server.js

Incarcarea modulelor

> incarca modulele express, path,fs , creaza aplicatia app, seteaza engine-ul pentru templating/view
> Deschide fisier server.js

- [ ] incarca modulul **express** cu `require('express')`
- [ ] incarca modulul **path** cu `require('path')`
- [ ] incarca modulul **fs** cu `require('fs')`
- [ ] creaza aplicatia app cu `const app = express()`
- [ ] seteaza view engine `app.set('view engine', 'ejs')`
- [ ] seteaza folderul de template-uri ejs `app.set('views', 'views')`

```
const express = require('express');
const path = require('path');
const fs = require('fs');
const app = express();
// Setări pentru EJS
app.set('view engine', 'ejs');
app.set('views', 'views');
```

Helpers

> creaza `helper`-ele pe care le folosim pe parcurs, 1 constanta `dataPath`, calea catre folderul cu date 2 functia `getListaFisiere()` care returneaza array-ul cu fisirele cu probleme

- [ ] afiseaza in consola valoarea lui `process.cwd()` si a lui `__dirname`

```
 // amandoua fac in fisierul server.js acelasi lucru dar diferenta e ca
//  process.cwd() va returna valoarea caii absolute catre radacina proiectului iar
//  __dirname e valoarea caii pentru un fisier oarecare oriunde s-ar afla in
structura de foldere ale aplicatiei
```

- [ ] afiseaza in consola valoarea lui `path.join(process.cwd(), 'data');` si apoi salveaza valoarea in constanta `dataPath`

```
 // Directorul unde sunt fișierele .txt
const dataPath = path.join(process.cwd(), 'data')
```

- [ ] afiseaza in consola vectorul cu fisierele citite din folderul **dataPath** `let listaFisiere = fs.readdirSync(dataPath, { withFileTypes: true })`
- [ ] pe vectorul rezultat aplica un filtru `let listaFisier = listaFisiere.filter(fis=>fis.isFile())` si afiseaza
- [ ] pe vectorul filtrat parcurge fiecare element si inlocuieste extensia cu sirul vid `let listaFisiere = listaFisiere.map(fis=>fis.name.replace('.txt',''))` si afiseaza
- [ ] creaza vectorul final prin **chaining** (inlantuire) `const listaFisiere = fs.readdirSync(dataPath, { withFileTypes: true }).filter(fis => fis.isFile()).map(fis => fis.name.replace('.txt', ''));` si il afisezi
- [ ] creaza o functie `getListaFisiere` care genereaza vectorul `listaFisiere` obtinut anterior si il returneaza

```
// Funcție helper pentru a obține lista fișierelor
function getListaFisiere() {
    return fs.readdirSync(dataPath, { withFileTypes: true })
    .filter(fis => fis.isFile())
    .map(fis => fis.name.replace('.txt', ''));
}
```

Crearea rutelor

> Avem doua rute: 1 `'/'` - ruta radacina, afiseaza lista de fisiere din `getListaFisiere` 2`'/probleme/:problema'` - afiseaza continutul fisierului corespunzator.

> Crearea unei rute consta din 3 pasi 1 alegerea metodei, in cazul nostru metoda GET, `app.get(A,B)` unde **A** e ruta iar **B** e functia callback apelata la momentul request-ului catre server pe acea ruta 2 Se inlocuieste **A** cu ruta propriu-zisa, `'/'` sau `'/probleme/:problema'` sau `'/products/:id'` 3 Se inlocuieste **B** cu functia callback care este apelata pe acel request, in general sub forma de arrow function dar care contine 2 parametri: primul este parametrul care contine datele de `request` trimise de browser si denumit simplu `req`, al doilea este parametrul `response` denumit `res` si care se ocupa de trimiterea datelor catre browser.

> ✅ Astfel functia callback are forma `(req,res)=>{return res.send('test')}` ✅ Iar ruta are forma `app.get('/',(req,res)=>{return res.send('test')})`

Crearea rutei '/'

- [ ] creaza ruta `'/'` folosind metoda get: `app.get('/',(req,res)=>{res.send('ruta radacina')})`
- [ ] Modificam ruta anterioara pentru a testa functia `getListaFisiere`:

```
app.get('/',(req,res)=>{
    const numeFisiere = getListaFisiere();
  console.log(numeFisiere)
  res.send('ruta radacina')
  })
```

- [ ] Modificam inca o data ruta `'/'` iar vectorul `numeFisiere` va fi converti in test cu JSON.stringify() si apoi trimis catre template-ul home.ejs.

```
app.get('/',(req,res)=>{
    const numeFisiere = getListaFisiere();
     let numeFisiereText = JSON.stringify(numeFisiere)
  res.render('home', { numeFisiereText, continutFisier: null });
  })
```

