

# Java Spring

# Security and Cloud

- To do list -

**Group:** 30434

**Team:** Baraian Tudor,  
Anca-Maria Giurgiu

## Contents

1.	Project Overview .....	3
2.	Architecture Overview.....	3
2.1.	Description of the Controller-Service-Repository Architecture.....	3
3.	Security Implementation .....	6
3.1.	JWT Authentication Overview .....	6
3.2.	Security Configurations .....	8
3.3.	Account Management .....	10
4.	Front-End Implementation .....	12
5.	Use Cases .....	22
6.	Class Diagram .....	22
7.	Deployment Diagram.....	23
8.	Conclusion.....	24

## 1. Project Overview

Our project is a dynamic web application developed using Spring Boot, integrating essential features of Spring Security. It's designed as a To-Do application, aimed at helping users manage their tasks and schedules with ease. The application utilizes a combination of HTML, CSS and JavaScript for the front end, ensuring a user-friendly interface, while the backend is powered by the robust and versatile capabilities of Spring Boot and Spring Security.

## 2. Architecture Overview

### 2.1. Description of the Controller-Service-Repository Architecture

Our application is structured around the Controller-Service-Repository pattern, a widely adopted architectural style in modern web applications, especially in Spring Boot projects. This pattern is instrumental in promoting separation of concerns, enhancing code maintainability, and facilitating scalability.

#### **Controllers**

- **Role:** Controllers act as the entry point for handling HTTP requests from the client side. They interpret user inputs and translate them into actions to be performed by the service layer.
- **Implementation:** In our project, controllers such as `TodayController`, `WeeklyController`, and `MonthlyController` are responsible for managing user interactions related to daily, weekly, and monthly tasks, respectively. They handle various HTTP requests (GET, POST, DELETE) and route them to appropriate service methods.

## Services

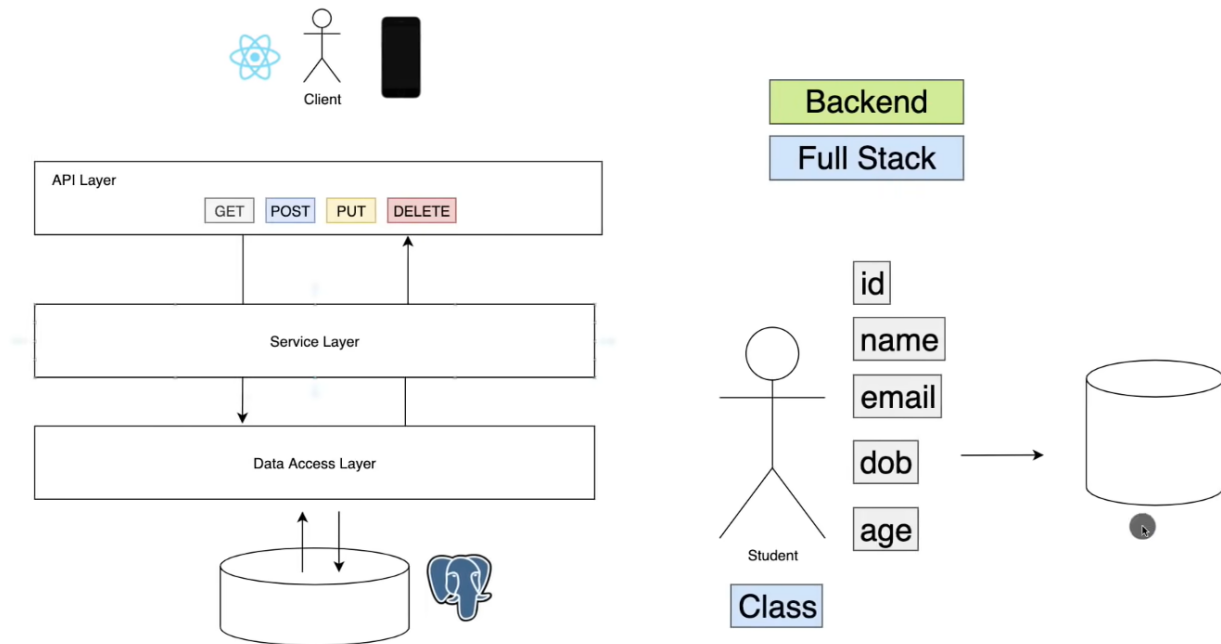
- **Role:** The service layer contains the business logic of the application. It acts as a bridge between the controllers and the repositories, ensuring that business rules and logic are correctly applied.
- **Implementation:** Services like `TodayService`, `WeeklyService`, and `MonthlyService` encapsulate the core business logic. They interact with the repository layer to retrieve, manipulate, and store data, while also implementing specific business rules and validations.

## Repositories

- **Role:** Repositories are responsible for data access and manipulation. They abstract the complexity of direct database interactions from the service layer.
- **Implementation:** Repository interfaces such as `TodayRepository`, `WeeklyRepository`, and `MonthlyRepository` extend Spring's `JpaRepository`, providing a rich set of methods for CRUD operations and database interactions. These repositories are used by the service layer to persist and retrieve data models like `Today`, `Weekly`, and `Monthly`.

## Benefits

- **Decoupling:** This architecture decouples the user interface (controllers), business logic (services), and data access (repositories), making the codebase more organized, understandable, and manageable.
- **Testability:** Each layer can be independently tested, improving the overall quality and reliability of the application.
- **Flexibility:** Changes in one layer, such as switching databases or modifying business logic, have minimal impact on other layers, enhancing the application's adaptability.
- **Reusability:** Common functionalities in the service layer can be reused across different controllers, reducing code redundancy.



