Student Budget Manager

Problem area:

The system is designed to be used in students' life to simplify supervision of their budget and overall spending. This system will be especially useful for students who live in dormitories or rent a flat together and share some spending, be it food, household items, rent or any other expenses.

Project goals:

- **Effortless Data Entry**: Develop a user-friendly interface that encourages accurate and consistent data entry through features like transaction imports and categorization suggestions.
- **Flexible Budgeting**: Allow users to customize their budgeting experience by offering features like multiple budgeting categories, goal setting, and the ability to handle complex financial situations.
- **Simplify Group Budgeting**: Streamline the process of creating and managing shared budgets for groups, reducing the effort required to collaborate on finances.
- **Fair and Transparent Debt Allocation**: Develop a clear and customizable system for splitting bills and allocating debt within groups. This could involve options for weighting expenses based on usage or splitting costs by category.
- **Promote Group Accountability**: Integrate features that encourage responsible participation within groups, such as reminders for data entry and notifications for outstanding balances.
- **Real-time Data Synchronization**: Ensure seamless data synchronization across devices to avoid conflicts and maintain accurate group finances.

System responsibilities:

- Track income and expenses with categorized entries.
- Allow users to add new transactions, update and delete them as needed.
- Provide customizable budget summaries with visualizations.
- Offer multi-currency support for income and expense tracking.
- Display summaries for specific time periods.
- Facilitate creation and management of user groups for shared budgeting.
- Enable adding existing transactions to groups (by time period, manual selection, or both).
- Allow adding new transactions to relevant groups during data entry.
- Display group budget summaries with visualizations.
- Track and display outstanding balances within a group.

- Suggest fair settlement amounts to balance group finances.

System users:

- **Guest:** This category represents anyone who visits website without registration.
- **User:** This category represents anyone who creates an account within the system.

Functionalities:

- Login/logout functionality.
- Password reset and recovery.
- User Profile Management:
- Edit profile details (name, email, currency preference, etc.).
- Change password.
- Add new earnings (one-time prizes, wages, etc.).
- Update existing earnings records.
- Delete earnings records.
- Add new spending (groceries, subscriptions, etc.).
- Categorize spending into predefined or custom categories.
- Update existing spending records.
- Delete spending records.
- Add other money-related records (loans, savings, transfers, etc.).
- Update and delete these records.
- Display overall financial summary.
- Show total earnings, total spending, and net balance.
- Visualize data using graphs and charts.
- Breakdown of expenses by categories.
- Filter summary by specified periods (last month, last 2 months, custom date range, etc.).
- Allow users to choose the currency for displaying summaries.
- Convert and display all records in the selected currency.
- Create a new budget group.
- Invite other registered users to join the group.
- Accept or decline group invitations.
- Select and add existing records to a group (by period or manual selection).
- Add new records to groups upon creation.
- Update and delete group-related records.
- Display a summary of the group's finances.

- Visualize group expenses using graphs and charts.
- Breakdown of group expenses by categories.
- Track who owes whom within the group.
- Display suggested settlements and outstanding balances.
- Notify users of group invitations and updates.
- Remind users of outstanding balances and suggested settlements.
- Set and update currency preferences.
- Customize spending categories.
- Set notification preferences (email, internal notification)
- Ensure data encryption for sensitive information.
- Implement user data privacy controls.
- Manage users and groups.
- Integrate with external financial services (possible later)
- Provide API access for users to integrate with other applications or services.

User requirements:

1. **Part one:**
   Tables:

- User: User_Id, Nickname, Email, Preferred_Currency_Id, Creation_Date, Password, Name (optional), Surname (optional). Purpose: Stores information about individual users of the system.
- Currency: Currency_Id, Name. Purpose: Defines different currency types used within the system.
- Transaction Category: Category_Id, Name. Purpose: Defines different category that can be assign to transaction
- Transaction: Transaction_Id, Amount, Date, Description (optional), Category_Id, Currency_Id, User_Id Purpose: Records individual financial transactions made by users.
- Periodic_Transaction: Periodic_Transaction_Id, Amount, Creation_Date, Description (optional), Period, Category_Id, Currency_Id, User_Id Purpose: Records periodic financial transactions made by users (subscriptions, loans, etc.).
- Budget Group: Group_Id, Name, Creator_Id, Creation_Date. Purpose: Represents groups created for shared budgeting among users.
- User Group: User_Id, Group_Id. Purpose: Connects users to budget groups, indicating their membership.
- Group Transaction: Group_Transaction_Id, Transaction_Id, Group_Id. Purpose: Links individual transactions to specific budget groups.