> In acest moment trebuia sa fi avut deja pregatit template-ul `home.ejs`, deci ne vom ocupa de crearea template-ului

Template-uri

> Template-ul e folosit prin crearea unor placeholders in interior. Sistemul `ejs` functioneaza cu cativa placeholders: 1 `<% %>` ruleaza javascript in interiorul template-ului 2 `<%= %>` afiseaza ce e interior, aici inseram valorile trimise din obiectul parametru de pe pozitia a 2-a din `res.render(template,obiectCuValori)` 3 `<%- %>` afiseaza date inclusiv html (care nu este `escaped`)

Crearea template-ului home.ejs

- [ ] creaza un fisier `home.ejs` in folderul `Views` care contine structura standard HTML, cel inserat cu **Emmet autocomplete** folosind **'!'**
- [ ] In interiorul lui body insereaza placeholderul <%= numeFisiereText %>
- [ ] Modifica functia callback de pe ruta '/' astfel incat sa trimita vectorul neconvertit in text

```
app.get('/',(req,res)=>{
    const numeFisiere = getListaFisiere();
    res.render('home', { numeFisiere, continutFisier: null });
  })
```

- [ ] In interiorul lui body vom adauga acum blocul javascript de parcurgere a vectorului numeFisiere

```
<% for (let fis of numeFisiere) { %>
        <%= fis %>
<% } %>
```

- [ ] Cream link-urile

```
<% for (let fis of numeFisiere) { %>
```

```
            <a href="/probleme/<%= fis %>">
            <%= fis %>
          </a>

<% } %>
```

- [ ] Inseram link-urile in liste

```
<ul>
   <% for (let fis of numeFisiere) { %>
     <li>
       <a href="/probleme/<%= fis %>">
          <%= fis %>
       </a>
      </li>
<% } %>
</ul>
```

▶ In acest moment avem o lista in browser iar daca dam click pe orice link avem acces prin intermediul parametrului `:problema` din link-ul `<a href="/probleme/<%= fis %>">` ▶ La rularea javascript din template-ul home.ejs, daca in vector exista urmatoarea lista de probleme, `numeFisiere = ['ecgr2','sumaDivizori','medieVector']` ▶ Lista rezultata si trimisa in browser de catre template va avea forma

```
<ul>
    <li><a href = "/probleme/ecgr2">ecgr2</a></li>
    <li><a href = "/probleme/sumaDivizori">sumaDivizori</a></li>
    <li><a href = "/probleme/medieVector">medieVector</a></li>
</ul>
```

▶ Ceea ce urmeaza e pregatirea celei de a doua rute care sa intercepteze valoarea `numeProblema` din url-ul `/probleme/numeProblema` si care in baza acestei valori sa citeasca fisierul `numeProblema.txt` iar textul sa-l trimita catre template-ul `home.ejs`

▶ Deja ruta '/' contine apelul `res.render` cu valoarea `continutFisier`, doar ca intr-o prima faza `continutFisier` este `null`

```
        res.render('home', { numeFisiere, continutFisier: null });
```

▶ Deci trebuie sa pregatim `home.ejs` ca sa primeasca si valoarea lui `continutFisier`

- [ ] Adauga in body in home.ejs elementele `div.container`, `aside` si `main` in forma:

```
<body>
    <div class="container">
        <aside>
        <h3>Lista fișiere</h3>

        </aside>

        <main>

        </main>
    </div>
</body>
```

- [ ] Adauga in `aside` lista creata mai devreme

```
<body>
    <div class="container">
        <aside>
        <h3>Lista fișiere</h3>
            <ul>
                <% for (let fis of numeFisiere) { %>
                    <li>
                        <a href="/probleme/<%= fis %>">
                            <%= fis %>
                        </a>
                    </li>
                <% } %>
            </ul>
        </aside>

        <main>

        </main>
    </div>
</body>
```

- [ ] Adauga in `main` valoarea continutFisier

```
<body>
    <div class="container">
        <aside>
        <h3>Lista fișiere</h3>
            <ul>
                <% for (let fis of numeFisiere) { %>
                    <li>
                        <a href="/probleme/<%= fis %>">
                            <%= fis %>
                        </a>
                    </li>
```

```
                <% } %>
            </ul>
        </aside>

        <main>
            <pre><%= continutFisier %></pre>
         </main>
    </div>
</body>
```

▶ Observatie: Ca sa poata fi testata si trimiterea unui text drept continut fisier se poate da o valoare oareacare pentru continutFisier, `res.render('home', { numeFisiere, continutFisier: "Acesta este un test fisier problema" })`;

```
   app.get('/',(req,res)=>{
       const numeFisiere = getListaFisiere();
       res.render('home', { numeFisiere, continutFisier: "Acesta este un test
fisier problema" });
    })
```