## 3. Security Implementation

### 3.1. JWT Authentication Overview

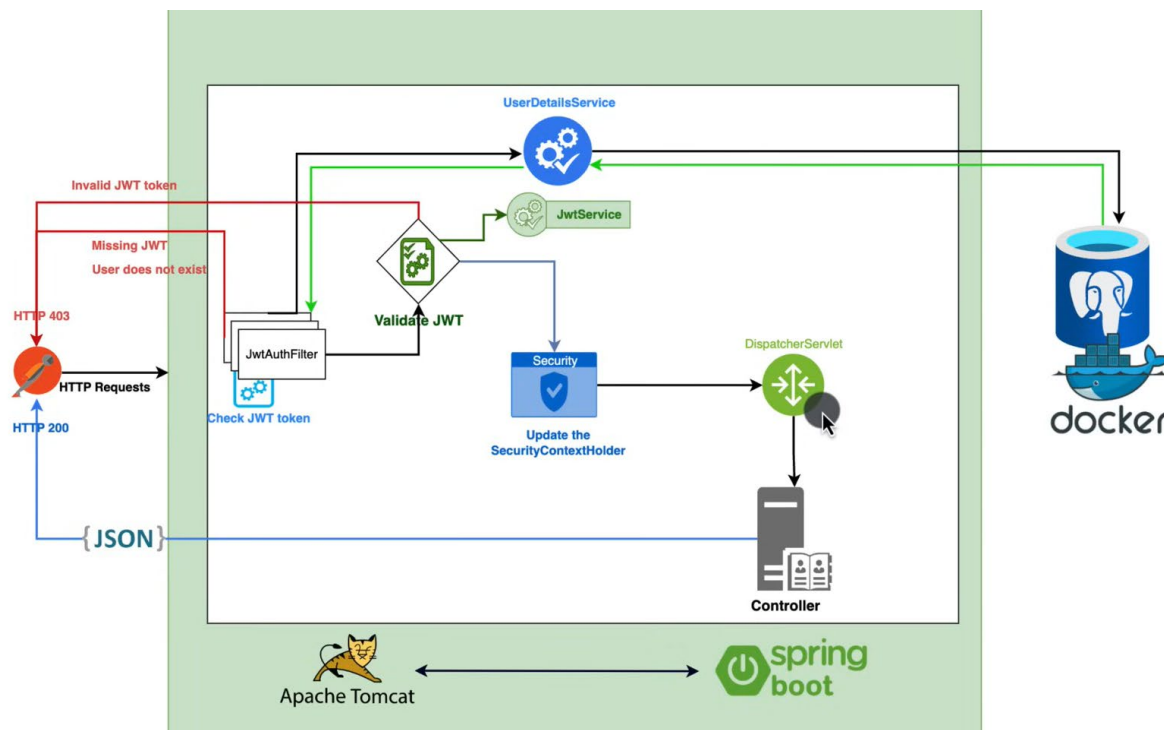
JSON Web Token (JWT) authentication is a cornerstone of our application's security mechanism. It provides a compact and self-contained way to securely transmit information between parties as a JSON object. This section provides an overview of how JWT authentication is implemented and utilized in our application.

#### How JWT Works

- **Token Creation:** When a user successfully logs in using their credentials, a JWT is generated by the server. This token contains encoded JSON data, including user details and an expiration time.
- **Token Structure:** A JWT typically consists of three parts: a header, a payload, and a signature. The header specifies the token type (JWT) and the signing algorithm. The payload contains the claims, which are statements about the user and additional data. The signature ensures the token's integrity and is generated using the header, payload, and a secret key.
- **Stateless Authentication:** Once the JWT is created, it is sent back to the client. For subsequent requests, this token is included in the request headers. The server then validates the token without needing to query the database, making the process stateless and efficient.

## Implementation in Our Application

- **Token Generation and Validation:** In our application, the `JwtService` class is responsible for generating and validating JWTs. It uses a secret key to create the signature and performs checks to ensure the token's validity and integrity.
- **User Authentication Flow:**
  - **Login:** The user logs in through the `AuthenticationController`, which delegates authentication to the `AuthenticationService`.
  - **Token Generation:** Upon successful authentication, `JwtService` generates a JWT, which is then sent back to the user.
  - **Token Usage:** The client includes this JWT in the HTTP Authorization header of subsequent requests.
  - **Token Verification:** The `JwtAuthenticationFilter` intercepts incoming requests, extracts the JWT from the header, and validates it using `JwtService`.
  - **Access Control:** If the token is valid, the user is granted access to the requested resources. Otherwise, an error response is returned.
- **Security Considerations**
  - **Confidentiality of Secret Key:** The security of JWT relies heavily on the confidentiality of the secret key used for signing tokens.
  - **Token Expiration:** Tokens are configured with an expiration time to mitigate the risks associated with stolen or intercepted tokens.
  - **HTTPS:** To prevent token interception, all communications between the client and server are secured using HTTPS.



## 3.2. Security Configurations

Effective security configurations are crucial for safeguarding our application against unauthorized access and potential security threats. In this section, we delve into the key security configurations implemented in our application, focusing on how they fortify the overall security posture.

### Core Components of Security Configuration

#### SecurityConfig Class

**Purpose:** SecurityConfig extends Spring Security's WebSecurityConfigurerAdapter and is the central place to configure security-related aspects of our application.

