**FunBankApp - Software Requirements Specification (SRS)**

**Page 1: Title Page**

**Project Name:** FunBankApp  
**Team Members:** Jarom Brads (Solo Developer)  
**Project Description:**  
FunBankApp is a secure, behaviorally-aware financial calculator designed for everyday users. It leverages user-submitted financial data and the Ollama API to deliver personalized summaries and basic investment advice via a chatbot interface. The application is being built with Spring Boot and hosted on a local SQL server, with plans for AWS deployment.

**Section 1: Introduction**

**Purpose:**  
FunBankApp aims to provide users with an intelligent, simplified view of their financial situation, offering helpful behavioral nudges via Ollama to promote smarter financial decisions.

**Scope:**  
The MVP includes secure user login, financial data input, and automated calculation of retirement estimates. I would like to include chatbot integration using Ollama as a requirement, but it does not seem possible in the current timeline. In order to include Ollama, I might move away from the API’s provide the service client side to focus less on the Cloud aspect. Frontend development, cryptocurrency integration, and cloud deployment are stretch goals.

**Overview:**  
Users register, input their financial information, and receive a retirement income estimate and basic behavioral advice. Information is encrypted and stored securely, and recommendations are presented through a chatbot.

**User Profiles:**  
Everyday users interested in budgeting, retirement planning, or learning about financial wellness.

**Workflows:**

1. User registers and logs in.
2. Inputs financial data.
3. System computes retirement projections.
4. Ollama responds with human-like financial advice.
5. The user clicks continue chatting and is then led to a chatbot window to receive advice based on their financial statements.

**Standards:**

* **Coding**: Follows RESTful standards.
* **Design**: Based on modular Spring Boot services.
* **Learning Model**: Ollama API integration (GPT).
* **Architecture**: Microservice-based with potential for client-side data hosting. Right now it is hosted on a local SQL server, but I would need to move that into the cloud for users to use it, so to keep costs down, I am planning on moving toward client-side data housing in JSON files. Another reason for this is I have found Ollama works well with JSON as compared to other data structures.
* **Quality Assurance**: Unit tests, integration tests, Postman testing(For API’s currently – might move away from Postman).
* **Legal & Security Risks**: Passwords encrypted via BCrypt. JWT for secure sessions. No production-level compliance (yet). As this is a test financial product, I need to do more research to find out if I need to do anything to be totally compliant with financial rules.

**Section 2a: Requirements**

1. **The system shall allow users to register and log in securely.**
   * **Success Measurement**: Users can authenticate with valid credentials. Passwords are stored encrypted.
2. **The system shall allow users to input and store financial account data.**
   * **Success Measurement**: Financial inputs are validated, saved to the database, and retrievable.
3. **The system shall generate a retirement income estimate based on user input.**
   * **Success Measurement**: Application computes projections without errors and provides results in real-time.
4. **The system shall encrypt all personally identifiable information (PII).**
   * **Success Measurement**: Confirm encrypted storage of passwords and sensitive fields in the DB. If I do this as a microservice, I do not know if I need them to login or if I should save any information.

**Section 2b: Stretch Requirements**

1. **The system will provide behavioral financial advice via Ollama to support users in volatile markets.**
2. **The system could integrate cryptocurrency wallets for complete financial overview.**
3. **The system should offer financial planning recommendations using a chatbot interface.**
4. **The system would store user data locally if operating in standalone mode, removing AWS dependency.**

**Section 3: Design Overview**

**Workflow:**

1. Register/Login
2. Enter income, savings, debts, and retirement goals.
3. API calls calculate estimates.
4. Ollama chatbot returns a narrative summary and advice.

**Resources:**

* **Backend**: Spring Boot
* **Database**: MySQL (JDBC) Possibly future (client side JSON)
* **Deployment**: Local, future AWS (RDS, EC2) Possibly future (client side JSON)
* **Auth**: Spring Security, JWT, BCrypt
* **AI**: Ollama API

**3rd Party Libraries:**

* Spring Boot, Lombok, MySQL Connector, Swagger, JWT, SpringDoc

**Architecture/Tiers:**

* Microservice Backend
* (Planned) Frontend Tier (HTML/React or simple Tailwind/CSS) I am really bad at design, so how good this looks will be a good measure of how successful I think it is.

**Data at Rest:**

* Stored in MySQL: hashed passwords, financial inputs.

**Data on the Wire:**

* JSON over HTTPS, JWT for session validation. Right now, I am using JWT, but I might switch to data client-side held with JSON

**Data State Flow Diagram:**

A diagram of a software system

AI-generated content may be incorrect.

As mentioned, data might change to JSON so I can focus on more cool features (ollama and crypto)

**HMI/HCI/GUI:**

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.  
I still don’t have a mvp for the chatbot area yet, as there is more I need to do, but here is a sandbox.html I made with a idea of how I want key parts to look.

**Section 4: Verification**

**Demo Plan:**

* Show login/registration flow
* Submit example financial profile
* Display retirement estimate and chatbot feedback

Here are some examples of tests, I am already done with, even though I might switch from AWS to JSON   
A screenshot of a computer

AI-generated content may be incorrect.  
A screenshot of a computer program

AI-generated content may be incorrect.

**Testing:**

* Unit tests using Spring Test Suite & Mockito
* Postman for manual API checks
* Security validation through JWT and BCrypt

**Sources / Citations / Resources**

* [Spring Security Password Encoding with BCrypt](https://www.baeldung.com/spring-security-registration-password-encoding-bcrypt)
* [Spring Boot JWT Authentication](https://www.javaguides.net/2024/01/spring-boot-security-jwt-tutorial.html)
* [Swagger/OpenAPI Setup](https://swagger.io/docs/)
* [Spring Security Best Practices GitHub](https://github.com/ZeroSTF/Spring-Security-Best-Practices)
* [Software Demo Video](https://youtu.be/hWKAgCgMUdQ)