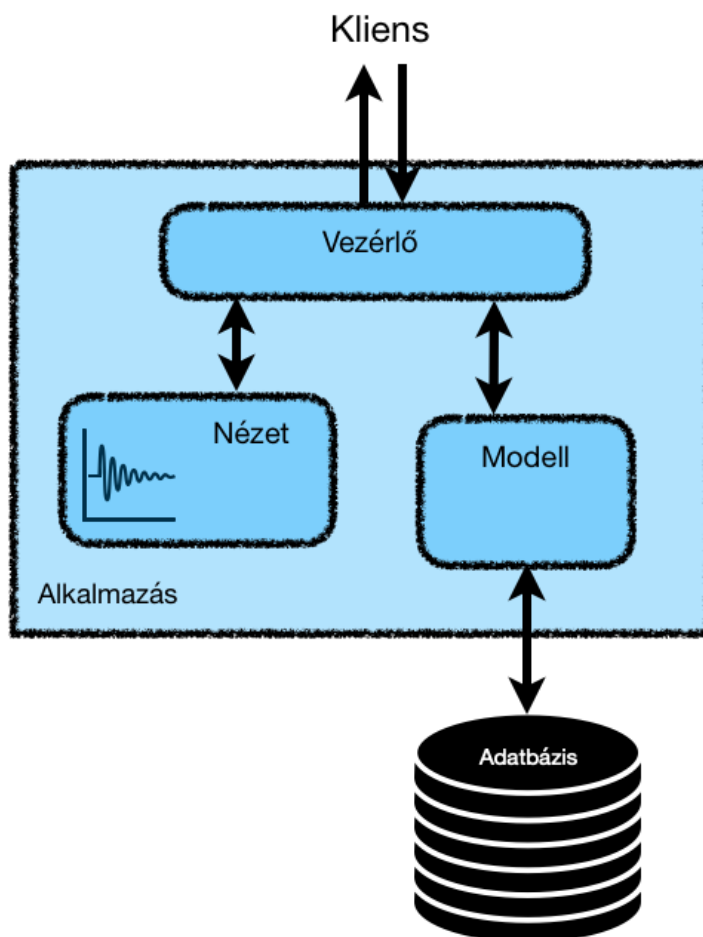


3. FEJEZET

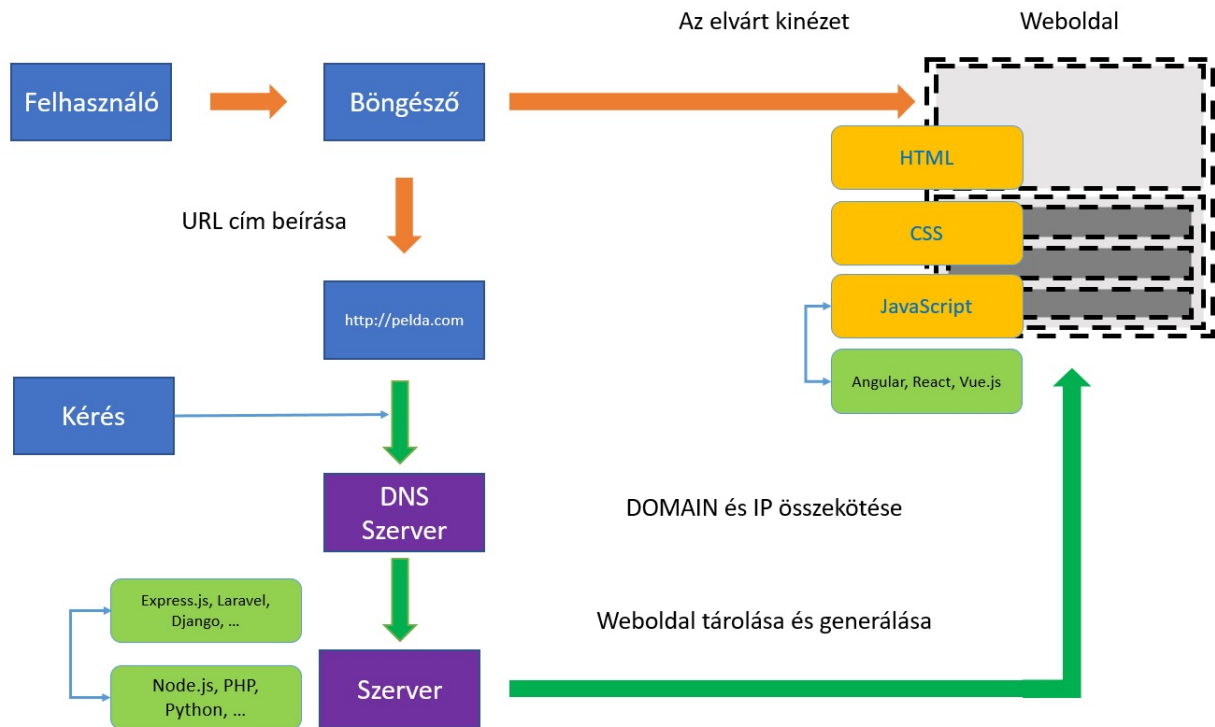
1. Webes alkalmazások: Webes alkalmazások felépítése, működési elve, kliens és szerver oldal, RESTful API-k, axios

