Aplicație BlackJack

Întocmit de: **Astaloș Rareș**
Pentru disciplina: **Proiectarea Aplicaților Web**

Data predării: **12//01//2025**

Cuprins

Capitolul 1. Introducere în Aplicația de Blackjack

Aplicația de **Blackjack** pe care am dezvoltat-o are ca scop oferirea unei experiențe de joc de cazino complet automatizată, în care un utilizator poate juca împotriva unui dealer controlat de algoritmi. Implementarea se bazează pe tehnologia **Spring Boot** pentru partea de backend și **React** pentru frontend, folosind standarde moderne de securitate și comunicare în timp real pentru a asigura o experiență fluidă și interactivă.

Titlul capitolului se bazează pe Heading 1 style, numerotat cu o cifra (x. Nume capitol), font Times New Roman de 14, Bold.

Ce se scrie aici:

- Contextul
- Conturarea domeniului exact al temei
- reprezintă cca. 5% din lucrare

1.1. Scopul Aplicației

Scopul principal al aplicației este de a simula un joc de **Blackjack** în care utilizatorul se poate autentifica, poate interacționa cu dealerul controlat de un algoritm și poate juca conform regulilor clasice ale jocului. Aplicația permite utilizatorilor să:

- Se autentifice în siguranță cu ajutorul unui sistem de **autentificare pe bază de token JWT**.
- Joace **Blackjack** împotriva unui dealer, care urmează regulile jocului standard (împărțirea cărților, trageri până la 17 sau mai mult pentru dealer, și stabilirea câștigătorului).
- Beneficieze de un **backend securizat** cu Spring Boot și de un **frontend interactiv** cu React.
- Folosească WebSocket pentru a permite comunicarea în timp real între client și server, facilitând actualizarea instantanee a stării jocului pe măsură ce utilizatorul joacă.

1.2. Arhitectura Aplicației

Backend (Spring Boot):

- Serviciile de backend sunt construite pe framework-ul **Spring Boot** și sunt responsabile pentru gestionarea autentificării utilizatorilor, logica de joc (inclusiv regulile Blackjack-ului), precum și pentru expunerea unui API RESTful pentru a comunica cu frontend-ul.
- Securitatea aplicației este asigurată prin **Spring Security** și **JWT**, care protejează rutele sensibile și asigură că doar utilizatorii autentificați pot accesa funcțiile jocului.
- **WebSocket** este folosit pentru a permite mesaje în timp real între client și server, oferind o experiență de joc dinamică.

Frontend (React):

- Partea de frontend este construită folosind **React**, o bibliotecă JavaScript puternică pentru construirea de interfețe utilizator dinamice.
- Frontend-ul interacționează cu backend-ul prin intermediul **Axios** pentru a face apeluri la API-ul de autentificare și pentru a manipula datele jocului.
- Utilizatorii se pot conecta la aplicație, pot vizualiza cărțile și scorul lor, pot lua decizii de joc (hit sau stand) și pot vedea în timp real rezultatul jocului

1.3. Funcționalități Cheie

Autentificare și Securitate:

- Utilizatorii trebuie să se autentifice cu un nume de utilizator și o parolă.
- Autentificarea este gestionată de **JWT** (JSON Web Tokens), asigurând astfel sesiuni sigure pentru utilizatori.
- Odată ce utilizatorul se autentifică, un token JWT este stocat în browser (localStorage), permițând accesul continuu la aplicație fără a fi necesar să se logheze de fiecare dată.

Jocul de Blackjack:

- Utilizatorii pot începe un joc nou, primind două cărți, iar dealerul primește, de asemenea, două cărți, dintre care una este ascunsă.
- Utilizatorii pot alege să **hit** (să primească o altă carte) sau **stand** (să rămână cu cărțile actuale).
- Dealerul joacă conform regulilor standard: va continua să tragă cărți până când ajunge la un scor de 17 sau mai mult.
- Câștigătorul este determinat pe baza scorului, iar dacă un jucător depășește 21, acesta pierde automat.

Comunicare în Timp Real:

- Aplicația folosește **WebSocket** pentru a permite actualizarea stării jocului în timp real între server și client.
- Acest lucru este esențial pentru experiența de joc, deoarece fiecare mișcare a jucătorului și acțiunile dealerului trebuie reflectate instantaneu în interfața utilizatorului.

1.4. Avantaje

- **Securitate**: Utilizarea token-urilor JWT asigură că doar utilizatorii autentificați pot accesa jocul. În plus, datele sensibile sunt protejate și autentificarea se face într-un mod scalabil și sigur.
- **Interactivitate**: Folosind React și WebSocket, utilizatorii beneficiază de o interfață dinamică și interactivă, iar rezultatele jocului sunt actualizate în timp real.
- **Ușurință în utilizare**: Interfața este simplă și intuitivă, permițând utilizatorilor să joace Blackjack fără a întâmpina dificultăți tehnice.