- Dept: User_Id, Group_Transaction_Id. Purpose: Tracks debt associated with group transactions for individual users within a group (who pays for what)
- Group Application: User_Id, Group_Id, Date, Application_Text (optional). Purpose: Manages applications sent by users to join specific budget groups.
- Notification: Notification_Id, Title, Description, User_Id. Purpose: Stores notification information sent to users within the system.
- Group Invitation: Id, Link, Group_Id, Expiration_Date. Purpose: Temporaly stores invitation links for the group with the given group id.

Relations:

User:

- Many-to-One with Currency: A user has a preferred currency.
- One-to-Many with Transaction: A user can have multiple transactions.
- One-to-Many with Periodic_Transaction: A user can have multiple periodic transactions.
- One-to-Many with Notification: A user can receive multiple notifications.
- One-to-Many with Dept: A user can have multiple debts.
- One-to-Many with Budget_Group: A user can be an owner of multiple groups.
- Many-to-Many with Budget_Group through User_Group: A user can belong to multiple budget groups.
- Many-to-Many with Budget_Group through Group_Application: A user can send many applications to budget groups.
- Many-to-Many with Group_Transaction through Dept: A user owns percentage of dept of the given group transaction.

Periodic_Transaction:

- Many-to-One with Currency: A periodic transaction was performed in the given currency.
- Many-to-One with Transaction_Category: A periodic transaction belongs to specified category.

Transaction:

- Many-to-One with Currency: A transaction was performed in the given currency.
- Many-to-One with Transaction_Category: A transaction belongs to specified category.
- Many-to-Many with Budget_Group through Group_Transaction: Multiple transactions can be related to multiple budget groups.

Budget_Group:

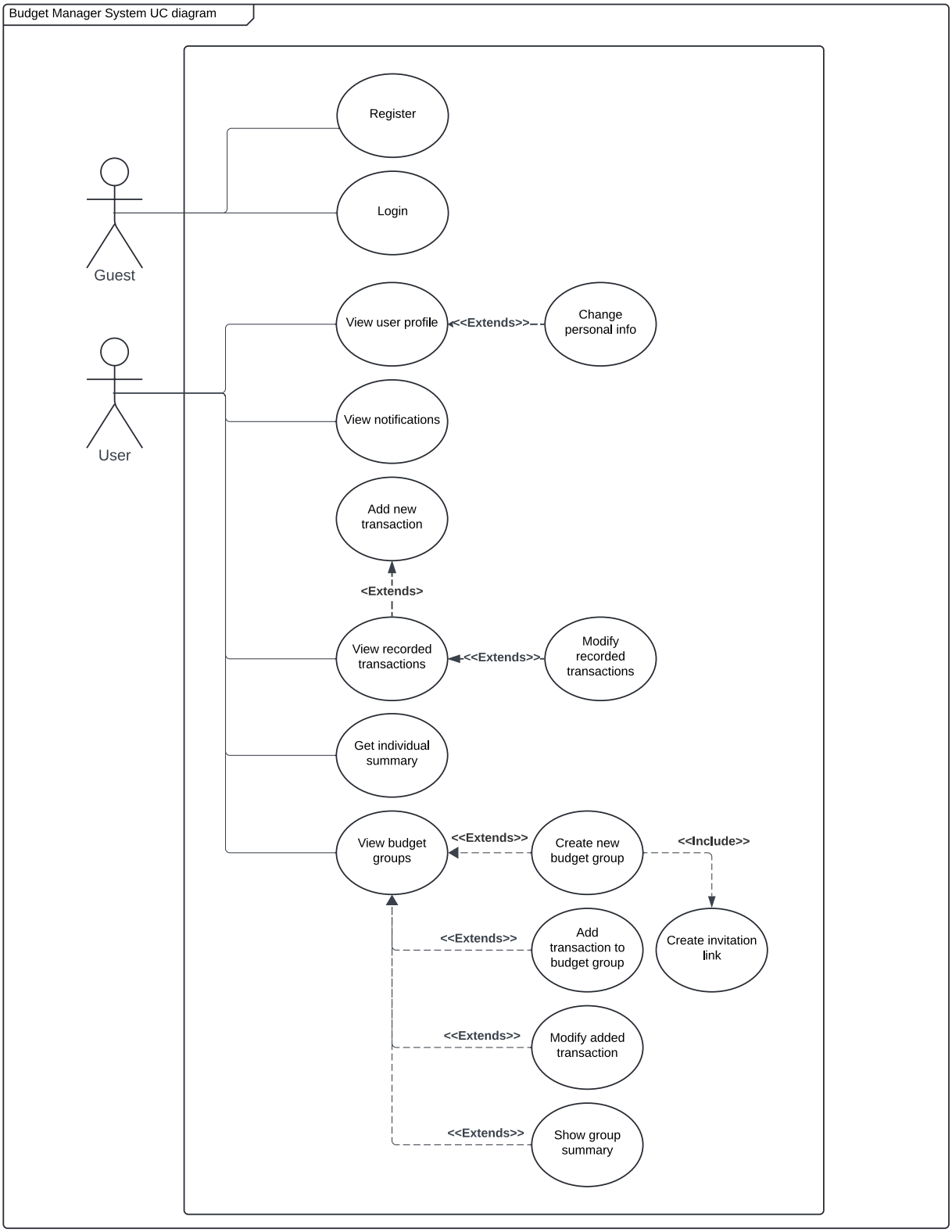
- One-to-Many: A budget group can have multiple invitation links to it at the same time.