**Key Configurations:**



- **Authentication Provider:** Integration with a custom authentication provider to manage user authentication processes.
- **Session Management:** Configuration of session creation policy, typically set to stateless to work seamlessly with JWT authentication.
- **CSRF Protection:** Disabling CSRF (Cross-Site Request Forgery) protection, which is a common practice when using JWT, as it inherently protects against CSRF attacks.
- **CORS (Cross-Origin Resource Sharing):** Setting up CORS configurations to allow or restrict requests based on origin, headers, and methods, which is crucial for applications interacting with different domains.

#### JwtAuthenticationFilter Class

**Purpose:** JwtAuthenticationFilter is responsible for intercepting HTTP requests and extracting the JWT token for validation.

**Functionality:**

- **Token Extraction:** Extracts the JWT token from the request headers.
- **Token Validation:** Utilizes JwtService to validate the extracted token.
- **User Authentication:** If the token is valid, the filter sets the authentication in the security context, allowing the user to access protected resources.

#### JwtService Class

**Role:** Central to JWT operations, JwtService handles the generation, parsing, and validation of JWT tokens.

**Key Methods:**

- **generateToken:** Creates a new JWT token based on user details.
- **validateToken:** Checks the validity of a token, ensuring it's not expired and is correctly signed.

- `extractUsername`: Retrieves the username or other credentials from the token payload.

### 3.3. Account Management

Account management is a critical component of our application's security framework, ensuring that user information is handled securely and efficiently. This section outlines the mechanisms and strategies we employ for managing user accounts, from registration to authentication and maintenance.

#### User Registration and Account Creation

- **Process:** Users can create an account through a registration form. The `AuthenticationController` handles the registration requests.
- **Data Handling:** User details, including usernames and passwords, are collected and stored securely. Passwords are encrypted using a strong hashing algorithm before being stored in the database.
- **Validation:** Input validation is performed to ensure data integrity and prevent common security vulnerabilities like SQL injection.

#### User Authentication

- **Login Mechanism:** The `AuthenticationController` also manages login requests. Upon successful login, a JWT token is generated and sent back to the user, marking the beginning of an authenticated session.
- **Password Security:** Passwords are never stored in plain text. During the login process, the provided password is hashed and compared with the stored hash to verify user identity.

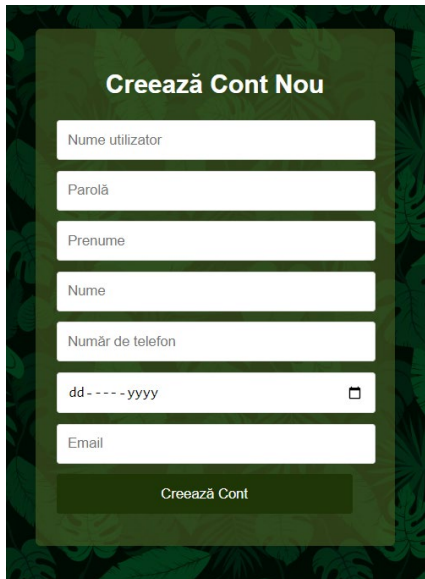
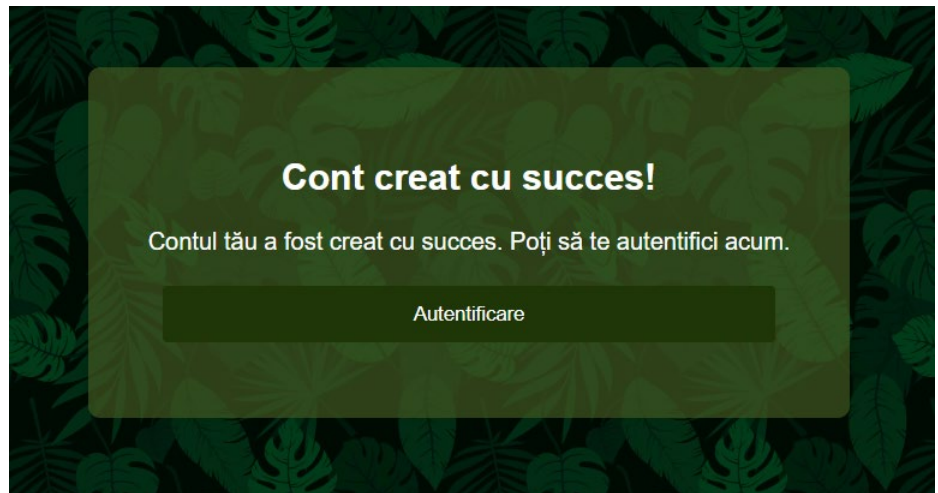
#### Role-Based Access Control (RBAC)

- **Implementation:** The application implements RBAC to define what resources a user can access. Roles like 'USER', 'ADMIN', etc., are assigned to users, and access permissions are granted based on these roles.
- **Enforcement:** Access control is enforced at both the controller and service levels, ensuring that users can only perform actions that their roles permit.

#### Account Maintenance and Security

- **Profile Updates:** Users can update their profile information. Changes are handled securely to maintain data integrity.
- **Account Recovery:** Features for password reset and account recovery are in place, providing users with the ability to regain access to their accounts in case of forgotten credentials.
- **Activity Monitoring:** User activities, especially related to login and critical operations, are monitored and logged for security purposes.

