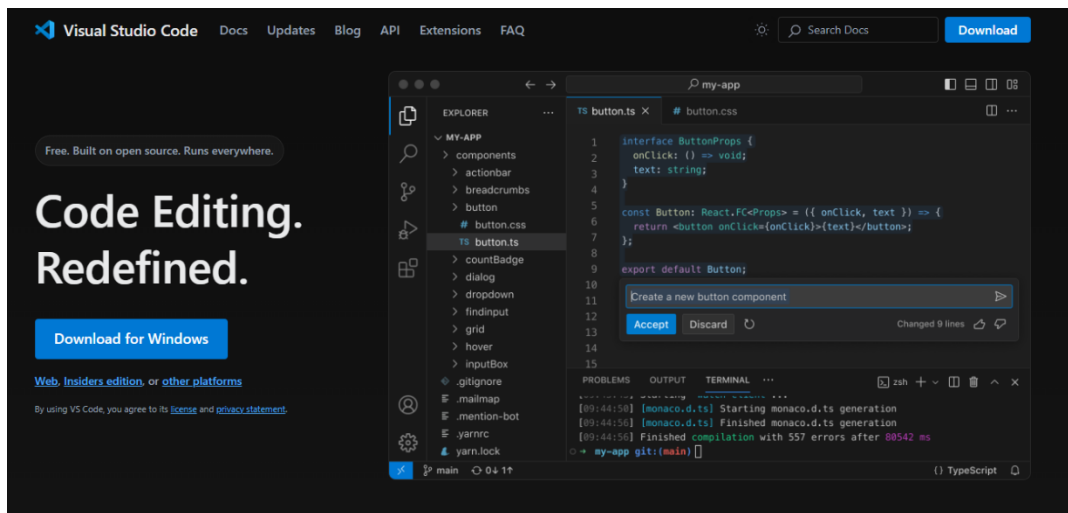


Vue.js CRUD Application cu API Express.js

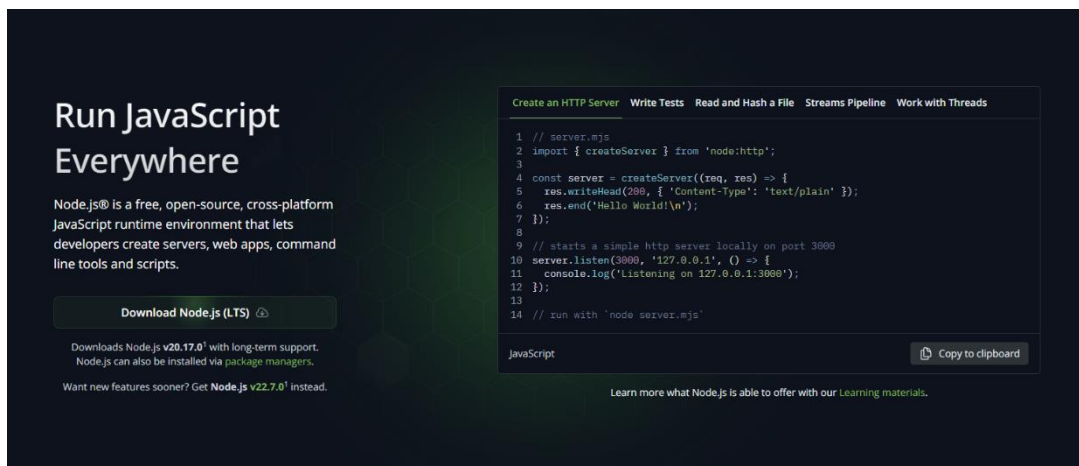


În urma parcurgerii acestei documentații, vom crea un **API** și o aplicație client care va simula o aplicație de cashflow management, având o baza de date **MySQL**. Aplicația va implementa validare de autentificare și criptare de parole.

1. Descarcați Visual Studio Code: <https://code.visualstudio.com/>



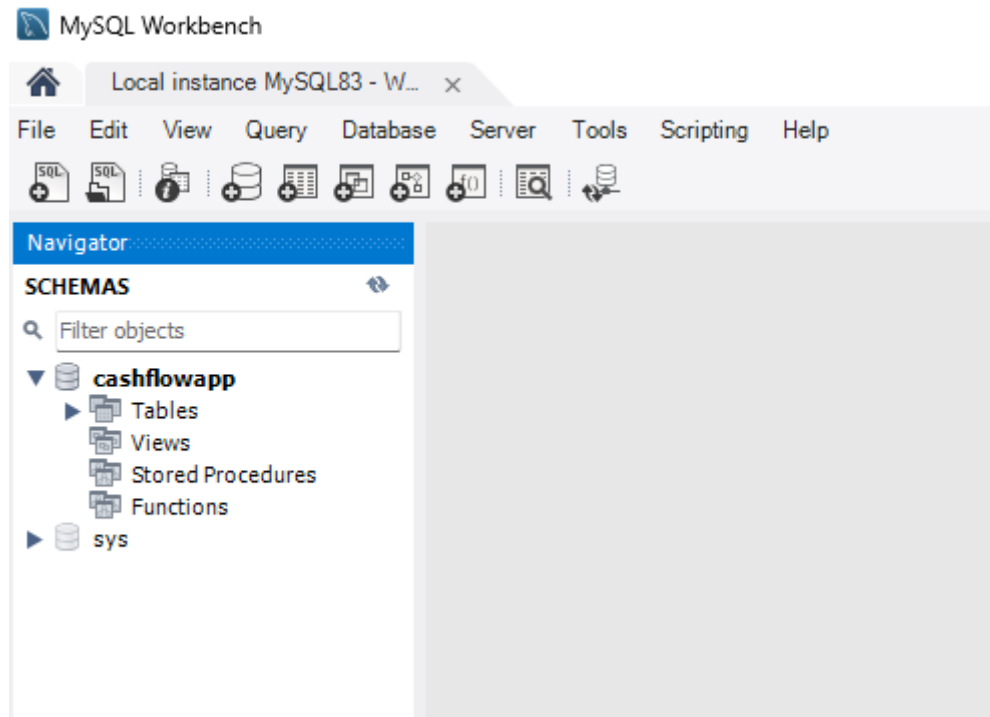
2. Descarcați Node.js: <https://nodejs.org/en>



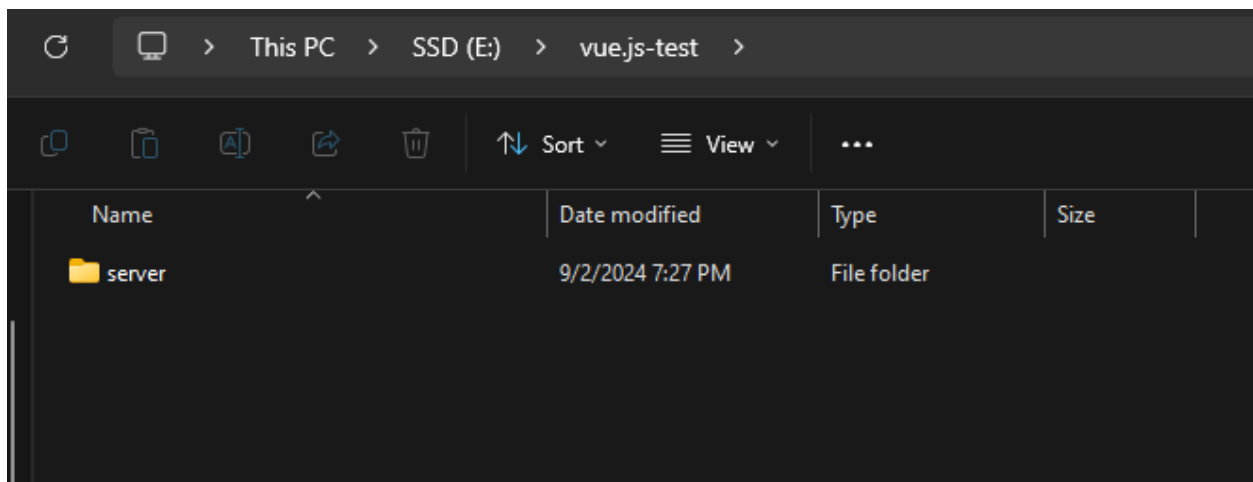
3. Descarcați și configurați serverul local:

<https://dev.mysql.com/downloads/mysql/>

Configurați user-ul **root** și atribuiți o parolă acestuia (va fi folosită ulterior la conectare). După conectare, folosiți **MySQL Workbench** pentru a crea o bază de date numită **cashflowapp**.

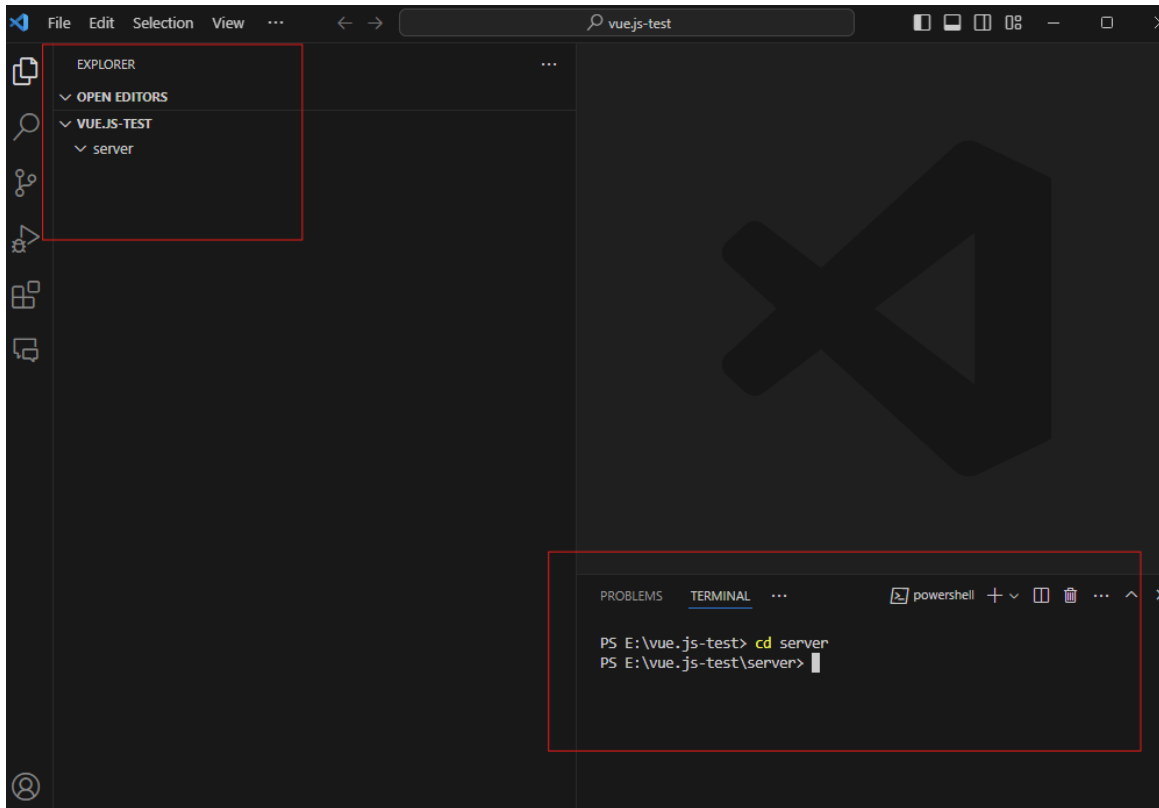


4. Creați un director în care se va stoca aplicația



În interiorul directorului selectat, creați un subdirector numit server. Acesta se va deschide cu VS Code.

