Personal Budget Tracker

This web application is designed to help users track, plan and manage their personal and shared budgets, manage budgets across multiple categories and visualize spending habits through charts and reports.

Required core entities

- **Users** – individuals registered in the system, each assigned to specific roles (many for a user).
- **Categories** – budget categories such as “Food”, “Gas”, “Entertainment”, etc.
- **Transactions** – expense or income records assigned to specific categories, with attributes like amount, date, type, description, currency, etc.

User Roles

1. **Administrator** – responsible for managing user accounts, roles, and overall system configuration.
2. **Management** – authorized to create budget categories, assign specific types of attributes to categories, and oversee regular user statistics.
3. **Regular User** – can act as:
    - **Budget owner** – manages personal budget of himself or shared budget and creates spending goals and limits.
    - **Contributor** – can add or edit transactions within shared budget (for example, family budget).
4. **Control User** – can browse statistics and other types of data but can’t change it.

Key Features

- Role-based permissions to ensure secure and structured access control in both domains: frontend human interaction and rest endpoints usage.
- Creation of categories and assignment of managers.
- Centralized **SQLite** database to store and manage all users and other entities
- Income and expense tracking
- Export or import of transaction history to CSV/PDF
- Shared budget tracking so that users can collaborate on shared budgets (for example, roommates, family, partners, etc.).
- Budget goals and limits so users can achieve goals or limit themselves on budget
- Multi-Currency support so users can have multiple types of currencies in same reports
- Dark and light theme of web application
- Income vs. Expense Reports with graphical chart

Business Rules

The system enforces business rules based on user roles. Each role grants specific responsibilities and restrictions to ensure normal flow in the context of your project.

Administrator

- Manages **users** within the system (e.g. login/emails, passwords etc.).
- Assigns and updates **roles** for each user.
- A user can hold **one role (administrator, management, regular user or unregistered/control user)**

Management

- Creates and modifies categories of budget
- Assigns types of users **in the project context** from the pool of Regular Users. A regular user can have multiple types at the same time.
- Can reassign/change the type if necessary.
- Does not directly create or manage lower functionality of the project.

Regular User

Depending on their type in a given, a Regular User may act as **Budget Owner** or **Contributor**

1. **As Budget Owner**
    - Creates personal budgets and assigns limits per category
    - Adds, edits and deletes transactions
    - Reviews summary reports
    - Adjusts goals and limits
    - Shares budget access with contributors
2. **As Contributor**
    - Be a part of multiple shared budgets
    - Adds or edits transactions in shared budgets
    - Views charts and spending summaries
    - Cannot modify budget limits or goals
    - Can edit their own transactions
    - Filter and sort their own transactions

Every user can be Budget Owner and/or Contributor of different budgets.

Control User

- Can view statistical data
- Can view financial tips about budgets and savings
- Can't access or edit user data

Other notes:

- Transactions cannot exist without being linked to a specific category.
- A budget must always have exactly one Budget Owner but can include multiple Contributors.

Common features required by all projects

Data Integrity and Validation

- All mandatory fields (e.g., user name, project title, task description) must be validated before saving.
- Unique identifiers (e.g., usernames, project IDs) must be enforced at the database level.
- Relationships between entities must be preserved.
- Invalid or incomplete data cannot be persisted.

Usability and User Experience

- The system must provide a clear, responsive web interface accessible across modern browsers.
- Actions should be simple and intuitive, following established UI/UX conventions.
- Error messages must be descriptive, guiding the user on how to resolve issues.
- **Large datasets -** are displayed in a way that is user-friendly and does not overuse browser resources (e.g. pagination on server side)
- **Sorting, filtering** - Filtering and sorting must be done on the server side.
- **Technologies** - must be implemented using **Angular**, data is retrieved from the backed **REST service** according to the standards. Using Angular material is allowed.
- Data should be presented graphically (charts) in at least two places.

Security and Access Control

- Authentication is required for all users, except for the unregistered user who has limited access to only view some small portions of the things in the context of the project.
- Authorization is role-based, ensuring users only access functions permitted by their role(s) on backend API. The frontend is tailored to the backends' capabilities within a given set of roles.

- Multiple types per registered user are supported, with the system applying the union of their permissions.
- Sensitive actions (e.g., user role changes, project manager reassignment) require confirmations to ensure user intentions.
- Passwords must be stored in a secure form not plain text.

Audit and Traceability

- All critical changes (user role updates, project manager reassignments, task status approvals) must be logged.
- The system should provide visibility into who performed specific actions and when.
- A special GUI for log browsing is not obligatory.

Other requirements

- As a starting point teachers' demo projects can be used.
- **Running the application on a production server -** NodeJS server started on a port number given by teachers to the student. Backed must have a necessary **REST** service that provides access to the resources needed by the frontend. Backend also hosts the angular application. The application is loaded when the user opens the root of the project in the browser (e.g. http://spider.foi.hr:12000) Every communication is then handled via **recommended standards**. REST service implementation should follow the conventional implementation using HTTP methods for specific purposes. (i.e. GET - retrieve data, POST - add data, DELETE - deleting data, and PUT - update of data)
    - **Installation and modules** - project must be installed on the server spider.foi.hr. Access via the SSH must be restricted to all other students.
    - **Frameworks/modules/libraries**: In the assignment, only the frameworks/modules/libraries that have been covered in class are allowed to be used on both the client-side and server-side. Specifically for NodeJS, only those modules that are globally installed on the server spider.foi.hr may be used, and no additional modules may be installed on the server-side. A list of available modules can be obtained with: `npm -g list`.
    - **Assignment structure**: The directory should only contain files and subdirectories related to the assignment; everything else must be removed (e.g., scripts from exercises). Relative paths must be used throughout the assignment. Multimedia files uploaded to the server must adhere to the maximum size limits: 500KB for images and 1MB for videos!
- **Help and Questions:** All questions regarding the assignment should be posted in the dedicated assignment forum on the Moodle system. You can also ask the instructor during lab sessions or during consultations scheduled by the instructor at the respective institution. We recommend that you do not share your code with other teams, do not show your code to other teams, and do not work on the assignment together with other teams, even if you are roommates or have known each other for 10 years. Copying assignments is prohibited; if it is determined that the assignment was copied from another student(s), everyone will receive 0 points.

Documentation Requirements

To ensure maintainability, knowledge transfer, and effective use of the system, comprehensive documentation must be prepared and maintained. The documentation should cover the following areas:

Authors documentation

- **Self Evaluation** - Table of the filled out evaluation form (self-evaluation) of what features are implemented. If a feature is implemented, partially lower the number of points in the category accordingly.
- **Authors page** - The authors page must have an image like student ID, passport or driver's licence documents, name and surname, student ID number and email. Email is a link that, when clicked, opens an email client defined in the browser settings. Design the rest of the page as desired.

Technical Documentation

- **System Architecture**: description of the overall architecture, including frontend, backend, and database components, along with integration points.
- **Data Model**: entity-relationship diagrams and explanations of relationships between users, categories etc.
- **API Documentation**: complete specifications of available endpoints, request/response formats, authentication, and error handling.

General Requirements

- Prefer **standard formats** (Markdown, HTML, or PDF) to ensure readability across platforms.
- Documentation should be accessible from the production deployment as a static HTTP content.