## 4. Front-End Implementation

- The signUp.html document creates a user registration form styled with CSS. It features fields for a username, password, first name, last name, phone number, birth date, and email. The background is a repeating image, and the design is focused on a simple and functional layout. The JavaScript code attached to the form prevents its default submission behavior and triggers a function to handle form data submission to a server. The form data is sent via a POST request to **/account/signUp** in JSON format, capturing user details for registration. Upon successful registration, the script redirects the user to a page denoted as "successAccount."
- succesAccount.html displays a message notifying the user that their account has been successfully created. The background is set as a repeating image, maintaining consistency with the previous pages. The content is centered within a container and styled for readability. The page includes a button that, when clicked, triggers a JavaScript function to redirect the user to a login page located at **/account/login**. Overall, the design focuses on clarity and provides a seamless transition for users to proceed to the login section after creating their account.



**Autentificare**

Nume utilizator

Parolă

Conectare

Creează un cont nou



**Autentificare**

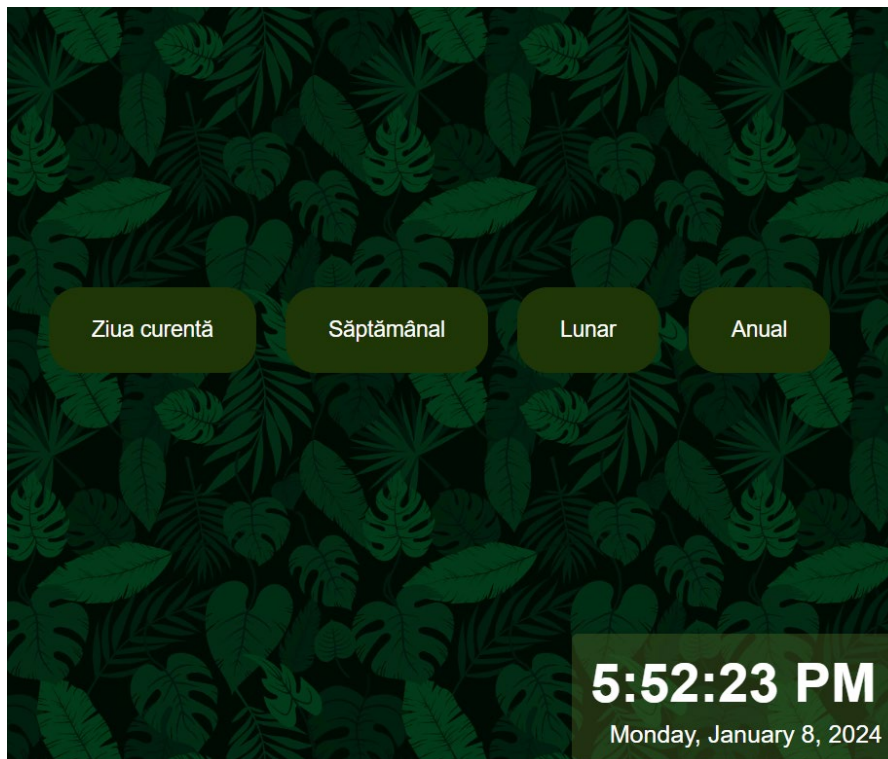
anca

.....

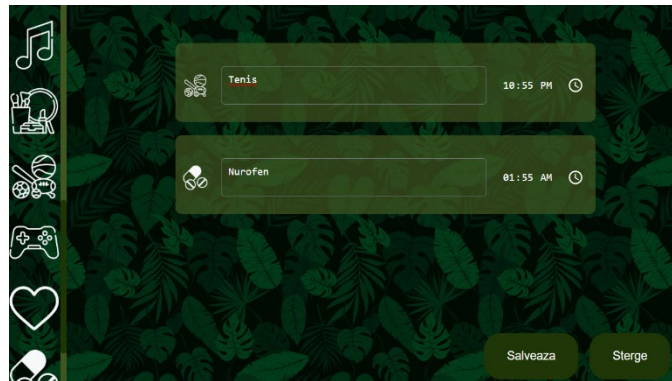
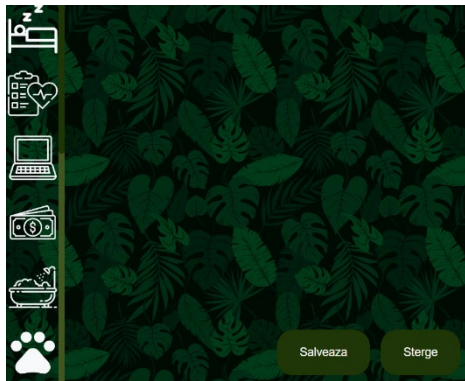
Conectare

Creează un cont nou

- logIn.html document is a basic login interface styled using CSS. It consists of a login container with fields for a username and password, along with buttons for login and creating a new account. The background is set to a repeating image, and the overall design is focused on simplicity and functionality. The login form uses JavaScript to handle user input, validate it, and perform actions like redirecting users to different pages based on their interactions. The code employs fetch requests to handle login authentication and authorization with a server, utilizing JSON web tokens (JWT) for authentication.

