugu, `db.find()` komandoa erabiliko dugu:

```

> db.bezeroak.find()
{ "_id" : ObjectId("...08fb9"), "izena" : "Ane", "abizena"
  ⇨ : "Uriarte", "email" : "ane@ni.eus" }
{ "_id" : ObjectId("...08fba"), "izena" : "Jon", "abizena"
  ⇨ : "Juanenea", "email" : "jon@ni.eus" }
{ "_id" : ObjectId("...08fbb"), "izena" : "Oihane",
  ⇨ "abizena" : "Lete", "email" : "oihane@ni.eus" }

```

Dokumentu bakoitzari, berez dituen atributuez gain, mongo-k identifikatzaile bat esleitu dio, *ObjectId* bat.

Badago dokumentu bat bere id edo beste atributu batzuen balioen arabera bi-

latzeko aukera. Adibidez, id-aren edo izenaren arabera bilatzeko:

```
db.bezeroak.find( { _id:
  ↳ ObjectId("5ed4c9cb64dd7facd8008fb9") } )
{ "_id" : ObjectId("5ed4c9cb64dd7facd8008fb9"), "izena" :
  ↳ "Ane", "abizena" : "Uriarte", "email" : "ane@ni.eus"
  ↳ }

db.bezeroak.find( { izena: "Ane" } )
{ "_id" : ObjectId("5ed4c9cb64dd7facd8008fb9"), "izena" :
  ↳ "Ane", "abizena" : "Uriarte", "email" : "ane@ni.eus"
  ↳ }

db.bezeroak.find(
  { izena: { $regex: /^A/ } } )

{ "_id" : ObjectId("5ed4c9cb64dd7facd8008fb9"), "izena" :
  ↳ "Ane", "abizena" : "Uriarte", "email" : "ane@ni.eus"
  ↳ }
{ "_id" : ObjectId("5ed4e879ff9c131dfbe6d23b"), "izena" :
  ↳ "Aitor", "abizena" : "Martikorena", "email" :
  ↳ "aitor@ni.eus" }
{ "_id" : ObjectId("5ed4e9328a54491e39140b49"), "izena" :
  ↳ "Aitor", "abizena" : "Martikorena", "email" :
  ↳ "aitor@ni.eus" }
)
```

Azkenengo adibidean, izena A hizkiak hasten diren bezero guztiak bilatzen ditugu (letra larriak eta xeheak ezberdintzen dira).

Adibide horretan ohartzen bagara, Aitor Martikorena birritan agertzen da. Bigarrena ezabatzeko, `db.deleteOne` metodoa erabil dezakegu:

```
db.bezeroak.deleteOne( { _id:
  ↳ ObjectId("5ed4e9328a54491e39140b49") } )
```

Konexioa ixteko, `Ctrl+d` sakatu edo `quit()` idatziko dugu.

19.5 MongoDB eta NodeJS lotzen

MongoDB eta NodeJS lotzeko, *mongojs* paketea instalatuko dugu.

```
npm install mongojs --save
```

Orain, [mongojs](https://mongojs.com/)² dokumentazioari jarraituz, nodetik konexio bat irekiko dugu:

²<https://mongojs.com/>

```
const mongojs = require('mongojs')
const db = mongojs(connectionString, [collections])
```

Gure kasurako doitzen badugu, honela geratuko da:

```
1 const mongojs = require('mongojs')
2 const db = mongojs('bezeroakdb', ['bezeroak'])
```

Lerro horiek `index.js` fitxategian txertatu behar ditugu.

Orain, eskuz idatzi ditugun bezeroen datuak `index.js` fitxategitik ezabatu eta `mongo` erabiliz jasoko ditugu. Horretarako, `app.get` bidea honela utziko dugu:

```
1 app.get("/", function(req, res) {
2   db.bezeroak.find( function (err, docs) {
3     if (err) {
4       console.log(err)
5     } else {
6       console.log(docs);
7       res.render('index', {
8         'izenburua': 'EJS probatzen',
9         'bezeroak': docs
10      })
11    }
12  })
13 });
```

19.6 Datuen txertaketak mongo-n

`app.post` bidea moldatu egin behar dugu inprimakitik datozen datuak mongo-n gordezkeko, `db.bilduma.insert` erabiliz, lehen ikusi dugun bezala (orain parametro gisa *callback* funtzio bat pasatuko diogu)

```
1 app.post('/bezeroa', function(req, res) {
2   const bezeroBerria = {
3     izena : req.body.izena,
4     abizena: req.body.abizena,
5     email: req.body.email
6   };
7
8   console.log(bezeroBerria);
9
10  db.bezeroak.insert( bezeroBerria, function(err) {
11    if (err) {
12      console.log(err);
13    } else {
```

```

14         res.redirect('/');
15     }
16 })
17 })
    
```

Garrantzitsua da 15. lerroan dugun `res.redirect('/')`. Horrek, datu-basean txer-taketa egin ondoren, webgunearen erro-katalogora birbideratuko gaitu, *app.get* bi-detik sartuko gara eta bezero guztien datuak bistaratuko ditugu.

19.7 Ariketa

Bezeroen datuak bistaratzean, bezero bakoitzaren izenaren ondoan, bistaratu “Eza-batu” eta “Editatu” estekak (ikus 19.3. irudia).