Az MVC a Model View Controller szavakból összetevődő kifejezés egy rendkívül népszerű szoftvertervezési mintára. Egy webfejlesztő szemszögéből nézve az eltérő nyelvek esetén nincs két egyforma megvalósítási forma, sok programozó szerint igazi MVC nem is értelmezhető a webfejlesztésben. Az általános koncepciója az, hogy elválasszuk egymástól a funkcionalitást, a vezérlést, a felhasználói felületet, stb. az alkalmazásunkban. Ez megadja a lehetőséget, hogy fejlesztőket csoportosítsunk egyetlen problémára. Az egyetlen hátránya az adatok megfelelő struktúrájának tervezésének és implementálásának időigényességéből adódik. A Model az adatok manipulálásáért felel, alapvetően az alkalmazásunk agyaként funkcionál (az üzleti logikát megvalósító réteg). A Model feladata az, hogy és leírjuk és definiáljuk azokat a szabályokat, melyek alapján elérhetjük azokat. Általában kapcsolatban áll valamilyen típusú adatbázissal (NoSQL vagy SQL) és intézi a lekérdezéseket. Továbbá kapcsolatot tart fent a Controller-el azon keresztül, hogy az is lekérhet adatokat a Model-en keresztül, hogy frissítse a View-t. Az MVC View része az alkalmazásunk kinézetéért / felhasználói felületéért felel.



A View definíció szerint HTML, CSS a Controller-től származó dinamikusan változó adatok összessége. Elhatárolódik egymástól a megjelenítés az adatszerkezetektől. Az adatokat kizárólag a Model osztályainak példányain keresztül (az objektum-orientált programozási környezetben) érhetjük el. Az összes View irányítását egy Modellel is meg lehet oldani. Az éppen használt keretrendszertől függően, a template engine eltérhet, ami a dinamikus átvitelért és megjelenítésért felel. Szintiszta HTML-el dolgozva a felhasználói oldalon a megjelenítésünk

jellege statikus, dinamikusan változó adatokat nem lehet megjeleníteni. Template engine használatával megoldódik ez a problémánk. Végezetül maradt a Controller, ami összefoglalóan a bemeneti eszközök és csatornák felügyeletéért felel, lényegében összeköti őket. Ez azt az esetet is takarhatja, amikor egy felhasználó egy oldalt meglátogat és rákattintva egy linkre kérést indítványoz. Természetesen egy böngészőből nem indítható akármilyen kérés. Mint már tudjuk, a Controller egyfajta köztes állomásként viselkedik a Model és a View között. A Controller megkéri a Model-t arra, hogy adatot kérdezzen le az adatbázisból, ezt követően fogja az adatokat és betöltjük a View-be a template engine számára.



A REST a REpresentational State Transfer rövidítése. Ez az elosztott hipermedia rendszerek építészeti stílusa, és Roy Fielding először ismertette 2000-ben híres disszertációjában.

Mint minden fejlesztési stílusnak, a REST-nek is megvan a saját 6 irányadó korlátozása, amelyeknek teljesülniük kell, ha egy interfészt RESTful-nak kell nevezni. Ezen alapelvek felsorolása az alábbiakban található.

A REST irányadó elvei

- **Ügyfél – szerver** - Ha elkülönítjük a felhasználói felület aggályait az adattárolási aggodalmaktól, javítottuk a felhasználói felület hordozhatóságát több platformon keresztül, és javítottuk a skálázhatóságot a szerver összetevőinek egyszerűsítésével.
- **Állapot nélküli** - Az ügyféltől a szerverig terjedő minden kérésnek tartalmaznia kell a kérelem megértéséhez szükséges összes információt, és nem használhatja ki a kiszolgálón tárolt környezet előnyeit. A munkamenet állapotát ezért teljes mértékben az ügyfél tartja.
- **Gyorsítótárazható** - A gyorsítótár-korlátozások megkövetelik, hogy a kérésre adott válasz implicit vagy explicit módon gyorsítótárazható vagy nem gyorsítótárazható legyen. Ha a válasz gyorsítótárban van, akkor az ügyfél gyorsítótárának joga van arra, hogy újra felhasználja a válasz adatait későbbi, egyenértékű kérésekhez.
- **Egységes interfész** - Ha a szoftverfejlesztés általános elvét alkalmazza az összetevő interfészre, egyszerűsödik a rendszer általános architektúrája és javul az interakciók láthatósága. Az egységes felület eléréséhez több építészeti korlátozásra van szükség az alkatrészek viselkedésének irányításához. A REST-

et négy interfész-korlátozás határozza meg: az erőforrások azonosítása; erőforrások manipulálása reprezentációk révén; ötleíró üzenetek; és a hypermedia, mint az alkalmazás állapotának motorja.