5. Se va deschide un terminal în VS Code și se va selecta directorul server



6. Creați template-ul de server folosind Express.js

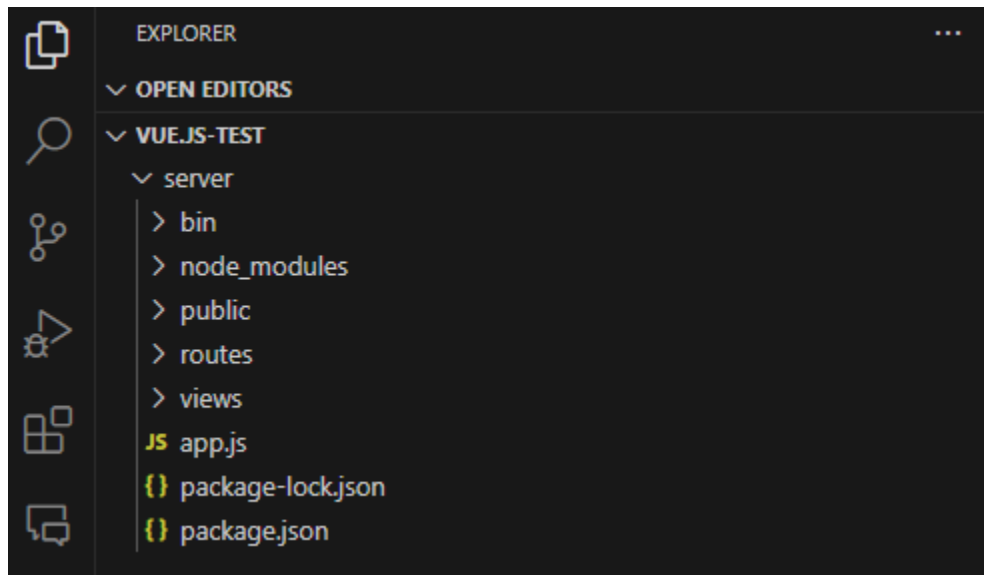
În terminal se vor rula următoarele comenzi:

- **npx express-generator**
- **npm install**
- **npm audit fix --force** (daca exista sau pana nu mai exista vulnerabilitati)

Aplicația de server va putea fi pornită cu **npm start** și va rula la URL-ul:

<http://localhost:3000/>.

După finalizarea pașilor, directorul server va arata precum în poza următoare:



7. Instalați Swagger UI pentru documentația/testarea API-ului

În terminalul directorului server, rulați următoarele comenzi:

- **npm install swagger-ui-express**
- **npm install swagger-jsdoc**

8. Configurați CORS (cross-origin resource sharing) pentru a putea realiza HTTP request-urile externe

În terminalul directorului **server** se va rula următoarea comandă:

- **npm install cors**

În fișierul **app.js** din directorul **server**, se vor face următoarele modificări pentru a permite aplicației de client, care va fi realizată în pașii următori, să acceseze endpoint-urile API-ului.

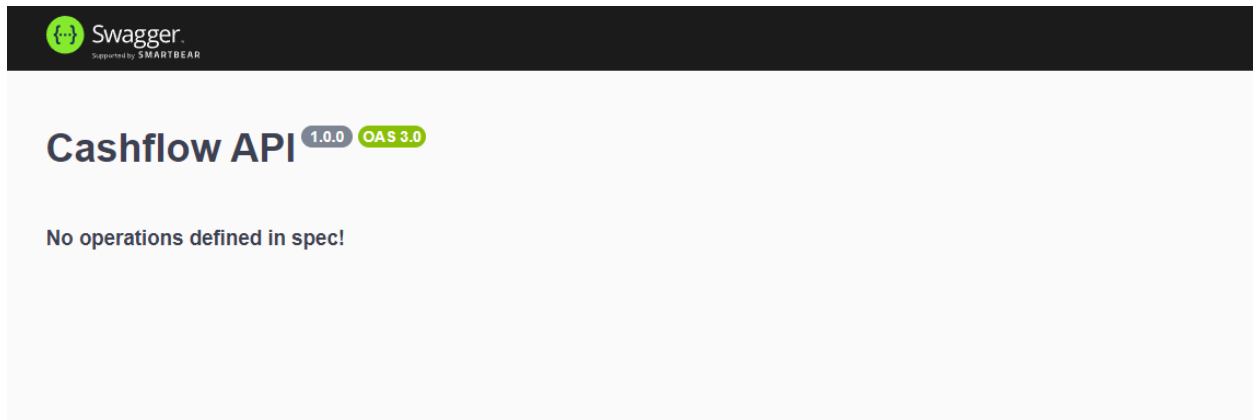
```
JS app.js X
server > JS app.js > [?] options > [?] definition > [?] info
1  var createError = require('http-errors');
2  var express = require('express');
3  var path = require('path');
4  var cookieParser = require('cookie-parser');
5  var logger = require('morgan');
6  const cors = require('cors');
7
8  var indexRouter = require('./routes/index');
9  var usersRouter = require('./routes/users');
10
11 var app = express();
12
13 // cors setup
14 app.use(cors({
15   exposedHeaders: ['Authorization']
16 }));
17
```

9. Configurați Swagger-UI

În fișierul **app.js** din directorul **server**, se vor realiza următoarele modificari pentru a permite documentarea/testarea de endpoint-uri pentru API.

```
2
3 // cors setup
4 app.use(cors({
5   exposedHeaders: ['Authorization']
6 }));
7
8 // swagger setup
9 const swaggerUi = require('swagger-ui-express');
10 const swaggerJsdoc = require('swagger-jsdoc');
11
12 const options = {
13   definition: {
14     openapi: '3.0.0',
15     info: {
16       title: 'Cashflow API',
17       version: '1.0.0',
18     },
19   },
20   apis: ['./routes/*.js'], // files containing annotations as above
21 };
22
23 const openapiSpecification = swaggerJsdoc(options);
24 app.use('/api', swaggerUi.serve, swaggerUi.setup(openapiSpecification));
```

API-ul va putea fi accesat la URL-ul: <http://localhost:3000/api/>



10. Realizați legatura cu serverul MySQL

În terminalul fișierului server rulați următoarea comandă:

- **npm install mysql**

În fișierul **app.js** din directorul **server**, se vor declara variabilele responsabile pentru interacțiunea cu serverul **MySQL**.

```
36 app.use('/api', swaggerUi.setup(swaggerUi.setup(openapiSpec));
37
38 // mysql setup
39 const mysql = require('mysql');
40 const db = mysql.createConnection({
41   host: 'localhost',
42   user: 'root',
43   password: 'c4shfl0w4pp',
44   database: 'cashflowapp'
45 });
```

11. Conectați-vă la baza de date, creați tabelele și faceți conexiunea accesibilă din toate rutele

În fișierul **app.js** din directorul **server**, realizați următoarele modificări. Următorul snippet de cod se va ocupa și de crearea tabelelor necesare, în caz că acestea nu există deja în baza de date **cashflowapp** din **MySQL**. Codul se va situa în continuarea setup-ului inițial din pasul 10.

Cod sursă: <https://pastebin.com/6keP3Lsn>

```

42 db.connect((err) => {
43   if (err) {
44     console.log(err);
45   }
46   else {
47     console.log('Connected to database');
48     const createUsersTableQuery = `CREATE TABLE IF NOT EXISTS users (
49       idUsers INT NOT NULL AUTO_INCREMENT,
50       username VARCHAR(30) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_ci DEFAULT NULL,
51       password VARCHAR(40) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_ci DEFAULT NULL,
52       PRIMARY KEY (idUsers)
53     ) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci`;
54     const createLogTableQuery = `CREATE TABLE IF NOT EXISTS log (
55       idcashflowLog INT NOT NULL AUTO_INCREMENT,
56       idUser INT NOT NULL,
57       idUserSelected INT NOT NULL,
58       description VARCHAR(500) NULL,
59       type VARCHAR(45) NULL,
60       value FLOAT NOT NULL,
61       currency VARCHAR(45) NOT NULL,
62       date DATETIME NOT NULL,
63       PRIMARY KEY (idcashflowLog),
64       INDEX FK_Users_idx (idUser ASC) VISIBLE,
65       CONSTRAINT FK_UsersLog
66         FOREIGN KEY (idUser)
67         REFERENCES cashflowapp.users (idUsers)
68         ON DELETE RESTRICT
69         ON UPDATE NO ACTION)`;
70     db.query(createUsersTableQuery, (err, result) => {
71       if (err) {
72         console.log(err);
73       }
74       else {
75         console.log('Users table checked/created successfully.');

```

In continuarea codului de conectare, vom face variabila db accesibilă din orice ruta:

```

84   }
85   })
86 }
87 });
88
89 // make db accessible to routes
90 app.use((req, res, next) => {
91   req.db = db;
92   next();
93 });
94

```

12. Creați un template de aplicație client folosind Vue.js

În terminal, selectați directorul de baza al proiectului, iar în terminal se vor realiza următoarele operații (referință: <https://github.com/vuejs/create-vue>)

```
PS E:\vue.js-test> npm create vue@latest
```

```
Vue.js - The Progressive JavaScript Framework
```

```
✓ Project name: ... cashflow
✓ Add TypeScript? ... No / Yes
✓ Add JSX Support? ... No / Yes
✓ Add Vue Router for Single Page Application development? ... No / Yes
✓ Add Pinia for state management? ... No / Yes
✓ Add Vitest for Unit Testing? ... No / Yes
✓ Add an End-to-End Testing Solution? » No
✓ Add ESLint for code quality? ... No / Yes
✓ Add Prettier for code formatting? ... No / Yes
✓ Add Vue DevTools 7 extension for debugging? (experimental) ... No / Yes
```

```
Scaffolding project in E:\vue.js-test\cashflow...
```

```
Done. Now run:
```

```
cd cashflow
npm install
npm run format
npm run dev
```

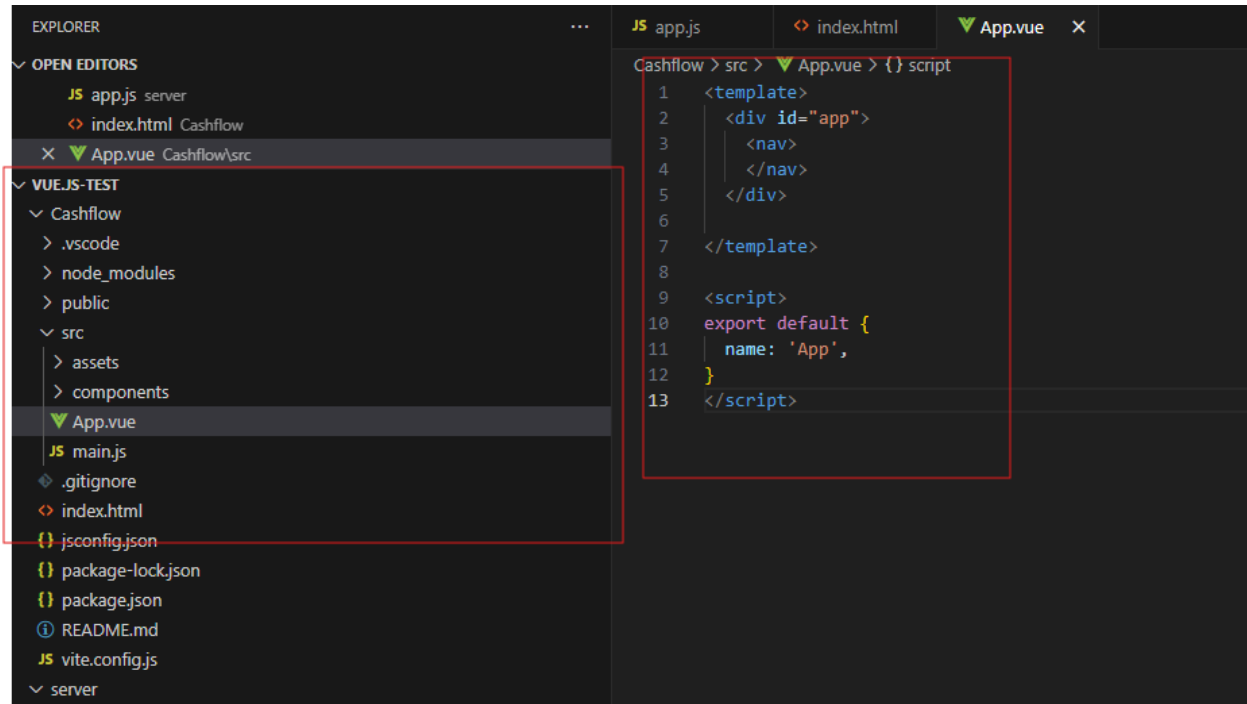
13. Configurare Index.html din client și importare CSS Font Awesome

În fișierul **index.html** din proiectul **Cashflow** creat cu Vue, anterior, se vor realiza următoarele modificări:

```
JS app.js  <> index.html
Cashflow > <> index.html > html
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5    <meta charset="UTF-8">
6    <link rel="icon" href="/favicon.ico">
7    <meta name="viewport" content="width=device-width, initial-scale=1.0">
8    <!-- Font Awesome CSS-->
9    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.5.2/css/all.min.css"
10      integrity="sha512-SnH5WK+bZxgPHs44uWIX+LLJA9/2PkPKZ5QiAj6Ta86w+fsb2TkcmfRyVX3pBnMfCV7oQPJk19QevSCWr3W6A=="
11      crossorigin="anonymous" referrerpolicy="no-referrer" />
12    <title>Cashflow</title>
13  </head>
14
15  <body id="content">
16    <div class="app-global" id="app"></div>
17    <script type="module" src="/src/main.js"></script>
18  </body>
19
20 </html>
```