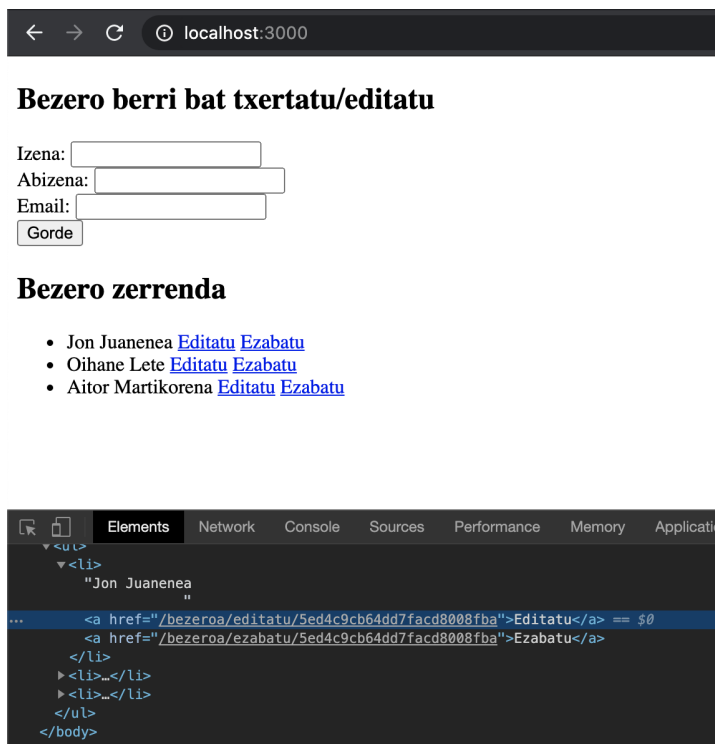
1. “Ezabatu” estekan sakatzean, erabiltzaileari alerta bat pantailaratu behar zaio, ezabaketa egiteko ziur dagoen egiazta dezan. Baietz erantzunez gero, aukeratutako bezeroa datu-basetik ezabatu eta zerrenda eguneratuta bistaratu behar da.
2. “Editatu” botoian sakatzean, bezeroen datuak (izena, abizena, helbide elektronikoa) inprimakian agertuko dira, erabiltzaileak edita ditzan. “Gorde” botoian sakatzean, erabiltzaile horren datuak datu-basean eguneratuko dira. Honelako kodea erabil dezakegu:

```

1 db.bezeroak.update(
2   "query": { "_id" :
3     ↳ ObjectId("5ed4c9cb64dd7facd8008fbb") },
4   "update": {
5     $set : {
6       "izena" : "Oihane",
7       "abizena" : "Letamendia",
8       email: "Oihane@ni.eus" } }, function()
9     ↳ {
10      console.log("Eguneraketa ondo
11        ↳ burutu da";
12      // berbideraketa egin
13    });
    
```

Soluzioa, [GitHubeko biltegian](https://github.com/juananpe/html5liburua/blob/master/index.js) aurki dezakezu³.

³<https://github.com/juananpe/html5liburua/blob/master/index.js>

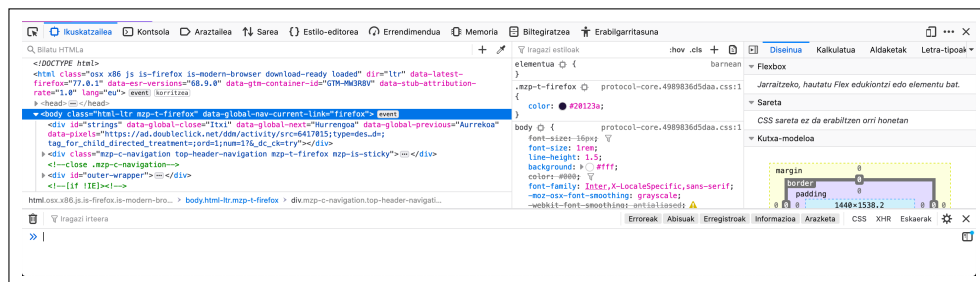


19.3. irudia: Bezeroak editatu eta ezabatzeko estekak izango ditugu. Ohart zaitez esteka bakoitzean bezeroaren identifikatzailea sartu dugula.

20. Araztailea

20.1 Firefox-en Web Kotsola

Webgune bat garatzean ezinbestekoa egingo zaigu programatutako kodea araztea eta zuzentzea. Horretarako, nabigatzaileak berak hainbat tresna laguntzaile eskaintzen dizkigu. Firefox-ek Web Kotsola deritzona. Chrome-ren kasuan DevTools (*Developer Tools*).



20.1. irudia: Firefox-en Web Kotsolak informazio andana eskainiko digu ikusten ari garen web orriaren inguruan.

Kotsola irekitzeko, Firefox-en, Hobespenak / Web Garapena / Web Kotsola hautatu. Bertan, hainbat fitxa aurkituko dugu. Ikuskatzailea izeneko erlaitza izango da goi-eskuineko aldean ikusiko dugun lehenengoa. Bertan sakatzean, 20.1. irudiko beheko 3 panelak ikusiko ditugu: uneko orriaren HTML kodea (bertan edozein nodo ezabatu edo aldatu dezakegu eta nabigatzailean berehala ikusi emaitza), unean hautatuta dugun HTML DOM zuhaitzen elementuaren CSS estiloak eta, eskuinaldean, elementu horri aplikatzen zaizkion ertzak eta marjinak, besteak beste. ESC tekla sakatuz gero, kotsola irekiko da (beheko panela). Kotsolan gure JavaScript kodeen zirriborroak landu ditzakegu, eta uneko orrian kargatu diren

scripten funtzioak eskuz exekutatatu.

Behealdeko kontsolaren panela oso txikia iruditzen bazaigu, **Kontsola** izeneko fitxan sakatu eta kontsola erabat irekiko da.

Dena den, gure programatzaile-lanak batez ere **Araztailea** (*debugger*) izeneko fitxan izango dira. Web aplikazio profesionalak lantzeko, ezinbestekoa da araztailea ondo menderatzea. Hurrengo atalean araztailea nola erabiltzen den ikasiko dugu, adibide simple batekin.