- Réteges rendszer - A rétegelt rendszerstílus lehetővé teszi, hogy az építészeti hierarchikus rétegekből álljon, korlátozva az alkotóelemek viselkedését úgy, hogy az egyes alkotóelemek ne tudják „látni” azon a közvetlen rétegen túl, amellyel kölcsönhatásba lépnek.
- Igényelt kód (opcionális) - A REST lehetővé teszi az ügyfél funkcionalitásának kibővítését, kódok letöltésével és végrehajtásával appletek vagy szkriptek formájában. Ez egyszerűsíti az ügyfeleket azáltal, hogy csökkenti az előzetesen beépítendő szolgáltatások számát.

A REST információ legfontosabb absztrakciója egy erőforrás. Bármely megnevezhető információ lehet erőforrás: dokumentum vagy kép, időbeli szolgáltatás, egyéb erőforrások gyűjteménye, nem virtuális objektum (például egy személy), és így tovább. A REST erőforrás-azonosítóval azonosítja az összetevők közötti interakcióban részt vevő erőforrást.

2. Vue UI-k (Bootstrap-VUE, Buefy, Vuetify, ...), VUEX, Vue Router

Miért is érdemes foglalkozni a Vue UI könyvtárakkal? Nos, a Vue (vagy Vue.js) egy olyan népszerű JavaScript keretrendszer, amivel dinamikus alkalmazásokhoz fejleszthetünk frontendet. Amióta 2014-ben megfogalmazódott a koncepciója, a rá következő két évre jelentős fejlődésen esett át pozitív irányban. Népszerűsége robbanásszerűen növekedett (növekszik még a mai napig is), komoly vetélytársa lett az Angular-nak és a React-nak. Ennek az az oka, hogy az előbb említett két meghatározó keretrendszer legjobb elemeit gyűrték össze. Az egyik ilyen jótékony tulajdonság a forráskód komponensekre-való szétválasztásának (és azok újrafelhasználhatóságának) a lehetősége, mely jelentősen lecsökkentheti a fejlesztési időt.

A BootstrapVue könyvtár segítségével teljesen reszponzív, a mobile-first irányelveknek eleget tevő alkalmazásokat építhetünk a Vue keretrendszer alatt. Több, mint 40 rendelkezésre álló bővítménnyel és több, mint 75 testre szabható UI komponenssel az egyik leggyakrabban használt könyvtára, ami a Bootstrap 4.3. elemeit és rácsrendszerét örökölte. A két rendszer elemeit szabadon keverhetjük, az osztályok és a különböző funkcionalitások ugyanúgy működnek mindenféle fennakadás nélkül. A Bootstrap rácsrendszere egy csomó konténer, sor és oszlop (container, row, column) testre szabásával az egyes komponenseket / tartalmat a megfelelő helyekre tudjuk helyezni. Ez a könyvtár ugyanúgy flexbox-t használ, ahogy a Vuetify is. A legfontosabb elemeket és tulajdonságait néhány sor keretén belül bemutatjuk. A konténerek lényegében azt csinálják, hogy a két tag közé helyezett tartalmat (legyen az szöveg, vagy kép) középre rendezi. Amíg `<b-container></b-container>` esetén a tartalom szélessége fix, addig `<b-container fluid></b-container fluid>` esetén a tartalom a weboldal teljes szélességében terül szét.

A tartalmat rendezhetjük oszlopokba (`<b-col></b-col>` közé beszúrva) és minden egyes sor közvetlen leszármazottja a `<b-row></b-row>` soroknak. A `<b-row></b-row>` sorokat továbbá horizontálisan lehet úgy csoportosítani, hogy egy különálló oszlopot alkossanak. A flexbox-nak köszönhetően a rácsrendszer úgy működik, hogy az oszlopok alapértelmezetten egyforma méretűek. A `<b-row></b-row>` komponenseket mindig a `<b-container></b-container>` tag-ek közé kell beszúrni.

