

1. Tổng quan cách ứng dụng hoạt động

Ứng dụng này là một hệ thống quản lý quán cà phê, với backend (Spring Boot) cung cấp API và frontend (React) hiển thị giao diện. Các chức năng chính bao gồm:

- **Đăng nhập/dăng ký:** Người dùng đăng nhập hoặc đăng ký tài khoản.
- **Phân quyền (role):** ADMIN và USER có quyền truy cập khác nhau.
- **Quản lý người dùng (ADMIN):** ADMIN có thể tạo/sửa/xóa người dùng.
- **Quản lý bàn (ADMIN và USER):** ADMIN có thể CRUD bàn, USER chỉ xem và đặt/hủy bàn.
- **Đăng xuất:** Người dùng có thể đăng xuất.

Ứng dụng chạy trên Docker với docker-compose.yml, backend chạy trên cổng 8080 và frontend chạy trên cổng 3000.

2. Luồng hoạt động tổng quát

a. Khởi động ứng dụng

- **Backend:**

- File CoffeeShopApplication.java là điểm khởi động của Spring Boot:

java

Sao chép

```
@SpringBootApplication

public class CoffeeShopApplication {

    public static void main(String[] args) {
        SpringApplication.run(CoffeeShopApplication.class,
        args);
    }
}
```

- Khi khởi động, backend tạo tài khoản admin mặc định (nếu chưa có):

java

Sao chép

```
@Bean
```

```

CommandLineRunner initData(UserRepository userRepository,
PasswordEncoder passwordEncoder) {

    return args -> {

        if (userRepository.findByUsername("admin").isEmpty()) {

            User rootAdmin = new User();
            rootAdmin.setUsername("admin");
            rootAdmin.setPassword(passwordEncoder.encode("123456"));
            rootAdmin.setRole("ADMIN");
            userRepository.save(rootAdmin);

            System.out.println("Admin account created with
username: admin, password: 123456");
        }
    };
}

```

- Backend chạy trên cổng 8080 (theo application.properties).
- **Frontend:**
 - File index.js là điểm khởi động của React:

javascript

Sao chép

```

const root =
ReactDOM.createRoot(document.getElementById('root'));

root.render(
<React.StrictMode>
<App />

```

```
</React.StrictMode>
```

```
) ;
```

- o File App.js định nghĩa các route:

javascript

Sao chép

```
function App() {  
  
  return (  
  
    <Router>  
  
      <Routes>  
  
        <Route path="/login" element={<Login />} />  
  
        <Route path="/register" element={<Register />} />  
  
        <Route path="/admin" element={<ProtectedRoute  
          requiredRole="ADMIN"><AdminDashboard /></ProtectedRoute>} />  
  
        <Route path="/user" element={<ProtectedRoute  
          requiredRole="USER"><UserDashboard /></ProtectedRoute>} />  
  
        <Route path="/" element={<Login />} />  
  
      </Routes>  
  
    </Router>  
  
) ;  
}
```

b. Luồng chính

1. Người dùng truy cập ứng dụng, mặc định vào trang /login (Login.js).
2. Người dùng đăng nhập hoặc đăng ký:
 - o Đăng nhập gọi API /api/auth/login (xử lý bởi AuthController.java).

- Đăng ký gọi API /api/auth/register (cũng do AuthController.java xử lý).
3. Sau khi đăng nhập, người dùng được chuyển hướng dựa trên vai trò (role):
 - ADMIN → /admin (AdminDashboard.js).
 - USER → /user (UserDashboard.js).
 4. **Phân quyền:**
 - Backend kiểm tra role qua SecurityConfig.java.
 - Frontend kiểm tra role qua ProtectedRoute.js.
 5. Người dùng thực hiện các chức năng (CRUD người dùng, quản lý bàn, đặt bàn, đăng xuất).
 6. Đăng xuất gọi API /api/auth/logout và chuyển về trang đăng nhập.
-

3. Chức năng phân quyền (role) và cách liên kết

Phân quyền được thực hiện ở cả backend và frontend, dựa trên vai trò ADMIN và USER.

a. Backend: Định nghĩa và kiểm tra role

- **Định nghĩa role:**
 - Role được lưu trong thực thể User (User.java):

java

Sao chép

```
@Entity  
  
@Table(name = "users")  
  
@Data  
  
public class User {  
  
    @Id  
  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
  
    private Long id;  
  
    private String username;  
  
    private String password;  
  
    private String role; // ADMIN hoặc USER
```