1.5. Obiectivele Viitoare

- Implementarea unui sistem de **istoric al jocurilor**, unde utilizatorii pot vizualiza rezultatele anterioare ale jocurilor.
- Posibilitatea de a juca **multiplayer**, unde mai mulți jucători pot participa simultan într-un joc.
- Adăugarea unui **mod de joc cu inteligență artificială**, în care utilizatorul joacă contra unui dealer controlat de un AI mai sofisticat.

Capitolul 2. Arhitectura Aplicației

Arhitectura aplicației de Blackjack este bazată pe o abordare tipică de aplicație full-stack, având un backend realizat în **Spring Boot** și un frontend în **React**. Aplicația utilizează tehnologiile moderne pentru a oferi o experiență de joc interactivă și în timp real. În această secțiune vom detalia structura arhitecturală a aplicației, interacțiunea dintre componente și modul în care acestea se leagă între ele.

2.1. Descriere Generală a Arhitecturii

Aplicația este construită pe o arhitectură client-server, unde backend-ul și frontend-ul sunt separate și interacționează prin API-uri REST și WebSocket. Aceasta permite extinderea și întreținerea fiecărei componente independent, făcând aplicația mai flexibilă și mai ușor de scalat.

Componentele principale ale aplicației sunt:

- **Backend**:
  - **Spring Boot**: Framework-ul principal utilizat pentru dezvoltarea backend-ului. Acesta gestionează logica de business, autentificarea utilizatorilor, manipularea datelor și comunicarea cu frontend-ul.
  - **WebSocket**: Utilizat pentru a permite comunicarea în timp real între server și client, permițând actualizarea instantanee a stării jocului.
  - **MySQL**: Baza de date pentru stocarea informațiilor despre utilizatori și istoricul jocurilor.
  - **Spring Security**: Folosit pentru a securiza API-urile și pentru autentificarea utilizatorilor folosind JWT.
- **Frontend**:
  - **React**: Framework-ul principal utilizat pentru dezvoltarea interfeței utilizatorului. Acesta oferă o experiență dinamică și reactivă pentru utilizator, actualizând automat UI-ul atunci când starea jocului se schimbă.
  - **Axios**: Folosit pentru a efectua cereri HTTP către backend, pentru autentificare și manipularea datelor.
  - **WebSocket (client-side)**: Utilizat pentru a gestiona conexiunile WebSocket pe partea clientului și pentru a primi mesaje de actualizare a stării jocului în timp real.

2.2. Cum Interacționează Componentele Aplicației

- **Frontend-ul (React)** trimite cereri de autentificare către backend-ul **Spring Boot** printr-un API REST. După ce utilizatorul se autentifică cu succes, backend-ul returnează un token JWT care este stocat în **localStorage** pe client. Acest token este utilizat pentru a autentifica și autoriza cererile ulterioare.

- Când utilizatorul pornește jocul, **React** inițiază o sesiune de joc, interacționând cu backend-ul pentru a crea un nou joc. Backend-ul gestionează logica jocului (distribuirea cărților, calculează scorurile) și transmite actualizările stării jocului prin **WebSocket** către client.

- **WebSocket** este folosit pentru a trimite mesaje în timp real între server și client, astfel încât starea jocului (mâinile jucătorului, mâinile dealerului, scorurile și rezultatele) să fie actualizată instantaneu pe toate instanțele clientului conectate. Acesta permite un flux constant de informații pentru a face experiența de joc interactivă și fără întârzieri semnificative.

2.3. Fluxul de date în Aplicație

Autentificare:

- Utilizatorul furnizează un nume de utilizator și o parolă.
- React trimite o cerere POST către API-ul de login al backend-ului.
- Backend-ul validează datele și returnează un token JWT.
- Frontend-ul salvează token-ul în **localStorage** și îl folosește pentru autentificarea ulterioarelor cereri.

Inițierea Jocului:

- Când utilizatorul apasă butonul „Start Game”, React trimite o cerere către backend pentru a începe un nou joc.
- Backend-ul creează un nou joc și returnează un set de date (mâinile jucătorului și ale dealerului, deck-ul de cărți etc.).
- Frontend-ul utilizează WebSocket pentru a primi mesaje de la server cu actualizările stării jocului (cărtile distribuite, scorurile).

Comunicarea în Timp Real:

- În timpul jocului, fiecare acțiune (hit, stand) din partea jucătorului trimite mesaje către server prin WebSocket.
- Serverul procesează aceste acțiuni și trimite actualizări către toate instanțele de client conectate.
- WebSocket permite actualizarea în timp real a stării jocului pe frontend, fără a fi necesare refresh-uri ale paginii.

