

# BlanceBuddy - LifestyleApp

Camelia Morari

July 2024

# Contents

|          |   |          |
|----------|---|----------|
| <b>1</b> | <b>Introduction</b>                                     | <b>3</b> |
| 1.1      | Application Description . . . . .                       | 3        |
| 1.2      | Used Technologies . . . . .                             | 3        |
| 1.3      | Functionalities . . . . .                               | 3        |
| 1.4      | User Stories . . . . .                                  | 4        |
| 1.5      | Interfaces, Module Communication and Protocol . . . . . | 7        |
| 1.6      | Message Flow . . . . .                                  | 7        |
| 1.7      | Scalability and Security Considerations . . . . .       | 11       |
| 1.8      | Server Specifications: . . . . .                        | 11       |
| 1.8.1    | Security Considerations: . . . . .                      | 11       |
| 1.9      | Application Diagram . . . . .                           | 12       |
| 1.10     | Architecture Elements . . . . .                         | 13       |
| 1.10.1   | Use Case Diagram . . . . .                              | 13       |
| 1.10.2   | Class Diagram . . . . .                                 | 14       |
| 1.10.3   | Entity Relationship Diagram . . . . .                   | 14       |
| 1.10.4   | Lifecycle of objects . . . . .                          | 14       |
| 1.11     | Design Patterns . . . . .                               | 15       |
| 1.11.1   | Database Management (Singleton Pattern) . . . . .       | 15       |
| 1.11.2   | Observer Pattern . . . . .                              | 15       |
| 1.11.3   | Factory Pattern . . . . .                               | 16       |
| 1.11.4   | Decorator Pattern . . . . .                             | 16       |
| 1.12     | Prototype . . . . .                                     | 17       |
| 1.12.1   | Landing Page . . . . .                                  | 17       |
| 1.12.2   | Login Page . . . . .                                    | 17       |
| 1.12.3   | Register Page . . . . .                                 | 18       |
| 1.12.4   | Home Page . . . . .                                     | 18       |
| 1.12.5   | Goals Page . . . . .                                    | 19       |
| 1.12.6   | Current Goal Page . . . . .                             | 19       |
| 1.12.7   | Edit Goal Page . . . . .                                | 20       |
| 1.12.8   | Create New Goal Page . . . . .                          | 20       |
| 1.12.9   | Add Progress Page . . . . .                             | 21       |
| 1.12.10  | Habit Page . . . . .                                    | 21       |
| 1.12.11  | New Habit Page . . . . .                                | 22       |
| 1.12.12  | History Page . . . . .                                  | 22       |
| 1.12.13  | Report Page . . . . .                                   | 23       |
| 1.12.14  | Settings Page . . . . .                                 | 23       |
| 1.13     | Setup . . . . .   | 24       |
| 1.13.1   | Prerequisites . . . . .                                 | 24       |
| 1.13.2   | Configuration . . . . .                                 | 24       |
| 1.13.3   | Running the Application . . . . .                       | 25       |

# 1 Introduction

## 1.1 Application Description

BalanceBuddy is a mobile application designed to help users improve their lifestyle by monitoring their daily habits. The app tracks various aspects of a person's daily routine such as steps taken, water intake, coffee consumption and food habits. The user will be able to input a goal in the beginning of a day regarding any desired field. At the end of each day, a report is sent to the user, summarizing their activities, progress towards the goal and some recommendations for improvement. BalanceBuddy aims to encourage users change their lifestyle into a healthier and more balanced one.

## 1.2 Used Technologies

The project is built on Windows 10, and the technologies used for this application are:

- Frontend: React Native 0.73 + Node.js v22
- Backend: Java 17 + Spring boot 3.2.3
- Database: MySQL - version 8.0.31
- API Testing: Postman - version 11.4.0
- Authentication: JWT (JSON Web Tokens)
- Version Control: GitHub - version 3.2

These technologies have been selected because they make up a robust and efficient technology stack that ensures scalability, flexibility and security for the application. React Native is chosen for the frontend being a great software framework for building mobile applications using native UI components. JavaSpring is employed for the backend because of its powerful features for building scalable and reliable applications. MySQL is used as the database for its performance and ease of integration with JavaSpring. Postman is an excellent tool for API testing, offering a user-friendly interface and comprehensive testing capabilities. JWT is utilized for providing a secure and stateless method of verifying users. Finally, Git is selected for version control and dynamic development due to its powerful branching and merging capabilities.

## 1.3 Functionalities

The main functionalities of the application are:

- **Authentication:**  
This functionality allows users to create a new account, to log in or to log out from an already existing account. Users can sign up by providing their email and password. They can log in using their credentials and log out when finished. Sessions are managed using JWT tokens.
- **Habit Tracking:**  
This functionality enables users to track their daily habits. The app collects the data regarding steps, which are automatically recorded using a sensor, water and food intake, coffee consumption and others by manual entry.
- **Goal Setting:**  
Users can set and manage their personal goals for various habits. They can define targets for each habit and update them later on. A notification system will remind users to work towards their goals.
- **Daily Reports:**  
A detailed summary is generated at the end of each day, including the user activities throughout the day and their progress. A recommendation is offered based on the user's performance alongside an encouraging quote.
- **User History:**  
A dashboard that provides a comprehensive overview of the user's progress with visual representations such as graphs and charts. These represent the progress and the user is able to view their past activities and habits.

- **Notifications:**

Notifications are sent from time to time to keep users engaged and to remind them to stay on track with their goals. An extra notification is sent at the end of the day to inform the user when their daily report is ready.

## 1.4 User Stories

### 1. USER STORY #1 - User Registration

**As a** new user,

**I want to** create an account,

**So that** I can start using the application.

Acceptance Criteria:

AC#1 - Link to Register

Scenario: A new user is navigating to the Login page.

**Given** I am a new user,

**When** I go to the Login page,

**Then** I should see a hyperlink to Register.

AC#2 - Register Form

Scenario: A new user is accessing the registration form.

**Given** I am a new user,

**When** I follow the hyperlink to Register,

**Then** I should see a registration form with the following fields:

| FieldName        | FieldType                                      | Mandatory/Optional Field |
|------------------|--|--------------------------|
| Firstname        | Free text, [6, 256] characters                 | Mandatory                |
| Lastname         | Free text, [6, 256] characters                 | Mandatory                |
| Email            | Free text, max 256 chars, email format         | Mandatory                |
| Password         | Special chars("*, min 6 chars, password format | Mandatory                |
| Confirm Password | Should be the same with "Password" field       | Mandatory                |

AC#3 - New Account Creation

Scenario: A new user has successfully created an account.

**Given** I am a new user,

**When** I fill out the registration form correctly and click Register,

**Then** I should see a "Registration Successful" message, and my account should be created.

Possible errors:

- If the required fields do not meet the validation criteria, an error message is displayed on the screen informing the user what should be changed.
- If some missing fields are detected, another error message is displayed, informing the user.
- If the passwords do not match, a new error message is shown to the user.

### 2. USER STORY #2 - User Login/Logout

**As a** registered user,

**I want to** log in and out from the application,

**So that** my data is protected.

Acceptance Criteria:

AC#1 - Login Form

Scenario: A user who is already registered wants to access its account, by filling up the login form.

**Given** I am a registered user,

**When** I go to the Login page,

**Then** I should see a login form with fields for Username and Password.

AC#2 - Successful Login

Scenario: Registered user logging in successfully.

**Given** I am a registered user,  
**When** I enter valid credentials and click Login,  
**Then** I should be redirected to the Main page of the application.

AC#3 - Logout

Scenario: Logged-in user wants to log out.

**Given** I am a logged-in user,  
**When** I click on the Logout button,  
**Then** I should be logged out and redirected to the Login page.