}

- Role được khởi tạo khi tạo tài khoản admin mặc định (xem CoffeeShopApplication.java ở trên) hoặc khi đăng ký (AuthController.java).
 - **Kiểm tra role:**
 - File SecurityConfig.java định nghĩa các quyền truy cập dựa trên role:

java

Sao chép

@Bean

```
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http
        .cors(cors ->
            cors.configurationSource(corsConfigurationSource()))
        .csrf(csrf -> csrf.disable())
        .headers(headers -> headers.frameOptions(frame ->
            frame.disable()))
        .authorizeHttpRequests(auth -> auth
            .requestMatchers("/api/auth/**", "/h2-console/**").permitAll() // Không cần xác thực
            .requestMatchers("/api/admin/users/**").hasRole("ADMIN") // Chỉ ADMIN truy cập
            .requestMatchers("/api/admin/tables/**").hasAnyRole("ADMIN", "USER") // ADMIN và USER truy cập
            .requestMatchers("/api/user/tables/**").hasAnyRole("USER", "ADMIN") // ADMIN và USER truy cập
        );
}
```

```
        .requestMatchers("/api/user/**").hasAnyRole("USER",
"ADMIN") // ADMIN và USER truy cập

        .requestMatchers("/api/menu/**").hasAnyRole("USER",
"ADMIN") // ADMIN và USER truy cập

        .anyRequest().authenticated() // Các request khác
cần xác thực

)

.formLogin(form -> form

    .loginProcessingUrl("/api/auth/login")

    .successHandler((request, response, authentication)
-> response.setStatus(200))

    .failureHandler((request, response, exception) -> {

        response.setStatus(401);

        response.getWriter().write("Tên người dùng hoặc
mật khẩu không đúng!");

    })

    .permitAll()

)

.logout(logout -> logout

    .logoutUrl("/api/auth/logout")

    .logoutSuccessHandler((request, response,
authentication) -> response.setStatus(200))

    .permitAll()

);
```

```
        return http.build();
```

```
}
```

- o Đoạn code trên yêu cầu:
 - /api/admin/users/** chỉ dành cho ADMIN.
 - /api/admin/tables/** và /api/user/** cho phép cả ADMIN và USER.
- o Để kiểm tra role, Spring Security sử dụng UserDetailsService (cũng trong SecurityConfig.java):

java

Sao chép

```
@Bean
```

```
public UserDetailsService userDetailsService(UserRepository
userRepository) {
```

```
    return username -> {
```

```
        User user = userRepository.findByUsername(username)
```

```
            .orElseThrow(() -> new
```

```
UsernameNotFoundException("User not found"));
```

```
        return
```

```
            org.springframework.security.core.userdetails.User
```

```
                .withUsername(user.getUsername())
```

```
                .password(user.getPassword())
```

```
                .roles(user.getRole()) // Gán role từ database
```

```
                .build();
```

```
    };
```

```
}
```

- o Khi người dùng đăng nhập, Spring Security lấy role từ database và gán vào Authentication object.

b. Frontend: Kiểm tra role và điều hướng

- **Kiểm tra role:**

- File ProtectedRoute.js kiểm tra role của người dùng để bảo vệ route:

javascript

Sao chép

```
const ProtectedRoute = ({ children, requiredRole }) => {  
  
    const [isAuthenticated, setIsAuthenticated] =  
        useState(null);  
  
    const [userRole, setUserRole] = useState(null);  
  
  
  
  
    useEffect(() => {  
  
        const checkAuth = async () => {  
  
            try {  
  
                const response = await  
                    axios.get('http://localhost:8080/api/auth/current-user', {  
  
                        withCredentials: true,  
  
                });  
  
                setIsAuthenticated(true);  
  
                setUserRole(response.data.role); // Lấy role từ  
                API  
  
            } catch (err) {  
  
                console.error('Auth check failed:', err);  
  
                setIsAuthenticated(false);  
  
            }  
  
        };  
  
        checkAuth();  
    }, [isAuthenticated]);  
  
    return isAuthenticated ? children : null;  
};
```



```
}
```

```
    return children;
```

```
} ;
```

- o Đoạn code trên:

- Gọi API /api/auth/current-user để lấy thông tin người dùng, bao gồm role.
 - Nếu không xác thực, chuyển hướng về /login.
 - Nếu role không khớp (ví dụ USER truy cập /admin), chuyển hướng về /user hoặc /login.

- Sử dụng **ProtectedRoute** trong App.js:

- o ProtectedRoute được dùng để bảo vệ các route /admin và /user:

javascript

Sao chép

```
<Route path="/admin" element={<ProtectedRoute  
requiredRole="ADMIN"><AdminDashboard /></ProtectedRoute>} />  
  
<Route path="/user" element={<ProtectedRoute  
requiredRole="USER"><UserDashboard /></ProtectedRoute>} />
```

- o /admin yêu cầu role ADMIN, /user yêu cầu role USER.

c. Luồng đăng nhập và lấy role

- Đăng nhập (Login.js):