14. Initializare App.vue ca un empty template

Se va deschide fișierul **App.vue** din subdirectorul **src** corespunzator proiectului de client și se vor face următoarele modificări:



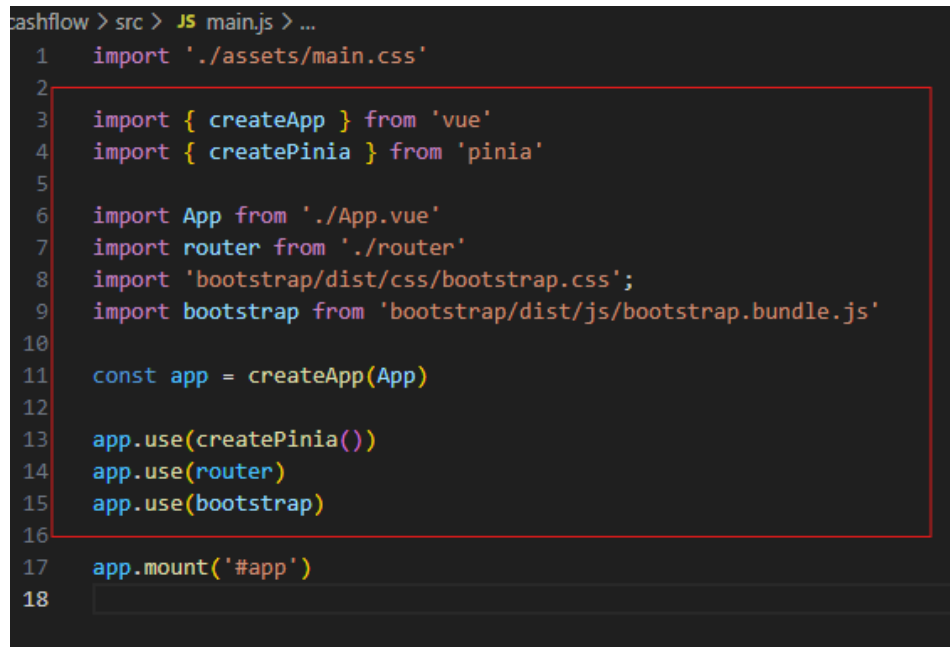
15. Instalați Bootstrap în aplicația client

În terminalul fișierului client se va rula următoarea comandă:

- **npm install bootstrap@5**

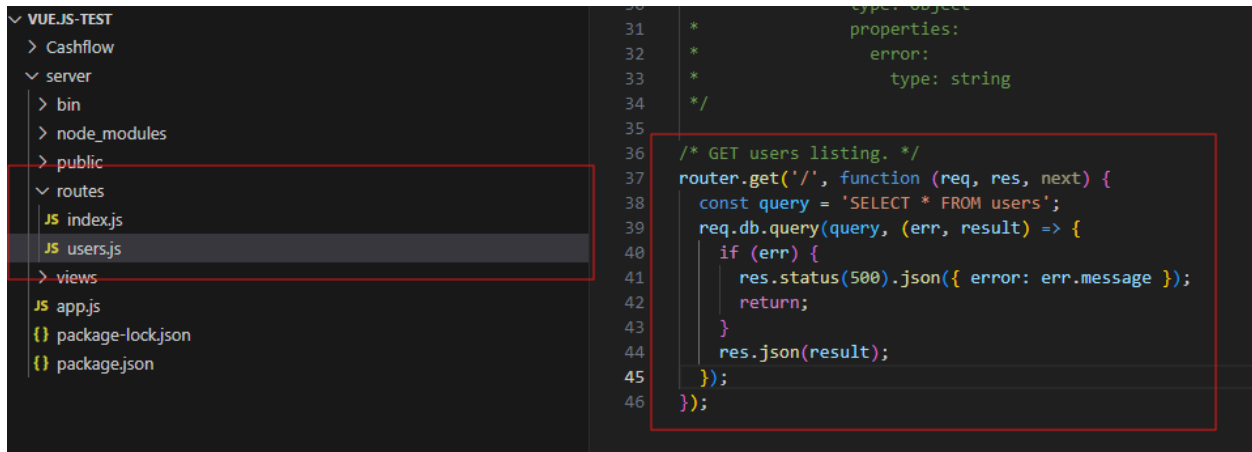
16. Importați Bootstrap la nivelul aplicației

În fișierul **main.js** din aplicația client, se vor face următoarele modificări pentru importarea bootstrap și folosirea routerelor.



17. Incepeți configurarea endpoint-urilor pentru API: metoda get pentru users

În fișierul **users.js**, din subdirectorul **routes** al serverului, se va realiza codul pentru endpoint-ul de get pentru users.

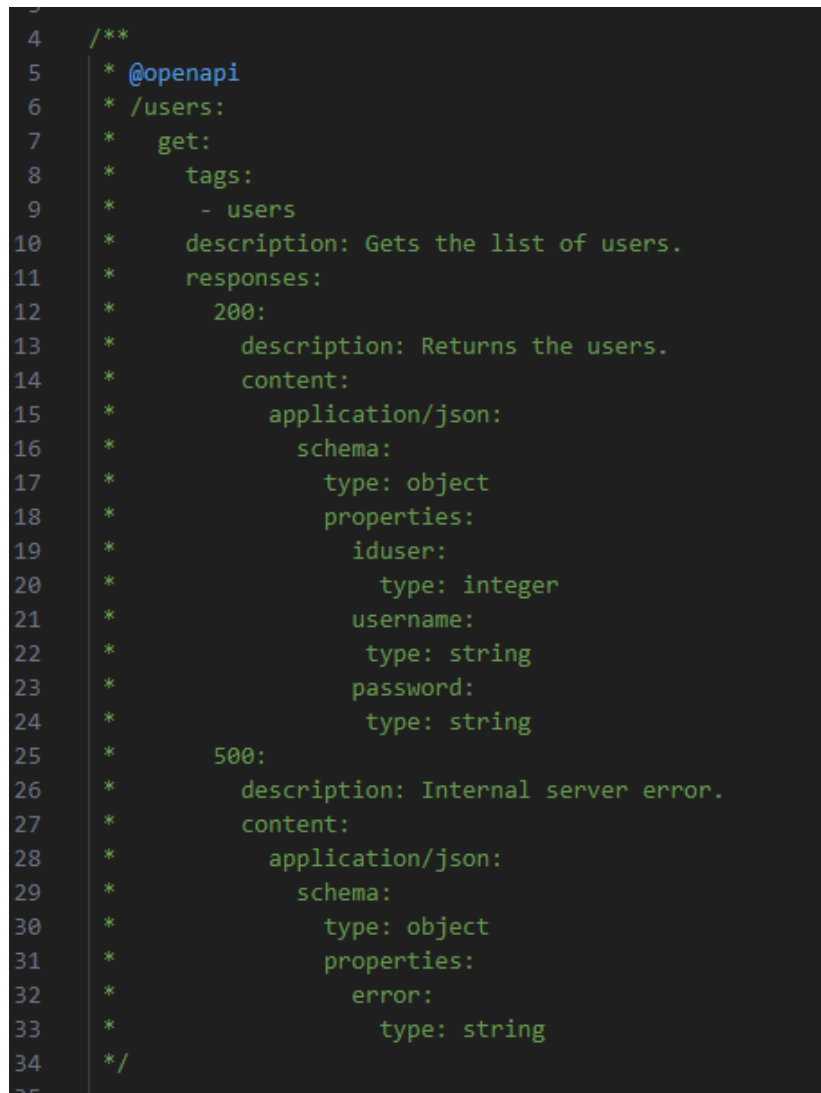


```
30      type: object
31      *      properties:
32      *      error:
33      *          type: string
34      */
35
36  /* GET users listing. */
37  router.get('/', function (req, res, next) {
38    const query = 'SELECT * FROM users';
39    req.db.query(query, (err, result) => {
40      if (err) {
41        res.status(500).json({ error: err.message });
42        return;
43      }
44      res.json(result);
45    });
46  });
```

18. (OPȚIONAL) Cod pentru documentația Swagger a metodei

Codul va fi scris respectând indentarea, deasupra metodei.

Cod sursă: <https://pastebin.com/mVFFiHWY>



```
4  /**
5   * @openapi
6   * /users:
7   *   get:
8   *     tags:
9   *       - users
10    *     description: Gets the list of users.
11    *     responses:
12    *       200:
13    *         description: Returns the users.
14    *         content:
15    *           application/json:
16    *             schema:
17    *               type: object
18    *               properties:
19    *                 iduser:
20    *                   type: integer
21    *                 username:
22    *                   type: string
23    *                 password:
24    *                   type: string
25    *       500:
26    *         description: Internal server error.
27    *         content:
28    *           application/json:
29    *             schema:
30    *               type: object
31    *               properties:
32    *                 error:
33    *                   type: string
34    */
35
```

19. Configurare endpoint get user după nume

În același fișier **users.js**, se va configura endpoint-ul de get user după username.

```
90
97 router.get('/:name', function (req, res, next) {
98   const query = 'SELECT idUsers FROM users WHERE username = ?';
99   if (!req.params.name) {
100     res.status(400).json({ error: 'The request has missing information!' });
101     return;
102   }
103   req.db.query(query, [req.params.name], (err, result) => {
104     if (err) {
105       res.status(500).json({ error: err.message });
106       return;
107     }
108     res.json(result);
109   });
110 });
```

20. (OPȚIONAL) Cod pentru documentația Swagger a metodei

Codul va fi scris respectând indentarea, deasupra metodei.

Cod sursă: <https://pastebin.com/3h8Rxs3m>

```
48 /**
49  * @openapi
50  * /users/{name}:
51  *   get:
52  *     tags:
53  *       - users
54  *     description: Gets the user id with the given username.
55  *     parameters:
56  *       - in: path
57  *         name: name
58  *         schema:
59  *           type: string
60  *           required: true
61  *     responses:
62  *       200:
63  *         description: Returns the user id.
64  *         content:
65  *           application/json:
66  *             schema:
67  *               type: object
68  *               properties:
69  *                 iduser:
70  *                   type: integer
71  *                 username:
72  *                   type: string
73  *                 password:
74  *                   type: string
75  *       400:
76  *         description: Error caused by an inappropriate input.
77  *         content:
78  *           application/json:
79  *             schema:
80  *               type: object
81  *               properties:
82  *                 error:
83  *                   type: string
84  *       500:
85  *         description: Internal server error.
86  *         content:
87  *           application/json:
88  *             schema:
89  *               type: object
90  *               properties:
91  *                 error:
92  *                   type: string
93  */
```

21. Instalați JWT (JSON Web Token)

În terminalul aplicației de server, se va instala JWT, folosind următoarea comandă:

- `npm install jsonwebtoken`

22. Configurare endpoint de add user

În fișierul **users.js** din subdirectorul routes al aplicației server, se vor face următoarele modificări:

```
ver > routes > JS users.js > ...
1  var express = require('express');
2  var router = express.Router();
3  const jwt = require('jsonwebtoken');
4  const crypto = require('crypto');
5
6  /**
```

Variabila **JWT** va fi folosită pentru verificarea logării, iar **crypto** va fi folosită pentru criptarea parolei userilor.

În continuare, se va începe prima parte din codul pentru endpoint-ul de addUser.

```
169 router.post('/addUser', function (req, res, next) {
170   const insertQuery = 'INSERT INTO users (username, password) VALUES (?, ?)';
171   const checkUsername = 'SELECT COUNT(idUsers) AS count FROM users WHERE username = ?';
172
173   if (!req.body.username || !req.body.password) {
174     res.status(400).json({ success: false, error: 'The request must have an username and password!' });
175     return;
176   }
177   if (req.body.username.length < 5) {
178     res.status(400).json({ success: false, error: 'The username must have at least 5 characters!' });
179     return;
180   }
181   if (req.body.password.length < 5) {
182     res.status(400).json({ success: false, error: 'The password must have at least 5 characters!' });
183     return;
184   }
185   if (req.body.username.length > 30) {
186     res.status(400).json({ success: false, error: 'The username must have at most 30 characters!' });
187     return;
188   }
189   if (req.body.password.length > 30) {
190     res.status(400).json({ success: false, error: 'The password must have at most 30 characters!' });
191     return;
192   }
193
194   const hasUpperCase = /[A-Z]/.test(req.body.password); // regex test for uppercase
195   const hasSpecialChar = /[!@#%$^&*(),.?":{}|<>]/.test(req.body.password); // regex test for special char
196   if (!hasUpperCase || !hasSpecialChar) {
197     res.status(400).json({ error: 'The password must have at least one uppercase letter and one special character!' });
198     return;
199   }
200 }
```

Aici se vor realiza verificările de baza , precum și niste regex check-uri pentru caractere speciale și un caracter uppercase.