Încheierea Jocului și Rezultatul:

- Când jocul se încheie (jucătorul ajunge la scorul maxim sau își depășește scorul), serverul trimite rezultatul jocului către client prin WebSocket.
- Frontend-ul afișează mesajul corespunzător (în funcție de câștigător).

2.4. Scalabilitate și Extensibilitate

Arhitectura aplicației permite scalabilitate și extensibilitate:

- **Scalabilitate**: Backend-ul poate fi scalat prin implementarea de microservicii (dacă este necesar) pentru a separa logica jocului de gestionarea utilizatorilor sau a datelor. WebSocket permite gestionarea în timp real a mai multor jocuri simultan.
- **Extensibilitate**: Aplicația poate fi extinsă ușor cu funcționalități suplimentare, cum ar fi integrarea unui AI pentru jucători, adăugarea unui sistem de puncte sau leaderboard, sau implementarea unui sistem multiplayer.

Capitolul 3. Funcționalități Cheie

Aplicația de Blackjack oferă o serie de funcționalități esențiale care permit utilizatorilor să joace în mod interactiv și să se conecteze cu alți jucători, având o experiență fluidă și captivantă. În acest capitol, vom detalia principalele funcționalități ale aplicației, explicând fiecare caracteristică în parte și modul în care aceasta contribuie la experiența utilizatorului.

3.1. Autentificare și Autorizare

Una dintre funcționalitățile fundamentale ale aplicației este sistemul de autentificare și autorizare a utilizatorilor. Aceasta permite accesul securizat la aplicație și protejează datele utilizatorilor.

Autentificare: Utilizatorul se poate conecta la aplicație folosind un nume de utilizator și o parolă. Aplicația folosește un sistem de autentificare bazat pe **JSON Web Tokens (JWT)**, care oferă un mod sigur de a verifica identitatea utilizatorului fără a stoca sesiuni pe server.

- **Fluxul autentificării**:
  - Utilizatorul introduce numele de utilizator și parola în formularul de login.
  - Backend-ul validează credențialele și generează un token JWT.
  - Token-ul este salvat în **localStorage** pe client și utilizat pentru autentificarea cererilor ulterioare.

Autorizare: După autentificare, utilizatorul are acces la jocuri, iar permisiunile acestuia sunt gestionate pe baza rolului asociat contului (de exemplu, jucător, admin).

3.2. Crearea și Gestionarea Jocului

Aplicația permite crearea unui joc de Blackjack, care poate fi jucat între jucător și dealer. Jocul respectă regulile standard ale Blackjack-ului, iar utilizatorul poate lua decizii în cadrul fiecărei runde.

- **Crearea jocului**: După autentificare, utilizatorul poate iniția un joc printr-o cerere către backend. Serverul va crea un joc nou, va distribui cărțile și va stabili mâna inițială a jucătorului și dealerului.
- **Gestionarea jocului**:

- **Mâinile jucătorului și ale dealerului**: Fiecare joc include două mâini: una pentru jucător și alta pentru dealer. Jucătorul primește două cărți și decide dacă vrea să tragă o altă carte (hit) sau să rămână cu cele deja primite (stand).
- **Logica jocului**: Jocul respectă regulile Blackjack-ului, unde dealerul trebuie să continue să tragă cărți până când ajunge la un total de 17 sau mai mult, iar jucătorul poate alege să își oprească acțiunile la orice moment.

3.3. Interacțiuni în Timp Real (WebSocket)

Un aspect important al aplicației este interacțiunea în timp real între jucător și dealer. Acest lucru este posibil datorită implementării **WebSocket** pe server și client, care permite comunicarea bidirecțională instantanee.

- **Comunicarea în timp real**: După ce jocul este început, serverul și clientul comunică prin WebSocket pentru a actualiza starea jocului în timp real. Astfel, când jucătorul face o acțiune (hit, stand), serverul trimite un mesaj către client, care actualizează interfața utilizatorului fără a fi necesar un refresh al paginii.
  - **Exemple de interacțiuni**:
    - Jucătorul apasă pe butonul „Hit” sau „Stand”, iar aplicația trimite un mesaj prin WebSocket către server.
    - Serverul procesează acțiunea și trimite o actualizare către toate instanțele de client, indicând noul status al jocului (cărtile distribuite, scorul curent, etc.).
- **Actualizări ale stării jocului**: Atunci când dealerul joacă sau când jocul se încheie, WebSocket permite actualizarea în timp real a statusului jocului pentru toți participanții.

3.4. Sistem de Punctaj și Rezultate

La finalul fiecărei runde, aplicația calculează scorul jucătorului și al dealerului conform regulilor standard ale Blackjack-ului. Aceste scoruri sunt folosite pentru a determina câștigătorul.