2. **Part two:**

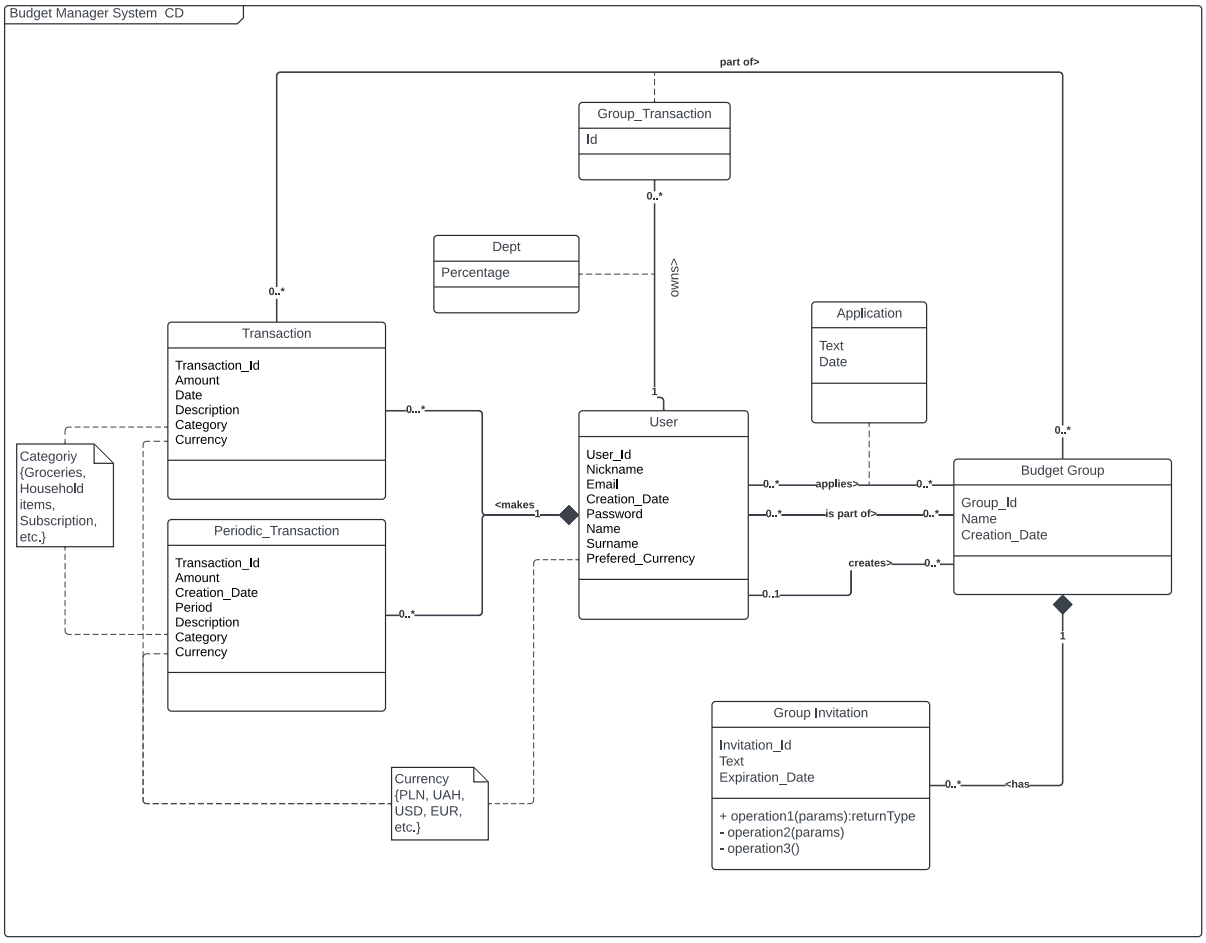
- Login/Register (Guest)
- View user profile
- Change user information
- View notifications
- Add new transaction
- View recorded transcations
- Modify recorded transaction
- Get individual summary
- View budget group list
- Create new budget group
- Create invitation link
- Accept/Decline users to budget group
- Add transaction to budget group
- Configure group transaction distribution
- Get group summary