Continuarea codului de endpoint, cu password hashing:

```
1 // Hash the password using MD5
2 const hashedPassword = crypto.createHash('md5').update(req.body.password).digest('hex');
3
4 req.db.beginTransaction((err) => {
5
6   if (err) {
7     res.status(500).json({ success: false, error: err.message });
8     return;
9   }
10
11   req.db.query(checkUsername, [req.body.username], (err, result) => {
12     if (err) {
13       res.status(500).json({ success: false, error: err.message });
14       return;
15     }
16     if (result[0]['count'] > 0) {
17       res.status(400).json({ success: false, error: 'The username already exists!' });
18       return;
19     }
20
21     req.db.query(insertQuery, [req.body.username, hashedPassword], (err, result) => {
22       if (err) {
23         res.status(500).json({ success: false, error: err.message });
24         return;
25       }
26
27       req.db.commit((err) => {
28         if (err) {
29           return req.db.rollback(() => {
30             res.status(500).json({ error: err.message });
31           });
32         }
33         res.json({ success: true, message: 'User added successfully!' });
34       });
35     });
36   });
37 });
```

23. (OPȚIONAL) Cod pentru documentația Swagger a metodei

Codul va fi scris respectând indentarea, deasupra metodei.

Deoarece metodele au o descriere din ce în ce mai complexă, iar acest proces nu este unul de o importanță majoră, se recomandă copy-paste-ul deasupra metodei addUser de pe link-ul următor: <https://pastebin.com/1FUbRz5j>

24. Configurare endpoint de login, și exportare router.

În același fișier **users.js**, se va adăuga codul pentru endpoint-ul care se va ocupa cu logarea user-ului.

```
293 router.post('/login', function (req, res, next) {
294   const { username, password } = req.body;
295   const query = 'SELECT * FROM users WHERE username = ?';
296
297   // Hash the password using MD5
298   const hashedPassword = crypto.createHash('md5').update(password).digest('hex');
299
300   req.db.query(query, [username], (err, results) => {
301     if (err) {
302       res.status(500).json({ success: false, message: err.message });
303       return;
304     }
305     if (results.length > 0) {
306       const user = results[0];
307       if (user.password == hashedPassword) {
308         const token = jwt.sign({ id: user.idUsers, username: username }, 'cashflow-key', { expiresIn: '24h' });
309         res.json({ success: true, token: token });
310       } else {
311         res.status(401).json({ success: false, message: 'Incorrect login details' });
312       }
313     } else {
314       res.status(401).json({ success: false, message: 'Incorrect login details' });
315     }
316   });
317 });
318
319 module.exports = router;
```

Aici se va folosi și package-ul JWT, care va crea în local storage un token, folosind o cheie privată (în cazul nostru: “cashflow-key”), cu o dată de expirare (în cazul nostru 24h). Signatura acestui token va fi verificată la fiecare accesare care necesită autentificare. Dacă această verificare eșuează, user-ul va fi trimis pe pagina de login, care o vom scrie ulterior.

25. Export la fișierul configurat

La finalul fișierului **users.js**, vom adăuga următoarea linie de cod pentru a realiza exportul fișierului.

```
408   });
409 });
410
411 module.exports = router;
```

26. (OPȚIONAL) Cod pentru documentația Swagger a metodei

Codul va fi scris respectând indentarea, deasupra metodei.

Deoarece metodele au o descriere din ce în ce mai complexă, iar acest proces nu este unul de o importanță majoră, se recomandă copy-paste-ul deasupra metodei login de pe link-ul următor: <https://pastebin.com/MNbu8tG4>

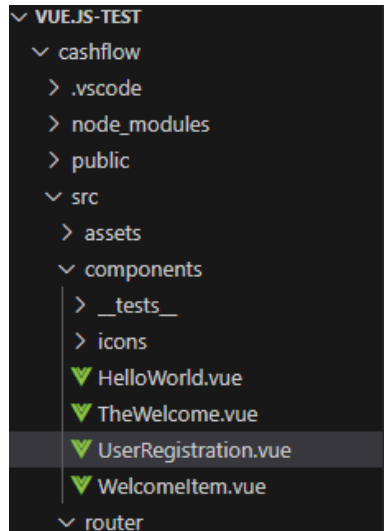
27. Instalare axios pentru apelare de endpoint-uri mai ușoară

În terminalul aplicației de server, se va apela următoarea comandă:

- **npm install axios**

28. Adăugare componenta pentru register a utilizatorului

În fișierul aplicației client, în subdirectorul **components**, se va crea un fișier **UserRegistration.vue**.



29. Template pentru UserRegistration.vue

În interiorul fișierului creat, se va configura partea de template, obligatorie pentru o componentă vue.

Cod sursă: <https://pastebin.com/P4EfMjgK>

```
1 <template>
2   <div class="container-fluid login-container">
3     <form class="main-form">
4       <h1 class="text-center">Register</h1>
5       <div class="mb-3">
6         <label for="inputUsername" class="form-label">Username</label>
7         <input v-model="username" type="text" class="form-control" id="inputUsername" required>
8       </div>
9       <div class="mb-3">
10        <label for="inputRepeatPassword" class="form-label">Password</label>
11        <input v-model="repeatPassword" type="password" class="form-control" id="inputRepeatPassword"
12          aria-describedby="pwHelp" required>
13      </div>
14      <div class="mb-3">
15        <label for="inputPassword" class="form-label">Repeat password</label>
16        <input v-model="password" type="password" class="form-control" id="inputPassword" aria-describedby="pwHelp"
17          required>
18      </div>
19      <div id="pwHelp" class="form-text">We'll never share your password with anyone else.</div>
20      <button @click="register($event)" type="submit" class="btn btn-primary">Register</button>
21    </form>
22
23
24
25
26    <div class="toast role="alert" aria-live="assertive" aria-atomic="true" data-bs-autohide="true"
27      :class="{ 'show': showToast }" style="position: absolute; top: 0; right: 0;">
28      <div class="toast-header">
29        <strong class="me-auto">Notification</strong>
30        <button type="button" class="ml-2 mb-1 btn-close" @click="showToast = false"></button>
31      </div>
32      <div class="toast-body">
33        {{ toastMessage }}
34      </div>
35    </div>
36
37  </div>
38 </template>
```

30. Configurare route pentru pagina de register

În subdirectorul **router**, din aplicația client, se va accesa fișierul **index.js**. Aici se va crea o rută default și o rută pentru pagina de register. De asemenea, se va scrie codul care verifică dacă o pagina necesită autentificare și va verifica existența token-ului.

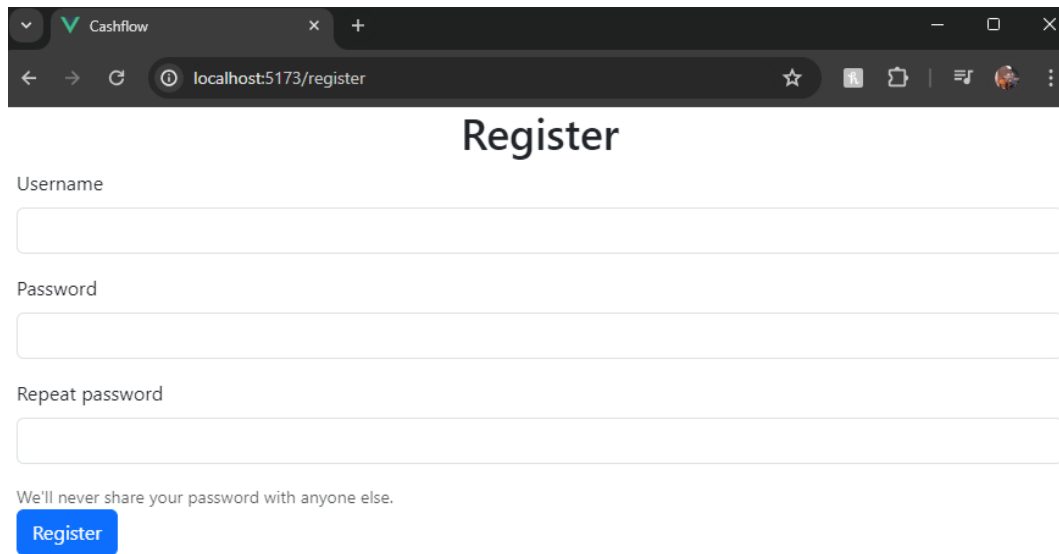
```
1 import { createRouter, createWebHistory } from 'vue-router'
2 import UserRegistration from '../components/UserRegistration.vue';
3
4 const routes = [
5   { path: '/', redirect: '/dashboard' },
6   { path: '/register', component: UserRegistration }
7 ];
8
9 const router = createRouter({
10   history: createWebHistory(),
11   routes // short for `routes: routes`
12 });
13
14 router.beforeEach((to, from, next) => {
15   if (to.matched.some(record => record.meta.requiresAuth) && !localStorage.getItem('user-token')) {
16     next('/login');
17   } else {
18     next();
19   }
20 });
21
22 export default router
23 |
```

31. Configurare router link in App.vue

În fișierul **App.vue** din aplicația client, configurați router link-ul pentru a putea încărca pagina scrisa anterior.

```
cashflow > src > App.vue > {} script
1 <template>
2   <div id="app">
3     <nav>
4       <RouterLink to="/register"></RouterLink>
5     </nav>
6     <RouterView />
7   </div>
8
9 </template>
10
11 <script>
12   export default {
13     name: 'App',
14   }
15 </script>
```


La o accesare a URL-ului, pagina va arata în felul următor:



Cashflow

localhost:5173/register

Register

Username

Password

Repeat password

We'll never share your password with anyone else.

Register



32. Configurare metode și apelare API cu axios

Se va declara un element **script** după cel de template din **UserRegistration.vue**. Această parte de cod va fi responsabilă pentru apelarea endpoint-urilor din API.

```

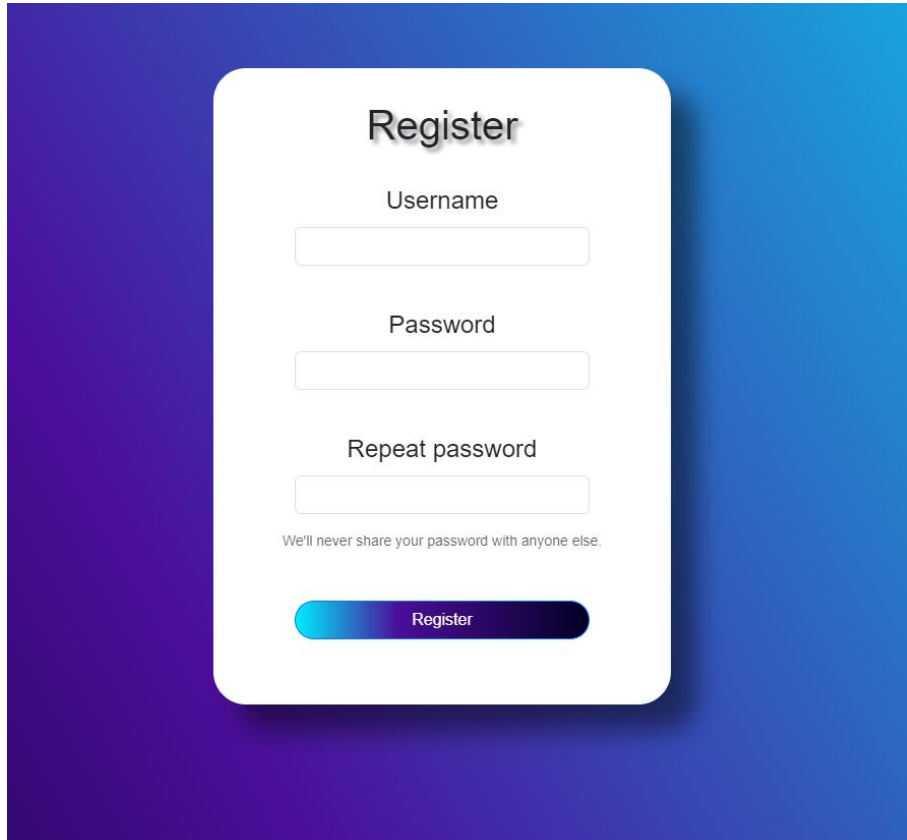
41 import axios from 'axios';
42 export default {
43   name: 'UserRegistration',
44   props: {
45     msg: String
46   },
47   data() {
48     return {
49       username: '',
50       password: '',
51       repeatPassword: '',
52       showToast: false,
53       toastMessage: ''
54     };
55   },
56   methods: {
57     register(event) {
58       event.preventDefault();
59       if (this.password !== this.repeatPassword) {
60         this.toastMessage = 'Passwords do not match!';
61         this.showToast = true;
62
63         setTimeout(() => {
64           this.showToast = false;
65         }, 5000);
66
67         return;
68       }
69       axios.post('http://localhost:3000/users/addUser', {
70         username: this.username,
71         password: this.password
72       })
73         .then(response => {
74           if (response.data.success) {
75             // The registration was successful
76             this.$router.push('/login');
77           }
78         })
79         .catch(error => {
80           this.toastMessage = 'Invalid registration: ' + error.response.data.error;
81           this.showToast = true;
82           setTimeout(() => {
83             this.showToast = false;
84           }, 5000);
85         });
86     }
87   }
88 }

```

După realizarea acestui pas, pagina de register va funcționa corect și user-ul va fi trimis către endpoint-ul de login, dacă înregistrarea de utilizator a reușit. Se poate observa cum metoda register, care este asignată butonului apelează endpoint-ul asociat din API, validarea register-ului realizându-se în backend. Dacă validarea eșuează, se va afișa un toast message pe ecran cu motivul.