- **Scorul jucătorului și al dealerului**: Fiecare carte are o valoare numerică (cărtile de 2-10 sunt echivalente cu valoarea lor nominală, iar cărțile de față (J, Q, K) au valoarea 10, iar Asul poate fi 1 sau 11, în funcție de ce este mai avantajos).
  - Scorul jucătorului este calculat pe baza cărților pe care le deține.
  - Dealerul trebuie să joace conform regulilor (hitting până la 17 sau mai mult), iar scorul acestuia este calculat în mod similar.
- **Rezultatul jocului**: La sfârșitul fiecărei runde, aplicația anunță câștigătorul (jucător sau dealer) pe baza comparației scorurilor. De asemenea, aplicația poate afișa un mesaj de încheiere a jocului și opțiunea de a juca din nou.

3.5. Securitate

Securitatea aplicației este o prioritate, iar aplicația utilizează mai multe tehnici pentru a proteja datele utilizatorilor și pentru a preveni accesul neautorizat:

- **Autentificare bazată pe JWT**: Token-ul JWT asigură că doar utilizatorii autentificați pot accesa funcționalitățile sensibile ale aplicației.
- **Protecția împotriva atacurilor CSRF**: Aplicația implementează mecanisme pentru a proteja utilizatorii de atacuri Cross-Site Request Forgery (CSRF).
- **Stocarea în siguranță a datelor**: Datele sensibile, cum ar fi parolele utilizatorilor, sunt stocate într-o formă criptată pe server, utilizând algoritmi de criptare siguri.

3.6. Interfața Utilizatorului (UI/UX)

Aplicația are o interfață ușor de utilizat și atractivă, care permite utilizatorilor să se concentreze pe joc. Aceasta include:

- **Ecranul de login**: Permite utilizatorilor să se autentifice cu ușurință folosind un nume de utilizator și o parolă.
- **Ecranul jocului**: Afișează mâinile jucătorului și dealerului, scorurile curente și butoane pentru a lua decizii (Hit/Stand).
- **Mesaje de stare**: Informații vizuale pentru a indica progresul jocului (exemplu: câștigătorul rundei).

Capitolul 4. Backend (Spring Boot)

Backend-ul aplicației este construit folosind **Spring Boot**, un framework Java care facilitează crearea de aplicații de tip server-side. În acest capitol, vom explora arhitectura backend-ului, modulele utilizate și cum interacționează acestea cu frontend-ul și baza de date pentru a oferi funcționalitățile aplicației de Blackjack.

4.1. Arhitectura Backend-ului

Aplicația backend este organizată într-o arhitectură modulară, bazată pe principii de **programare orientată pe servicii**. Principalele module care compun backend-ul aplicației sunt:

- **Controlerele (Controllers)**: Responsabile pentru gestionarea cererilor HTTP de la client (frontend) și coordonarea proceselor asociate.
- **Serviciile (Services)**: Conțin logica de business a aplicației, manipulând datele și interacționând cu repository-urile.
- **Repository-urile (Repositories)**: Interfatază baza de date, realizând operațiuni de citire și scriere a datelor.
- **Modelele (Models)**: Reprezintă entitățile de date din aplicație (de exemplu, utilizatori, jocuri, sesiuni de joc).

4.2. Controlerele (Controllers)

Controlerele din backend sunt componentele care expun API-urile RESTful ale aplicației. Acestea sunt responsabile pentru gestionarea cererilor din partea clientului, validarea datelor și redirecționarea lor către serviciile corespunzătoare.

Exemple de controlere:

1. **AuthController**: Gestionează autentificarea și înregistrarea utilizatorilor.
   - **Login**: Autentifică utilizatorul și generează un token JWT.
   - **Register**: Permite crearea unui cont de utilizator nou.
2. **GameController**: Gestionează crearea și progresul jocului de Blackjack.
   - **Create Game**: Creează un joc nou pentru un utilizator.
   - **Make Move**: Permite utilizatorului să facă o mișcare (hit/stand) în cadrul unui joc activ.
   - **Get Game Status**: Returnează starea curentă a jocului (mâinile jucătorului și dealerului, scorul curent etc.).

Exemplar de cod pentru un controller:

```
@RestController
@RequestMapping("/api/auth")
@RequiredArgsConstructor
public class AuthController {

    private final UserService userService;
    private final AuthenticationManager authenticationManager;
    private final JwtUtil jwtUtil;

    @PostMapping("/register")
    public ResponseEntity<?> register(@RequestBody AuthRequest request)
{
        User user = userService.register(request.getUsername(),
request.getPassword());
        return ResponseEntity.ok(user);
    }

    @PostMapping("/login")
    public ResponseEntity<?> login(@RequestBody AuthRequest request) {
        User user = userService.authenticate(request.getUsername(),
request.getPassword());
        if (user != null) {
            String token = jwtUtil.generateToken(user.getUsername());
            String username = user.getUsername();
            System.out.println(username);
            return ResponseEntity.ok(new
AuthResponse(token,username,"/game")); // Redirecționare spre pagina
jocului
        }
        return ResponseEntity.status(401).body("Invalid credentials");
    }
}
```

4.3. Serviciile (Services)

Serviciile din aplicație sunt locul unde se află logica de business. Ele interacționează cu repository-urile pentru a manipula datele și pentru a realiza operațiuni pe entitățile de la nivelul bazei de date.