### 3. USER STORY #3 - Daily Steps Tracking

**As a** user,  
**I want to** track my daily steps,  
**So that** I can monitor my physical activity.

Possible errors:

- If some missing fields are detected, another error message is displayed, informing the user.
- If the credentials are invalid, an error message appears informing the user about the mistake.

Acceptance Criteria:

AC#1 - Steps Input

Scenario: A user wants to track its daily steps.

**Given** I am a logged-in user,  
**When** I go to the Tracking steps page,  
**Then** I should see a field displaying the total number of steps I made today.

### 4. USER STORY #4 - Water Intake Logging

**As a** user,  
**I want to** log my water intake,  
**So that** I stay hydrated.

Acceptance Criteria:

AC#1 - Water Intake Input

Scenario: A user wants to track its daily water intake.

**Given** I am a logged-in user,  
**When** I go to the Water Intake page,  
**Then** I should see an input field to enter my daily water intake in liters or glasses or other measuring technique.

AC#2 - Water Intake Logging

Scenario: User successfully logging water intake.

**Given** I am a logged-in user,  
**When** I enter my water intake and click Save,  
**Then** my water intake should be logged and displayed in my activity history.

Possible errors:

- Input is a non-numeric value, so an error is thrown.

### 5. USER STORY #5 - Goal Setting

**As a** user,  
**I want to** set daily, weekly or monthly, goals  
**So that** I can improve my lifestyle.

Acceptance Criteria:

AC#1 - Goal Setting Form

Scenario: User wants to access the goal setting form in order to set up a personal goal.

**Given** I am a logged-in user,

**When** I go to the Goal Setting page,  
**Then** I should see a form to set goals for different parameters.

AC#2 - Save Goals

Scenario: User saves newly created goal.

**Given** I am a logged-in user,  
**When** I set my goals and click Save,  
**Then** my goals should be saved and displayed on my Dashboard.

AC#3 - Edit Goals

Scenario: User wants to modify something to the existing goal.

**Given** I am a logged-in user,  
**When** I edit my goals and click Save,  
**Then** my goals should be saved and displayed on my Dashboard.

Possible errors:

- Input is a non-numeric value, so an error is thrown.

#### 6. USER STORY #6 - Daily Report

**As a** user,  
**I want to** receive a daily report of my activity,  
**So that** I can analyze my progress.

Acceptance Criteria:

AC#1 - Daily Report View

Scenario: User wants to see its daily report.

**Given** I am a logged-in user,  
**When** I go to the Daily Report page,  
**Then** I should see a summary of my activities throughout the day.

#### 7. USER STORY #7 - User History

**As a** user,  
**I want to** see a overview of my progress,  
**So that** I can easily track my habits.

Acceptance Criteria:

AC#1 - History View

Scenario: User wants to view the history of its past activities.

**Given** I am a logged-in user,  
**When** I go to the History page,  
**Then** I should see a detailed history of my activity including graphical representations like graphs and charts.

#### 8. USER STORY #8 - Notifications

**As a** user,  
**I want to** receive timely notifications,  
**So that** I do not forget to input parameters or see the daily report.

Acceptance Criteria:

AC#1 - Notification Settings

Scenario: User wants to receive notifications from the application.

**Given** I am a logged-in user,  
**When** I go to the Notification Settings page,  
**Then** I should see options to set reminders for receiving notifications.

AC#2 - Notification Alerts

Scenario: User receives notifications in case of an event.

**Given** I am a logged-in user,  
**When** it is time to log steps or water intake,  
**Then** I should receive a notification reminding me to do so.

## 1.5 Interfaces, Module Communication and Protocol

- **Login/Register:** Here the users can sign up or log in filling up the required fields.
- **User History:** Displays the user's progress with charts and graphs
- **Goal Setting:** Users can set and edit their daily, weekly or monthly goals.
- **Daily Report Page:** Shows a summary of the user's activities during the day and progress towards their goals.
- **Tracking Page:** Allows user to log in their tracking parameters.

The user can freely navigate between the pages as long as it is logged in. The Frontend-Backend communication is handled via RESTful APIs over HTTPS. The frontend sends requests to the backend, which processes the data and returns responses. For the authentication, JWT tokens are used to manage user sessions and secure API endpoints.

The protocol used for this application is HTTPS for the connection between server and client and for sending requests, on port 80.

## 1.6 Message Flow

### 1. User log in:

- **Description:** The user enters their username and password into the login form.
- **Type of Operation:** POST
- **Endpoint:** POST /api/auth/login
- **Request Body:**

```
{
  "username": "user@example.com",
  "password": "password123"
}
```
- **Expected Body Response:**

```
{
  "token": "JWT_TOKEN_HERE"
}
```
- **Expected Status Code:** 200 OK

### 2. User registration:

- **Description:** User enters their information to create a new account.
- **Type of Operation:** POST
- **Endpoint:** POST /api/auth/register
- **Request Body:**

```
{
  "firstname": "John",
  "lastname": "Doe",
  "username": "newuser@example.com",
  "password": "password123",
  "confirmPassword": "password123"
}
```
- **Expected Body Response:**

```
{
  "id": "NEW_USER_ID"
}
```

- **Expected Status Code:** 201 Created

3. User enters homepage:

- **Description:** User accesses the homepage.
- **Type of Operation:** GET
- **Endpoint:** GET /homepage
- **Request Body:** None
- **Expected Body Response:** HTML content of the homepage
- **Expected Status Code:** 200 OK

4. User views daily report:

- **Description:** User selects to view their daily report.
- **Type of Operation:** GET
- **Endpoint:** GET /dailyReport/{user\_id}
- **Request Body:** None
- **Expected Body Response:** HTML content of the daily report page
- **Expected Status Code:** 200 OK

5. Add a habit:

- **Description:** User inputs information regarding a new habit.
- **Type of Operation:** POST
- **Endpoint:** POST /habit/add
- **Request Body:**

```
{
  "userId": "12345",
  "habitType": "waterIntake",
  "value": 250,
  "date": "07-08-2024"
}
```

- **Expected Body Response:**

```
{
  "status": "success",
  "message": "Habit added successfully"
}
```
- **Expected Status Code:** 200 OK

6. Log Sleep:

- **Description:** User inputs parameters regarding their sleeping schedule.
- **Type of Operation:** POST
- **Endpoint:** POST /sleep/log
- **Request Body:**

```
{
  "userId": "12345",
  "habitType": "waterIntake",
  "value": 250,
  "date": "07-08-2024"
}
```



- **Expected Body Response:**

```
{
  "status": "success",
  "message": "Sleep logged successfully"
}
```

- **Expected Status Code:** 200 OK

#### 7. Tracking Exercise:

- **Description:** User logs details about their physical exercise.
- **Type of Operation:** POST
- **Endpoint:** POST /exercise/log
- **Request Body:**

```
{
  "userId": "67890",
  "exerciseType": "running",
  "duration": 30,
  "date": "07-08-2024"
}
```

- **Expected Body Response:**

```
{
  "status": "success",
  "message": "Exercise logged successfully"
}
```

- **Expected Status Code:** 200 OK

#### 8. Set a goal:

- **Description:** User sets a daily goal for a specific habit.
- **Type of Operation:** POST
- **Endpoint:** POST /goal/set
- **Request Body:**

```
{
  "userId": "12345",
  "habitType": "steps",
  "targetValue": 10000,
  "period": "daily"
}
```

- **Expected Body Response:**

```
{
  "status": "success",
  "message": "Goal set successfully"
}
```

- **Expected Status Code:** 200 OK

#### 9. Edit a goal:

- **Description:** User edits an existing goal.
- **Type of Operation:** PUT
- **Endpoint:** PUT /goal/edit
- **Request Body:**

```
{
  "userId": "12345",
  "habitType": "steps",
  "targetValue": 12000,
  "period": "daily"
}
```

- **Expected Body Response:**

```
{
  "status": "success",
  "message": "Goal edited successfully"
}
```

- **Expected Status Code:** 200 OK

#### 10. Delete a goal:

- **Description:** User deletes a goal.

