***Phase II: User Requirements and Application Specifications***

***Submission Deadline: 18.03.2024, 23:59***

***Group members : Ergita Curri,Adelajda Torollari***

* **1.Chosen Development Model: Agile**

**Justification:** We have chosen Agile as our development model for its flexibility and adaptability to changes in project requirements. Agile allows us to iterate quickly, gather feedback from stakeholders, and make adjustments to deliver a high-quality product that meets user needs.

1. **Flexibility:** Agile allows us to adapt to changing requirements and incorporate feedback quickly.
2. **Iterative Development:** We can deliver working software in short, regular intervals, getting early feedback from stakeholders.
3. **Customer Collaboration:** Agile promotes close collaboration with end-users, ensuring the final product meets their needs.
4. **Early and Continuous Delivery:** We can deliver a minimum viable product early and continuously add features based on user feedback.
5. **Focus on Individuals and Interactions:** Agile prioritizes communication and collaboration among team members, leading to a more efficient development process.
6. **Customer Satisfaction:** Agile aims to maximize customer satisfaction by delivering valuable software quickly and responding to changing requirements.

In conclusion, the Agile development model is well-suited for the Budget Buddy project due to its flexibility, adaptability, focus on customer collaboration, and emphasis on delivering value quickly and continuously. It provides a framework that allows us to respond to changing requirements and deliver a high-quality product that meets the needs of our users.

* **2. User Requirements:**

**Stakeholders:**

* End-Users: Individuals managing personal finances.
* Role: Primary users of the application, interested in easily managing and monitoring their finances.
* Clients: Our software engineering class project team.
* Role: Responsible for the successful outcome of the project.
* Developers: Myself (Ergita) and Adelajda.

Role: Responsible for developing and testing the application.

**Discussion:** End-users seek a simple and intuitive interface for managing finances. Developers aim to create a functional and secure application. Financial advisors and the instructor provide guidance and feedback throughout the development process.

* **3. User Stories:**
* As an end-user, I want to easily track my daily expenses and categorize them for better understanding.
* As an end-user, I want to set customized budgets based on my income and spending habits, with notifications for deviations.
* As an end-user, I want to set financial goals and track my progress using available tools and visual representations.

***4. Functional Requirements:***

* **Brief Description:**
* Expense Tracking: The system should allow users to input and categorize their daily expenses.
* Budget Management: Users should be able to create customized budgets and receive notifications for budget deviations.
* Financial Goal Setting: The application should assist users in setting and tracking financial goals.

**Acceptance Criteria:**

-User can add, categorize, and track daily expenses.

-User can create, customize, and receive notifications for budget deviations.

-User can set, track, and visualize progress towards financial goals.

-Administrator can access and manage user data for support and feedback.

***Non-Functional Requirements:***

* **Brief Description:**
* User-Friendly Interface: The application should have an intuitive and easy-to-use interface.
* Security: The application should ensure the security of user data. The system should be user-friendly, secure, and efficient.

**Acceptance Criteria:**

-The system provides an intuitive and simple interface for users.

-The system encrypts user data and follows best practices for security.

-The system is responsive and performs efficiently under varying user loads.

**5. Application Specifications:**

**Architecture:**

For the Budget Buddy application, we have chosen a client-server architecture with a web-based frontend and a cloud-based backend.

**Components:**

1. **Client-Side (Frontend):**
   * Developed using HTML, CSS, and JavaScript.
   * Utilizes React.js framework for building interactive user interfaces.
   * Responsible for presenting the user interface and handling user interactions.
2. **Server-Side (Backend):**
   * Developed using Node.js with Express.js framework.
   * Responsible for processing business logic, handling user requests, and interacting with the database.
   * Utilizes RESTful APIs for communication between the client and server.
3. **Database:**
   * Uses MySQL as the relational database management system (RDBMS).
   * Stores user data including expenses, budgets, and notifications.
   * Designed with tables for users, expenses, budgets, and notifications, with appropriate relationships and constraints.

**Interactions:**

* The client (frontend) interacts with the server (backend) through RESTful APIs to perform actions such as adding expenses, creating budgets, and setting financial goals.
* The server processes these requests, interacts with the database to retrieve or update data, and sends the appropriate responses back to the client.
* The database stores and manages user data, allowing for efficient retrieval and manipulation of financial information.

**High-Level Diagram:**



**Database Model:**

For the Budget Buddy application, we have designed a relational database model using MySQL to store and manage user data efficiently. The database model includes tables for users, expenses, budgets, and notifications, with appropriate relationships and constraints.

**Tables:**

1. **Users Table:**
   * Contains information about users such as user ID, username, email, and hashed password.
   * Primary Key: UserID
   * Unique Constraint: Email
2. **Expenses Table:**
   * Stores information about user expenses including expense ID, user ID (foreign key), date, amount, category ID (foreign key), and description.
   * Primary Key: ExpenseID
   * Foreign Key: UserID
   * Foreign Key: CategoryID
3. **Budgets Table:**
   * Stores user-defined budgets with fields for budget ID, user ID (foreign key), category ID (foreign key), budget amount, and time period.
   * Primary Key: BudgetID
   * Foreign Key: UserID
   * Foreign Key: CategoryID
4. **Notifications Table:**
   * Stores notifications for users about budget deviations or financial goals with fields for notification ID, user ID (foreign key), notification message, and timestamp.
   * Primary Key: NotificationID
   * Foreign Key: UserID

**Relationships:**

* One-to-Many relationship between Users and Expenses: One user can have many expenses.
* One-to-Many relationship between Users and Budgets: One user can have many budgets.
* One-to-Many relationship between Users and Notifications: One user can have many notifications.