20.2 Araztailea erabiltzen

```

1 <html> <head> <title>Proba</title> <script
  ↪ src="proba.js"></script> </head>
2 <body>
3   Araztailearekin lanean.
4 </body>
5 </html>

```

Demagun 3 puntuz osatutako arraya sortu dugula:

```

1 function Point(x,y) {
2   this.x = x;
3   this.y = y;
4 }
5
6 let puntuak = [new Point(5,0), new Point(11,1), new
  ↪ Point(2,2)];

```

Jarraian, array horren puntu batzuk ezabatzeko agindua jasotzen dugu. Zehazki, x koordinatua 10 baino balio handiagoa dutenak. Otu ahal zaigun lehenengo saiakera honako kodea izan daiteke. Baina ez da zuzena, zergatik?

```

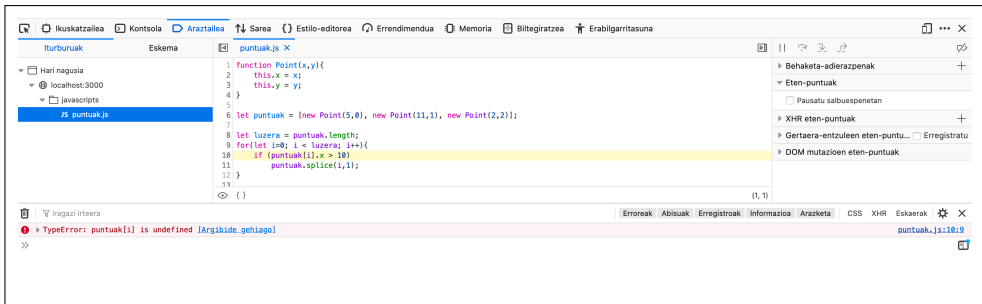
1 let luzera = puntuak.length;
2 for(let i=0; i < luzera; i++){
3   if (puntuak[i].x > 10)
4     puntuak.splice(i,1);
5 }

```

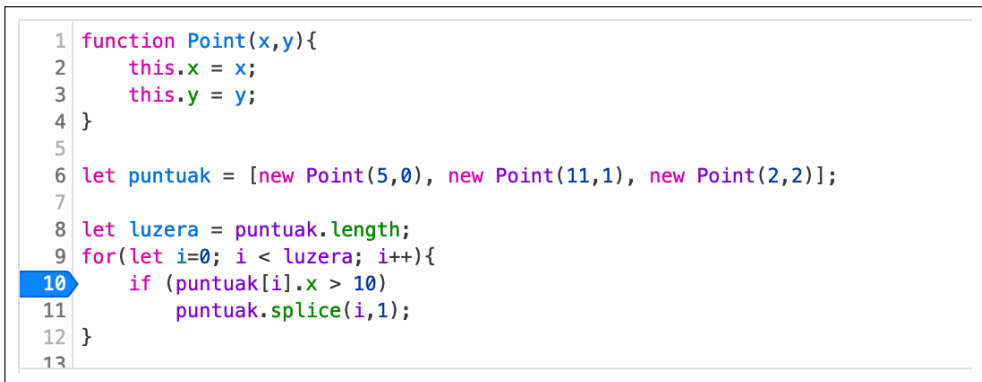
Exekutatuz gero, `if (puntuak[i].x > 10)` lerroan errore bat jasoko dugula ikusiko dugu (ikus [20.2.](#) irudia):

Eten-puntu bat (*breakpoint* bat) jarriko dugu lerro horretan jakin ahal izateko zer gertatu den. Hamargarren lerroan (lerro-zenbakiaren gainean) klik bat eginenez, urdinez markatuko da eten-puntua.

Orain orria berriro kargatzean, kodea martxan jarri eta eten-punturaino helduko da, bertan exekuzioa etenez. Araztailearen panelak ere irekiko dira. Eskuinal-



20.2. irudia: Araztailea nola erabiltzen den ikastea ezinbestekoa da taxuzko web aplikazio bat garatu nahi badugu.



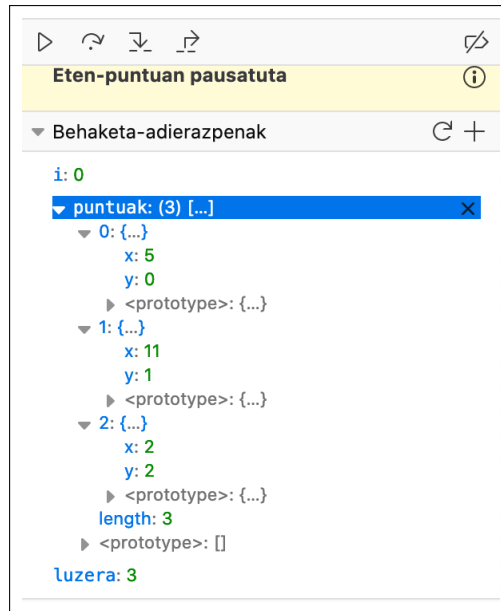
20.3. irudia: *Breakpoint* edo eten-puntuak lerro-zenbakiaren gainean klik eginez zehaztuko ditugu.

deko panelean, behaketa-adierazpenak azpiatalaren barruan, **+** botoian sakatu eta **i** aldagaiaren balioa ikusi nahi dugula adieraziko dugu. Baita **luzera** eta **puntuak** arrayaren edukia ere (ikus 20.4. irudia).

Ohart zaitez **i** aldagaiaren balioa, eten-puntuaren, une honetan 0 dela. **Luzera**, 3. Eta, beraz, arraya zeharkatzen hasi gara, oraindik ez dugu ezer ezabatu bertatik.