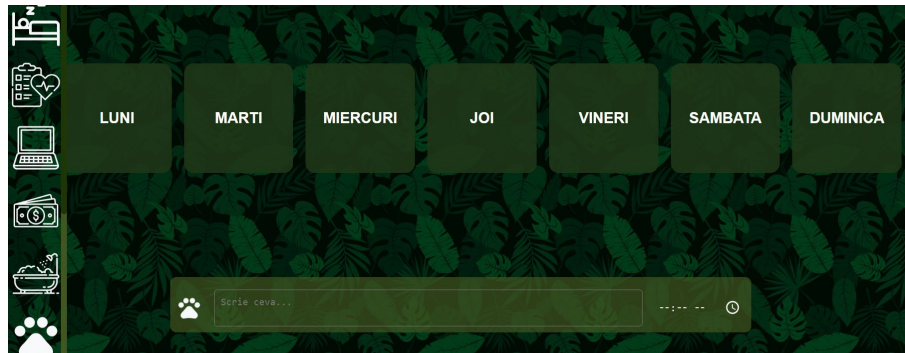
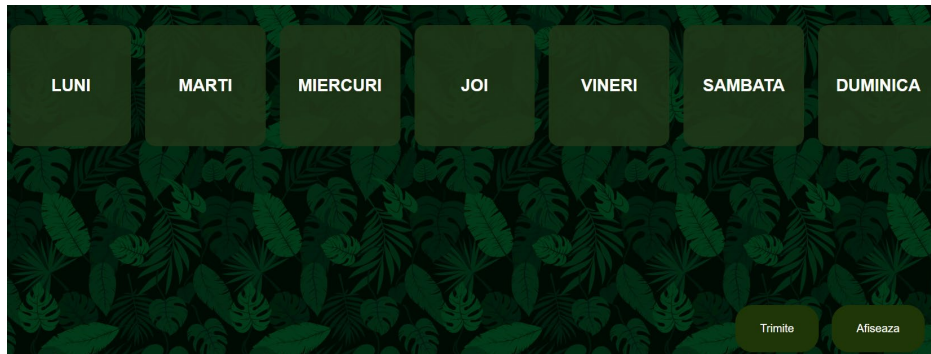


- This HTML document creates a page with four buttons styled with CSS. Each button represents a choice for different time intervals: today, weekly, monthly, and annual. The background is a repeating image, and the layout is centered on the screen. Additionally, there's a section at the bottom right corner displaying the current date and time using JavaScript's Date object.
- The JavaScript functions attached to the buttons use window.location.href to redirect users to different pages denoted as /today, /weekly, /monthly, and /annual respectively. The time and date are updated in real-time using setInterval along with toLocaleTimeString and toLocaleDateString methods.



- The today.html document creates a web interface that includes a sidebar with images, a container for creating rectangles based on selected images, and two buttons for saving and deleting selected rectangles.
- The sidebar contains several image icons. When an image is clicked, it generates a rectangular section in the #rectContainer. Each rectangle consists of an image, a textarea for writing text, and an input field for time selection.
- The createRectangle(event) function adds a new rectangle to the container when an image from the sidebar is clicked. The save() function collects data from the created rectangles, such as the image name, text, and selected time, and sends it to a server endpoint /saveData using a POST request.
- The displayTodayData(idAccount) function seems to populate the #rectContainer with previously saved data retrieved from the server endpoint /getTodayData/:idAccount. It creates rectangles similar to those generated via the sidebar, displaying images, text, and time inputs.
- Other functions like sendDayOfWeek(), sendDayOfMonth(), and deleteSelectedRectangles() handle sending specific data to the server and deleting selected rectangles respectively, interacting with server endpoints like /saveDayOfWeek, /saveDayOfMonth, and /deleteData.
- This HTML file involves a complex interface that interacts with a server to handle data management and updates based on user interactions and previously saved data.





- Weekly.html is a web page for a weekly planner or scheduler. It includes a layout with squares representing each day of the week and a sidebar with selectable images. When a day square is clicked, it displays the sidebar with images. Clicking on an image generates a rectangular section with an image, a text area, and a time input field.

Here's a breakdown of the functionality:

- The .sideBar contains images representing different tasks or events.
- .main-content includes squares representing each day of the week. When clicked, it shows the sidebar with images related to the selected day.
- When an image in the sidebar is clicked, a rectangular section with the image, a text area for notes, and a time input field is created and displayed.
- save() function seems to handle the data entered in the rectangular section, collects the image, text, time, and day selected, and prepares it to be sent to a server endpoint (/addWeeklyData) using a POST request.



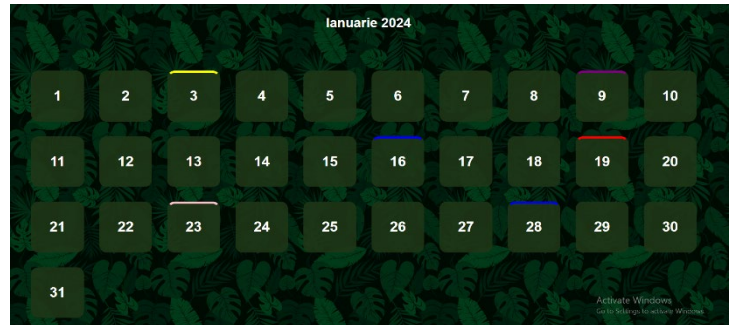
Ziua	08:00-10:00	10:00-12:00	12:00-14:00	14:00-16:00	16:00-18:00	18:00-20:00	Altele
Luni					Sedinta		
Marti	Serviciu						
Miercuri		Invatat					
Joi	Curatenie						Baie
Vineri							
Sambata							
Duminica						Concert	

Sterge  
Activeaza

Ziua	08:00-10:00	10:00-12:00	12:00-14:00	14:00-16:00	16:00-18:00	18:00-20:00	Altele
Luni					Sedinta		
Marti							
Miercuri		Invatat					
Joi	Curatenie						Baie
Vineri							
Sambata							
Duminica						Concert	