☐ In cele din urma verificam daca `continutFisier` are text si daca are il afisam, iar daca nu, asta insemnand ca abia a fost accesata ruta radacina '/' si nu exista nici un fisier selectat. Deci vom indica userului sa aleaga un fisier din lista

```
<body>
    <div class="container">
        <aside>
        <h3>Lista fișiere</h3>
            <ul>
                <% for (let fis of numeFisiere) { %>
                    <li>
                        <a href="/probleme/<%= fis %>">
                            <%= fis %>
                        </a>
                    </li>
                <% } %>
            </ul>
        </aside>

        <main>
            <% if (continutFisier) { %>
                <h3>Conținutul fișierului:</h3>
                <pre><%= continutFisier %></pre>
            <% } else { %>
                <h3>Selectează un fișier din listă pentru a-i vedea conținutul.
</h3>
            <% } %>
        </main>
```

```
    </div>
</body>
```

> ▶ Asta inseamna ca in div-ul `container` general vor exista doua subcontainere, `aside` cu rol de menu pentru lista de probleme si `main` cu rol de container pentru continutul text al fisierului problema

> ❗ Desi componenta CSS va determina modul vizual de asezarea in pagina, nu e importanta pentru intelegerea mecanismului deci va fi adaugata la final. Se va adauga in tag-ul `style` din tag-ul `head`

Ruta '/probleme/:problema

> Ne va interesa sa extragem valoarea din `url` de dupa al doilea simbol slash '/', adica valoarea 'ecgr1' din url-ul `'/probleme/ecgr1'` si apoi se va citi fisierul `ecgr1.txt` iar continutul va fi trimis catre template-ul `home.ejs`

- [ ] Se obtine valoarea din parametru folosind obiectul params

```
app.get('/probleme/:problema', (req, res) => {
    const problema = req.params.problema;
    res.send(problema)
})
```

- [ ] Se creaza calea catre fisier in baza parametrului si a caii catre folderul `data`

```
app.get('/probleme/:problema', (req, res) => {
    const problema = req.params.problema;
    const problemaPath = path.join(dataPath, `${problema}.txt`);
    res.send(problemaPath)
})
```

- [ ] Se incearca citirea fisierului, daca se reuseste, se trimite in browser

```
app.get('/probleme/:problema', (req, res) => {
    const problema = req.params.problema;
    const problemaPath = path.join(dataPath, `${problema}.txt`);

    let continutFisier;
    try {
    continutFisier = fs.readFileSync(problemaPath, 'utf-8');
    } catch (err) {
    continutFisier = 'Eroare: fișierul nu a fost găsit sau nu poate fi
citit.';
    }

    res.send(continutFisier)
})
```

- [ ] La final punem toate elementele in obiectul pe care il trimitem catre `res.render('home', {numeFisiere,continutFisier})`, atat lista de fisiere cat si continutul fisierului, fiecare afisata separat, lista in `aside`, fisierul text in `main`

```
    app.get('/probleme/:problema', (req, res) => {
        const problema = req.params.problema;
        const problemaPath = path.join(dataPath, `${problema}.txt`);

        let continutFisier;
        try {
        continutFisier = fs.readFileSync(problemaPath, 'utf-8');
        } catch (err) {
        continutFisier = 'Eroare: fișierul nu a fost găsit sau nu poate fi
citit.';
        }

        const numeFisiere = getListaFisiere();

        res.render('home', { numeFisiere, continutFisier });
    })
```

Stilizarea CSS

> Stilurile vor fi adaugate in `style` din `head`. Nu vom incerca acum fisiere externe .css din cauza ca trebuie activat un asa-zis middleware pentru servirea fisierelor statice. Deci ne vom rezuma la inserarea css-ului direct in template. Bineinteles fiecare are libertatea sa faca ce stilizare doreste

```
body {
font-family: Arial, sans-serif;
background-color: #f3f3f3;
}

.container {
border: 2px solid green;
display: flex;
padding: 20px;
background-color: white;
max-width: 1000px;
margin: 30px auto;
box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

aside,
main {
background: rgb(16, 138, 179);
color: white;
padding: 20px;
min-height: 300px;
min-width: 300px;
```

```
}

aside {
margin-right: 20px;
}
ul {
list-style: none;
padding: 0;
}

li {
margin-bottom: 10px;
}

a {
color: white;
text-decoration: none;
font-weight: bold;
}

a:hover {
text-decoration: underline;
}

pre {
background-color: #003a4a;
color: #e0e0e0;
padding: 10px;
white-space: pre-wrap;
border-radius: 4px;
}
```