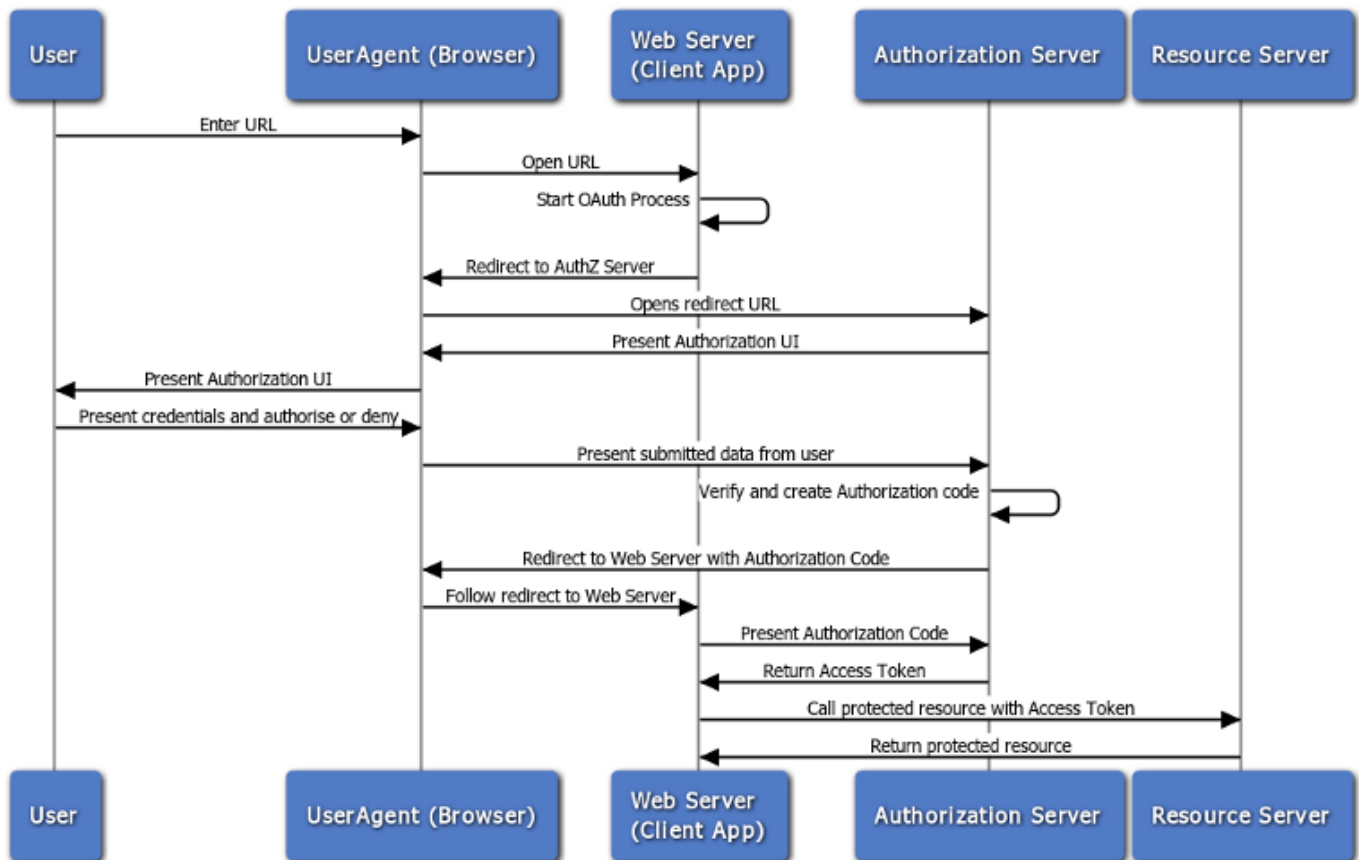
Az oszlopok (relatív) vastagságát százalékban tudjuk megadni, 100% a szülő elem szélessége, amibe bele van ágyazva.

5 rácsszintet tudunk definiálni és mindegyik egy-egy töréspontot definiál: cols, sm, md, lg, xl. Mindegyik ilyen média-töréspont egy-egy pixel tartománynak feleltethető meg.

Mindegyik média-törésponthoz rendelhetünk egy (1-12 közé eső) számot. Ez alapján meghatározhatjuk a "b-col egység nagyságát" attól függően, hogy mekkora a képernyő mérete. A nagyság alatt relatív értéket értünk, amit a képernyő pixelben megadott teljes szélességéhez viszonyítunk. 1024 pixeles képernyő szélesség esetén egy ilyen oszlop egység lefelé kerekítve 85 pixeles lesz. Ha azt szeretnénk, hogy az összes képernyő méret esetén ugyanúgy nézzen ki a rácsrendszer felépítése, akkor szám helyet egy csillagot (*) adunk értékül.

3. Google API-k, autentikáció, Cloud platformok (HEROKU, Firebase, ...)

Az OAuth 2.0 az iparági szabványos engedélyezési protokoll. Az OAuth 2.0 az ügyfelek fejlesztői egyszerűségére összpontosít, miközben speciális engedélyezési folyamatokat biztosít webes alkalmazások, asztali alkalmazások, mobiltelefonok és nappali eszközök számára. Ezt a specifikációt és annak kiterjesztéseit az IETF OAuth munkacsoport fejleszti.



