CSE 593 Applied Project Proposal – Personal Finance Management App

Jared Ham – Graduate Student ASU

9/2/202

Contents

Summary

This project will focus on creating a personal finance management app that aims to provide users with the ability to take control of their finances. Users will be able to add and manage bank accounts, enter transactions and create budgets. They will also be able review financial reports and compare them against their financial planning goals. All together this app aims to provide a place where users can go to get educated about personal finance and give them the tools needed to have financial security.

The application will be built using the client server architecture pattern and will leverage technologies such as: ReactJS, Java Spring Boot, Keycloak, PostgresSQ and Docker.

Introduction

Overview and Objectives

This project will create a financial management app that will allow users to add, track and categorize transactions in bank accounts, credit cards, and investment accounts. The categories available to track will be completely customizable and subcategories will also be able to be created. Users will also be able to create and add monthly budgets as well as savings goals to aid in their financial planning. The app will also provide users with a set of financial planning calculator to help them achieve their goals. Examples of this would be an investment calculator which can calculate predicted future value based on an initial balance and periodic contributions. These calculators will be able to utilize values from accounts that users are tracking to make this process as painless as possible. Overall, the project sets out to create a single location where users can go for all their financial planning needs.

Existing Designs

There are a fair number of apps that currently exist that offer a similar set of services to their users. One big one is Intuit Mint [1], which allows users to track and manage account balances, transactions and create budgets. A great benefit of Mint is that users can login to there accounts through Mint so that their transactions are automatically synced instead of having to manually enter them into the system. Another example would be GnuCash [2], which is an open source accounting software. This is a slight down step from Mint in terms of ease of use and accessibility. This software is a stand alone application for a computer so it cannot be accessed from anywhere. But it does offer a few more features than Mint such as the ability to create invoices and generate custom reports.

This project will focus on being more like Mint, but will add on features that Mint does not offer such as financial planning tools. The project will also not focus on being able to sync accounts automatically from the start but if time allows this option will be explored.

Project Scope

The reality is that this project will need to be completed by a single person, working 12-15 hours per week for 14 weeks. So, it will not be able to accomplish all the things that a product like Intuit Mint or GnuCash can provide. But it should be able to resemble and function in a similar fashion just not as ironed out. For this reason, the scope of the project will focus mostly on the development and not as much on the deployment and hosting side. An effort will be made to deploy the application to be available for use anywhere on the internet, but it will be considered a nice to have with the main goal being able to easily spin up the application and make it accessible on a local network.

This means that the overall scope of the project is to create an application that allows users to create an account, login and be able to manage their personal finances in an effective way. This will include the ability to manually add single and batch transactions, create budgets, and use financial planning calculators. All of this should be available to do from a web browser and the User Interface should be simplistic and easy to use.

Methodology

System Architecture

This project will adopt the Client-Server architecture [3], utilizing a spring boot backend and a ReactJS web based front end. It will also make use of a Postgres [4] database and will use Keycloak [5] for user management and authentication.