Exekuzioa etenda dago. Hurrengo lerroarekin jarraitzeko, pauso bat aurrera egin behar dugu, alegia, 20.5. irudiko 2. gezian sakatu (edo Ctrl+F10). Botoi horri ingelesez “*Step into*” deritzo.

Baldintza betetzen ez denez (**puntuak[0].x = 5** da), ez gara 11. lerroan sartuko eta zuzenean **for** begiztara egingo dugu jauzi. Ctrl+F10 sakatu berriro eta **i** aldagaiaren balioa unitate batean gehituko da (**i=1**). Orain (Ctrl+F10 berriro), **for** begiztaren baldintza aztertuko da (**i < luzera**) eta betetzen denez, hurrengo pau-



20.4. irudia: Programa pausoz pauso egikaritzean, aldagaien balio guztiak monitorizatu ahal ditugu *debugger*-aren behaketa-adierazpenak atalean.

soan berriro *if* baldintzan sartuko gara. Orain baldintza beteko da (`puntuak[1].x = 11`) eta, beraz, `puntuak.splice(1,1)` exekutatuko da. Horrek `puntuak` arrayari elementu bat kenduko dio.

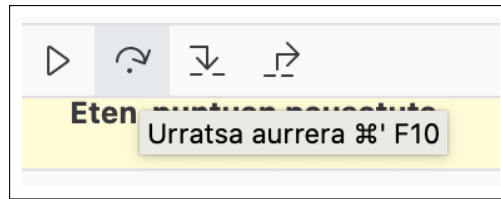
Baina adi, `luzera` aldagaiaren balioa ez da eguneratzen! Ikus 20.6. irudia. Hots, arrayan hasieran baino elementu bat gutxiago badugu ere, `luzera` aldagaia ez da aldatu. Hurrengo exekuzio-lerroan, `i++` egin eta `i=2` bilakatuko da. Arrayak soilik 2 elementu ditu, beraz, `if (puntuak[2].x = 0)` aztertzean, errore bat jasoko dugu (`puntuak[2]` elementua ez baita existitzen!).

Konpontzeko, agian beste soluzio hau otu ahal zaigu:

```
1 let puntuak = [new Point(5,0), new Point(11,1), new
  ↪ Point(2,2)];
2 for(let i=0; i < puntuak.length; i++){
3   if (puntuak[i].x > 10)
4     puntuak.splice(i,1);
5 }
```

Adibidean emandako puntuekin ondo doa, baina ez da soluzio zuzena ere. Zergatik? Puntu berri bat txertatuz gero:

```
let puntuak = [new Point(5,0), new Point(11,1), new
```



20.5. irudia: F10 tekla sakatzean urrats bat aurrera egingo dugu gure kodean. F8 (*Play*) tekla sakatuz exekuzioa aurrera egingo du hurrengo eten-puntua aurkitu arte.



20.6. irudia: Array baten edukia behaketa-adierazpenekin oso modu erosoan ikusi ahalko dugu.

```
↩ Point(15,1), new Point(2,2)];
```

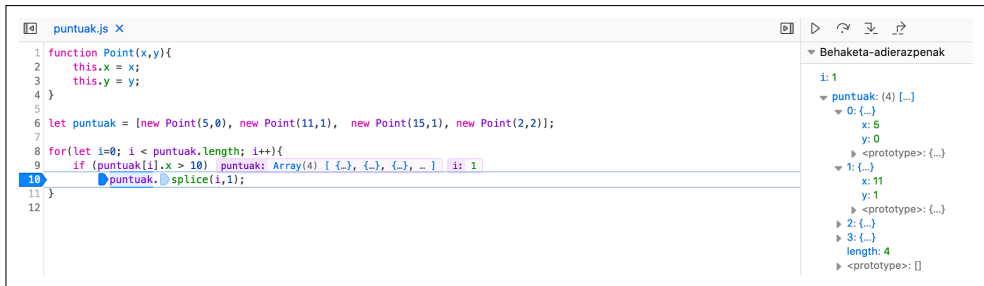
eta *debugger*-arekin pausoz pauso exekutatuz gero, hau ikusiko dugu:
i=1 denean, `puntuak[1]` ezabatu egingo dugu. Une honetan arrayaren edukia hau da:

```
[Point(5,0), Point(15,1)]
```

Hurrengo iterazioan `i++` egitean, `i=2` bilakatuko da eta *for* begiztatik irten egin-gara, azkenengo puntua (15,1) tratatu gabe!

Egoera zuzentzea ez da zaila: arraya lehenengo elementutik aztertzen hasi ordez, azkenengo elementutik lehenengo elementura zeharkatuko dugu.

```
for (let i=puntuak.length-1; i>=0; i--){
  if (puntuak[i].x > 10)
    puntuak.splice(i,1);
}
```

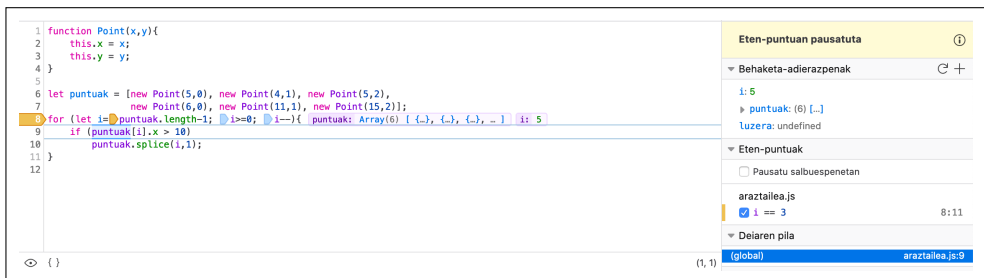


20.7. irudia: Exekuzioa urratsez urrats zeharkatzean, araztaileak lerro bakoitzaren ondoan bertan aldatu diren aldagaien balioak adieraziko ditu, kodearen ondoan —ikus 9. lerroaren eskuinaldea—.