**Constraints:**

* Primary Keys: Each table has a primary key (UserID for Users table, ExpenseID for Expenses table, BudgetID for Budgets table, NotificationID for Notifications table).
* Foreign Keys: Expenses, Budgets, and Notifications tables have foreign keys referencing the Users table (UserID).

This database model allows for efficient storage and retrieval of user data, enabling Budget Buddy to effectively track expenses, manage budgets, and provide notifications to users.

Top of Form

**Technologies Used:**

1. **Frontend:**
   * **React.js:** Chosen for its component-based architecture, which allows for easier management of complex user interfaces. React.js also provides a virtual DOM for efficient updates and is well-suited for building single-page applications.
   * **HTML, CSS, JavaScript:** Standard web technologies used for structuring content, styling, and adding interactivity to the application's frontend.
2. **Backend:**
   * **Node.js:** Chosen for its non-blocking, event-driven architecture, which makes it lightweight and efficient for handling concurrent requests. Node.js also allows for easy scalability and is well-suited for building web applications.
   * **Express.js:** A minimalist web application framework for Node.js, chosen for its simplicity and flexibility in creating RESTful APIs and handling HTTP requests.
3. **Database:**
   * **MySQL:** Chosen as the relational database management system (RDBMS) for its reliability, performance, and scalability. MySQL provides robust features for managing relational data, making it suitable for storing financial information and user data.
4. **Other Technologies:**
5. **Frontend:**
   * **React.js:** Chosen for its component-based architecture, making it easier to manage complex UI components and improve code reusability. We are familiar with React.js, which will speed up development.
6. **Backend:**
   * **Node.js:** Chosen for its non-blocking, event-driven architecture, which makes it lightweight and efficient for handling concurrent requests. We have experience with Node.js, which will streamline development.
   * **Express.js:** Chosen as the web application framework for Node.js, providing a simple and minimalistic approach for building APIs. Its compatibility with Node.js and our familiarity with it make it a suitable choice.
7. **Database:**
   * **MySQL:** Chosen for its reliability, performance, and scalability. We have experience working with MySQL, making it a practical choice for managing financial data and user information.
8. **Other Technologies:**
   * **RESTful APIs:** Used for communication between the frontend and backend, providing a standardized way to exchange data. Our familiarity with RESTful APIs makes it a suitable choice for this project.
   * **JSON Web Tokens (JWT):** Used for secure authentication, ensuring that only authorized users can access protected routes. We have experience implementing JWT for authentication, making it a secure choice for user management.
9. **Development Tools:**
   * **Git:** Used for version control, allowing for collaboration and tracking changes in the codebase. We are familiar with Git, which will facilitate team collaboration and code management.
   * **VS Code:** Integrated development environment (IDE) for writing, editing, and debugging code. We are proficient in using VS Code, which will enhance our development workflow.

**Reasoning:**

* **Scalability:** The chosen technologies and frameworks are known for their scalability, allowing the application to handle a large number of users and transactions efficiently.
* **Compatibility:** The technologies chosen are compatible with each other, ensuring smooth integration and communication between frontend and backend components.
* **Performance:** Node.js and React.js are known for their performance optimizations, ensuring that the application is fast and responsive.
* **Community Support:** The chosen technologies have a large and active community, providing access to resources, libraries, and updates that can benefit the development process.
* **Security:** The use of JWT for authentication and bcrypt for password encryption enhances the security of the application, protecting user data from unauthorized access.

**User Interface Design:**

For the Budget Buddy application, the user interface (UI) will be designed to be intuitive, user-friendly, and visually appealing. Here is a description of the key components and features of the UI:

* **Dashboard:** Provides a summary of financial situation, including account balance, earnings, and spending. Includes charts and graphs for visual representation of financial data.
* **Expense Tracking:** Allows users to input and categorize daily expenses. Users can add, edit, and delete expenses, and view summaries by category.
* **Budget Management:** Users can create customized budgets based on income and spending habits. Receive notifications for budget limit exceedances.
* **Financial Goals:** Set and track financial goals, with progress tracking and recommendations.
* **Reports and Visualizations:** Includes reports and visualizations of financial data for insights into spending trends and budget summaries.

**Security Measures:**

* + **Encryption:** User data, including passwords and financial information, will be encrypted using industry-standard encryption algorithms (e.g., AES-256) to protect against unauthorized access.
  + **Authentication:** Secure authentication with JSON Web Tokens (JWT) will ensure only authorized users can access the system. Passwords will be hashed using bcrypt before storage.
  + **Secure Communication**: Data transmitted between client and server will be encrypted using HTTPS.
  + **Data Privacy:** User data will be securely stored with restricted access. Regular security audits will be conducted.
  + **User Permissions:** Role-based access control (RBAC) will restrict access based on user roles.
  + **Logging and Monitoring:** Logging and monitoring mechanisms will track suspicious activity for timely response.
  + **Regular Updates:** Regular updates will be applied to patch known vulnerabilities.