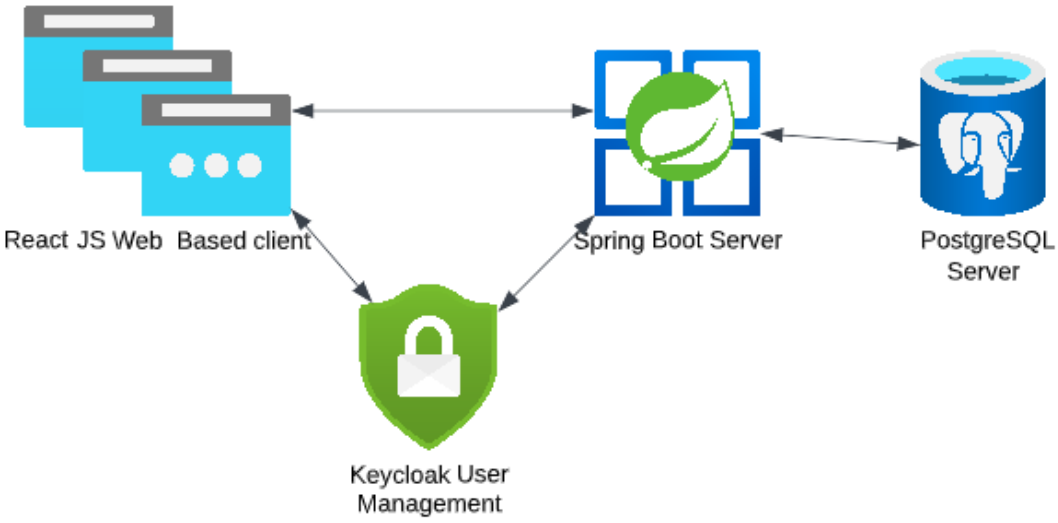

Figure 1: High Level Architecture

The spring boot server will have both secured and unsecured api endpoints. Most of the endpoints will be secured, with the exception of the login and account creation endpoints. All other endpoints will require authentication to access. The authentication will also limit the access of data to the data for the user making the request only. For example if user A is making a call to get account information, they will only be able to get access to their own account data. They would not be able to access data from user B's account. This authentication strategy will be handled mostly through keycloak which will provide logged in users with a JWT for their session. This JWT will allow them to call the secured endpoints and will provide the server with data about the user making the call. This allows the server to restrict access to only the user's data. Obviously this will all be managed behind the scenes by the client, but if a savvy user tried to make a call to the server using their auth token attempting to get another users data, they would not be able to.

The server will be designed to be run inside a Docker container using Ubuntu as the base image. This means that the server itself will need to be compatible with the linux OS, but since docker is being utilized it will be able to be run on any system which is able to host docker containers. This will also be true of the client, database and keycloak server. Adopting docker as the application environment gives a greater success of creating an application that can be easily spun up and which does not rely on the end user having to setup their own environment which is compatible with the application. This also makes development much easier. A docker-compose [6] file will be created which will allow the entire application to be launched on a docker host with a single command.

The server APIs will be RESTful [7] and will accept GET and POST type requests containing a JSON body with the necessary data for the request. For example, a createUser endpoint will be created that accepts the JSON object shown below. This endpoint will enable the client to request that a user account be created with the provided information.

```
{
    "username": "joey2",
    "password": "password",
    "email": "joey2@email.com",
    "firstname": "Joey",
    "lastname": "Jones"
}
```

Currently there is a plan to create a number of APIs but the createUser request is the only one that has been designed and created to date. Other APIs that will be created are shown in the table below. Note that this is not an exhausted list, just a list of APIs that are known to be needed at this point.

Endpoint	**Description**	**Secured**
createUser	Will handle the user creation logic.	No
addAccount	Handle adding an account to the users dashboard. Will be able to handle adding bank accounts, credit cards, brokerage, etc.	Yes
addSingleTransaction	Handle adding a transaction to an account. This endpoint will need to accept data about which account the transaction was associated with, the date it was made, the amount, and the budget category that it belongs to.	Yes
addBatchTransaction	Will handle adding multiple transactions at a time	Yes
createBudget	Will handle the creation of a Budget category. This will need to take in data such as the budget name and whether it is a subcategory of another budget	Yes
getAllUserAccounts	Will get all the accounts associated with the user and return them to the client	Yes
getAllUserBudgets	Will get all the budgets associated with the suer and return them to the client.	Yes

calculateInvestmentReturn	This endpoint will be able to calculate the future value of an investment based on an initial amount, return rate and periodic contributions	No

For any endpoint that is secured, a JWT bearer token must be included in the request. This means that the user will first have to login to their account before they will be able to retrieve any data. This will keep any bad actors from circumventing the web client and making calls directly to the server. Without an authenticated bearer token in the api request, the request will be refused. This authentication will all be handled by the keycloak server.

The logic on the server side will be broken down into a few different packages. We will have controllers which will act as the entry point for each endpoint and will define the interface of the endpoint. Controllers in most cases will make use of a handler which will contain all the business logic to handle the request. This design ensures that the interface of the endpoint is separated from the business logic and would allow for easy changes or additions to the business logic in the future. The handlers will often times need to access objects from the database. These database objects will be defined by DTOs (Data transfer objects) and there persistence will be managed by DAOs (Data Access Objects). This structure will create a similar flow of logic for each API request that is received and will look something like shown below. Essentially the client will call and endpoint which will grab the required data from the request then call the handler. The handler will then perform some sort of business logic which will likely create or alter a DTO. After the DTO is created/updated the DAO will be called to persist the data to the database. This architecture draws clean lines and will allow for things to be switched out at each level without effecting the rest of the application.