Exemple de servicii:

1. **AuthService**: Contine logica pentru autentificarea și înregistrarea utilizatorilor, generarea token-urilor JWT și validarea acestora.
2. **GameService**: Se ocupă cu crearea și gestionarea sesiunilor de joc Blackjack, actualizarea stării jocului pe măsură ce jucătorul și dealerul fac mișcări.

Exemplar de cod pentru un serviciu:

```
public class UserService {

    private final UserRepository userRepository;
    private final BCryptPasswordEncoder passwordEncoder;
    private final AuthenticationManager authenticationManager;
    private final UserDetailsService userDetailsService;

    public User register(String username, String password) {
        User user = new User();
        user.setUsername(username);
        user.setPassword(passwordEncoder.encode(password));
        return userRepository.save(user);
    }

    public User authenticate(String username, String password) {
        // Autentificarea utilizatorului folosind AuthenticationManager
        authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(username,
password)
        );
```

4.4. Repository-urile (Repositories)

Repository-urile sunt componentele care interacționează direct cu baza de date, folosind **JPA (Java Persistence API)** pentru a efectua operațiuni de stocare și recuperare a datelor. Spring Data JPA oferă o implementare automată a repository-urilor, reducând semnificativ volumul de cod necesar pentru manipularea datelor.

Exemple de repository-uri:

1. **UserRepository**: Permite interacțiunea cu tabelul users, pentru a salva și recupera datele utilizatorilor.
2. **GameRepository**: Gestionează entitatea Game și permite salvarea și recuperarea jocurilor active și a istoricului acestora.

Exemplar de cod pentru un repository:

```
package com.example.Proiect_blecjec;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.Optional;

public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findByUsername(String username);
}
```

4.5. Modelele (Models)

Modelele reprezintă entitățile de date care sunt stocate și gestionate în aplicație. Acestea sunt mapate pe tabelele din baza de date și sunt folosite de controlere și servicii pentru a manipula informațiile necesare.

Exemple de modele:

1. **User**: Reprezintă utilizatorii aplicației (cu câmpuri precum username, password și role).
2. **Game**: Reprezintă un joc de Blackjack, incluzând informații despre mâinile jucătorului și dealerului, scorurile și starea jocului.

Exemplar de cod pentru un model:

```
@Entity
@Table(name = "users")
@Data
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true, nullable = false)
    private String username;

    @Column(nullable = false)
    private String password;
}
```

4.6. Securitate (Spring Security)

Securitatea aplicației este asigurată de **Spring Security**, care protejează endpoint-urile API și gestionează autentificarea și autorizarea utilizatorilor.

- **JWT Authentication**: Token-urile JWT sunt folosite pentru a autentifica utilizatorii și pentru a permite accesul doar celor care au un token valid.
- **Configurarea Spring Security**: Aplicația folosește configurații personalizate pentru a proteja endpoint-urile sensibile și pentru a permite accesul doar utilizatorilor autentificați.

Exemplar de cod pentru configurarea securității:

```
@Configuration
@RequiredArgsConstructor
public class SecurityConfig {

    private final UserDetailsService userDetailsService;

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws
Exception {
        http
                .csrf().disable()
                .cors().configurationSource(corsConfigurationSource())
                .and()
                .authorizeHttpRequests()
                .anyRequest().permitAll();
        return http.build();
    }

    @Bean
    public CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration configuration = new CorsConfiguration();

configuration.setAllowedOrigins(List.of("http://localhost:3000"));
```

```
        configuration.setAllowedMethods(List.of("GET", "POST", "PUT",
"DELETE", "OPTIONS"));
        configuration.setAllowedHeaders(List.of("*"));
        configuration.setAllowCredentials(true);

        UrlBasedCorsConfigurationSource source = new
UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", configuration);
        return source;
    }
```

4.7. Baza de Date (MySQL)

Backend-ul aplicației folosește o bază de date **MySQL** pentru a stoca informațiile despre utilizatori, jocuri și istoricul acestora. **Spring Data JPA** este utilizat pentru interacțiunea cu baza de date, iar **Hibernate** este utilizat pentru maparea obiectelor Java pe tabelele din MySQL.

- **Configurare MySQL**: Conexiunea la baza de date este configurată în fișierul application.properties, iar aplicația utilizează un DataSource pentru a interacționa cu baza de date.