- o Người dùng nhập username và password, gửi POST request đến /api/auth/login:

javascript

Sao chép

```
const handleLogin = async (e) => {  
  
  e.preventDefault();  
  
  try {  
  
    const formData = new URLSearchParams();
```

```
        formData.append('username', username);

        formData.append('password', password);

    }

    await axios.post(
        'http://localhost:8080/api/auth/login',
        formData,
        {
            headers: { 'Content-Type': 'application/x-www-
form-urlencoded' },
            withCredentials: true
        }
    );

    const response = await
axios.get('http://localhost:8080/api/auth/current-user', {
    withCredentials: true
});

    const role = response.data.role;

    if (role === 'ADMIN') {
        navigate('/admin');
    } else if (role === 'USER') {
        navigate('/user');
    } else {
        setError('Vai trò không hợp lệ!');
    }
}

} catch (err) {
```

```

        const errorMessage = err.response?.data || 'Đăng nhập thất bại, vui lòng thử lại!';
        setError(errorMessage);
    }
};


```

- Sau khi đăng nhập thành công, gọi /api/auth/current-user để lấy role và điều hướng.
- **Backend xử lý đăng nhập (SecurityConfig.java):**
 - Đã định nghĩa endpoint /api/auth/login với formLogin (xem phần securityFilterChain ở trên).
 - Nếu đăng nhập thành công, trả về mã 200. Nếu thất bại, trả về mã 401 và thông báo lỗi.
- **Backend trả thông tin người dùng (AuthController.java):**
 - Endpoint /api/auth/current-user trả về thông tin người dùng, bao gồm role:

java

Sao chép

```

@GetMapping("/current-user")
public User getCurrentUser() {
    Authentication authentication =
        SecurityContextHolder.getContext().getAuthentication();
    String username = authentication.getName();
    User user = userRepository.findByUsername(username)
        .orElseThrow(() -> new RuntimeException("User not found"));
    return user; // Trả về đối tượng User, bao gồm role
}

```

d. Sử dụng role trong các chức năng

- **ADMIN (AdminDashboard.js):**
 - ADMIN có thể quản lý người dùng và bàn:

javascript

Sao chép

```
const fetchUsers = async () => {

    const data = await apiCall('get',
'http://localhost:8080/api/admin/users', null, setError);

    setUsers(data);

};

const fetchTables = async () => {

    const data = await apiCall('get',
'http://localhost:8080/api/admin/tables', null, setError);

    setTables(data);

};
```

- Gọi API /api/admin/users và /api/admin/tables, chỉ ADMIN được phép truy cập (theo SecurityConfig.java).
- **USER (UserDashboard.js):**
 - USER chỉ có thể xem và đặt/hủy bàn:

javascript

Sao chép

```
const fetchTables = async () => {

    const data = await apiCall('get',
'http://localhost:8080/api/user/tables', null, setError);

    setTables(data);

};

const handleBookTable = async (id, currentStatus) => {

    if (currentStatus === 'OCCUPIED') {
```

```

        setError('Bàn này đã được đặt, vui lòng chọn bàn
khác! ');
    }

    await apiCall('put',
`http://localhost:8080/api/user/tables/${id}`, { status:
'OCCUPIED' }, setError);

    await fetchTables();

};


```

- Gọi API /api/user/tables, USER được phép truy cập (theo SecurityConfig.java).
-

4. Liên kết giữa các file

Backend

- CoffeeShopApplication.java → Khởi động ứng dụng, tạo tài khoản admin mặc định → Dùng UserRepository.java để lưu user.
- SecurityConfig.java → Định nghĩa phân quyền và xác thực → Dùng UserRepository.java để lấy thông tin user.
- AuthController.java → Xử lý đăng nhập/đăng ký → Dùng UserRepository.java để lưu/lấy user.
- AdminController.java → Xử lý yêu cầu từ ADMIN → Dùng UserRepository.java và TableRepository.java.
- UserController.java → Xử lý yêu cầu từ USER → Dùng TableRepository.java.

Frontend

- App.js → Định nghĩa route → Dùng ProtectedRoute.js để kiểm tra role → Điều hướng đến Login.js, Register.js, AdminDashboard.js, UserDashboard.js.
- Login.js → Gửi yêu cầu đăng nhập → Gọi API /api/auth/login và /api/auth/current-user → Điều hướng dựa trên role.
- ProtectedRoute.js → Kiểm tra role → Gọi API /api/auth/current-user.
- AdminDashboard.js → Gọi API /api/admin/users và /api/admin/tables → Hiển thị giao diện quản trị.
- UserDashboard.js → Gọi API /api/user/tables → Hiển thị giao diện người dùng.

Docker

- docker-compose.yml → Khởi động cả backend và frontend → Dùng Dockerfile trong backend/ và frontend/.
-

5. Tóm tắt

- **Phân quyền (role):**
 - Backend: SecurityConfig.java kiểm tra role để cấp quyền truy cập API.
 - Frontend: ProtectedRoute.js kiểm tra role để bảo vệ route.
- **Luồng chính:**
 1. Đăng nhập (Login.js → /api/auth/login → SecurityConfig.java).
 2. Lấy role (/api/auth/current-user → AuthController.java).
 3. Điều hướng dựa trên role (ProtectedRoute.js → /admin hoặc /user).
 4. Thực hiện chức năng dựa trên role (AdminDashboard.js hoặc UserDashboard.js)