33. (OPȚIONAL) Stilizare pentru intreg proiectul.

În fișierul **main.css** , din subdirectorul **src/assets** al aplicației client, se vor introduce următoarele stiluri **CSS**. Stilurile conțin și componente ce le vom crea ulterior în aplicație. În urma aplicării stilurilor, pagina va arăta în felul următor:

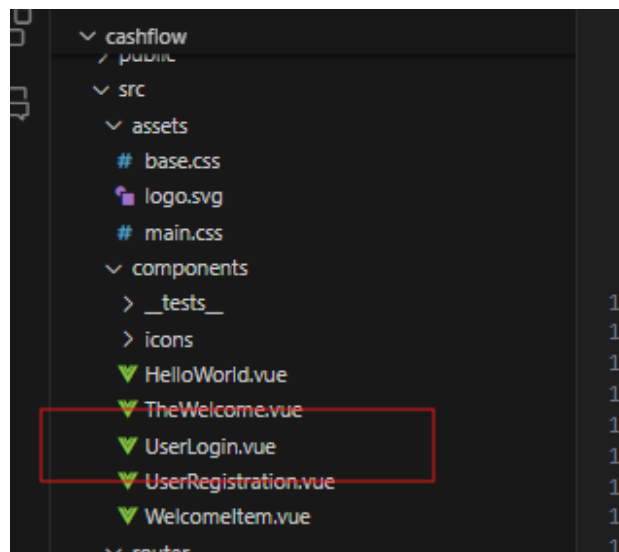


Conținutul fișierului **main.css** se poate găsi la link-ul următor:

<https://pastebin.com/RdM1rcRc>

34. Creare pagină de login

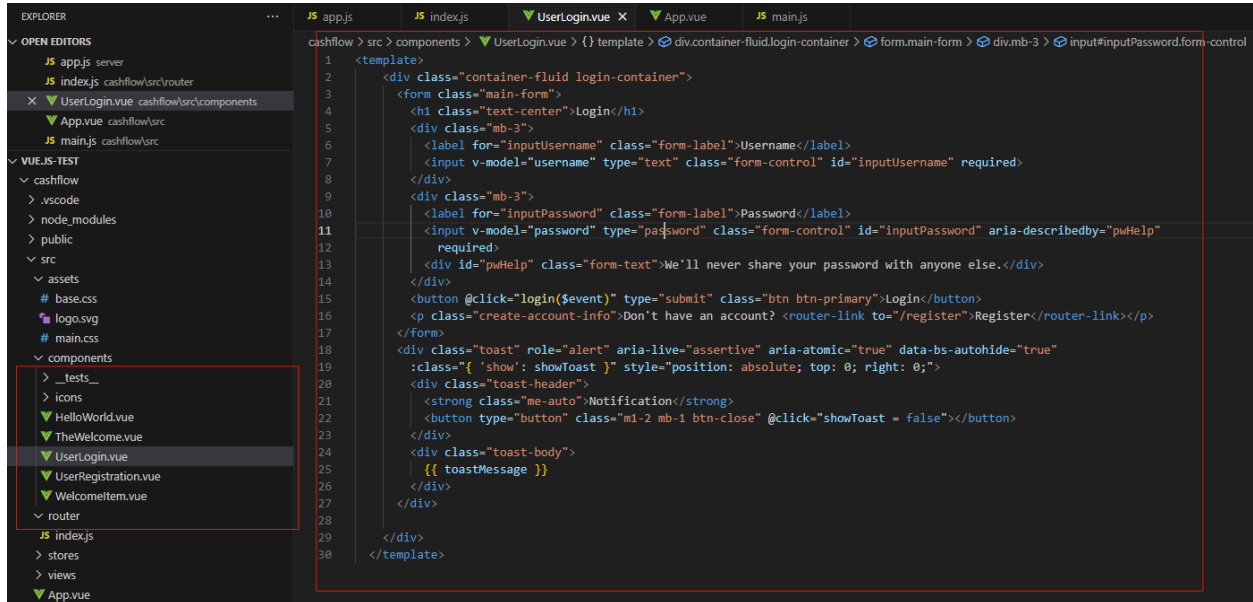
Precum pagina de register, vom crea un nou fișier **UserLogin.vue** în subdirectorul **components** din aplicația client.



35. Template-ul din UserLogin.vue

În pagina nou creată, se va adăuga componentă de template, necesară oricărei pagini Vue.

Cod sursă: <https://pastebin.com/YENRJ2sp>



```
1 <template>
2   <div class="container-fluid login-container">
3     <form class="main-form">
4       <h1 class="text-center">Login</h1>
5       <div class="mb-3">
6         <label for="inputUsername" class="form-label">Username</label>
7         <input v-model="username" type="text" class="form-control" id="inputUsername" required>
8       </div>
9       <div class="mb-3">
10        <label for="inputPassword" class="form-label">Password</label>
11        <input v-model="password" type="password" class="form-control" id="inputPassword" aria-describedby="pwHelp"
12          required>
13        <div id="pwHelp" class="form-text">We'll never share your password with anyone else.</div>
14      </div>
15      <button @click="login($event)" type="submit" class="btn btn-primary">Login</button>
16      <p class="create-account-info">Don't have an account? <router-link to="/register">Register</router-link></p>
17    </form>
18    <div class="toast" role="alert" aria-live="assertive" aria-atomic="true" data-bs-autohide="true"
19      :class="{ 'show': showToast }" style="position: absolute; top: 0; right: 0;">
20      <div class="toast-header">
21        <strong class="me-auto">Notification</strong>
22        <button type="button" class="m1-2 mb-1 btn-close" @click="showToast = false"></button>
23      </div>
24      <div class="toast-body">
25        {{ toastMessage }}
26      </div>
27    </div>
28  </div>
29 </template>
```

Această pagină este aproape identică cu cea de register, singurele diferențe fiind lipsa celor doua câmpuri necesare pentru validarea parolei, acum fiind nevoie doar de unul și introducerea unui element cu router link către pagina de **register**. O altă diferență este numele metodei atașate la buton, în loc de register, aceasta se va numi **login**.

36. Script-ul din UserLogin.vue

Vom parcurge pașii invers fata de anterior, iar acum vom scrie codul care va utiliza API-ul pentru realizarea operațiunii de logare. Vom declara un nou bloc script în continuare la cel template.

```

32 <script>
33 import axios from 'axios';
34
35 export default {
36   name: 'UserLogin',
37   props: {
38     msg: String
39   },
40   data() {
41     return {
42       username: '',
43       password: '',
44       showToast: false,
45       toastMessage: ''
46     };
47   },
48   methods: {
49     login(event) {
50       event.preventDefault();
51       axios.post('http://localhost:3000/users/login', {
52         username: this.username,
53         password: this.password
54       })
55         .then(response => {
56           if (response.data.success) {
57             // The login was successful
58             localStorage.setItem('user-token', response.data.token);
59             this.$router.push('/dashboard');
60           }
61         })
62         .catch(error => {
63           this.toastMessage = 'Invalid login: ' + error.response.data.message;
64           this.showToast = true;
65
66           setTimeout(() => {
67             this.showToast = false;
68           }, 5000);
69         });
70   }
71 }
72 }
73 </script>

```

Se poate observa cum metoda login, atașată butonului apelează endpoint-ul asociat din API, iar toată logica de logare se realizează în backend. Dacă logarea eșuează, un toast message va apărea pe ecranul utilizatorului.

37. Configurare route pentru pagina de login

În fișierul **index.js** al aplicației client, se va crea route-ul pentru pagina creata anterior:

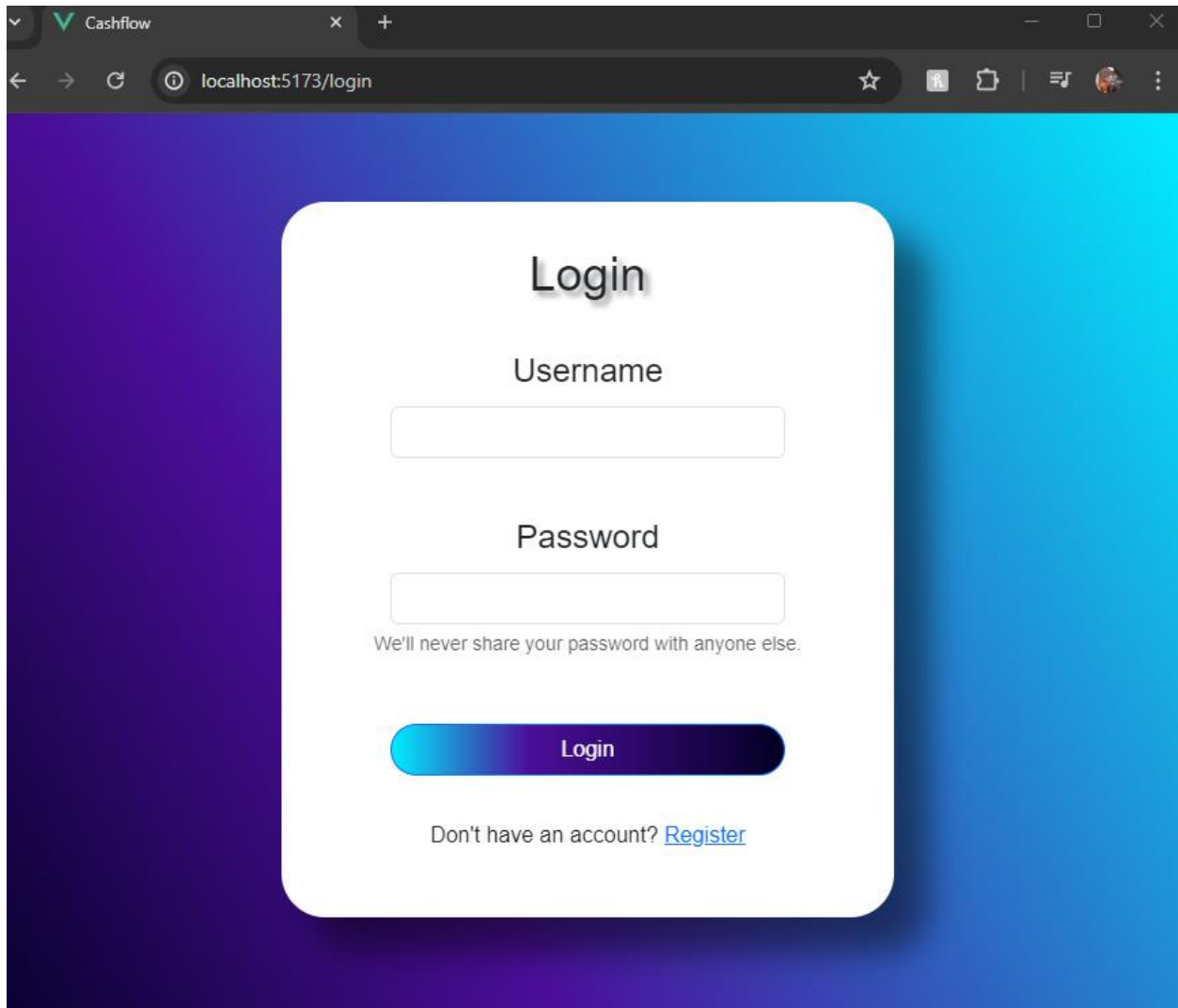
```
JS app.js  JS index.js  x  UserLogin.vue  App.vue  JS main.js
cashflow > src > router > JS index.js > ...
1  import { createRouter, createWebHistory } from 'vue-router'
2  import UserRegistration from '../components/UserRegistration.vue';
3  import UserLogin from '../components/UserLogin.vue';
4
5  const routes = [
6    { path: '/', redirect: '/dashboard' },
7    { path: '/register', component: UserRegistration },
8    { path: '/login', component: UserLogin }
9  ];
10
11  const router = createRouter({
12    history: createWebHistory(),
13    routes // short for `routes: routes`
14  });
15
16  router.beforeEach((to, from, next) => {
17    if (to.matched.some(record => record.meta.requiresAuth) && !localStorage.getItem('user-token')) {
18      next('/login');
19    } else {
20      next();
21    }
22  });
23
24  export default router
25
```

38. Configurare router link in App.vue

Precum pentru endpoint-ul de register, se va crea un router link în fișierul **App.vue** al aplicației client.