Gogoan izan ariketa ez zela arazoa konpontzea, baizik eta araztailea erabiltzen ikastea arazoaren arrazoia aurkitzeko.

20.2.1 Baldintzazko eten-puntuak

Demagun gure arrayan sei puntu sortu ditugula. Laugarren puntuan gaudenean kodearen portaera aztertu nahi dugu. Baina horraino heltzeko, pausoz pauso exekutatzea, prozedura neketsu eta traketsa bilaka daiteke (F10 hainbat aldiz sakatu beharko dugu... eta 4. posizioan egon ezean, 400. posizioan egonez gero, prozedura ezinezkoa litzakete). Zer egin? Kasu hauetan baldintzazko eten-puntuak oso baliagarriak bilakatuko zaizkigu. Alegia, eten-puntuek kodearen exekuzioa gertuko dute, soilik ezarritako baldintza betetzen denean. Horretarako, eten-puntuak ezarri, haien gainean eskuineko botoia sakatu eta Editatu baldintza hautatu. Eten-puntuak orain laranja marraztuko dira eta eskuinaldean, eten-puntuaren panelean, ezarritako baldintza agertuko da (kasu honetan `i==3`, ikus 20.8. irudia).



20.8. irudia: Baldintzazko eten-puntuak ezarri ondoren, ► botoian sakatu eta kodearen exekuzioa baldintza bete arte exekutatu da, puntu horretan etenez.

Bukatzeko ► ikurrean (edo F8 teklan) sakatu. Exekuzioak aurrera egingo du `i==3` baldintza bete arte.

```
1 let puntuak = [new Point(5,0), new Point(4,1), new
    ↪ Point(5,2), new Point(6,0), new Point(11,1), new
    ↪ Point(15,2)];
2 for (let i=puntuak.length-1; i>=0; i--){
3     if (puntuak[i].x > 10)
4         puntuak.splice(i,1);
5 }
```

21. Garapenerako ingurune bateratuak - IDEak

Eskolak ematen ditudanean beti galdetzen didate ea zein den nik gomendatzen dudana editorea HTML5en garatzeko (edo hobeto esanda, zein den erabiltzen dudana IDEa - ingelesezko *Integrated Development Environment*, alegia, garapenerako ingurune bateratua).

Bi ingurune gomendatzen ditut azken bolada honetan (nork daki etorkizunean zein izango den aukera!), **Microsoft VisualStudio Code**¹ eta **IntelliJ IDEA**². Egia esan bai bata nola bestea oso IDE onak dira, beraz, garatzaile bakoitzak, bere gustuen arabera, bata edo bestea aukera dezake. Kapitulu honetan bien inguruan oinarri-oinarrizko azalpen bat emango da, lanean hasteko lagundu nahian.

21.1 IntelliJ

IntelliJ IDEa bi lizentziapean banatzen da. Alde batetik, Ultimate Edition, software pribatiboa. Eta bestetik, Community Edition delakoa, software irekia. Tamalez, Community Edition ezin da erabili gure JavaScript proiektuak kudeatzeko. Zorionez, ikasleak bagara Ultimate edition delakoa dohainik lor dezakegu JetBrains-eko **webgunean** eskaera eginez³.

21.1.1 Hobespenak

IntelliJ konfiguratu egin behar da JavaScript proiektuekin lan egiteko lehendabiziko aldian. Zehazki, hobespen hauek:

1. **Preferences / Plugins / JavaScript debugger** (instalatuta eta gaituta dagoela egiaztatu)

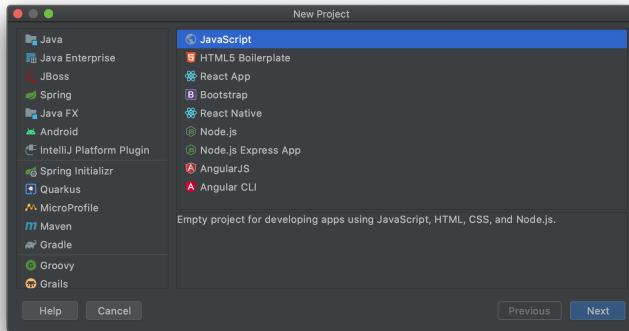
¹<https://code.visualstudio.com/>

²<https://www.jetbrains.com/idea/>

³<https://www.jetbrains.com/community/education/#students>

2. Preferences / Languages and Frameworks / JavaScript / EcmaScript 6 (gaituta dagoela egiaztatu)
3. Preferences / Editor / Code Style / JavaScript / Use file extension in module name (gaituta dagoela egiaztatu)

Jarraian, JavaScript edo HTML proiektu sinple bat sortzeko, Create a new project / JavaScript aukeratu dugu (ikus 21.1. irudia).



21.1. irudia: IntelliJ-n JavaScript proiektu mota ezberdinak garatzeko txantiloiak eskuragarri ditugu.

Ondoren, proiektuari izena esleitu, zer karpetan gorde nahi dugun zehaztu eta berehala izango dugu proiektuan lan egiteko aukera.