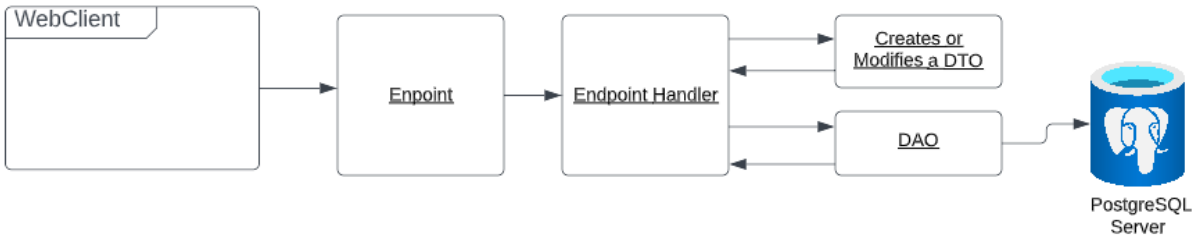

Figure 2: Server Architecture

Design Alternatives

At this stage the react client, spring boot web server and postgres DB have been created and are able to communicate with each other. So those components are set in stone. The keycloak server still needs to be set up and if anything goes wrong during that set up there are alternatives out there that can be used for user authentication. One such alternative is Auth0 [8], which is hosted in the cloud so no server setup would be required. They also provide a nice getting started with spring boot guide [9] so this will be a great backup if the keycloak set up does not go as planned.

Postgres was chosen because I have used mySQL in the past and I am looking to learn how to setup and use a different type of SQL database. That being said, if at any point the database begins to give me issues I can easily swap out the database being used since the program architecture is designed to be agnostic to the database being used.

Deliverable Considerations

The project had originally intended to use Heroku to deploy and host the application, however initial testing showed that the free dynos on Heroku did not have sufficient resources to run the client or the spring boot server. Further options will need to be researched to find a more viable solution to host this application. Alternatives such as AWS, Azure, and Digital Ocean will all be explored as good candidates for hosting this application. For development purposes a docker-compose file will be provided which will allow for easy, consistent spin up of the entire app on a developers local machine.

Release 0 will be able to be reused by any application wishing to have a client server architecture which will support user creation and login. It will serve as a good starting point for a any web based application.

Releases of the software will be made available as zip files on GitHub and Docker images for both the client and server will be pushed to docker hub and will be available for anyone to pull them down. These docker images in docker hub combined with a docker compose file available from GitHub will allow anyone with a docker host the ability to spin up the entire application on their host system.

Schedule

This project will utilize the Agile approach to software development and will create an official major release at the end of each sprint. Sprints will run for 2 weeks starting on Sundays and ending on Saturdays. The first sprint started on August 21st and will end on September 3rd. This first sprint will handle all the initial setup such as creating the spring boot and react apps, setting up the development environment and getting the client, server and DB talking to each other. Below in table 1 the release schedule is laid out.

Release	Date	Notes
Release 0	9/3/2022	The development environment will be set up. Client will be able to sign up users using a create user endpoint on the server.
Release 1	9/17/2022	Focus will be on getting the major features of the application built such as account and transaction management.
Release 2	10/1/2022	Will likely focus on refining the features added in Release 1 and adding budget management
Release 3	10/15/2022	Focus will be on application deployment and making it publicly accessible.
Release 4	10/29/2022	Financial planning calculators and tools will be added to the application.
Release 5	11/12/2022	If time allows options to allow users to sign into their bank accounts and allow transactions to auto synchronize will be explored.
Release 6	11/26/2022	This release will focus on cleaning up any remaining bugs and making sure the application is ready for final release.

Planning will revolve around user story creation, with an initial set of user stories being created as part of this initial planning process. Additional user stories will be created and planned as issues and new features are discovered as part of the development process. A list of initial user stories are shown below and are marked as either must have or nice to have.

Story	Need
As a developer, I want to be able to deploy the entire application with docker-compose	Must Have
As a user, I want to be able to create and login to my account	Must have
As a user I want to be able to be able to add a bank account	Must Have
As a user, I want to be able to manually enter and import a list of transactions	Must Have
As a user I want to be able to create and track a budget	Must Have
As a user, I want a tool to help me plan for retirement	Must Have
As a user, I want my stocks update in value automatically.	Nice to Have
As a user I want to be able to customize the information I see on my dashboard	Nice to have
As a user, I want my transactions to automatically sync from my banking institution.	Nice to have

References

[1] "Intuit Mint," [Online]. Available: https://mint.intuit.com/.

[2] "GnuCash," [Online]. Available: https://www.gnucash.org/.

[3] Y. Chen, Service-Oriented Computing and System Integration: Software, IoT, Big Data and AI as Services, 6th ed., Kendall Hunt Publishing, 2018.

[4] "PostgresSQL," [Online]. Available: https://www.postgresql.org/.

[5] "Keycloak," [Online]. Available: https://www.keycloak.org/.

[6] "Overview of Docker Compose," [Online]. Available: https://docs.docker.com/compose/.

[7] "What is a REST API," [Online]. Available: https://www.redhat.com/en/topics/api/what-is-a-rest-api.

[8] "Auth0 Docs," [Online]. Available: https://auth0.com/docs/manage-users.

[9] "Auth0 - Java Spring Boot," [Online]. Available: https://auth0.com/docs/quickstart/webapp/java-spring-boot/01-login.