3. **Part three:**
    - All sensitive user data should be encrypted.
    - All user inputs must be validated both client-side and server-side to prevent errors and fraudulent activities.
    - The system architecture must follow the open-closed principle for easier future functionality extensions.
    - System interface must be easy and intuitive.
    - The system must be compatible with modern web browsers.

Functional Requirements:



Description of the system structure:



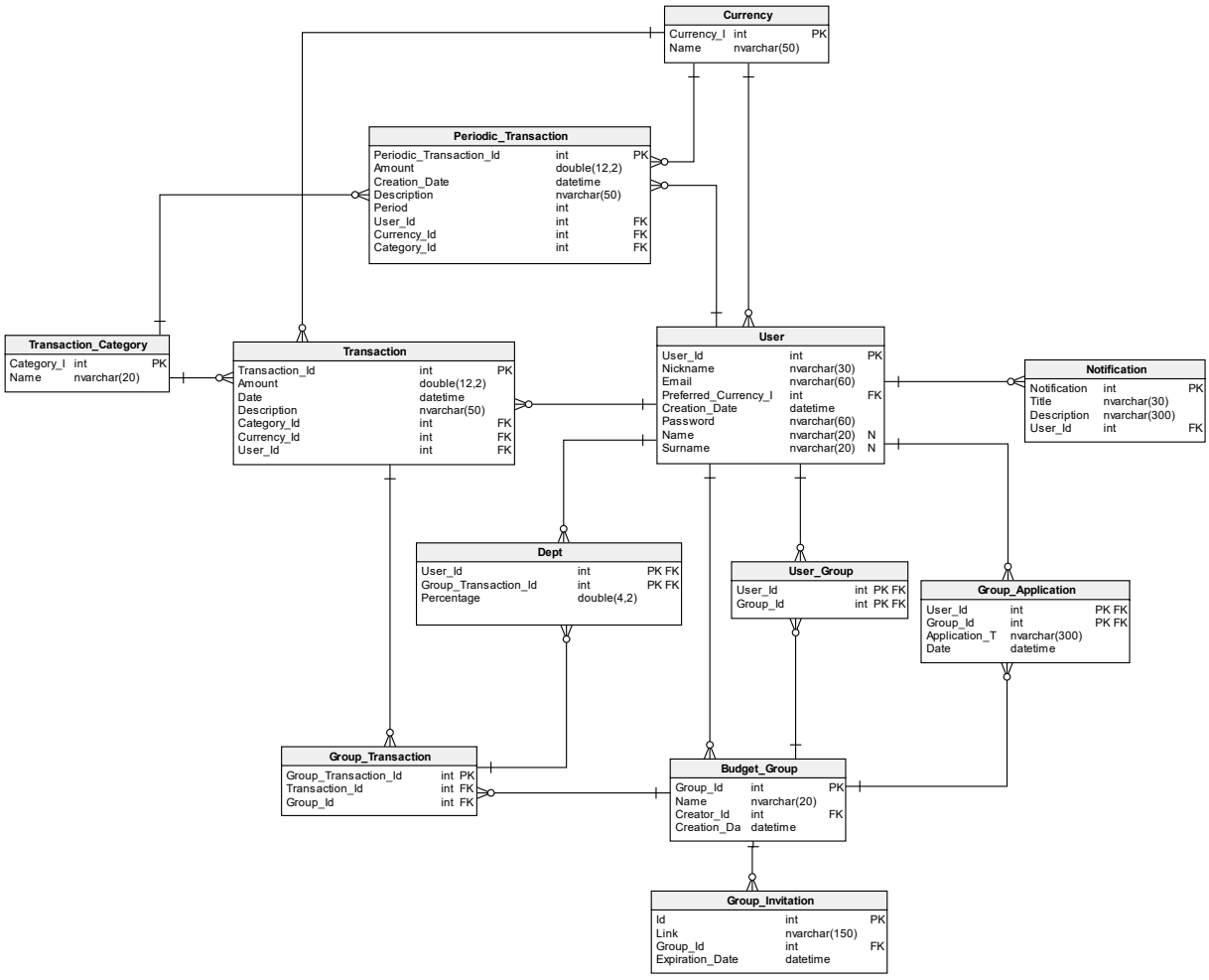
Non-Functional Requirements:

- Requirement: All sensitive user data should be encrypted.
  Metric: Make sure that all sensitive data is hashed using a strong hashing algorithm. Perform regular security checks and penetration tests to ensure invulnerability of the system.
- Requirement: All user inputs must be validated both client-side and server-side to prevent errors and fraudulent activities.
  Metric: Perform input validation accuracy tests and make sure that percentage of total valid inputs / total inputs submitted is higher than 95%
- Requirement: The system architecture must follow the open-closed principle for easier future functionality extensions.

Metric: Perform analysis of use of abstract classes, interfaces, coupling between classes, code churn during extension, and existing functionality test coverage

- Requirement: System interface must be easy and intuitive.
  Metric: Conduct user testing sessions where participants navigate the system and complete tasks. After the session, measure user satisfaction with a questionnaire that includes questions about how easy it was to learn the interface and how intuitive they found it to be.
- Requirement: The system must be compatible with modern web browsers.
  Metric: Use automated testing tools or manual testing to verify if the system functions correctly across all the target browsers.

Database schema:



API endpoints list with description:

- [GET]/api/manager/{user_id} – get User by Id.
- [PATCH]/api/manager/{user_id} – update User information by id.
- [GET]/api/manager/{user_id}/notifications – get list of notification of User with the given id.
- [POST] /api/manager/{user_id}/transactions – add new transaction to the User with the given id.
- [GET]/api/manager/{user_id}/transactions – get list of transactions of User with the given Id.
- [GET]/api/manager/{user_id}/summary/{begin_date}/{end_date} – get summary of the User with the given Id over period of time
- [PATCH] /api/ transactions/{id} – update the transaction with the given id.
- [GET]/api/manager/{user_id}/groups – list groups that the User with the given id is a part of.
- [POST]/api/manager/{user_id}/groups – create new budget group.
- [POST]/api/groups/{id}/invitation - create new invitation link for the group with the given Id.
- [POST]/api/groups/{id}/accept/{user_id} – accept User with the given id to the group.
- [DELETE]/api/groups/{id}/decline/{user_id} – decline User with the given id from joining the group.
- [POST]/api/groups/{id}/transactions/{id} – add transaction with the given Id to the group.
- [PATCH]/api/groups/{id}/transactions/{id} – modify the transaction of the group with the given Id.
- [GET]/api/groups/{id}/summary – get summary of the group with the given Id.

List of used technologies

- Gradle - for multi-language software development.
- Spring Boot - to build an application.
- Spring Boot DevTools – for fast application restart.
- Spring Core - provide the fundamental parts of the framework.
- Spring WEB - to create self-contained HTTP server by using embedded Tomcat.
- Spring Data JPA - to easily implement JPA-based repositories.
- Spring MVC - to implement Model-View-Controller pattern.
- Spring Security – to implement authentication and secure data storing.

- MySQL - to map Java classes to database tables and from Java data types to SQL data types.
- Thymeleaf – to processing HTML documents and provide data to them.
- HTML – to create web page content.
- CSS – to add style to web page.
- JavaScript – to create dynamic content for web pages.