Dena ondo konfiguratuta utzi dugula egiaztatzeko, kode hau idatziko dugu, pantailan canvas bat sortu eta bertan lauki beltz bat marrazten duena:

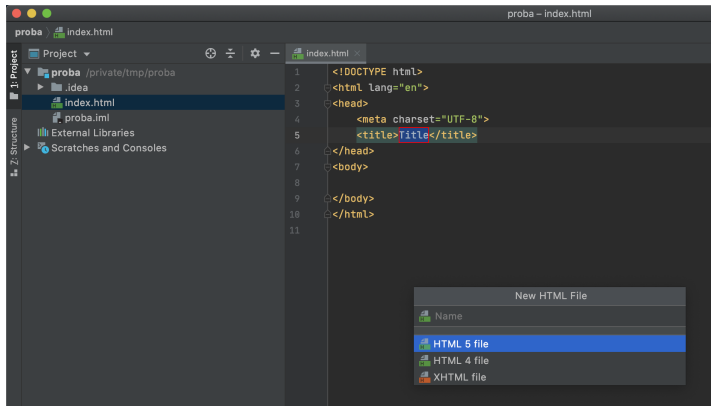
```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head><meta charset="UTF-8"><title>Lauki beltza</title>
4   <script type="module" src="js/index.js"></script>
5 </head>
6 <body><canvas id="arbela" width="640"
   ↪ height="480"></canvas></body></html>

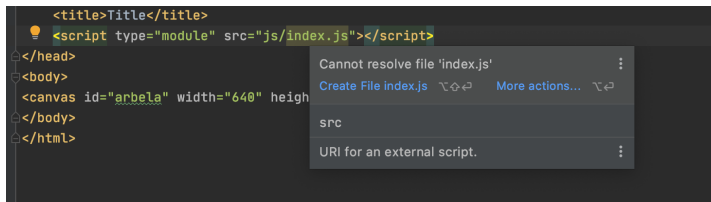
```

Orain, seigarren lerroan dagoen js/index.js fitxategia eskuz sor dezakegu, bai-na badago beste metodo azkarrago bat. Fitxategiaren izenaren gainean klik egin eta sagua bertan mantendu, 21.3. irudian dagoen testinguru-leihoak agertu arte. Bertan Create File index.js estekaren gainean sakatu eta IntelliJ-k automatikoki js/ karpeta sortu eta bertan index.js fitxategia gordeko du, hutsik.

Orain bai, JS fitxategia honako kodearekin beteko dugu:



21.2. irudia: **File / New HTML file / HTML5** aukeratu eta IntelliJ-k automatikoki HTML5en egindako kode-txantilo bat sortuko du.



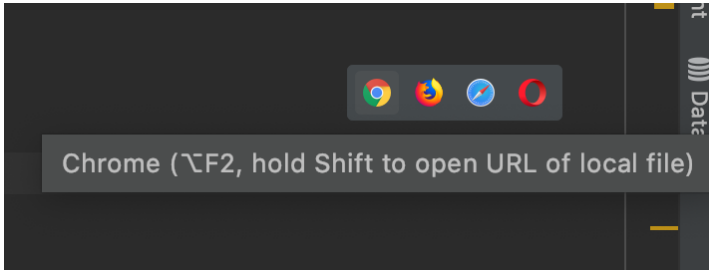
21.3. irudia: IntelliJ-k bonbilla horixka bat erakutsiko digu kodearen ondoan, kodea zuzentzeko gomendio bat duenean.

```
const canvas = document.getElementById('arbela');
const context = canvas.getContext('2d');
context.fillRect(0, 0, 50, 50);
```

Bukatzeko, aplikazioa nabigatzailean probatzeko, IDEak eskaintzen duen nabigatzailearen barran gure gustuko bat hautatuko dugu, adibidez, Chrome. Hautatutako nabigatzailea berehala irekiko da, gure JS aplikazioa bistaratuz, *IntelliJ*-k eskaintzen duen berezko HTTP zerbitzaria erabiliz, portu berezi batean (21.5. irudian 63342 portuan).

21.2 VSCode

Visual Studio Code (VSCode), Linux, macOS eta Windows-erako eskuragarri dagoen iturburu-kodea editatzeko garapen-ingurune bat da. Microsoft enpresak sortu zuen 2015ean, kode irekiko lizentziarekin (MIT). Visual Studio-rekin kodea edita-



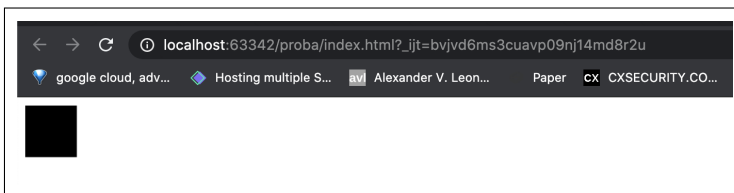
21.4. irudia: Alt-F2 sakatuz nabigatzaileen barra agertuko da. Bertan sisteman instalatuta dituzun nabigatzaileen ikurak ikusiko ditugu. Baten gainean sakatuz, nabigatzaile horretan irekiko da gure aplikazioa.

tzeaz gain, kodea arazteko, sintaxia nabarmentzeko, automatikoki betetzeko eta hobetzeko (*refactoring*) aukerak ematen ditu, besteak beste. Are gehiago, ehunka gehigarri eskaintzen ditu, editoreari funtzionaltasun berriak modu erraz batean emateko (adibidez, kodea aztertzeko, beste lengoia batzuetan programatzeko, editorearen itxura aldatzeko, datu-baseekin integratzeko, etab.)