4. Komplex skálázható webes rendszerek, webes kommunikációs interfészek (ws, sse)

SSE: Szerveroldali események

Hagyományosan, egy weboldalnak kérést kell küldenie a szervernek új adatok fogadására; vagyis az oldal adatokat kér a szerverről. A szerver által küldött eseményekkel a szerver bármikor új adatokat küldhet el egy weboldalra az üzeneteknek a weboldalra történő továbbításával. Ezeket a bejövő üzeneteket események + adatokként lehet kezelni a weboldalon.

WS:

A WebSockets egy fejlett technológia, amely lehetővé teszi egy interaktív kommunikációs munkamenet megnyitását a felhasználó böngészője és egy szerver között. Ezzel az API-val üzeneteket küldhet egy kiszolgálóra, és eseményvezérelt válaszokat kaphat anélkül, hogy a kiszolgálóra lekérdezni kellene a választ.

Példa SSE alkalmazására:

Szerver oldal - Node.js

```
var SSE      = require('sse'),
    { Svc,
      app }  = require('singlevue'),
    amoba    = new Svc('amoba'),
    reversi  = new Svc('reversi'),
    main     = new Svc(),
    ct       = [],
    port     = 3005;

app.get( '/amoba/' , (req,res)=> {
  res.send(amoba.vue({ title: `Amóba` } ) ) ;
});

app.get( '/' , (req,res)=> {
  res.send(main.vue({})) ;
});

app.get( '/reversi/' , (req,res)=> {
  res.send(reversi.vue({ title: `Reversi` } ) ) ;
});

app.post( '/' , (req,res) => {
  ct.map( (v,i) => {
    if ( v.id == req.body.id1 ||
        v.id == req.body.id2 )
      v.c.send( `${
        req.body.x
      }-${
        req.body.y
      }-${
        req.body.f
      }-${
        req.body.id1
      }-${
        req.body.id2
      }` )
  });
  res.sendJSON( {x: ct.length } );
});

app.listen(port, server => {
  console.log('server is running on http://localhost:'+port);
  new SSE(server)
    .on('connection', c => {
      let cs = {
        c,
        ts: Number(new Date()),
        id: Math.round(Math.random()*8999)+1000
      }
    })
});
```

```

        ct.push(cs);
        c.send('id-'+cs.id.toString());
    } )
});

setInterval( () =>
    ct = ct.filter( v =>
        (Number( new Date() ) - v.ts < 1000000)
    ), 1000000
);

```

Kliens oldal - Vue SFC:

```

<template>
  <div id="app">
    <h1 v-html="`${
      template.title
    } (ID: ${ id })`">dd</h1>
    <div class="mini">
      Aktív játékosok száma belépéskor: {{ conn }}
    </div>
    <table>
      <tr :key="j"
        v-for="(row,j) in arr">
        <td :key="i"
          v-for="(e,i) in row"
          @click="f(i,j)"
          :class="e" >{{e}}</td>
        </tr>
      </table>
    <hr>
    <input value="Ellenfél" disabled>
    <input placeholder="ID"
      v-model="opponent"
      :disabled="opponent==='-----'" />
    <hr>
    <span v-if="nyert" >
      <div v-html="nyert" ></div>
      <br><br>
      <button
        v-if="nyert!='Kapcsolat megszakadt!'"
        @click="uj()">Új játék</button>
      <a href="." v-else>Új játék</a>
      <br>
    </span>
    <span v-else>
      <table>
        <tr>
          <th colspan="3">
            Következő
          </th>
        </tr>

```

```

        <tr>
          <td :class="next==='X'? 'X': ''">X</td>
          <td> - </td>
          <td :class="next==='0'? '0': ''">0</td>
        </tr>
      </table>
    </span>
  </div>
</template>

<script>
//const base='/u/tnemeth_5/'; // inf-en
const base='/';
let next="X", rak=0,
    es = new EventSource(base+"sse");
function ures() {
  return Array(10)
    .fill(0)
    .map( v => Array(10).fill(' ') );
}
export default {
  data: {
    arr: ures(),
    nyert: false,
    conn: 0,
    id: 0,
    opponent: '',
    next
  },
  mounted() {
    es.onmessage = e => {
      let [x,y,p,id1,id2]=e.data.split('-') ;
      if (x==='uj') {
        this.arr=ures() ;
        this.nyert='' ;
      }
      else if (x==='id') {
        this.id=y;
      }
      else {
        if (--rak<0) rak=0;
        this.$set( this.arr[y],x,p );
        next=p==="X"? "0": "X";
        this.next=next;
        [[1,1],[1,0],[0,1],[-1,1]]
          .forEach( v => {
            let xp=Number(x),
                yp=Number(y),
                maxh=0;
            while ( this.arr[yp] &&
                    this.arr[yp][xp]===p) {
              xp+=v[0];
              yp+=v[1];
              maxh++;
            }
          }
      }
    }
  }
}