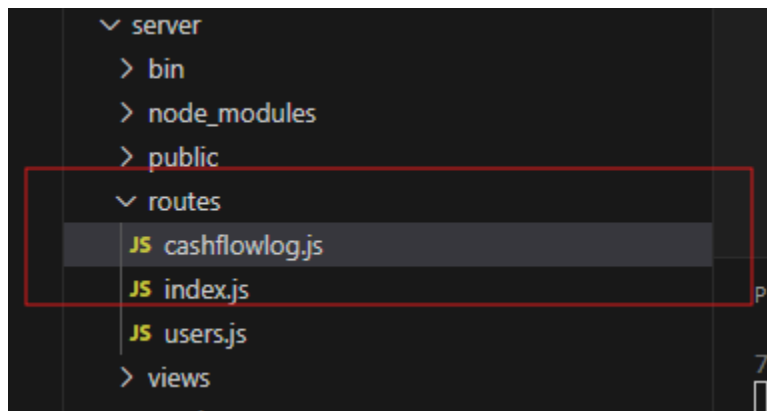
```
JS app.js  JS index.js  UserLogin.vue  App.vue  x  JS main.js
cashflow > src > App.vue > {} template > div#app > RouterView
1  <template>
2    <div id="app">
3      <nav>
4        <RouterLink to="/register"></RouterLink>
5        <RouterLink to="/login"></RouterLink>
6      </nav>
7      <RouterView />
8    </div>
9
10 </template>
11
12 <script>
13 export default {
14   name: 'App',
15 }
16 </script>
```

După configurările de mai sus, la accesarea URL-ului corespunzător, pagina de login va fi funcțională.



39. Creare endpoints pentru dashboard-ul aplicației

Vom crea în subdirectorul **routes**, din aplicația server, un fișier **cashflowlog.js**, care va fi folosit pentru definirea endpoint-urilor utilizate în gestionarea cashflow-ului.



40. Endpoint pentru get la log-uri

În fișierul nou creat, vom configura endpoint-ul de get:

```
1 var express = require('express');
2 var router = express.Router();
3 const jwt = require('jsonwebtoken');
4
5
6 router.get('/', function (req, res, next) {
7   const authHeader = req.headers.authorization;
8   if (!authHeader) {
9     res.status(401).json({ error: 'No authorization header' });
10    return;
11  }
12
13  const token = authHeader.split(' ')[1]; // get the token from the Authorization header
14  let userId;
15  try {
16    const decoded = jwt.verify(token, 'cashflow-key'); // verify the token
17    userId = decoded.id; // get the user ID from the decoded token
18  } catch (err) {
19    res.status(401).json({ error: 'Invalid token' });
20    return;
21  }
22  const query = `SELECT log.idcashFlowLog, log.idUserSelected, users.username, log.type, log.value, log.currency, DATE_FORMAT(log.date, '%Y-%m-%dT%T') as date FROM log
23  INNER JOIN users ON log.idUserSelected = users.idUsers
24  WHERE log.idUser = ? OR log.idUserSelected = ? ORDER BY log.date DESC`;
25  req.db.query(query, [userId, userId], (err, result) => {
26    if (err) {
27      res.status(500).json({ error: err.message });
28      return;
29    }
30    res.json(result);
31  });
32 });
33
```

Vom selecta intrările în cashflow care au fost create de user-ul logat (transfer de la user-ul logat către alt user), împreună cu cele în care user-ul logat este setat ca și destinatar (transfer de la alt user către user-ul logat).

Nota: Pentru fiecare endpoint din cashflow vom verifica JWT-ul pe care îl vom trimite într-un authorization header la apelarea acestuia.

41. (OPȚIONAL) Cod pentru documentația Swagger a metodei

Codul va fi scris respectând indentarea, deasupra metodei.

Se recomandă copy-paste-ul deasupra metodei get de pe link-ul următor:

<https://pastebin.com/t7aLhqg9>

42. Endpoint pentru insert la log-uri

În același fișier, vom configura endpoint-ul de insert:

```
51
52 router.post('/insertLog', function (req, res, next) {
53   const authHeader = req.headers.authorization;
54   if (!authHeader) {
55     res.status(401).json({ error: 'No authorization header' });
56     return;
57   }
58
59   const token = authHeader.split(' ')[1]; // get the token from the Authorization header
60   let userId;
61   try {
62     const decoded = jwt.verify(token, 'cashflow-key'); // verify the token
63     userId = decoded.id; // get the user ID from the decoded token
64   } catch (err) {
65     res.status(401).json({ success: false, error: 'Invalid token' });
66     return;
67   }
68   const { idUserSelected, type, value, currency, date } = req.body;
69   if (!idUserSelected || !type || !value || !currency /*|| !description*/ || !date) {
70     res.status(400).json({ success: false, error: 'Missing required fields' });
71     return;
72   }
73   if (value <= 0) {
74     res.status(400).json({ success: false, error: 'Value must be greater than 0' });
75     return;
76   }
77   //console.log(userId, " ", idEntity, " ", type, " ", value, " ", currency, " ", date);
78   if (type !== "Income" && type !== "Expense") {
79     res.status(400).json({ success: false, error: 'Invalid type' });
80     return;
81   }
82   if (currency !== "USD" && currency !== "EUR" && currency !== "RON") {
83     res.status(400).json({ success: false, error: 'Invalid currency' });
84     return;
85   }
86   const query = `INSERT INTO log (idUser, idUserSelected, type, value, currency, date) VALUES (?, ?, ?, ?, ?, ?)`;
87   req.db.query(query, [userId, idUserSelected, type, value, currency, date], (err, result) => {
88     if (err) {
89       res.status(500).json({ success: false, error: err.message });
90       return;
91     }
92     res.json({ success: true, message: 'Cashflow log inserted successfully' });
93   });
94 });
```

43. (OPȚIONAL) Cod pentru documentația Swagger a metodei

Codul va fi scris respectând indentarea deasupra metodei.

Se recomandă copy-paste-ul deasupra metodei insert de pe link-ul următor:

<https://pastebin.com/mXs2iN4d>

44. Endpoint pentru delete la log-uri

În același fișier, vom configura endpoint-ul de delete:

```
router.delete('/deleteLog/:idcashflowLog', function (req, res, next) {
  const authHeader = req.headers.authorization;
  if (!authHeader) {
    res.status(401).json({ error: 'No authorization header' });
    return;
  }

  const token = authHeader.split(' ')[1]; // get the token from the Authorization header
  let userId;
  try {
    const decoded = jwt.verify(token, 'cashflow-key'); // verify the token
    userId = decoded.id; // get the user ID from the decoded token
  } catch (err) {
    res.status(401).json({ error: 'Invalid token' });
    return;
  }

  const deleteQuery = 'DELETE FROM log WHERE idcashflowLog = ?';

  if (!req.params.idcashflowLog) {
    res.status(400).json({ error: 'The request has missing information!' });
    return;
  }

  req.db.query(deleteQuery, [req.params.idcashflowLog], (err, result) => {
    if (err) {
      res.status(500).json({ error: err.message });
      return;
    }

    if (result.affectedRows == 0) {
      res.status(400).json({ error: 'No cashflow log found with the provided id!' });
      return;
    }

    res.json({ message: 'Log deleted successfully!' });
  });
});
```

45. (OPȚIONAL) Cod pentru documentația Swagger a metodei

Codul va fi scris respectând indentarea deasupra metodei.

Se recomandă copy-paste-ul deasupra metodei delete de pe link-ul următor:

<https://pastebin.com/GnjnfD3Q>

46. Configurare endpoint de update pentru log-uri

În același fișier, vom configura endpoint-ul de update:

```
74
75 router.post('/updateLog/:idcashflowLog', function (req, res, next) {
76   const authHeader = req.headers.authorization;
77   if (!authHeader) {
78     res.status(401).json({ error: 'No authorization header' });
79     return;
80   }
81
82   if (!req.params.idcashflowLog) {
83     res.status(400).json({ error: 'The request has missing information!' });
84     return;
85   }
86
87   const token = authHeader.split(' ')[1]; // get the token from the Authorization header
88   let userId;
89   try {
90     const decoded = jwt.verify(token, 'cashflow-key'); // verify the token
91     userId = decoded.id; // get the user ID from the decoded token
92   } catch (err) {
93     res.status(401).json({ success: false, error: 'Invalid token' });
94     return;
95   }
96   const idcashflowLog = req.params.idcashflowLog,
97     idUserSelected = req.body.idUserSelected,
98     type = req.body.type,
99     value = req.body.value,
100     currency = req.body.currency,
101     date = req.body.date;
102   console.log(`${idcashflowLog} ${idUserSelected} ${type} ${value} ${currency} ${date}`);
103   if (!idcashflowLog || !idUserSelected || !type || !value || !currency || !date) {
104     res.status(400).json({ success: false, error: 'Missing required fields' });
105     return;
106   }
107   if (value <= 0) {
108     res.status(400).json({ success: false, error: 'Value must be greater than 0' });
109     return;
110   }
111   if (type !== "Income" && type !== "Expense") {
112     res.status(400).json({ success: false, error: 'Invalid type' });
113     return;
114   }
115   if (currency !== "USD" && currency !== "EUR" && currency !== "RON") {
116     res.status(400).json({ success: false, error: 'Invalid currency' });
117     return;
118   }
119   const query = `UPDATE log SET idUser = ?, idUserSelected = ?, type = ?, value = ?, currency = ?, date = ? WHERE idcashflowLog = ?`;
120   req.db.query(query, [userId, idUserSelected, type, value, currency, date, idcashflowLog], (err, result) => {
121     if (err) {
122       res.status(500).json({ success: false, error: err.message });
123       return;
124     }
125     res.json({ success: true, message: 'Cashflow log inserted successfully' });
126   });
127 });
128
```

47. (OPȚIONAL) Cod pentru documentația Swagger a metodei

Codul va fi scris respectând indentarea deasupra metodei.

Se recomandă copy-paste-ul deasupra metodei update de pe link-ul următor:

<https://pastebin.com/sS61Z4Ea>

48. Export la fișierul configurat

La finalul fișierului **cashflowlog.js**, vom adăuga următoarea linie de cod pentru a realiza exportul fișierului.

```
408     });  
409   });  
410  
411   module.exports = router;
```

49. Configurare router pentru log-urile cashflow in aplicația server

În fișierul **app.js** din aplicația server, vom adăuga următoarele linii pentru a putea avea acces la router-ul exportat.

```
var path = require('path');  
var cookieParser = require('cookie-parser');  
var logger = require('morgan');  
const cors = require('cors');  
  
var indexRouter = require('./routes/index');  
var usersRouter = require('./routes/users');  
var cashflowlogRouter = require('./routes/cashflowlog');  
  
var app = express();
```

```
110 app.use(express.static(path.join(__dirname, 'public')));  
111  
112 app.use('/', indexRouter);  
113 app.use('/users', usersRouter);  
114 app.use('/cashflowlog', cashflowlogRouter);  
115  
116 // catch 404 and forward to error handler  
117 app.use(function(req, res, next) {
```

Configurarea este identica cu cea de la router-ul pentru users.