- **Type of Operation:** DELETE

- **Endpoint:** POST /goal/delete

- **Request Body:**

```
{
  "userId": "12345",
  "habitType": "steps"
}
```

- **Expected Body Response:**

```
{
  "status": "success",
  "message": "Goal deleted successfully"
}
```

- **Expected Status Code:** 200 OK

#### 11. Generate a Daily Report:

- **Description:** Server generates a daily report for the user.

- **Type of Operation:** POST

- **Endpoint:** POST /report/generate

- **Request Body:**

```
{
  "userId": "12345",
  "date": "07-08-2024"
}
```

- **Expected Body Response:**

```
{
  "report": "REPORT_CONTENT"
}
```

- **Expected Status Code:** 200 OK

#### 12. Send Reminder Notification:

- **Description:** Server sends a reminder notification to the user.

- **Type of Operation:** POST

- **Endpoint:** POST /notification/reminder

- **Request Body:**

```
{
  "userId": "12345",
  "type": "waterIntakeReminder"
}
```

- **Expected Body Response:**

```
{
  "status": "success",
  "message": "Notification sent successfully"
}
```

- **Expected Status Code:** 200 OK

## 1.7 Scalability and Security Considerations

### 1.8 Server Specifications:

- CPU: 6 cores
- Logical processors: 12
- RAM: 32GB
- Storage: 250GB SSD

#### **Load Capacity:**

- Concurrent Users: up to 5000
- Requests per Second: up to 500

#### 1.8.1 Security Considerations:

##### **User Authentication and Authorization**

- **JWT Tokens:** Use JSON Web Tokens (JWT) for secure user authentication. JWTs are signed using the HS256 algorithm and include an expiration time to limit their validity.
- **Token Refresh:** Mechanism to refresh expired token that allows users to obtain a new JWT without re-authenticating.

##### **Data Encryption:**

- **HTTPS:** Ensure all communication between the client and server is encrypted using HTTPS to protect data.
- **Encryption at Rest:** Encrypt sensitive data stored in the database, such as passwords and user information.
- **Password Hashing:** Hash user passwords using a bcrypt, a hashing algorithm, before storing them in the database to prevent plaintext password storage.

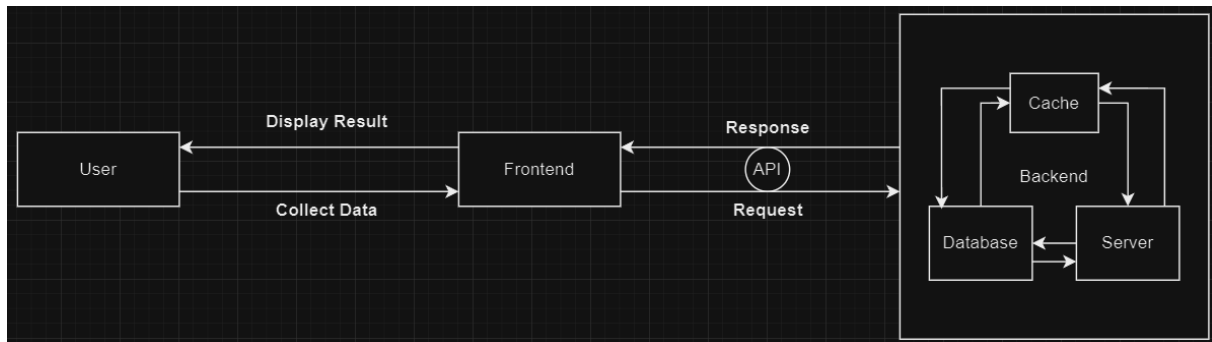
##### **Security Input Handling**

- **SQL Injection:** Use parameterized queries to protect against SQL injection attacks.
- **Hidden Password:** The password is not displayed, as well as multiple password criteria are applied (for example, the password must have at least 6 characters, at least one uppercase, one special character and one number).

##### **Error Handling:**

- **Generic Error Messages:** Avoid displaying detailed error messages to users. Use generic error messages to prevent leaking sensitive information about the system.
- **Error Logging:** Log detailed error messages on the server side for debugging and analysis purposes.

## 1.9 Application Diagram



**The User** interacts with the application through a user interface provided by the frontend. Actions such as collecting data (input habits, set goals, etc.) and displaying results (viewing reports, notifications, etc.) are performed by the user.

**The Frontend** is responsible for the user interface of the application. It collects data from the user and sends requests to the backend via API. It receives responses from the backend and displays the results to the user.

**The API** serves as a communication bridge between the frontend and the backend. It handles requests from the frontend and routes them to the appropriate backend services. It sends responses back to the frontend after processing the requests.

**The Backend** processes requests sent by the API and interacts with other backend components like the cache, database, and server. It contains the business logic and manages the core functionalities of the application.

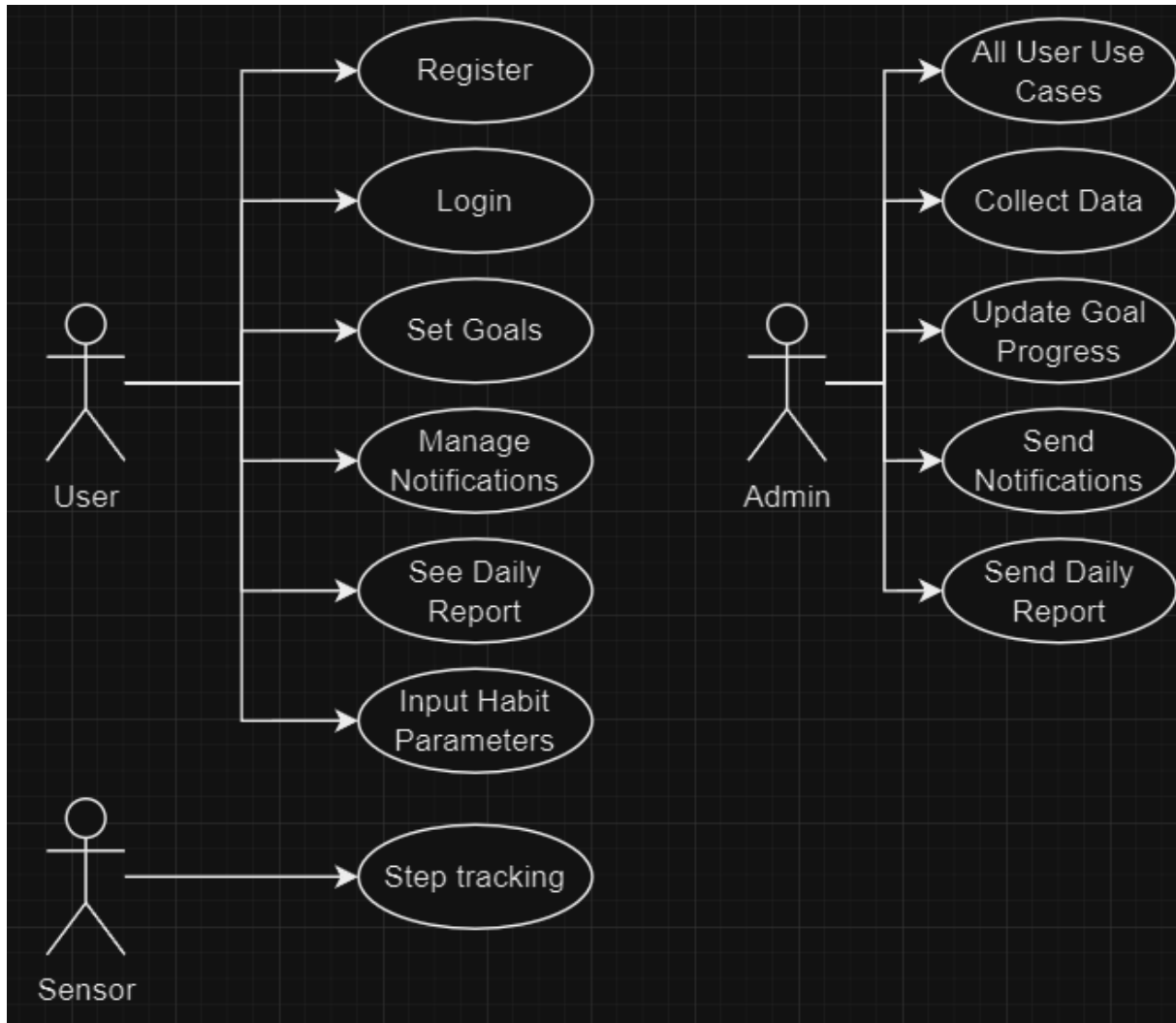
**The Server** hosts the backend application and serves as the primary component that processes requests. It handles the logic and data manipulation based on user requests.

**The Database** stores all the persistent data of the application, such as user information, goals, habits, and reports. The backend queries and updates the database to manage the data. MySQL is used for managing the database.

**The Cache** is used to store frequently accessed data, current goals and habits, temporarily to reduce the load on the database and improve performance. It provides faster data retrieval for the backend services.

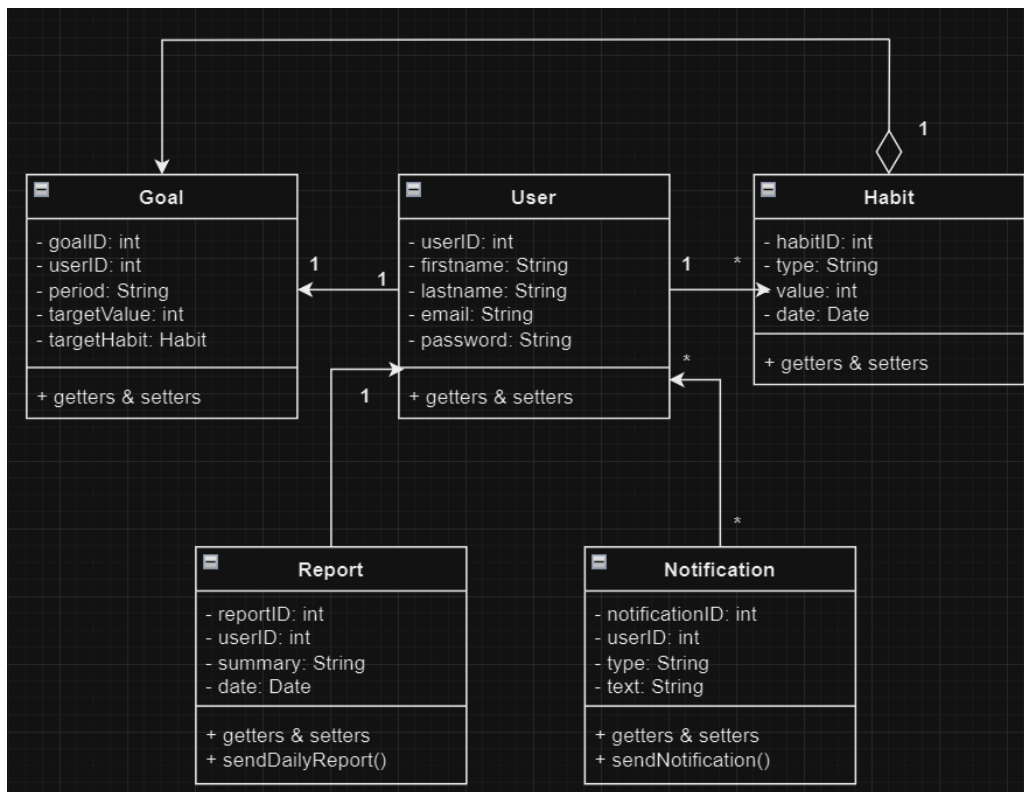
## 1.10 Architecture Elements

### 1.10.1 Use Case Diagram



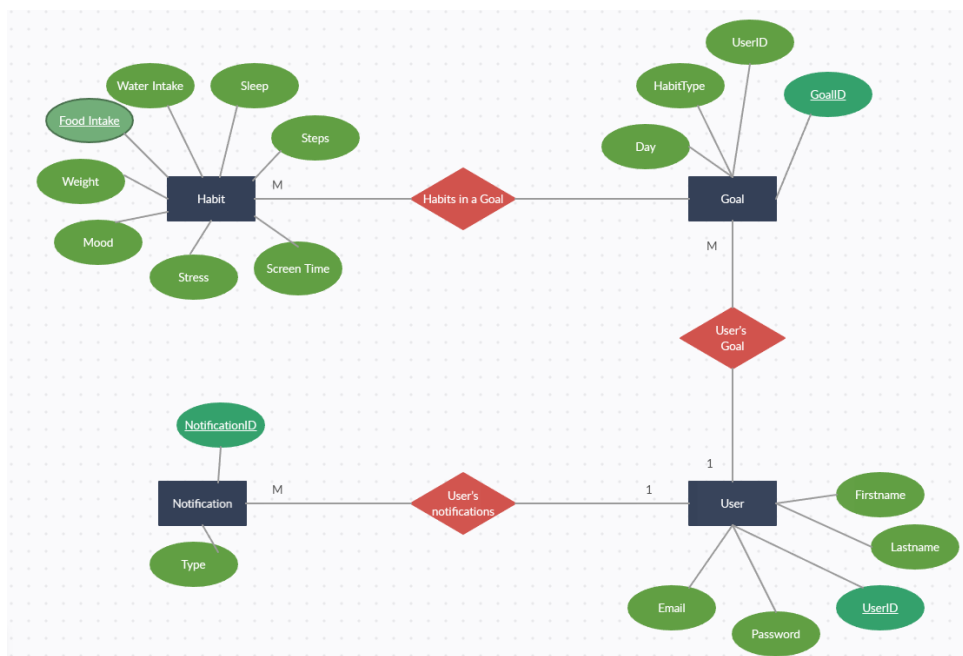
The use case diagram illustrates the actions that each actor of the application can do. The present actors are the User, Sensor and the Admin. The User represents the end-users of the BalanceBuddy application who interact with the system to manage their lifestyle habits. The Sensor represents the hardware or software sensors that track user steps. Finally, the Admin represents the backend server that processes data, sends notifications and manages user goals and reports. A user is able to register, login, logout, set goals, see the daily report and input habit parameters. The sensor sends the detected number of steps to the server, while the admin can do everything a user can, and also it collects all the data, updates the progress of the user's goal, sends notifications and daily reports.

### 1.10.2 Class Diagram



This is an orientative class diagram for the application. It provides a clear overview of the structure and interactions between different entities in the application highlighting how user data is managed, how habits and goals are tracked and how reports and notifications are generated and sent.

### 1.10.3 Entity Relationship Diagram



### 1.10.4 Lifecycle of objects

#### 1. User object

- **Creation:** When a user registers, a new User object is created.

- **Usage:** The User objects is used for login, setting goals, inputting habits, viewing reports and managing notifications.
- **Deletion:** When a user account is deleted, the User objects is removed form the database.

## 2. Habit object

- **Creation:** When a user logs a new habit, a Habit object is created.
- **Usage:** The Habit object is used to store and track habit data such as water intake, food consumption and so on.
- **Deletion:** When a habit entry is deleted by the user, the Habit object is removed from the database.

## 3. Goal Object

- **Creation:** When a user sets a new goal, a Goal object is created.
- **Usage:** The Goal object is used to track user goals and progress towards achieving them.
- **Deletion:** When a goal is deleted or achieved by the user, the Goal object is removed from the database.

## 4. Report Object

- **Creation:** When a daily report is generated, a Report object is created.
- **Usage:** The Report object is used to summarize the user's daily activities and progress.
- **Deletion:** Reports are stored for the user dashboard/history system, but can be deleted based on user preferences.

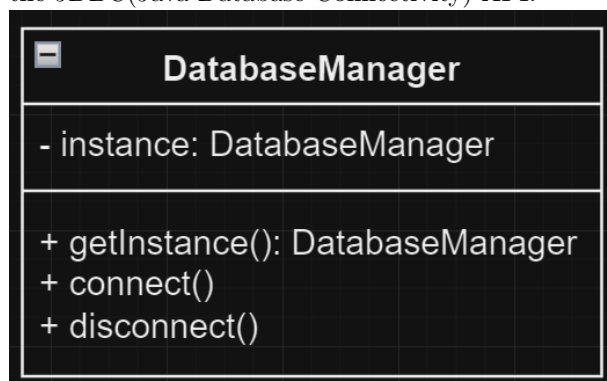
## 5. Notification Object

- **Creation:** When a notification is triggered, a Notification object is created.
- **Usage:** The Notification object is used to manage and send reminders and alerts to the user.
- **Deletion:** Notifications are removed after they are sent.

# 1.11 Design Patterns

## 1.11.1 Database Management (Singleton Pattern)

The Singleton pattern is used for managing the database connection and to ensure there is a single instance of the connection used across the application. The connection to the database is achieved using the JDBC(Java Database Connectivity) API.

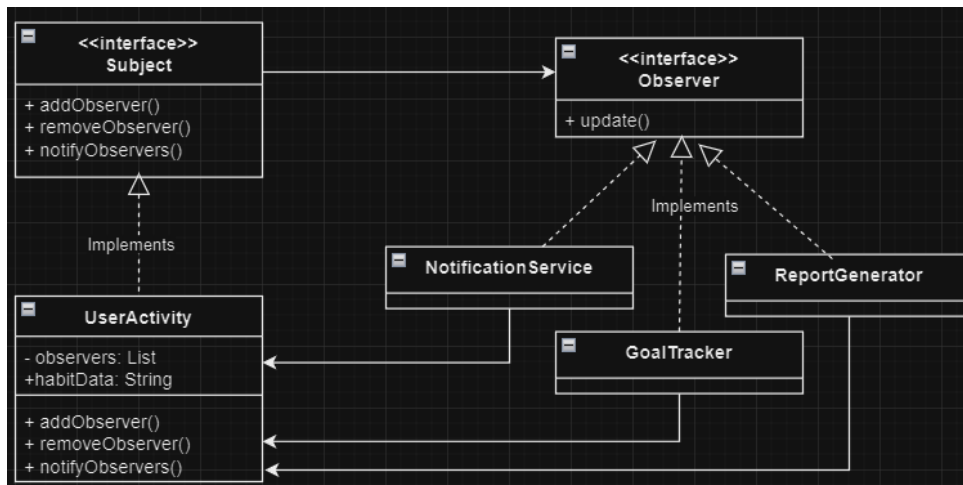


## 1.11.2 Observer Pattern

The Observer pattern is used for implementing the notification system where users can subscribe to certain habit reminders, to update the progress to a goal and to generate the report.

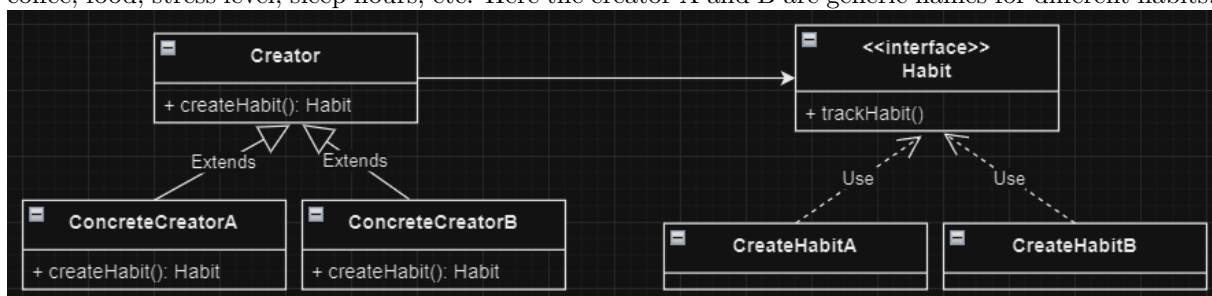
The subject is the `UserActivity` class, which holds the state and notifies observers when the state changes. When the `UserActivity` state changes, it notifies all attached observers, which then perform their specific updates. The observers are:

- NotificationService: Sends notifications to users based on habit data changes.
- ReportGenerator: Generates reports based on user activity.
- GoalTracker: Updates goal progress based on habit data changes.



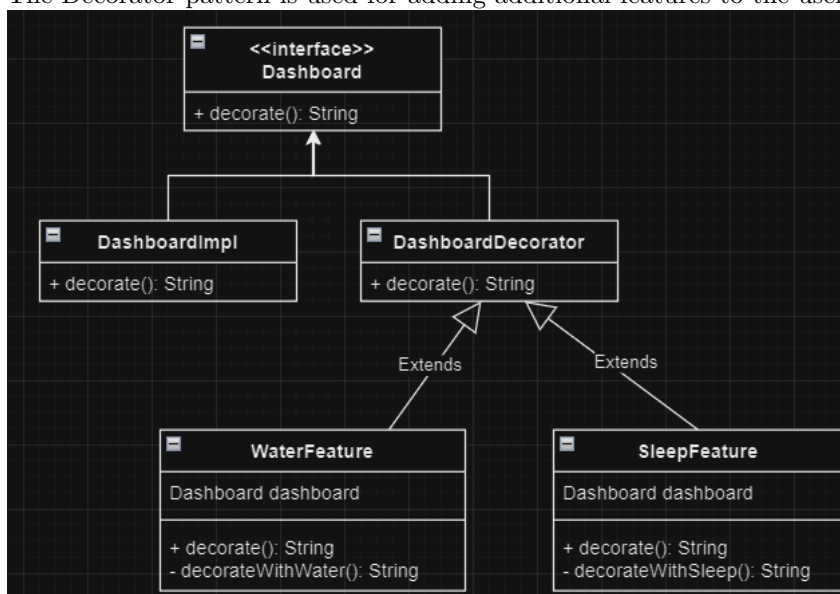
### 1.11.3 Factory Pattern

The Factory pattern is used for creating different types of habit log entries, for example steps, water, coffee, food, stress level, sleep hours, etc. Here the creator A and B are generic names for different habits.



### 1.11.4 Decorator Pattern

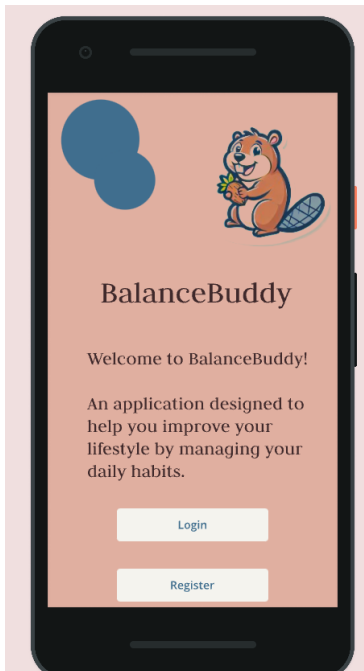
The Decorator pattern is used for adding additional features to the user dashboard dynamically.





## 1.12 Prototype

### 1.12.1 Landing Page

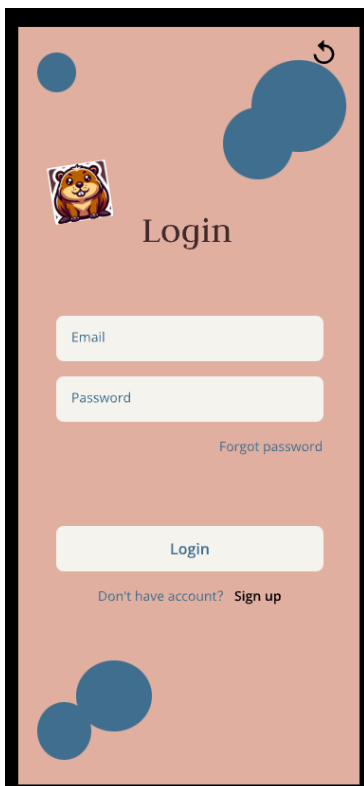


#### Landing Page Description

The landing page is the first interface that users encounter when they access the BalanceBuddy application. It provides a welcoming overview. The primary elements on this page include:

- A brief introduction to the app and its benefits.
- Button to log in.
- Button to register
- Visual elements to engage users and make navigation intuitive.

### 1.12.2 Login Page

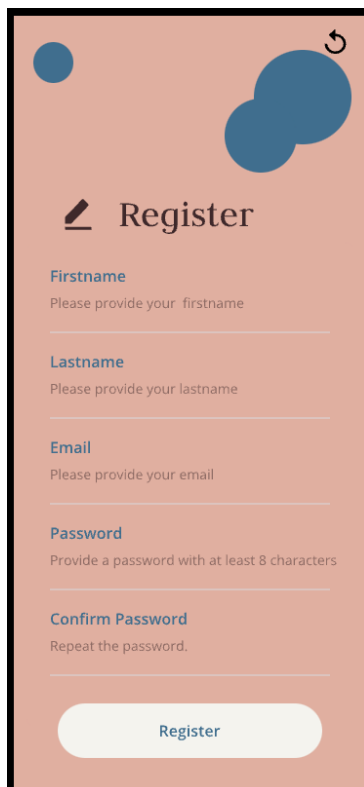


#### Login Page Description

The login page allows users to access their personal BalanceBuddy account. It includes:

- Fields for entering email and password.
- A link to reset the password if the user has forgotten it.
- A button to submit the login form.
- A link to the registration page for new users.
- A button for going back to the landing page.

### 1.12.3 Register Page

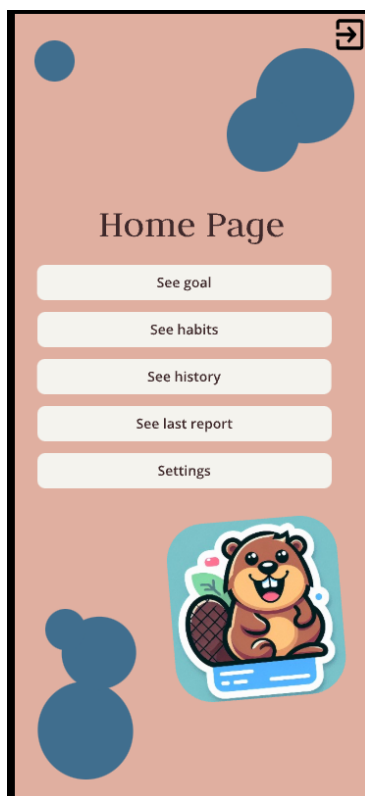
A mobile app mockup of a registration page. The background is a solid light orange color. At the top, there are three blue decorative circles of varying sizes. Below them, the word "Register" is displayed in a dark brown font, preceded by a small icon of a pencil and paper. The form consists of five input fields, each with a label in blue and a placeholder text in a smaller, lighter font. The fields are: "Firstname" (placeholder: "Please provide your firstname"), "Lastname" (placeholder: "Please provide your lastname"), "Email" (placeholder: "Please provide your email"), "Password" (placeholder: "Provide a password with at least 8 characters"), and "Confirm Password" (placeholder: "Repeat the password."). At the bottom of the form is a white rounded rectangular button with the word "Register" in blue. A small circular arrow icon is located in the top right corner of the page.

#### Register Page Description

The register page allows new users to create a BalanceBuddy account. It includes:

- Fields for entering personal information such as name, email, and password.
- A button to submit the registration form.
- A button for going back to the landing page.

### 1.12.4 Home Page

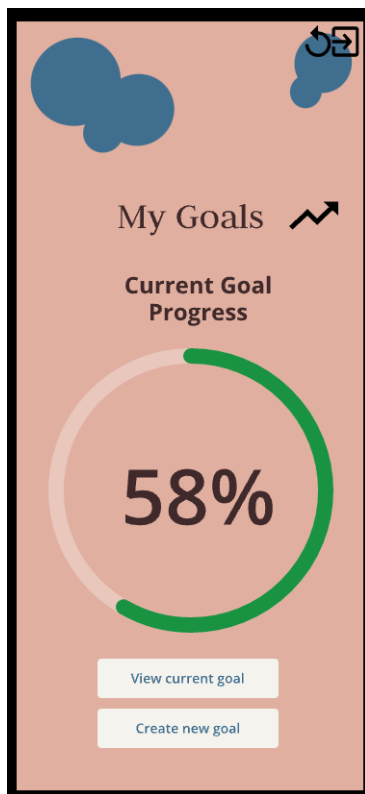


#### Home Page Description

The home page provides users with an overview of the application pages. It includes:

- A button directing the users to the goal page.
- A button directing the users to the habits page.
- A button directing the users to the history page.
- A button directing the users to the settings page.
- A button directing the users to the report page.
- A button for logging out.

#### 1.12.5 Goals Page



##### Goals Page Description

The goals page allows users to view and manage their lifestyle goals. It includes:

- The progress of the current goal.
- Option to view the current goal.
- A button to create new goals.
- Buttons for logging out and going back to the last accessed page.

#### 1.12.6 Current Goal Page

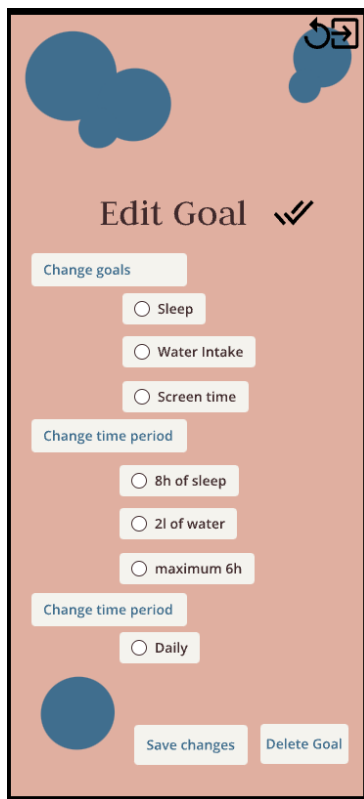
The screenshot shows a mobile app interface for 'Current Goal'. At the top, there's a title 'Current Goal' with a checkmark icon. Below it, the text 'Habits I want to improve:' is displayed. There are three radio button options: 'Sleep', 'Water Intake', and 'Screen time'. Below these, the text 'My goal is to achieve:' is displayed. There are three radio button options: '8h of sleep', '2l of water', and 'maximum 6h'. Below these, the text 'Daily.' is displayed. At the bottom, there's a motivational phrase 'I CAN DO THIS!' and three buttons: 'Edit Goal', 'Create new goal', and 'Add progress'. The background is a light orange color with blue cloud-like shapes in the top left and top right corners.

##### Current Goal Page Description

The current goal page provides details of a specific goal. It includes:

- The target habit and its parameters.
- Summary of the goal.
- Option to edit the goal.
- A button to create new goals.
- A button to add the progress to the goal.
- Buttons for logging out and going back to the last accessed page.

### 1.12.7 Edit Goal Page



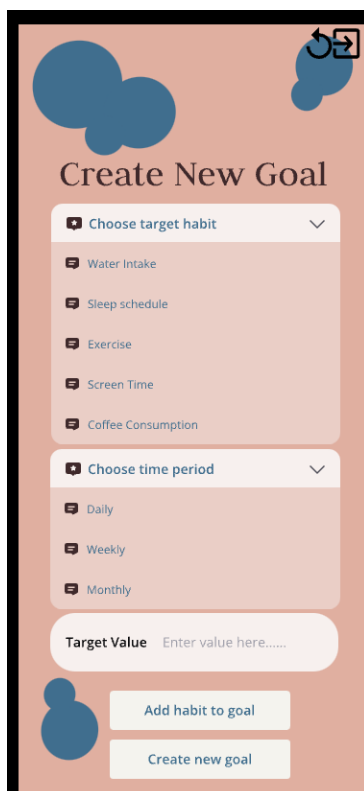
The 'Edit Goal' page features a light orange background with blue cloud-like shapes at the top. A title bar at the top contains the text 'Edit Goal' and a checkmark icon. Below this, there are three sections for editing goal parameters: 'Change goals' with radio buttons for 'Sleep', 'Water Intake', and 'Screen time'; 'Change time period' with radio buttons for '8h of sleep', '2l of water', and 'maximum 6h'; and another 'Change time period' section with a radio button for 'Daily'. At the bottom, there are two buttons: 'Save changes' and 'Delete Goal'. A blue circular profile picture placeholder is located on the left side of the bottom section.

#### Edit Goal Page Description

The edit goal page allows users to modify an existing goal. It includes:

- Fields to update the goal parameters such as target habit, value, and period.
- A button to save the changes.
- A button to delete the goal.
- Buttons for logging out and going back to the last accessed page.

### 1.12.8 Create New Goal Page



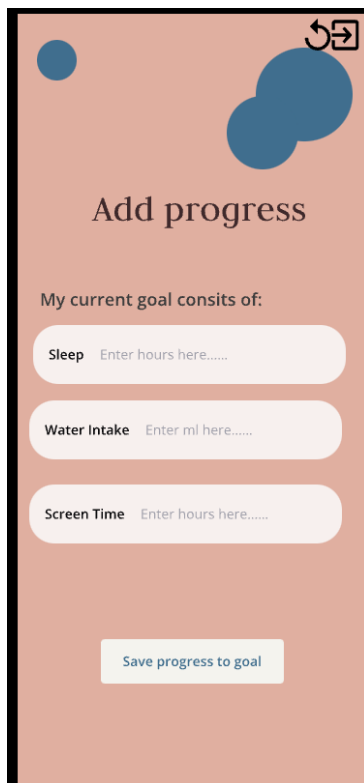
The 'Create New Goal' page has a light orange background with blue cloud-like shapes at the top. A title bar at the top contains the text 'Create New Goal'. Below this, there are two dropdown menus: 'Choose target habit' with options like 'Water Intake', 'Sleep schedule', 'Exercise', 'Screen Time', and 'Coffee Consumption'; and 'Choose time period' with options like 'Daily', 'Weekly', and 'Monthly'. Below these is a 'Target Value' input field with the placeholder text 'Enter value here.....'. At the bottom, there are two buttons: 'Add habit to goal' and 'Create new goal'. A blue circular profile picture placeholder is located on the left side of the bottom section.

#### Create New Goal Page Description

The create new goal page allows users to set up a new goal. It includes:

- Fields to specify the target habit, value, and period.
- A button to create the new goal.
- A button to add the habit to the goal that is in creation.
- Buttons for logging out and going back to the last accessed page.

### 1.12.9 Add Progress Page



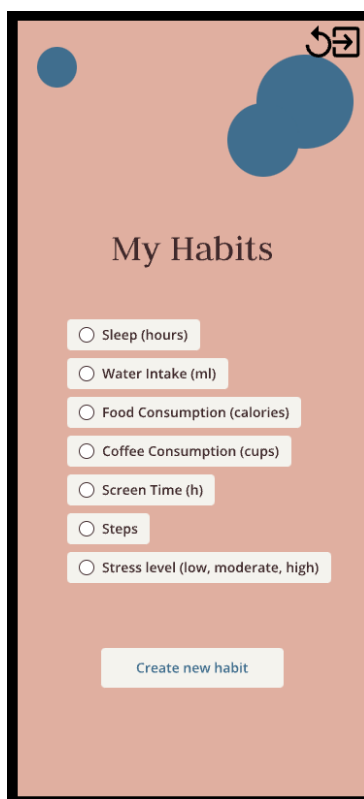
The 'Add progress' page features a light orange background with a dark orange header. At the top right, there are three blue circular icons and a navigation icon. The title 'Add progress' is centered. Below it, the text 'My current goal consists of:' is followed by three input fields: 'Sleep' with placeholder 'Enter hours here.....', 'Water Intake' with placeholder 'Enter ml here.....', and 'Screen Time' with placeholder 'Enter hours here.....'. A 'Save progress to goal' button is at the bottom.

#### Add Progress Page Description

The add progress page allows users to input the habit parameters for their current goal. It includes:

- Fields to enter the progress made towards the current goal.
- A button to save the progress.
- Buttons for logging out and going back to the last accessed page.

### 1.12.10 Habit Page



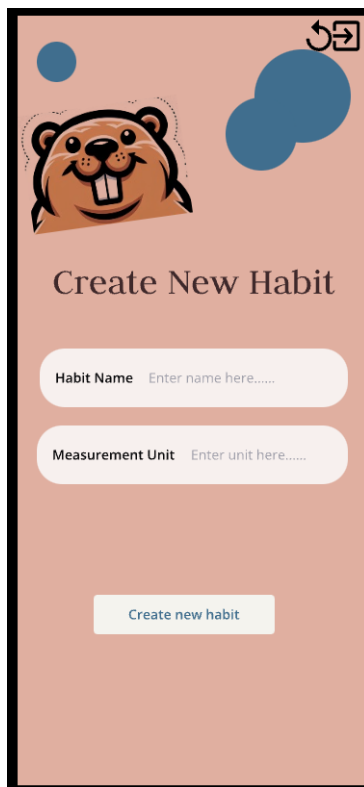
The 'My Habits' page features a light orange background with a dark orange header. At the top right, there are three blue circular icons and a navigation icon. The title 'My Habits' is centered. Below it, there is a list of habits with radio buttons: 'Sleep (hours)', 'Water Intake (ml)', 'Food Consumption (calories)', 'Coffee Consumption (cups)', 'Screen Time (h)', 'Steps', and 'Stress level (low, moderate, high)'. A 'Create new habit' button is at the bottom.

#### Habits Page Description

The habit page allows users to track and manage their daily habits. It includes:

- A list of habits that can be tracked by the user.
- Options to create a new habit.
- Buttons for logging out and going back to the last accessed page.