```

```

        }
        xp=Number(x);
        yp=Number(y);
        while ( this.arr[yp] &&
                this.arr[yp][xp]===p){
            xp-=v[0];
            yp-=v[1];
            maxh++;
        }
        if ( maxh>5)
            this.nyert = `Nyert: <b>${ p }</b>`;
    });

    };
};
es.onerror = e => {
    this.nyert="Kapcsolat megszakadt!";
    this.id='-';
    this.opponent='-----';
    es.close() ;
};
},
methods: {
    uj() {
        axios
            .post( base, {
                x: 'uj',
                y: 0,
                f: "!",
                id1: this.opponent,
                id2: this.id
            } );
    },
    f(i,j) {
        if ( !this.nyert &&
            this.arr[j][i] === " " &&
            this.opponent != this.id &&
            rak===0 && this.opponent.length===4 ) {
            rak=2;
            axios
                .post( base, {
                    x: i,
                    y: j,
                    f: next,
                    id1: this.opponent,
                    id2: this.id
                } )
                .then(resp => this.conn = resp.data.x );
        }
    }
}
};
</script>

<style>

```



```
h1 {
  height: 20px;
}
#app, h1, input {
  text-align: center;
}
button {
  font-size: 17px;
  cursor: pointer;
}
input {
  font-size: 20px;
  width: 75px;
}
.mini{
  font-size: 14px;
  height: 20px;
}
table {
  margin: 0 auto;
}
td {
  text-shadow: 0 0 2px black;
  user-select: none;
  width: 28px;
  height: 28px;
  cursor: pointer;
  border: solid 1px rgb(89, 109, 108);
  text-align: center;
  border-radius: 3px;
  background-color: #d7daa7;
  color: snow;
  box-shadow: 0 0 3px black;
}
td.0 {
  color: rgb(240, 239, 199);
  background-color: #197596;
}
td.X {
  color: rgb(192, 237, 159);
  background-color: #982907;
}
</style>
```

5. Platformfüggetlen alkalmazásfejlesztés: NW.js, ElectronJS, Cordova, ...

Az Electron egy nyílt forráskódú könyvtár, amelyet a GitHub fejlesztett platformokon átívelő asztali alkalmazások készítéséhez HTML, CSS és JavaScript segítségével. Az Electron ezt úgy valósítja meg, hogy a Chromiumot és a Node.js-t egyetlen futásidejévé egyesíti, és az alkalmazások csomagolhatók Mac, Windows és Linux rendszerekhez.

Az Electron 2013-ban kezdte meg a keretet, amelyre az Atom, a GitHub hackelhető szövegszerkesztője felépül. A kettőt 2014. tavaszán nyitották meg.

Elektron telepítése

Ezen a ponton magának az elektronnak kell települnie. Ennek az ajánlott módja az, ha fejlesztőfüggőségként telepíti azt az alkalmazásába, amely lehetővé teszi, hogy több alkalmazáson dolgozzon, különféle Electron verziókkal. Ehhez futtassa a következő parancsot az alkalmazás könyvtárából:

```
npm install --save-dev electronCopy
```

Az Electron telepítésére más eszközök léteznek. Kérjük, olvassa el a telepítési útmutatót, hogy megismerje a proxykkel, tükrökkel és az egyedi gyorsítótárakkal való használatát.

Az első elektron alkalmazás

Az elektron alkalmazásokat JavaScript-ben fejlesztették ki, a Node.js fejlesztésében alkalmazott alapelvekkel és módszerekkel. Az Electronban található összes API és szolgáltatás elérhető az elektronmodulon keresztül, amelyhez hasonlóan szükség lehet bármely Node.js modulra:

```
const electron = igényelni ('elektron')
```

Az elektron modul a névterek jellemzőit tárja fel. Például az alkalmazás életciklusát az electron.app útján kezeljük, az alkalmazás létrehozható az electron.BrowserWindow osztály használatával. Egy egyszerű main.js fájl vbiztosítja az alkalmazás futtatását és megnyithat egy ablakot:

```
const { app, BrowserWindow } = require('electron')

function createWindow () {
  // Create the browser window.
  let win = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      nodeIntegration: true
    }
  })

  // and load the index.html of the app.
  win.loadFile('index.html')
}

app.on('ready', createWindow)
```

6. JS könyvtárak és alkalmazásuk: VIS.JS, ChartJS, ngraph,...

Könnyű elkezdeni a Chart.js használatát. Csak annyit kell tennie, hogy az oldalon megtalálható szkript egy `<canvas>` csomóponttal együtt kerül a diagram megjelenítésére.

Ebben a példában vonaldiagramot hozunk létre egyetlen adatkészlethez, és ezt az oldalunkon jelenítjük meg.