Sterge  
Activeaza

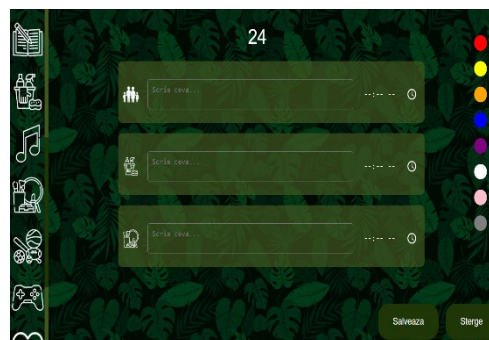
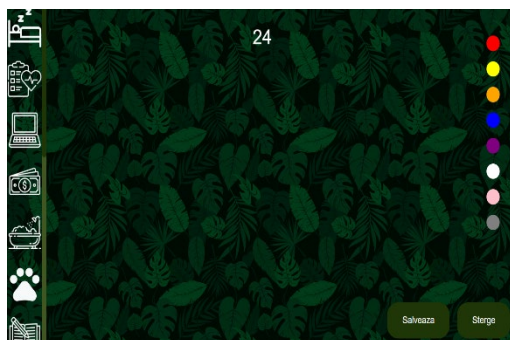
- The HTML structure involves a table with days of the week as rows and time intervals as columns. Each cell represents a specific day and time slot for activities. The JavaScript part contains functions that help populate, delete, and manage the data within the table.
- The fetchWeeklyData() function makes an asynchronous call to retrieve schedule data from the server, and populateTable() updates the HTML table with this data.
- There's also a feature to select and delete cells from the table. The toggleSelection() function highlights the selected cells, and the deleteSelectedCells() function deletes the content of the selected cells and triggers a backend call to update the database.



- The monthly.html document seems to represent a calendar where each day of the month is represented by a square element that can be clicked. When a square is clicked, it triggers a function `sendSquareNumber(number)` that sends the corresponding number (day of the month) to a server endpoint `/saveSquare` using a POST request.

Structure and functionalities:

- **HTML Structure:** It consists of a series of square div elements representing each day of the month, from 1 to 31. These divs have a class `square` and an `onclick` attribute that triggers the `sendSquareNumber()` function when clicked.
- **CSS Styling:** The styling includes setting a background image for the body and styling the square elements to make them visually appealing as calendar squares. Each square has a `border-top` color that may be dynamically set later based on data fetched from the server.
- **JavaScript Functions:**
  - `sendSquareNumber(number)`: Sends the clicked day number to the server endpoint `/saveSquare` via a POST request using the Fetch API.
  - `window.onpageshow`: Listens for the `onpageshow` event and reloads the page if it's persisted.
  - `window.onload`: When the window loads, it fetches data about colors from the server endpoint `/getColorList`. This data includes the color and the corresponding day number. The function then dynamically sets the `border-top` color of the squares based on the fetched data.



The addMonthly.html document creates a page where rectangles with images, text areas, and time inputs can be dynamically added. It includes functionalities to save, delete, and update these rectangles along with their associated data.

The functionalities and structure of the code:

- Page Structure:
  - Contains a sidebar with images that can be clicked to create rectangles.
  - Allows the selection of a color from circles displayed in a separate container.
  - Rectangles can be created with images, text areas for input, and time inputs.
  - Provides buttons for saving, deleting, and selecting rectangles.
- Functionality:
  - createRectangle(event): Creates a new rectangle upon clicking an image in the sidebar. Each rectangle includes an image, a text area, and a time input. Also, allows for the selection/deselection of rectangles.
  - save(): Gathers data from all the created rectangles (image, text, time, and day number) and sends it to a server endpoint /saveMonthly using a POST request.
  - saveColor(color): Saves the selected color and the associated day number using a POST request to /saveColor.
  - deleteMonthly(): Deletes the selected rectangles and their associated data by sending the information to the server endpoint /deleteMonthly using a DELETE request.
- Event Listeners:

- The functionality to select/deselect rectangles by clicking is added as an event listener to each rectangle.
- Initialization:
  - When the window loads, it fetches data about saved rectangles and populates the page with these rectangles along with their associated data.

The code is organized to allow the user to interact with the interface, create rectangles with different content, save these data elements, and delete selected ones. The data handling functionalities (saving, updating, deleting) are designed to communicate with a server or backend application for managing this information.

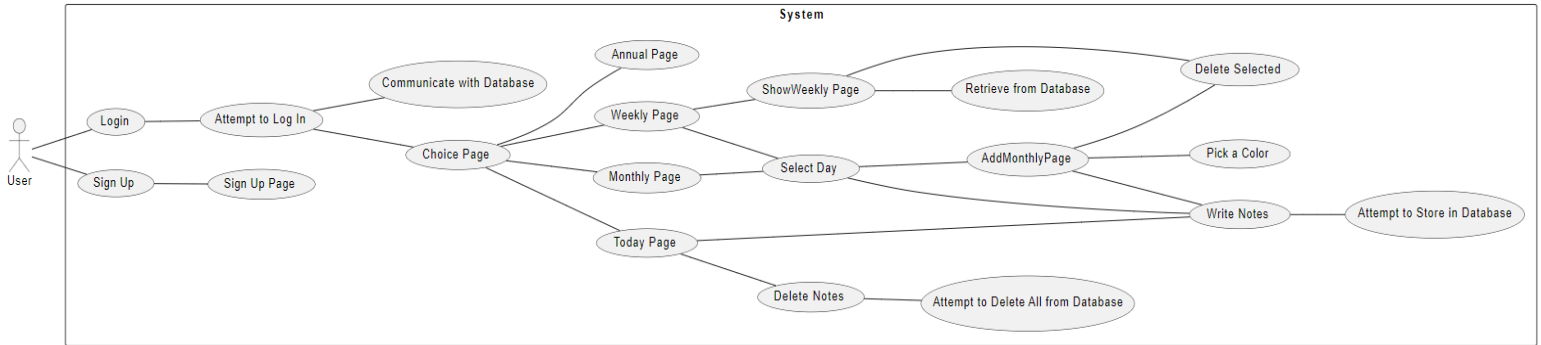


This code creates an annual calendar interface where users can view months and days for each month, select specific days, and add events to those days. Here's an overview of its structure and functionality:

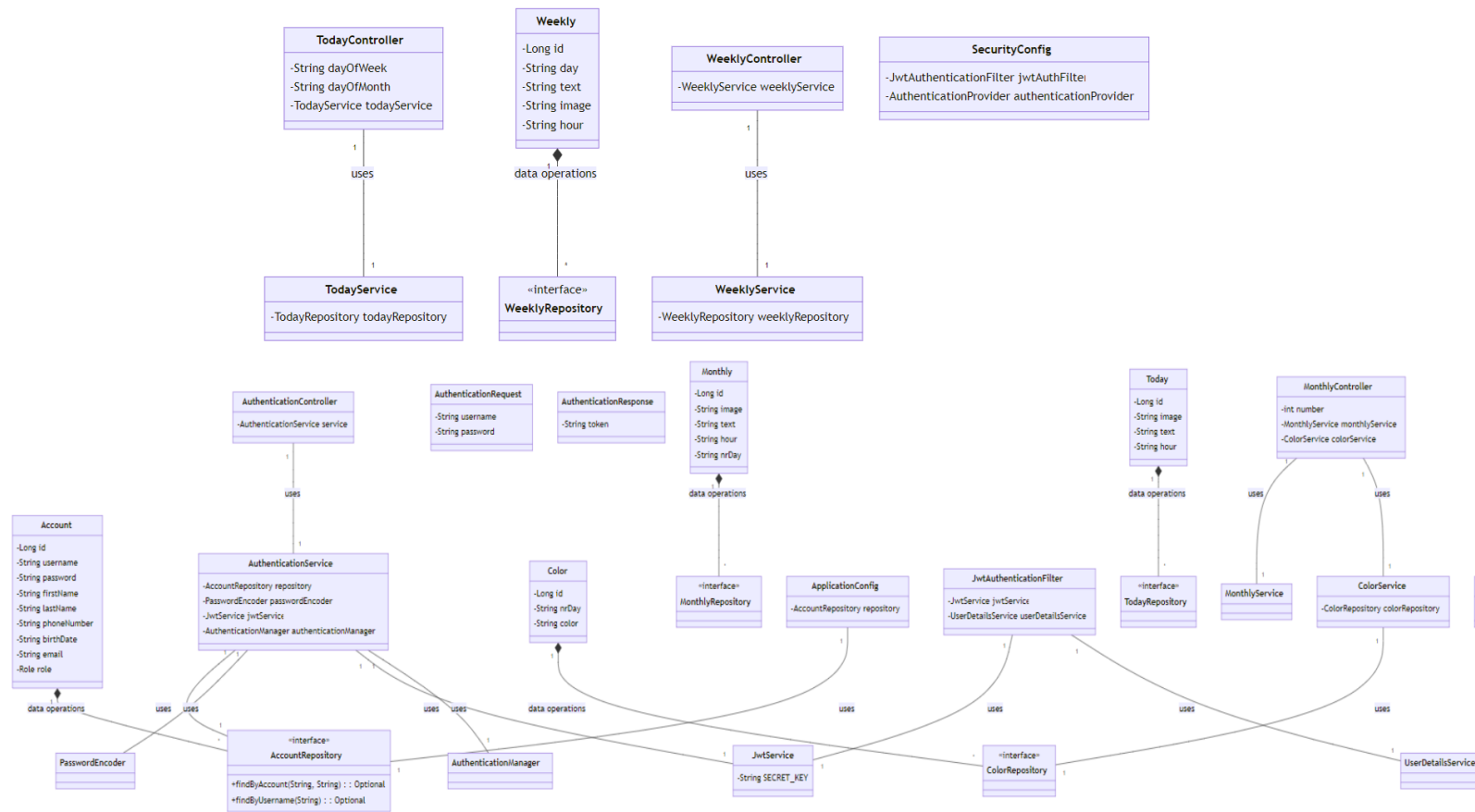
- Page Structure:
  - Displays the current year at the top of the page.
  - Creates a grid-based calendar structure for all 12 months.
  - Each month is displayed in a separate section, with days as clickable elements.
- Functionality:
  - Initializes the calendar by creating month and day elements dynamically.
  - Allows users to click on a day to select it, displaying an input field for adding events.
  - Events added to a selected day are displayed as text within that day.
  - The event input field becomes visible upon selecting a day, allowing users to add events to that day.
  - An event rectangle is displayed on the clicked day to indicate the event input area.
- JavaScript Functions:
  - `addEvent()`: Adds the text from the input field as an event to the selected day. Hides the input field and event rectangle afterward.

This calendar provides a basic structure for viewing months and days, allowing users to select days and add events to those specific dates. The events are displayed as text within the selected day.

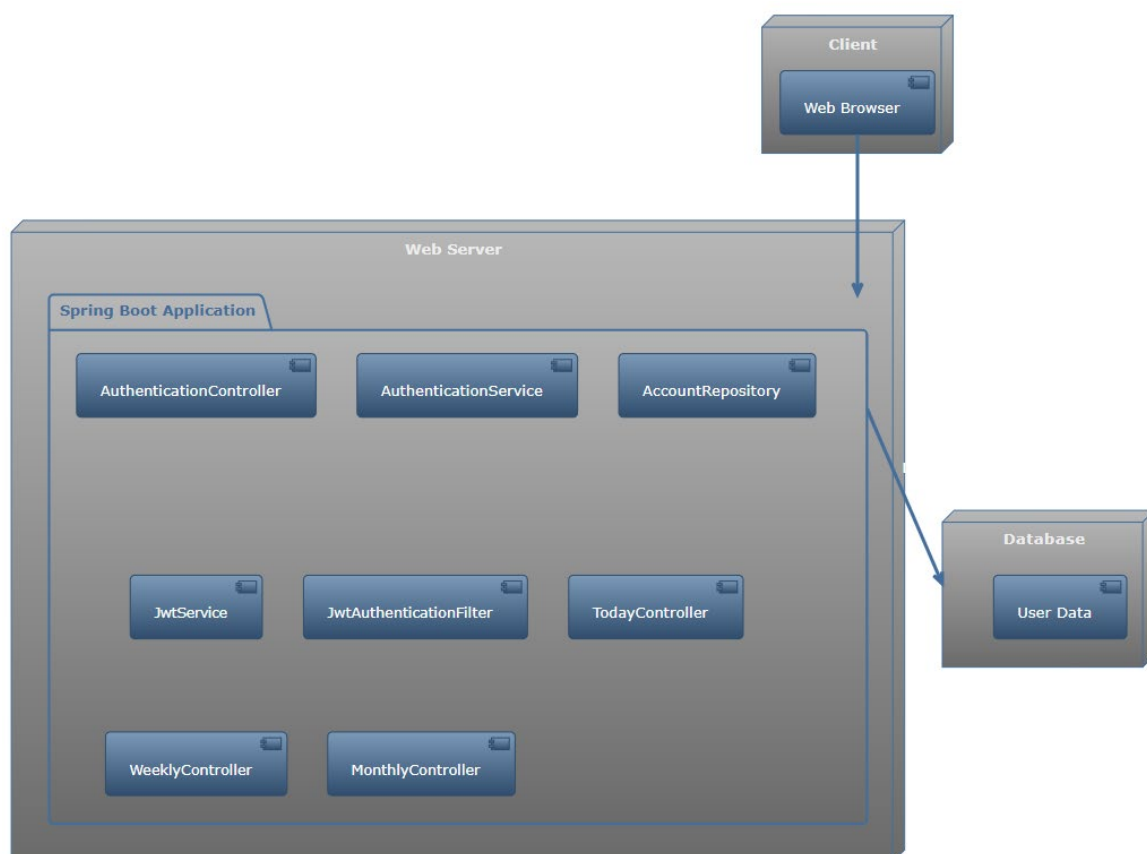
## 5. Use Cases



## 6. Class Diagram



## 7. Deployment Diagram





## 8. Conclusion

In summary, this documentation has provided a comprehensive overview of our Java Spring application, emphasizing its integration of Spring Boot and Spring Security within a robust Controller-Service-Repository architecture. Our focus on secure, efficient JWT authentication, alongside meticulous account management and security configurations, establishes a strong foundation for user data protection.

The front-end implementation, characterized by its user-friendly design, complements the backend's robustness, creating a harmonious user experience. The inclusion of use cases and diagrams has further illuminated the application's functionality and structure.

As we conclude, it's clear that our application exemplifies a successful blend of security, functionality, and user-centric design, making it a resilient and adaptable solution in the evolving landscape of web applications. This document not only serves as a detailed guide to our current system but also lays the groundwork for future enhancements and innovations.



## 9. Bibliography

<https://spring.io/guides/gs/spring-boot/>

<https://docs.spring.io/spring-security/reference/getting-spring-security.html>

<https://www.baeldung.com/spring-security-with-maven>

<https://central.sonatype.com/artifact/org.springframework.security/spring-security-web>

<https://stackoverflow.com/questions/76969773/why-spring-security-antmatchers-is-not-found>

<https://www.javacodegeeks.com/2018/03/spring-security-with-maven-tutorial.html>

[https://www.w3schools.com/html/html\\_tables.asp](https://www.w3schools.com/html/html_tables.asp)

<https://www.javatpoint.com/spring-cloud>

<https://medium.com/@AlexanderObregon/step-by-step-creating-your-first-spring-cloud-application-ef4dcfe277a4>

<https://www.geeksforgeeks.org/what-is-spring-cloud/>

<https://www.pluralsight.com/courses/spring-cloud-fundamentals>

<https://spring.io/microservices/>

<https://www.javatpoint.com/microservices>

<https://www.shecodes.io/athena/38917-how-to-display-the-current-date-and-time-in-javascript>

<https://stackoverflow.com/questions/69085519/how-can-i-display-current-time-in-html-input-type-time>

<https://docs.oracle.com/javase%2Ftutorial%2Fuiswing%2F%2F/components/paswordfield.html>

<https://stackoverflow.com/questions/13963899/how-to-retain-the-username-in-the-textbox-in-jsp-using-spring-java>

<https://hackernoon.com/using-postgres-effectively-in-spring-boot-applications>

<https://www.youtube.com/watch?v=A8qZUF-GcKo>

<https://www.youtube.com/watch?v=p485kUNpPvE&t=2716s>

<https://www.youtube.com/watch?v=QWOgkl4DuE8>

<https://www.youtube.com/watch?v=b9O9NI-RJ3o&t=119s>

<https://www.youtube.com/watch?v=KxqlJblhzfl>

[https://www.youtube.com/watch?v=her\\_7pa0vrg](https://www.youtube.com/watch?v=her_7pa0vrg)