Exemplu de configurare a bazei de date:

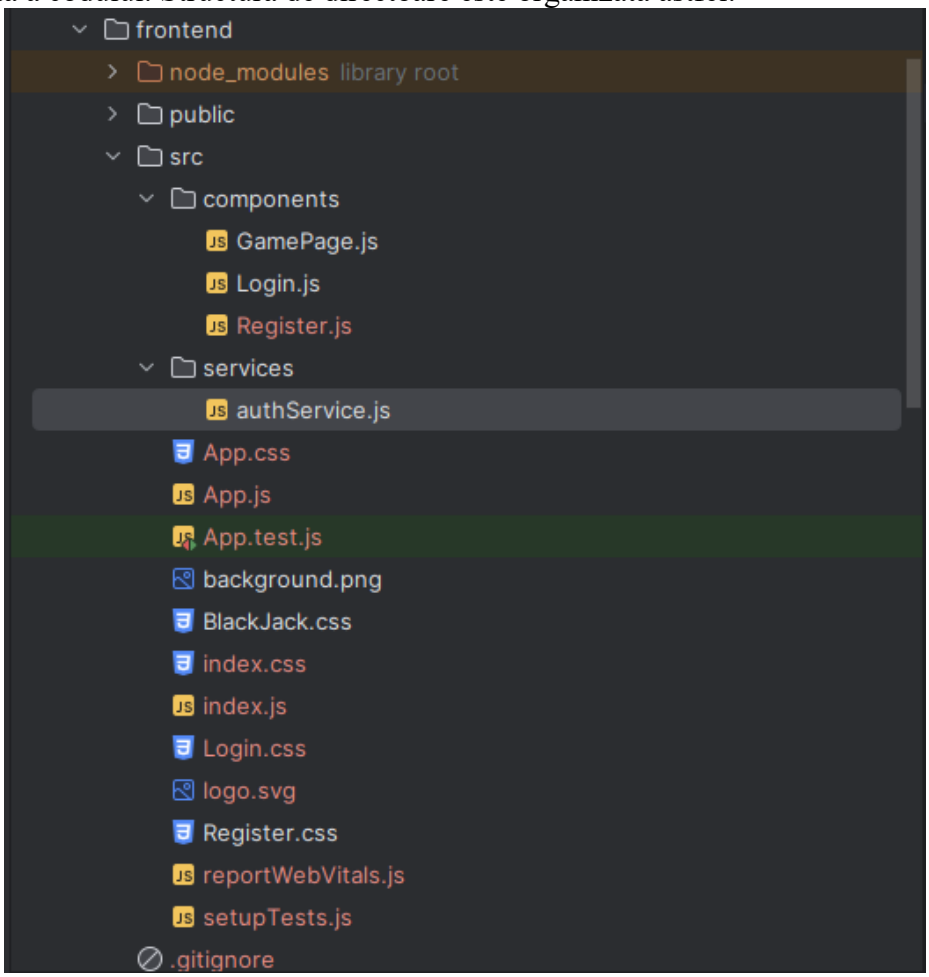
```
            server.port=8081

spring.datasource.url=jdbc:mysql://localhost:3306/proiect_paw
spring.datasource.username=root
spring.datasource.password=

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dia
lect
jwt.expirationMs=3600000
```

Capitolul 5. Frontend (React)

Frontend-ul aplicației este construit utilizând **React**, o bibliotecă JavaScript populară pentru dezvoltarea interfețelor de utilizator interactive și reactive. În acest capitol, vom explora arhitectura, principalele componente, funcționalitățile oferite și modul în care frontend-ul interacționează cu backend-ul.

5.1. Arhitectura Frontend-ului

Aplicația React este structurată pentru a susține scalabilitatea și organizarea eficientă a codului. Structura de directoare este organizată astfel:



- **components/**: Conține toate componentele React utilizate în aplicație.
- **services/**: Include fișierele care gestionează cererile HTTP către backend.
- **App.js**: Componenta principală care definește structura generală și rutele aplicației.

5.2. Componentele Principale

5.2.1. Login.js

Această componentă permite utilizatorilor să se autentifice în aplicație. Este conectată la backend-ul de autentificare printr-un serviciu HTTP (Axios).

Funcționalități:

- Colectarea numelui de utilizator și a parolei.
- Trimiterea datelor către endpoint-ul /api/auth/login din backend.
- Stocarea token-ului JWT și a altor informații relevante în localStorage.
- Redirecționarea utilizatorului către pagina principală după autentificare.