Példa, a Chart.js alkalmazása Vue környezetben:

```
<template>
  <div>
    <canvas ref="cc"></canvas>
  </div>
</template>
<script>
import Chart from 'chart.js'
var gs = { }
var config = {
  type: 'line',
  data: {
    labels: [1,2,3,4,6,7,8],
    datasets: [{
      label: 'Adatok',
      backgroundColor: 'rgba(255, 99, 132, 0.2)',
      borderColor: 'rgba(255, 99, 132, 1)',
      data: [0, 10, 5, 2, 20, 30, 45]
    }]
  },
  options: {}
}
export default {
  name: 'Diagram',
  mounted() {
    gs.cc = new Chart(
      this.$refs.cc.getContext('2d'),
      config
    )
  }
}
</script>
<style>
div {
  text-align: center;
}
</style>
```

7. Full stack web frameworks: NUXT

Mi a full-stack webes fremwork?

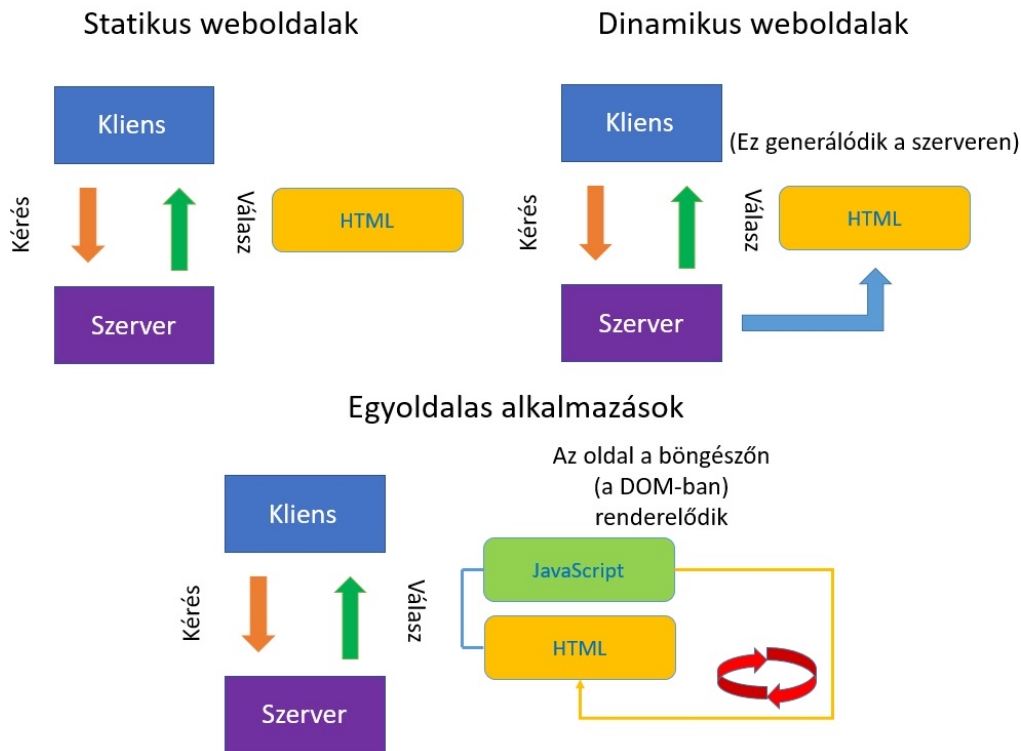
Egy ilyen kérdés megválaszolásához meg kell határozni egy egyszerű webes alkalmazás alapvető alkotóelemeit:

- adatbázis
- backend
- frontend

Minden olyan keretrendszer, amely egy programozási nyelvet használ, és rendelkezik eszközökkel a fenti 3 alkotóelemet kötelezően tartalmazza. Az adatbázisból kiindulva a keretrendszernek képesnek kell lennie arra,

hogy lekérdezze az adatbázist anélkül, hogy a fejlesztő tudná, melyik adatbázist használja vagy annak ismerje sajátosságait.

A háttérprogram felelős a beérkező kérések beszerzéséért és a válaszok felépítéséért az adatbázisból és az üzleti logikából származó adatok felhasználásával. A fejlesztőnek ismét képesnek kell lennie a bejövő kérelmek kezelésére anélkül, hogy foglalkoznia kellene a hálózati és csomagkezelési mechanizmusok legapróbb részleteivel.



A Frontend HTML, Javascript és CSS elemeket tartalmaz. A Javascript és a CSS a HTML elemektől függ, tehát a keretnek eszközöket kell biztosítania a HTML elemek és struktúrák könnyű létrehozásához. Az űrlapok, listák és menük a HTML-struktúrák egy részét képezik, amelyeket a keretrendszer hoz létre, anélkül, hogy a fejlesztő ezeket egyenként létrehozná.