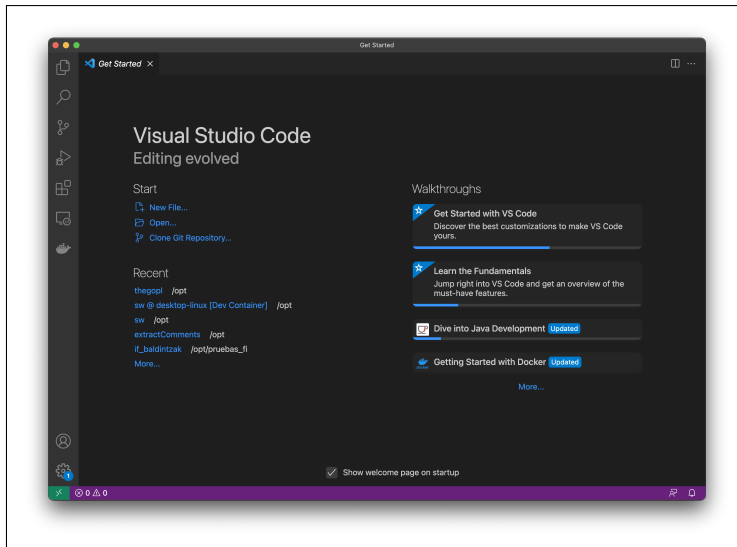
StackOverflow webgune famatuak urtero argitaratzen duen garatzaileen arteko 2021eko inkestaren arabera, Visual Studio Code izan zen garapen-ingurune ospetsuena (82.000 erantzunen artean, %70ek VSCode erabiltzen zutela esan zuen).

21.2.1 Nondik jaitsi

VSCode-k edozein sistema eragiletarako instalatzailea eskaintzen du (<https://code.visualstudio.com/>). Instalatu ondoren aplikazioa ireki eta ikusiko dugun lehenengo pantaila 21.6 irudian ikusten duguna izango da. Bertan, fitxategi berri bat sortu, karpeta ireki edo Git kode-biltegi batetik klonatzeko aukerak ditugu eskuragarri. Aurreko adibidearekin jarraituz, IDE honetan ere oinarritzko HTML



21.5. irudia: *IntelliJ*-k berezko HTTP zerbitzaria dauka eta bertatik eskainiko du gure JS aplikazioa. URLan ikusten den azken zatia ausazko kate luze bat da, cachearekin arazoak ekiditeko.

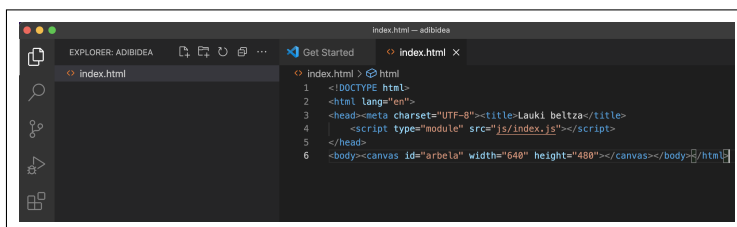


21.6. irudia: VSCode kode irekiko IDEa sistema eragile erabilienentzat dago eskuragarri.

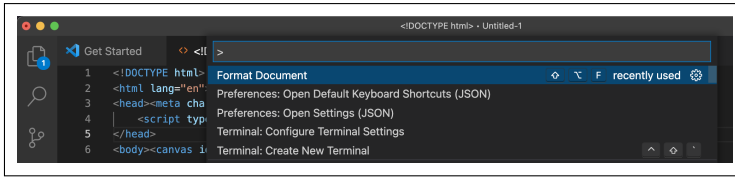
eta JavaScript fitxategiak sortuko ditugu, aplikazioarekin lehen urratsak ematen ikasteko.

21.2.2 Oinarrizko adibidea

Lehenengo eta behin, sortu karpeta bat zure disko gogorrean. Horretarako, VSCode-n "Open Folder..." aukeratu eta jarraian irekitzen den pantailan karpeta berria sortzeko aukera izango duzu. Jarraian, HTML fitxategi berri bat sortuko dugu "New File..." botoian sakatuz. Bertan irekitzen den leihotxoan "Text File Built-in" aukeratu. Jarraian, "Select a language" estekan klik egin eta HTML hautatu.

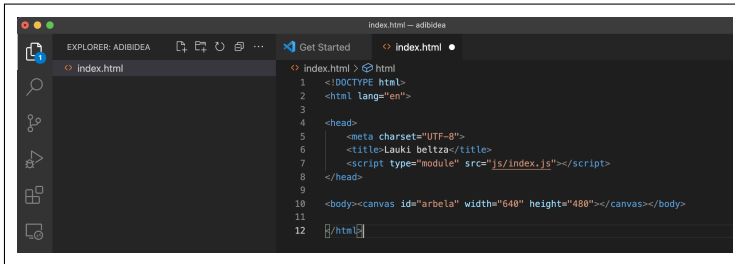


21.7. irudia: VSCode-n, ezkerrean fitxategien arakatzailerak agertuko zaigu. Eskuinaldean editatzen ari garen kodea, sintaxia-nabarmentzearekin.



21.8. irudia: VSCode-k eskaintzen duen komando-paletarekin egin nahi dituzun eragiketak zuzenean idatz ditzakezu, menuetan eragiketeta non dagoen bilatu beharrik gabe.

Aurreko adibidearen kode bera kopia (IntelliJ-en kasuan erabili dugun berbera), eta fitxategia gorde index.html izenarekin, 21.7 irudian dugun pantaila lortuz. Bertan, hiru puntu nabarmenduko ditugu:



21.9. irudia: Kodearen formateatze automatikoa aplikatu ondoren, nabarmena da hobekuntza.

- Aurreko editorean gertatzen zen bezala, `js/index.js` fitxategia sortu gabe dagoenez, bertan sortzeko aukera ematen digu, azpimarratuta agertzen den fitxategiaren izenaren gainean kokatuz eta "Follow Link" aukeratuz. Ondoren, "Create file" botoian sakatu eta fitxategia automatikoki sortuko da.
- Kodearen sintaxia koloreztatuak balizko akatsak berehala antzematen laguntzen digu.
- *Copy&paste* egin dugunez, kodearen formatua ez da hoberena. Hori konpontzeko **Control-Shift-P** sakatuko dugu VSCode-n aginduak sartzeko *Command-Palette* deritzon leihoa irekitzeko (21.8 irudia). Bertan gaudelarik, "Format Document" hautatuko dugu, kodeari formatua emateko. Emaizta 21.9 irudian ikus daiteke.