Exemplu de cod:

```
const Login = () => {
    const [username, setUsername] = useState('');
    const [password, setPassword] = useState('');
    const navigate = useNavigate();

    const handleLogin = async (e) => {
        e.preventDefault();
        try {
            const response = await
axios.post('http://localhost:8081/api/auth/login', {
                username,
                password,
            });

            if (response.status === 200) {
                localStorage.setItem('token', response.data.token);

localStorage.setItem("numeUtilizator",response.data.username);
                navigate(response.data.redirectUrl);  // Redirecționare
spre pagina jocului
            }
        } catch (error) {
            alert('Invalid credentials!');
        }
    };

    return (
        <div className="login-container">
            <div className="login_box">
                <h2>Login</h2>
                <form onSubmit={handleLogin}>
                    <div className="input-group">
                        <label htmlFor="username">Username:</label>
                        <input
                            type="text"
                            id="username"
                            value={username}
                            onChange={(e) =>
setUsername(e.target.value)}
```

```
                            placeholder="Enter your username"
                            required
                        />
                    </div>
                    <div className="input-group">
                        <label htmlFor="password">Password:</label>
                        <input
                            type="password"
                            id="password"
                            value={password}
                            onChange={(e) =>
setPassword(e.target.value)}
                            placeholder="Enter your password"
                            required
                        />
                    </div>
                    <button type="submit" className="register-
btn">Login</button>
                </form>
            </div>
        </div>
    );
};

export default Login;
```

5.2.2. Register.js

Această componentă gestionează înregistrarea utilizatorilor. Permite utilizatorilor să își creeze un cont, trimițând datele către endpoint-ul /api/auth/register.

5.2.3. GamePage.js

Componenta **Game** este responsabilă pentru afișarea și gestionarea unui joc de Blackjack.

Funcționalități:

- Obține starea curentă a jocului de la backend.
- Afișează mâna utilizatorului și a dealerului.
- Permite utilizatorului să facă mișcări (hit/stand).
- Primește rezultatele jocului de la backend.

5.3. Interacțiunea cu Backend-ul

Frontend-ul comunică cu backend-ul prin cereri HTTP realizate folosind **Axios**. Fiecare serviciu din directorul `services/` definește funcții specifice pentru aceste cereri.

Exemplu de serviciu:

```
import axios from 'axios';

const API_URL = 'http://localhost:8081/api/auth/';

const register = (username, password) => {
    return axios.post(API_URL + 'register', {
        username,
        password
    });
};

const login = (username, password) => {
    return axios.post(API_URL + 'login', {
        username,
        password
    });
};

const logout = () => {
    localStorage.removeItem('token');
};
```

5.4. Gestionarea Securității (JWT)

Frontend-ul utilizează **token-uri JWT** pentru a autentifica utilizatorii la fiecare cerere către backend. Aceste token-uri sunt stocate în `localStorage` și sunt atașate la header-ul cererilor HTTP.

Exemplu de funcție pentru gestionarea token-ului:

```
useEffect(() => {
    const token = localStorage.getItem('token');
    const numeUtilizator = localStorage.getItem('numeUtilizator');
    setUsername(numeUtilizator);
    if (!token) {
        setIsAuthenticated(false);
    } else {
        const decodedToken = jwtDecode(token);
        axios.get('http://localhost:8081/api/user/username', {
            headers: {
                Authorization: `Bearer ${token}`
            }
        })
            .then(response => {
                console.log(response.data);
```

```
            })
            .catch(error => {
                console.error("Error fetching username:", error);
                setIsAuthenticated(false);
            });
    }
}, []);
```

5.5. Rute și Navigare

Navigarea în aplicație este gestionată folosind **React Router**. Fiecare pagină a aplicației este asociată cu o rută specifică.

Exemplu de configurare a rutelor:

```
import React from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-
dom';
import './App.css';  // Importă stilurile din App.css
import Login from './components/Login';
import Register from './components/Register';
import GamePage from "./components/GamePage";
function App() {
  return (
      <Router>
        <div className="App">
          <Routes>
            <Route path="/login" element={<Login />} />
            <Route path="/register" element={<Register />} />
            <Route path="/" element={<h1>Welcome to Blackjack</h1>} />
              <Route path="/game" element={<GamePage />} />
          </Routes>
        </div>
      </Router>
  );
}

export default App;
```

Capitolul 6. Securitate

Securitatea este un aspect esențial al aplicației, având în vedere faptul că aceasta gestionează autentificarea utilizatorilor și accesul la date sensibile. Acest capitol detaliază mecanismele implementate atât pe partea de backend, cât și pe partea de frontend pentru a asigura integritatea, confidențialitatea și disponibilitatea datelor.

6.1. Mecanisme de Autentificare și Autorizare

Autentificare cu JWT

Aplicația utilizează **JSON Web Tokens (JWT)** pentru autentificarea utilizatorilor. Acest mecanism oferă o metodă eficientă și sigură de a identifica utilizatorii autentificați.