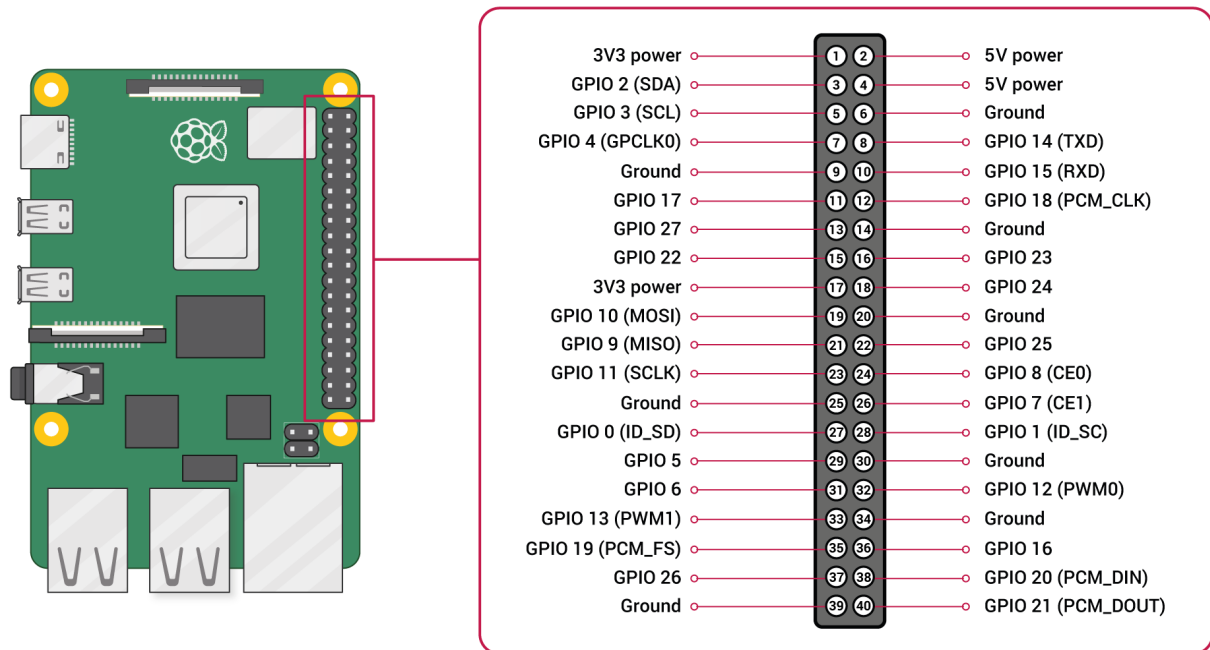
Tehát a könyvtárak úgy vannak konfigurálva, hogy zökkenőmentesen működjenek együtt a fentiek teljesítése érdekében. Ezenkívül az a tény, hogy mindezeket csak egy programozási nyelven történő kód írásával lehet elérni, különbözteti meg őket az egyszerű eszközkészletektől.

A Nuxt egy erőteljes moduláris architektúrán alapul. Több mint 50 modul közül választhatunk, hogy gyorsabb és könnyebb legyen a fejlesztést. A PWA előnye, hogy további modulokkal könnyen bővíthető, pl.: Google Analytics, vagy a webhelytérkép, ezáltal a fejlesztés leegyszerűsíthető.

8. Webes IOT és beágyazott rendszerek programozása, RPI-GPIO

A Raspberry Pi egy aprócska, többcélú számítógép.

Raspberry Pi és Node.js A Raspberry Pi egy sor GPIO (általános célú bemenet / kimenet) érintkezőt tartalmaz, ezek felhasználhatók csodálatos módon való interakcióra a való világgal.



Szükséges hardver:

- Raspberry Pi számítógép
- MicroSD memóriakártya (Ajánlunk egy 10-es osztályt, 16 GB vagy annál nagyobb)
- MicroSD – SD memóriakártya-adapter (általában a MicroSD-kártyához tartozik)
- Mikro USB-tápegység a Raspberry Pi táplálásához (legalább 2,5A ajánlott)
- WiFi / Ethernet kábelcsatlakozás a Raspberry Pi-hez (Nem szükséges a Raspberry Pi 3-hoz, mivel beépítette a WiFi-t)
- Működő számítógép internettel és SD memóriakártya-olvasóval (a Raspberry Pi operációs rendszerének (operációs rendszer) a memóriakártyára történő bejuttatásához használható).
- Windows, Mac vagy Linux számítógépet is használhatunk: HDMI monitor, USB billentyűzet (ezekre csak ideiglenesen szükségünk van a Raspberry Pi első indításához) Az oktatóanyag későbbi fejezeteiben speciális érzékelőket vagy eszközöket használunk, amelyeket a Raspberry Pi-hez csatlakoztatunk.