### 1.12.11 New Habit Page



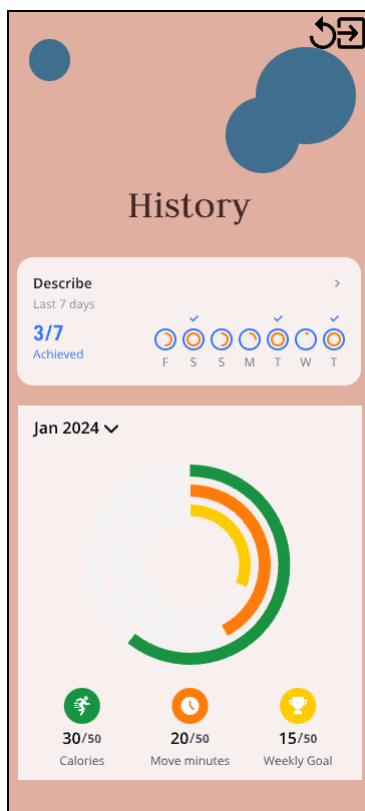
The 'New Habit Page' is a mobile app screen with a light orange background. At the top left is a cartoon beaver head. At the top right is a back arrow icon. The title 'Create New Habit' is centered. Below it are two input fields: 'Habit Name' with placeholder text 'Enter name here.....' and 'Measurement Unit' with placeholder text 'Enter unit here.....'. At the bottom is a 'Create new habit' button.

#### Create New Habit Page Description

The new habit page allows users to add a new habit to be tracked. It includes:

- Fields to specify the habit name and the measurement unit of the habit.
- A button to save the new habit.
- Buttons for logging out and going back to the last accessed page.

### 1.12.12 History Page

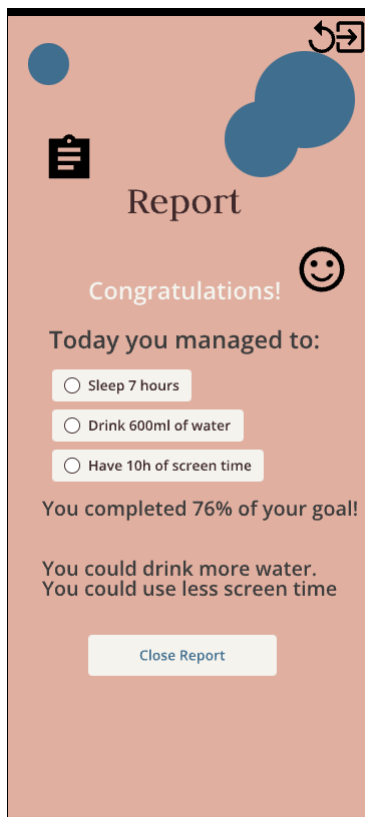


#### History Page Description

The history page provides users with a record of their tracked habits and goals. It includes:

- A chronological list of past habits and goals.
- Summary statistics and progress reports.
- Buttons for logging out and going back to the last accessed page.

### 1.12.13 Report Page

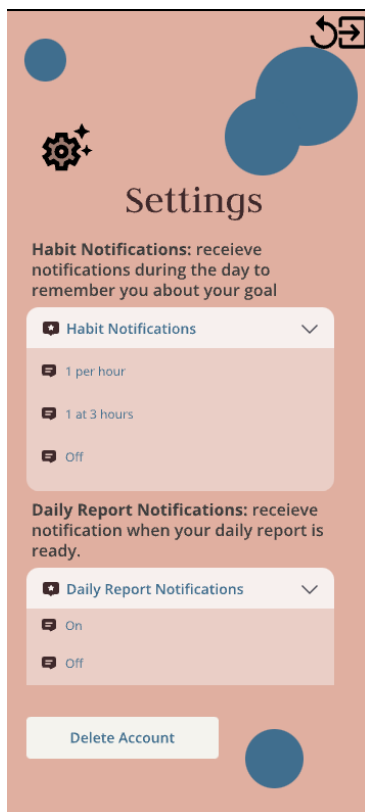


#### Report Page Description

The report page provides users with detailed reports of their progress. It includes:

- Summaries of the user's habit tracking and goal progress.
- Button to close the report.
- Buttons for logging out and going back to the last accessed page.

### 1.12.14 Settings Page



#### Settings Page Description

The settings page allows users to customize their BalanceBuddy experience. It includes:

- Settings to configure notifications and reminders.
- Button for deletion of account.
- Buttons for logging out and going back to the last accessed page.

## 1.13 Setup

This section provides a comprehensive guide on setting up the BalanceBuddy project environment, including installation instructions, configuration steps, and necessary setup for databases and other components.

### 1.13.1 Prerequisites

Before proceeding with the setup, ensure you have the following installed and configured on your system:

- Install Java 17 from [oracle.com](https://www.oracle.com/in/java/technologies/javase-downloads.html)
- Node.js and npm: Install Node.js from [nodejs.org](https://nodejs.org/), npm will be installed alongside Node.js.
- Install MySQL from [downloads.mysql](https://downloads.mysql.com/archives/community)

After installing all the required technologies follow these steps:

1. Clone the repository

```
git clone https://github.com/cami2803/BalanceBuddy-Lifestyle-Application.git
cd BalanceBuddy-Lifestyle-Application
```

2. Ensure Java 17 is correctly installed and configured in your PATH environment variable, the following way (Image taken from [docs.oracle.com](https://docs.oracle.com/en/java/javase/17/install/setting-path.html)):

To set the `JAVA_HOME` and `PATH` environment variables in Microsoft Windows:

1. Click **Start, Control Panel, System**, and then **Advanced system settings**.
2. In the **System Properties** dialog box, on the Advanced tab, click **Environment Variables**.
3. Add the `JAVA_HOME` environment variable:
  - a. In the **System Variables** section, click **New**.
  - b. In the **Variable name** field, enter `JAVA_HOME`.
  - c. In the **Variable value** field, enter the location where the JDK software is installed (for example, `C:\Program Files\Java\<java_version>`)
  - d. Click **OK**.
4. Update the `PATH` environment variable to include the location of the Java executable files:
  - a. In the **System Variables** section, select the `PATH` variable, and click **Edit**.
  - b. In the **Variable value** field, insert `%JAVA_HOME%\bin;` in front of all the existing directories. Do not delete any existing entries; otherwise, some existing applications may not run.
  - c. Click **OK**.
5. Exit the Control Panel.

3. Open a terminal or command prompt and navigate to the backend directory of the cloned repository.
4. Build and run the backend using Maven:

```
cd backend
mvn spring-boot:run
```

5. Open another terminal or command prompt window and navigate to the frontend directory of the cloned repository.
6. Install frontend dependencies and start the development server:

```
cd frontend
npm install
npm start
```

### 1.13.2 Configuration

1. Navigate to the backend directory.
2. Open the `application.properties` file (located in `src/main/resources`) and configure the MySQL database connection:

```
spring.datasource.url=jdbc:mysql://localhost:3306/balancebuddy-lifestyle-application
spring.datasource.username=root
spring.datasource.password=password
```

Replace username and password with your MySQL credentials.



### 1.13.3 Running the Application

1. Ensure MySQL is running, then run the backend application using Maven:

```
cd backend
mvn spring-boot:run
```

The backend server will start running on port 8080 by default (<http://localhost:8080>).

2. Open a new terminal or command prompt window, navigate to the frontend directory, and start the frontend development server:

```
cd frontend
npm start
```

The frontend development server will start and open in your default web browser.

#### Ports Used:

- Frontend: Runs on port 3000 by default (<http://localhost:3000>).
- Backend: Runs on port 8080 by default (<http://localhost:8080>).
- Database: MySQL runs on port 3306 (<jdbc:mysql://localhost:3306/balancebuddy-lifestyle-application>).