Visual Studio Code eta laster-teklak

VSCoDe-k erabiltzen dituen laster-teklak oso baliagarriak dira IDE honek eskaintzen dituen milaka funtzioen artean erabiltzenak berehala exekutatu ahal izateko. Arazoa da laster-tekla horiek ere asko direla. Horrez gain, ezberdinak izango dira erabiltzailearen sistema eragilearen arabera. Arazo hauek konpontzeko VSCoDe-k *txuleta* (*cheatsheet*) batzuk eskaintzen ditu, Linux^a, macOS^b eta Windows^c sistemetarako.

^a<https://code.visualstudio.com/shortcuts/keyboard-shortcuts-linux.pdf>

^b<https://code.visualstudio.com/shortcuts/keyboard-shortcuts-macos.pdf>

^c<https://code.visualstudio.com/shortcuts/keyboard-shortcuts-windows.pdf>

22. Kontzeptuen aurkibidea

<article>, 15
<aside>, 15
<audio>, 75, 80
<header>, 15
<nav>, 15
<section>, 15
<video>, 75
<video> controls, 77
=>, 43
selectedIndex, 51

addEventListener, 48
AJAX, 56, 115
API, 25
app.get(), 138
app.post(), 138
app.use, 138
araztailea, 164
arc(), 68, 69
Array, 28, 29
arrow, 43
article, 17
aside, 17
autofocus, 105
autoplay, 77
beginPath(), 67

behaketa-adierazpen, 166
breakpoint, 164

cache.addAll, 127
caches, 126
caches.match, 128
canPlayType(), 78, 79
canvas, 65
cheatsheet, 177
class, 41
clear, 87
clearRect, 67
clearRect(), 67
closePath(), 68
codec, 76
color, 109
concat(), 29
const, 25
constructor, 41
cookie, 85
createServer, 133

db.createCollection, 157
db.deleteOne, 159
db.find(), 158
debugger, 164
DevTools, 163

DOCTYPE, 14
drawImage(), 70

ECMAScript, 41
ejs, 142
elevationService(), 98
email, 105
emit, 118
ES2015, 41, 43
eten-puntu, 164
express, 137
express.urlencoded, 139
extends, 43

fetch, 126
fetch APIa, 57
fill(), 68
fillRect, 66, 67
fillText(), 71
filter(), 43
footer, 17
forEach, 145
form, 105

Geolocation API, 93
GET, 138
get(), 42
getContext(), 66
getCurrentPosition(), 92
getElementById(), 36
getItem, 87
gezi funtzioa, 43
google.maps.LatLng, 96
google.maps.Map, 96

header, 17
herentzia, 43
HTMLMediaElement, 75
http, 133

IDE, 171

IETF, 12
Ikuskatzailea, 163
indexOf(), 29
inline, 37
innerHTML, 36
inprimakiak, 105
install, 127
IntelliJ, 171

Jest, 147
join(), 29
JSON, 56
JSON.parse, 57
JSON.stringify, 59

key, 87
kontainer, 76

laster-teklak, 177
lastIndexOf(), 29
length, 87
let, 25
lineTo(), 68
load(), 78
localStorage, 85, 86
loop, 77

map(), 43
middleware, 138
mongo, 156
MongoDB, 155
mongodb, 137
mongojs, 159
moveTo(), 67
MP4, 76

nav, 17
new, 41
Node.js, 133
node_modules, 134
nodemon, 141

- NoSQL, 155
- noSQL, 137
- npm.init, 134
- number, 108
- ObjectId, 158
- offline, 123
- OGG, 77
- onblur, 47
- oncanplay, 48
- onclick, 47
- ondrag, 47
- onfocus, 47
- onkeypress, 47
- onload, 38, 47, 48, 50
- onmessage, 117
- onmouseover, 47
- onoffline, 48
- onopen, 116
- onsubmit, 110
- package.json, 135
- parseFloat(), 98
- pause(), 78
- placeholder, 105
- Play, 169
- play(), 78
- polling, 115
- pop(), 30
- POST, 139
- poster, 77
- postMessage, 100
- preload, 77
- programazio funtzionala, 44
- Promesak, 57
- Promise, 57
- proxy, 123
- push, 88
- push(), 30
- query string, 140
- Quirks mode, 14
- range, 52, 108
- reduce(), 43
- refactoring, 174
- register, 126
- rejected, 128
- removeEventListener, 49
- removeItem, 87
- req.body, 139
- req.params, 140, 141
- req.query, 141
- request, 133
- require, 133
- required, 110
- resolved, 128
- respondWith, 126
- response, 133
- responseText, 57
- reverse(), 30
- RGB, 109
- route, 138
- scope, 126
- search, 109
- section, 17
- send, 117
- serve, 134
- Service Worker, 123
- set(), 42
- setInterval, 53
- setItem, 87
- setTimeout, 53
- shift(), 30
- show dbs, 156
- slider, 105, 108
- socket.io, 117
- spinbox, 105, 108
- stateless, 85
- step, 108

step-into, 165

stroke(), 68

strokeRect, 67

strokeRect(), 67

SuperTest, 147

terminate, 101

this, 41

url, 105

use, 156

var, 25

Vorbis, 77

VP8, 77

VSCode, 171, 173

W3C, 12

waitUntil, 127

Web Kantsola, 163

Web Storage API, 86

Web Workers API, 99

WebM, 76

WebSocket, 115

WHATWG, 13

Worker, 100

ws, 116

wss, 116

XHR, 56, 115

XMLHttpRequest, 56

XmlHttpRequest, 115