- **Generarea token-ului**:
  - După autentificarea cu succes, serverul generează un token JWT semnat cu o cheie secretă.
  - Token-ul include informații precum numele de utilizator și rolul utilizatorului (dacă există), iar acesta este trimis către client.
- **Păstrarea token-ului**:
  - Pe frontend, token-ul este stocat în localStorage, oferind acces ușor pentru cererile ulterioare.
- **Validarea token-ului**:
  - Fiecare cerere către backend care necesită autentificare include token-ul JWT în header-ul Authorization folosind formatul: Bearer <token>.
  - Serverul validează token-ul, verificând semnătura și data de expirare.

6.2. Măsuri de Securitate în Backend

6.2.1. Hashing-ul parolelor

Parolele utilizatorilor sunt securizate folosind hashing cu algoritmul **BCrypt**.

- La înregistrare, parola este convertită într-un hash ireversibil și stocată în baza de date.
- La autentificare, parola introdusă de utilizator este comparată cu hash-ul stocat.

Exemplu de utilizare BCrypt:

```
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

public String hashPassword(String password) {
    BCryptPasswordEncoder encoder = new BCryptPasswordEncoder();
    return encoder.encode(password);
}
```

6.2.2. Validarea intrărilor

Toate datele primite de la client sunt validate pentru a preveni atacurile de tip **SQL Injection** sau **XSS (Cross-Site Scripting)**.

6.2.3. Configurarea CORS

Pentru a controla accesul la resursele backend-ului din alte domenii, este configurată politica **CORS (Cross-Origin Resource Sharing)**.

Exemplu de configurare:

```
public void addCorsMappings(CorsRegistry registry) {
    registry.addMapping("/**")
            .allowedOrigins("http://localhost:3000") // Frontend-ul
care rulează pe portul 3000
            .allowedMethods("GET", "POST", "PUT", "DELETE")
            .allowCredentials(true);
}
```

6.3. Măsuri de Securitate în Frontend

6.3.1. Stocarea Token-urilor

Token-urile JWT sunt stocate în localStorage pentru utilizare ușoară, dar acest lucru poate expune aplicația la atacuri de tip **XSS**. Măsurile implementate includ:

- Validarea strictă a intrărilor în formulare pentru a preveni scripturile malițioase.
- Utilizarea HTTPS pentru a proteja traficul dintre client și server.

6.3.2. Gestionarea expirării token-ului

Token-urile au o durată de viață limitată. Frontend-ul gestionează expirarea token-ului prin verificarea periodică a validității acestuia.

Exemplu de funcție pentru verificarea expirării:

```
export const isTokenExpired = (token) => {
        const payload = JSON.parse(atob(token.split('.')[1]));
        return payload.exp * 1000 < Date.now();
};
```

6.3.3. Logout-ul utilizatorilor

La deconectare, token-ul JWT este eliminat din localStorage, iar utilizatorul este redirecționat către pagina de login.

Capitolul 7. Concluzii și Îmbunătățiri

7.1. Concluzii

Proiectul realizat combină tehnologiile moderne precum **Spring Boot** și **React** pentru a crea o aplicație web securizată, scalabilă și ușor de utilizat. Integrarea funcționalităților de autentificare, gestionare a utilizatorilor și un sistem interactiv de joc demonstrează capacitatea de a combina eficient backend-ul și frontend-ul pentru a oferi o experiență completă utilizatorilor.

7.1.1. Aspecte pozitive ale aplicației

- **Arhitectură modulară**:
- Separarea clară între frontend și backend facilitează întreținerea și extinderea aplicației.
- **Securitate îmbunătățită**:
- Utilizarea JWT pentru autentificare și autorizare asigură o protecție sporită.
- **Scalabilitate**:
- Aplicarea principiilor REST și utilizarea tehnologiei React permit adăugarea de noi funcționalități cu efort minim.
- **Interfață prietenoasă**:
- Design-ul interactiv și intuitiv facilitează utilizarea aplicației de către utilizatori, indiferent de nivelul lor tehnic.

7.1.2. Lecții învățate

- Importanța validării datelor de intrare pentru prevenirea vulnerabilităților de securitate.
- Gestionarea erorilor este esențială pentru o experiență bună a utilizatorului.
- O comunicare eficientă între frontend și backend este cheia pentru un sistem robust.

7.2. Concluzia finală

Aplicația reprezintă un exemplu solid de integrare a tehnologiilor moderne pentru crearea unui sistem web complet. Cu o arhitectură scalabilă, funcționalități cheie bine implementate și măsuri de securitate solide, acest proiect este o bază excelentă pentru extindere și dezvoltare ulterioară.

Continuarea dezvoltării prin adăugarea de noi funcționalități și îmbunătățiri va crește valoarea aplicației și satisfacția utilizatorilor.