După acest lucru, la un restart al aplicației server, interfață vă arata în felul următor:

Cashflow API

1.0.0 OAS 3.0[Authorize](#)

cashflowlog



GET

`/cashflowlog`

POST

`/cashflowlog/insertLog`

DELETE

`/cashflowlog/deleteLog/{idcashflowLog}`

POST

`/cashflowlog/updateLog/{idcashflowLog}`

users



GET

`/users`

GET

`/users/{name}`

POST

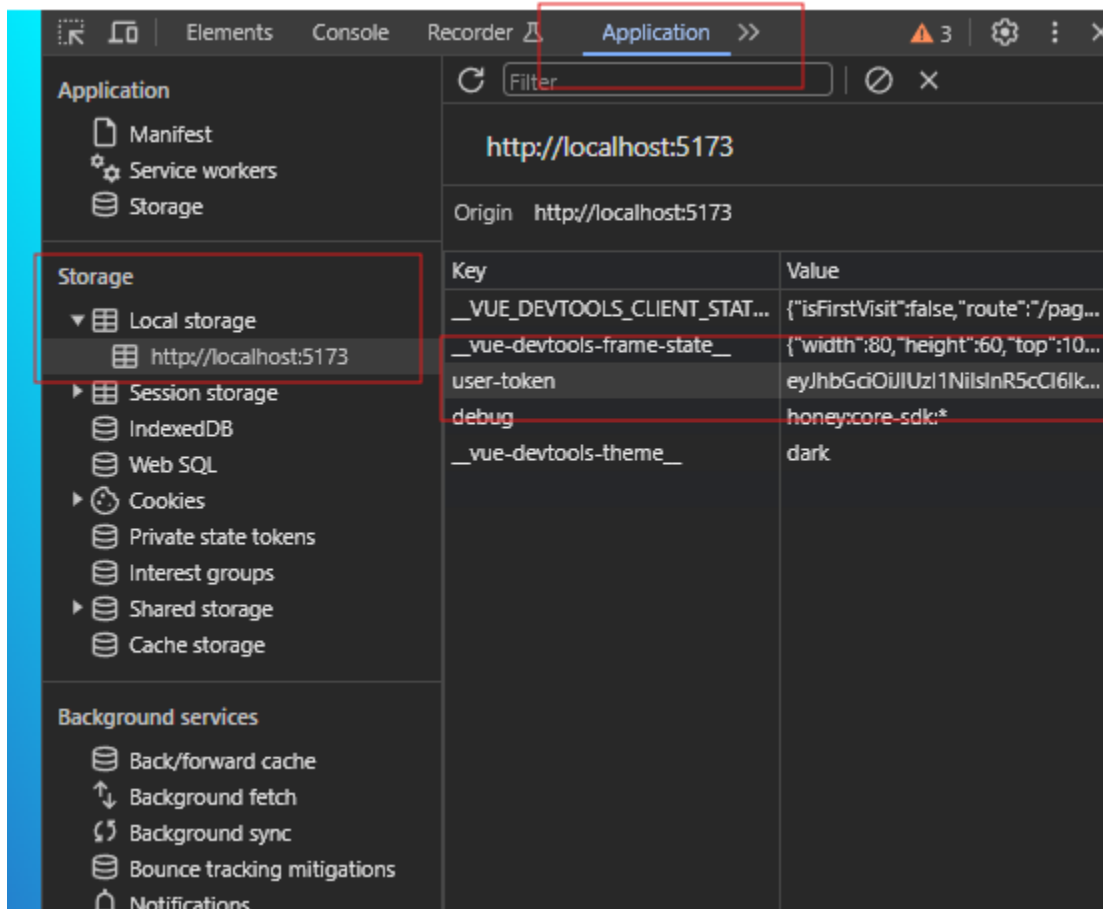
`/users/addUser`

POST

`/users/login`

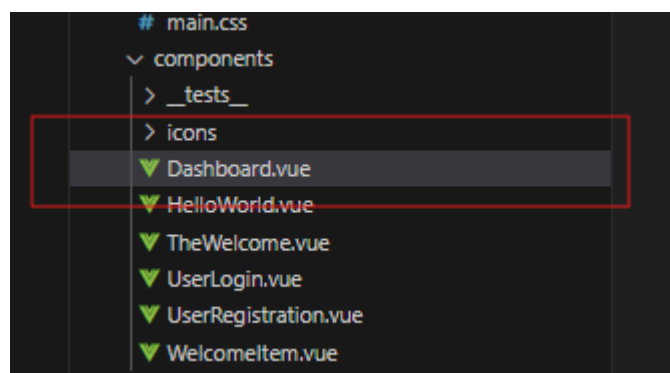
Pentru testarea noilor endpoint-uri create, va fi nevoie de un JWT valid. Acest lucru este implementat pentru limitarea accesului la endpoint-urile critice și a permite apelarea acestora doar de useri care au acces la acestea.

Pentru a obține un token valid, acesta se poate lua din local storage după o sesiune de login efectuată cu succes. Acesta se poate găsi la secțiunea **application, local storage**, după accesarea hot key-ului **CTRL+Shift+I** în browser.



50. Creare pagina Dashboard

Precum paginile de register și login, creați în aplicația client, în subdirectorul **components**, un fișier numit **Dashboard.vue**.



51. Template pentru Dashboard.vue

Configurați template-ul pentru fisierul creat. Prima parte a template-ului va fi formata din elemente html care utilizeaza bootstrap, mai exact elementul accordion. In acest mod fiecare intrare in cashflow va putea avea un dropdown, unde va putea fi editata.

```
<template>
<div id="cashflow-log">
  <h1 class="text-center cashflowlog-heading">Cashflow</h1>
  <div class="accordion" id="accordionCashflow">
    <div v-for="(log, index) in cashflowLog" :key="log.idcashflowLog" class="accordion-item cashflow-element"
      :style="{ animationDelay: index / 4 + 's' }">

      <!-- ACCORDION INFO -->
      <h2 class="accordion-header" id="headingOne" @click="openLog(log)">
        <button class="accordion-button collapsed" type="button" data-bs-toggle="collapse"
          :data-bs-target="#collapse" + log.idcashflowLog" aria-expanded="false"
          :aria-controls="'collapse' + log.idcashflowLog">
          Transaction ID: {{ log.idcashflowLog }} &nbsp;<b
            :class="{ 'text-danger': log.type == 'Expense', 'text-success': log.type == 'Income' }">{{
              log.type }}</b>
          &nbsp;  
          Name: {{ log.username }} Value: {{ log.value }} Currency: {{ log.currency }}
          Date: {{ log.date }}
        </button>
      </h2>

      <!-- ACCORDION BODY COLLAPSED -->
      <div :id="'collapse' + log.idcashflowLog" class="accordion-collapse collapse"
        :aria-labelledby="heading" + log.idcashflowLog" data-bs-parent="#accordionCashflow">
        <div class="accordion-body">

          <!-- USER -->
          <div class="input-group mb-3">
            <div class="input-group-prepend">
              <label class="input-group-text" :for="'username' + log.idcashflowLog">Name</label>
            </div>
            <select :id="'username' + log.idcashflowLog" class="name-select form-control"
              aria-label="Name" aria-describedby="inputGroup-sizing-default"
              v-model="log.idUserSelected" @change="inputChanging()">
              <option v-for="user in users" :key="user.idUsers" :value="user.idUsers">
                {{
                  user.username }}</option>
            </select>
          </div>

          <!-- TYPE -->
          <div class="input-group mb-3">
            <div class="input-group-prepend">
              <label class="input-group-text" :for="'typeName' + log.idcashflowLog">Type</label>
            </div>
            <select :id="'typeName' + log.idcashflowLog" class="type-select form-control"
              aria-label="Type" aria-describedby="inputGroup-sizing-default" v-model="log.type"
              @change="inputChanging()">
              <option value="Income">Income</option>
              <option value="Expense">Expense</option>
            </select>
          </div>

          <!-- VALUE -->
          <div class="input-group mb-3">
            <div class="input-group-prepend">
              <span class="input-group-text" id="inputGroup-sizing-default">Value</span>
            </div>
            <input :id="'value' + log.idcashflowLog" type="number" class="value-input form-control"
              aria-label="Value" aria-describedby="inputGroup-sizing-default" v-model="log.value"
              @change="inputChanging()">
          </div>

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

După cum se poate observa, se vor defini variabilele care sunt ținute minte în baza de date, toate fiind incluse într-un container **div** care utilizează elementul **v-for**

(specific Vue.js) pentru a crea un număr de **N** elemente de acest tip. De asemenea elementele au un **Id unic**, adaugandu-se la Id-ul fiecărui element, Id-ul stocat în log-urile cashflow-ului.

Continuare:

```
3
4
5      <!-- CURRENCY -->
6      <div class="input-group mb-3">
7        <div class="input-group-prepend">
8          <label class="input-group-text"
9            :for="'currencyName' + log.idcashFlowLog">Currency</label>
10        </div>
11        <select :id="'currencyName' + log.idcashFlowLog" class="currency-select form-control"
12          aria-label="Currency" aria-describedby="inputGroup-sizing-default"
13          v-model="log.currency" @change="inputChanging()">
14          <option value="RON">RON</option>
15          <option value="EUR">EUR</option>
16          <option value="USD">USD</option>
17        </select>
18      </div>
19
20      <!-- DATE -->
21      <div class="input-group mb-3">
22        <div class="input-group-prepend">
23          <span class="input-group-text" id="inputGroup-sizing-default">Date</span>
24        </div>
25        <input :id="'date' + log.idcashFlowLog" type="datetime-local"
26          class="date-input form-control" aria-label="Value"
27          aria-describedby="inputGroup-sizing-default" v-model="log.date"
28          @change="inputChanging()">
29      </div>
30
31      <!-- DELETE BUTTON -->
32      <div class="delete-button-container">
33        <button v-if="!showUpdateButton" class="btn-delete">
34          <span class="delete-message">CONFIRM DELETE</span>
35          <svg xmlns="http://www.w3.org/2000/svg" width="25" height="25" fill="none"
36            viewBox="0 0 24 24" stroke="currentColor" stroke-width="2"
37            @click="toggleDeleteConfirmation($event, log)">
38            <path stroke-linecap="round" stroke-linejoin="round"
39              d="M19 71-.867 12.142A2 2 0 0116.138 21H7.862a2 2 0 01-1.995-1.858L5 74v6m4-6v6m1-10V4a1 1 0 00-1-1h3M4 7h16" />
40          </svg>
41        </button>
42      </div>
43
44      <!-- UPDATE BUTTON -->
45      <transition name="fade">
46        <button v-if="showUpdateButton" @click="updateCashFlowLog(log)"
47          class="btn btn-primary">Update</button>
48      </transition>
49    </div>
50  </div>
51
52  <button type="button" class="btn-insert" data-bs-toggle="modal" data-bs-target="#insertModal"
53    @click="resetInsertInfo()">Insert</button>
54
```

Aceasta a fost partea de display principala a informațiilor care vor veni din baza de date. Pentru insert, vom alege un alt element specific bootstrap, mai exact modal popup. La apăsarea butonului de insert, vom deschide un tab de tip modal popup unde se vor putea adauga informații care se vor insera în database-ul MySQL creat. Structura modal popup-ului va fi foarte asemănătoare cu codul scris anterior, acesta fiind efectiv formată din aceleași elemente replicate sub altă formă.

Structură modal popup:

```
15 <!-- MODAL POPUP -->
16 <div class="modal fade" id="insertModal" tabindex="-1" aria-labelledby="InsertPopup" aria-hidden="true">
17   <div class="modal-dialog modal-dialog-centered">
18     <div class="modal-content">
19       <div class="modal-header">
20         <h5 class="modal-title" id="InsertPopup">Insert log</h5>
21         <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>
22       </div>
23       <div class="modal-body">
24
25         <!-- USER INSERT -->
26         <div class="input-group mb-3">
27           <div class="input-group-prepend">
28             <label class="input-group-text" for="usernameInsert">Name</label>
29           </div>
30           <select id="usernameInsert" class="name-select form-control" aria-label="Name"
31             aria-describedby="inputGroup-sizing-default" v-model="idUserInsert">
32             <option v-for="user in users" :key="user.idUsers" :value="user.idUsers">
33               {{
34                 user.username }}</option>
35             </select>
36         </div>
37
38         <!-- TYPE INSERT -->
39         <div class="input-group mb-3">
40           <div class="input-group-prepend">
41             <label class="input-group-text" for="typeNameInsert">Type</label>
42           </div>
43           <select id="typeNameInsert" class="type-select form-control" aria-label="Type"
44             aria-describedby="inputGroup-sizing-default" v-model="typeInsert">
45             <option value="Income">Income</option>
46             <option value="Expense">Expense</option>
47           </select>
48         </div>
49
50         <!-- VALUE INSERT -->
51         <div class="input-group mb-3">
52           <div class="input-group-prepend">
53             <span class="input-group-text" id="inputGroup-sizing-default">Value</span>
54           </div>
55           <input id="valueInsert" type="number" class="value-input form-control" aria-label="Value"
56             aria-describedby="inputGroup-sizing-default" v-model="valueInsert">
57         </div>
58
59         <!-- CURRENCY INSERT -->
60         <div class="input-group mb-3">
61           <div class="input-group-prepend">
62             <label class="input-group-text" for="currencyNameInsert">Currency</label>
63           </div>
64           <select id="currencyNameInsert" class="currency-select form-control" aria-label="Currency"
65             aria-describedby="inputGroup-sizing-default" v-model="currencyInsert">
66             <option value="RON">RON</option>
67             <option value="EUR">EUR</option>
68             <option value="USD">USD</option>
69           </select>
70         </div>
```

Continuare modal popup, alături de buton logout și toast pentru notificări:

```
<!-- DATE INSERT -->
<div class="input-group mb-3">
  <div class="input-group-prepend">
    <span class="input-group-text" id="inputGroup-sizing-default">Date</span>
  </div>
  <input type="datetime-local" class="date-input form-control" aria-label="Value"
    aria-describedby="inputGroup-sizing-default" v-model="dateInsert">
  </div>
</div>

<!-- BUTTONS FOR MODAL -->
<div class="modal-footer">
  <button type="button" class="btn-cashflow-close" data-bs-dismiss="modal">Close</button>
  <button type="button" class="btn-save" @click="insertCashflowLog()">Save changes</button>
</div>
</div>
</div>
<button @click="logout()" class="btn-logout"> <i class="fa-solid fa-right-from-bracket"></i> Logout</button>

<div class="toast" role="alert" aria-live="assertive" aria-atomic="true" data-bs-autohide="true"
  :class="{ 'show': showToast }" style="position: absolute; top: 0; right: 0;">
  <div class="toast-header">
    <strong class="me-auto">Notification</strong>
    <button type="button" class="m1-2 mb-1 btn-close fade-in" @click="showToast = false"></button>
  </div>
  <div class="toast-body">
    {{ toastMessage }}
  </div>
</div>
</template>
```

Cod sursă template dashboard: <https://pastebin.com/WvnVZ8Sg>

52. Script-ul pentru pagina dashboard

Acum vom continua cu scrierea blocului script din Dashboard.vue:

```
<script>
import axios from 'axios';

export default {
  name: 'CashFlowLog',
  data() {
    return {
      cashflowLog: [],
      users: [],

      // toast
      showToast: false,
      toastMessage: '',
      showUpdateButton: false,

      // insert log
      idUserInsert: 0,
      typeInsert: '',
      valueInsert: 0,
      currencyInsert: '',
      dateInsert: '',

      // current log
      currentId: 0,
      currentType: '',
      currentValue: 0,
      currentCurrency: '',
      currentDate: '',

      showDeleteConfirmation: false,
    }
  },

  created() {
    this.getUsers();
    this.getCashflow();
  },
  methods: {
    logout() {
      localStorage.removeItem('user-token');
      this.$router.push('/login');
    },
  },
}
```

Vom defini datele necesare din pagina si metodele apelate cand pagina se creeaza, pe care le vom define in continuare, alaturi de metoda de logout.

Continuare definire metode:

```
getCashflow() {
  const token = localStorage.getItem('user-token'); // get the token from local storage
  axios.get('http://localhost:3000/cashflowlog', {
    headers: {
      Authorization: `Bearer ${token}` // send the token in the Authorization header
    }
  })
    .then(response => {
      this.cashflowLog = response.data;
    })
    .catch(error => {
      localStorage.removeItem('user-token');
      this.toastMessage = 'Invalid login: ' + error.response.data.message;
      this.showToast = true;

      setTimeout(() => {
        this.showToast = false;
      }, 5000);
      this.$router.push('/login');
      return;
    });
},

getUsers() {
  axios.get('http://localhost:3000/users')
    .then(response => {
      this.users = response.data;
    })
    .catch(error => {
      console.error(error);
    });
},

inputChanging() {
  this.showUpdateButton = true;
},

resetInsertInfo() {
  this.idUserInsert = 0;
  this.typeInsert = '';
  this.valueInsert = 0;
  this.currencyInsert = '';
  this.dateInsert = '';
},
```

Se poate observa cum metoda de get pentru users și cashflow, care se apelează la crearea paginii, apelează endpoint-urile definite anterior din API, iar logica necesară se realizează în backend. Dacă token-ul nu e validat, user-ul e trimis pe pagina de **login**.

Metodă pentru insert log:

```
insertCashflowLog() {
  console.log(this.idUserInsert, this.typeInsert, this.valueInsert, this.currencyInsert, this.dateInsert);
  const token = localStorage.getItem('user-token'); // get the token from local storage
  if (this.idUserInsert == 0 || this.typeInsert == '' || this.valueInsert == 0 || this.currencyInsert == '' || this.dateInsert == '') {
    this.showToast = true;
    this.toastMessage = 'Please fill all fields';
    setTimeout(() => {
      this.showToast = false;
    }, 5000);
  } else {
    axios.post("http://localhost:3000/cashflowlog/insertLog", {
      idUserSelected: this.idUserInsert,
      type: this.typeInsert,
      value: this.valueInsert,
      currency: this.currencyInsert,
      date: this.dateInsert
    }, {
      headers: {
        Authorization: `Bearer ${token}` // send the token in the Authorization header
      }
    })
    .then(response => {
      console.log(response.data);
      if (response.data.success) {
        this.showToast = true;
        this.toastMessage = 'Log inserted successfully';
        setTimeout(() => {
          this.showToast = false;
        }, 5000);

        this.getCashflow();
      }
    })
    .catch(error => {
      this.showToast = true;
      this.toastMessage = 'Error inserting log: ' + error.response.data.error;
      setTimeout(() => {
        this.showToast = false;
      }, 5000);
    });
  }
},
```

În continuare, vom defini metoda pentru update a unui log.

```

updateCashflowLog(log) {
  const token = localStorage.getItem('user-token'); // get the token from local storage
  axios.post(`http://localhost:3000/cashflowlog/updateLog/${log.idcashflowLog}`, {
    idUserSelected: log.idUserSelected,
    type: log.type,
    value: log.value,
    currency: log.currency,
    date: log.date
  },
  {
    headers: {
      Authorization: `Bearer ${token}` // send the token in the Authorization header
    }
  })
  .then(response => {
    if (response.data.success) {
      this.showToast = true;
      this.toastMessage = 'Log updated successfully';

      // update the current log values to the new ones so it doesn't reset when closing the log
      this.currentType = log.type;
      this.currentValue = log.value;
      this.currentCurrency = log.currency;
      this.currentDate = log.date;
      setTimeout(() => {
        this.showToast = false;
      }, 5000);
      this.getCashflow();
    }
  })
  .catch(error => {
    this.showToast = true;
    this.toastMessage = 'Error updating log: ' + error.response.data.error;
    setTimeout(() => {
      this.showToast = false;
    }, 5000);
  });
  this.showUpdateButton = false;
},

```

La această metodă, se va pasa și Id-ul log-ului, pentru a putea fi realizate operațiunile pe intrarea corectă.

În continuare vom defini metodele auxiliare, pentru deschiderea log-ului, starea butonului de delete și pentru ștergerea unui log. Butonul de ștergere log vă avea doua stări și va avea nevoie de două click-uri consecutive pentru ștergerea unei intrări, de aceea vom avea nevoie de o funcție separate pentru resetarea stării butonului dacă cumva se vă selecta o altă intrare sau se vă începe operațiunea de update.

```

openLog(log) {
  if (this.currentId == 0) {
    this.currentId = log.idcashflowLog;
    this.currentType = log.type;
    this.currentValue = log.value;
    this.currentCurrency = log.currency;
    this.currentDate = log.date;
    //console.log('currentId: ${this.currentId} currentType: ${this.currentType} currentValue: ${this.currentValue} currentCurrency: ${this.currentCurrency} currentDate: ${this.currentDate}')
  } else {
    if (this.currentId == log.idcashflowLog) {
      //reset the log in the array
      log.type = this.currentType;
      log.value = this.currentValue;
      log.currency = this.currentCurrency;
      log.date = this.currentDate;

      // reset the current log values
      this.currentId = 0;
      this.currentType = '';
      this.currentValue = 0;
      this.currentCurrency = '';
      this.currentDate = '';
    } else {
      this.currentId = log.idcashflowLog;
      this.currentType = log.type;
      this.currentValue = log.value;
      this.currentCurrency = log.currency;
      this.currentDate = log.date;
    }
  }

  this.showUpdateButton = false;
  this.showDeleteConfirmation = false;
},
toggleDeleteConfirmation(event, log) {
  if (!this.showDeleteConfirmation) {
    event.stopPropagation();
    this.showDeleteConfirmation = !this.showDeleteConfirmation;
    return;
  } else {
    this.deleteLog(log);
  }
  this.showDeleteConfirmation = false;
},
},

```

Metoda de delete și finalizarea blocului script:

```

17   deleteLog(log) {
18     if (this.showDeleteConfirmation) {
19       const token = localStorage.getItem('user-token'); // get the token from local storage
20       axios.delete(`http://localhost:3000/cashflowlog/deleteLog/${log.idcashflowLog}`, {
21         headers: {
22           Authorization: `Bearer ${token}` // send the token in the Authorization header
23         }
24       })
25       .then(response => {
26         if (response.data.message) {
27           this.showToast = true;
28           this.toastMessage = 'Log deleted successfully';
29           setTimeout(() => {
30             this.showToast = false;
31           }, 5000);
32           this.getCashflow();
33         }
34       })
35       .catch(error => {
36         this.showToast = true;
37         this.toastMessage = error.response.data.error;
38         setTimeout(() => {
39           this.showToast = false;
40         }, 5000);
41       });
42     }
43   }
44 }
45 }
46 </script>

```

Cod sursă bloc script: <https://pastebin.com/WvnVZ8Sg>

53. Configurare route pentru pagina de login

În fișierul **index.js** al aplicației client, se va crea route-ul pentru pagina creată anterior:

```
1 import { createRouter, createWebHistory } from 'vue-router'
2 import UserRegistration from '../components/UserRegistration.vue';
3 import UserLogin from '../components/UserLogin.vue';
4 import Dashboard from '../components/Dashboard.vue';
5
6 const routes = [
7   { path: '/', redirect: '/dashboard' },
8   { path: '/register', component: UserRegistration },
9   { path: '/login', component: UserLogin },
10  { path: '/dashboard', component: Dashboard, meta: { requiresAuth: true } }
11 ];
12
13 const router = createRouter({
14   history: createWebHistory(),
15   routes // short for `routes: routes`
16 });
17
18 router.beforeEach((to, from, next) => {
19   if (to.matched.some(record => record.meta.requiresAuth) && !localStorage.getItem('user-token')) {
20     next('/login');
21   } else {
22     next();
23   }
24 });
25
26 export default router
```

Se va impune și necesitatea autentificării, iar dacă token-ul lipsește, user-ul va fi trimis pe pagina de login la accesarea endpoint-ului **/dashboard**.

54. Configurare router link in App.vue

Precum pentru endpoint-urile anterioare, se va crea un router link în fișierul **App.vue** al aplicației client.

```
<template>
  <div id="app">
    <nav>
      <RouterLink to="/register"></RouterLink>
      <RouterLink to="/login"></RouterLink>
      <RouterLink to="/dashboard"></RouterLink>
    </nav>
    <RouterView />
  </div>
</template>

<script>
export default {
  name: 'App',
}
</script>
```


După terminarea acestui pas, aplicația este finalizata și gata de testat în întregime.

Notă: Aplicația este doar un proiect simplist pentru demonstrarea funcționalității și creării de proiecte Vue.js.

Referință github:

<https://github.com/BogdanBargaoanu/Vue.js-Introduction>

Redactor: Bârgăoanu Bogdan-Alexandru



UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA