Personal Finance Management Application

Table of Contents

1. Executive Summary

The Personal Finance Management Application is a comprehensive web-based solution designed to help users track their income, expenses, and budgets effectively. This application aims to provide users with insights into their financial health, enabling them to make informed decisions about their spending and saving habits.

Key highlights:

- Full-stack web application with React frontend and Spring Boot backend
- Secure user authentication and data protection
- Real-time transaction tracking and budget management
- Email notifications for important financial events
- Data visualization for better financial insights

- Hosted on AWS for scalability and reliability

2. Project Overview

Objective: To create a user-friendly, secure, and feature-rich personal finance management tool that helps users take control of their financial life.

Target Audience: Individuals looking to manage their personal finances more effectively, from young adults starting their financial journey to experienced users seeking a comprehensive financial tracking tool.

Core Functionalities:

- User account management
- Income and expense tracking
- Budget creation and monitoring
- Financial reports and analytics
- Email notifications for transactions and budget alerts

Technologies Used:

- Frontend: React, Material-UI
- Backend: Java 11, Spring Boot
- Database: MySQL
- ORM: Hibernate
- Build Tool: Maven
- Version Control: Git
- Deployment: AWS EC2

3. Technical Architecture

The application follows a modern, scalable architecture:

Frontend:

- Framework: React.js
- UI Library: Material-UI
- State Management: React Context API
- HTTP Client: Axios

Backend:

- Framework: Spring Boot (Java 11)
- Database: MySQL
- ORM: Hibernate
- Authentication: JWT (JSON Web Tokens)
- Email Service: Mailgun

Hosting:

- Platform: Amazon Web Services (AWS)
- Services Used: EC2, RDS (optional for database)

4. Features and Functionality

20. **User Authentication**
    - Secure signup and login process
    - Password encryption
    - JWT-based session management
21. **Dashboard**
    - Overview of financial status
    - Recent transactions
    - Budget progress
22. **Transaction Management**
    - Add, edit, and delete transactions
    - Categorize transactions
    - Filter and search functionality
23. **Budget Planning**
    - Create monthly/yearly budgets
    - Set budget limits for different categories
    - Visual representation of budget utilization
24. **Reports and Analytics**
    - Income vs Expense charts
    - Category-wise spending analysis
    - Historical trend analysis
25. **Notifications**
    - Email alerts for new transactions
    - Budget limit warnings
    - Customizable notification settings

5. Development Process

The project follows an Agile development methodology:

26. **Planning**: Requirement gathering and sprint planning
27. **Design**: UI/UX design and technical architecture design
28. **Development**: Iterative development in 2-week sprints
29. **Testing**: Continuous integration and automated testing
30. **Deployment**: Continuous deployment to staging environment
31. **Review**: Sprint review and stakeholder feedback
32. **Iteration**: Incorporate feedback and begin next sprint

6. Backend Implementation

Project Structure

```
backend/
├── src/
│   ├── main/
│   │   ├── java/
│   │   │   └── com/
│   │   │       └── example/
│   │   │           └── financeappbackend/
│   │   │               ├── config/
│   │   │               ├── controller/
│   │   │               ├── dto/
│   │   │               ├── model/
│   │   │               ├── repository/
│   │   │               ├── security/
│   │   │               ├── service/
│   │   │               └── FinanceApplication.java
│   │   └── resources/
│   │       └── application.properties
│   └── test/
└── pom.xml
```

Key Components and Their Functions

33. **FinanceApplication.java**: Main entry point for the Spring Boot application.
34. **Config/**:
    - WebSecurityConfig.java: Configures security settings, CORS, and JWT filter.
    - MailConfig.java: Sets up JavaMailSender for email notifications.
35. **Controller/**:
    - TransactionController.java: Handles CRUD operations for transactions.
    - BudgetController.java: Manages budget-related operations.
    - UserController.java: Handles user profile operations.
    - AuthController.java: Manages authentication processes.
36. **DTO/**:
    - Contains Data Transfer Objects for API requests and responses.
37. **Model/**:
    - User.java: Represents user entity.
    - Transaction.java: Represents transaction entity.
    - Budget.java: Represents budget entity.
38. **Repository/**:
    - TransactionRepository.java: JPA repository for Transaction entity.
    - BudgetRepository.java: JPA repository for Budget entity.
    - UserRepository.java: JPA repository for User entity.
39. **Security/**:
    - JwtUtil.java: Utility for JWT token generation and validation.
    - JwtAuthenticationFilter.java: Filter for JWT-based authentication.
40. **Service/**:
    - TransactionService.java: Business logic for transaction operations.
    - BudgetService.java: Business logic for budget operations.
    - UserService.java: Business logic for user-related operations.
    - EmailService.java: Handles email sending functionality.

7. Frontend Implementation

Project Structure

```
frontend/
├── public/
├── src/
│   ├── components/
│   ├── contexts/
│   ├── services/
│   ├── App.js
│   └── index.js
├── package.json
└── README.md
```

Key Components and Their Functions

41. **Components/**:
    - Dashboard.js: Main dashboard view showing financial overview.
    - TransactionList.js: Displays and manages list of transactions.
    - TransactionForm.js: Form for adding/editing transactions.
    - BudgetList.js: Displays and manages list of budgets.
    - BudgetForm.js: Form for creating/editing budgets.
    - Login.js: Handles user login.
    - Register.js: Handles user registration.
    - Profile.js: Displays and manages user profile.
42. **Contexts/**:
    - AuthContext.js: Manages global authentication state.
43. **Services/**:
    - api.js: Contains functions for making API calls to the backend.
44. **App.js**: Main component setting up routing and global contexts.
45. **index.js**: Entry point of the React application.

8. Database Design

- **Tables**:
    - users: id, username, email, password, created_at, updated_at
    - transactions: id, user_id, amount, category, date, type
    - budgets: id, user_id, category, amount, period
- **Relationships**:
    - One-to-Many: User to Transactions
    - One-to-Many: User to Budgets
- **Indexes**:

    - transactions(user_id, date)
    - budgets(user_id, category)
- **Constraints**:
    - Foreign Key: transactions.user_id references users.id
    - Foreign Key: budgets.user_id references users.id

9. API Design

- RESTful API structure
- Endpoints:
    - /api/auth: Authentication endpoints
    - /api/transactions: CRUD operations for transactions
    - /api/budgets: CRUD operations for budgets
    - /api/users: User-related operations
- Request/Response Format: JSON
- Error Handling: Consistent error response structure
- Versioning: API version included in URL (e.g., /api/v1/)

10. Authentication and Authorization

- **JWT (JSON Web Tokens)**:
    - Library: jjwt 0.9.1
    - Token Expiration: 24 hours
- **Password Encryption**: BCrypt
- **Role-based Access Control**:
    - USER: Regular user permissions
    - ADMIN: Full access to all features
- **Secure Routes**: All routes except /api/auth/* require authentication

11. Email Notification System

- **Email Service**: Mailgun
- **Implementation**: JavaMailSender with Mailgun SMTP
- **Triggered Events**:
    - New account creation
    - Password reset requests
    - Large transactions

    - Budget limit warnings
- **Email Templates**: Thymeleaf for dynamic email content

12. Data Visualization

- **Library**: Chart.js with react-chartjs-2
- **Types of Charts**:
    - Line Chart: Income vs Expenses over time
    - Pie Chart: Expense distribution by category
    - Bar Chart: Monthly budget vs actual spending

13. Performance Optimization

- **Backend**:
    - Database Query Optimization: Using JPA named queries and indexing
    - Caching: Spring Cache with EhCache for frequently accessed data
    - Connection Pooling: HikariCP for efficient database connections
- **Frontend**:
    - Code Splitting: React.lazy() for component-level code splitting
    - Memoization: React.memo() for expensive computations
    - Asset Optimization: Compression and minification of JS/CSS files

14. Security Measures

- **HTTPS**: SSL/TLS encryption using Let's Encrypt
- **CORS**: Configured to allow only specific origins
- **XSS Protection**: React's built-in XSS protection + custom input sanitization
- **CSRF Protection**: Spring Security's CSRF token implementation
- **SQL Injection Prevention**: Use of prepared statements and JPA
- **Rate Limiting**: Custom implementation using bucket4j

15. Setup and Installation

15.1 AWS Setup

46. Create an AWS Account:
    - Go to aws.amazon.com and click "Create an AWS Account"
    - Follow the prompts to set up your account

47. Launch an EC2 Instance:
    - Navigate to EC2 in the AWS Console
    - Click "Launch Instance"
    - Choose "Ubuntu Server 22.04 LTS"
    - Select t2.xlarge for the instance type
    - Configure instance details as needed
    - Add storage (recommend at least 30GB)
    - Add tags (e.g., Name: personal-finance-management)
    - Configure security group to allow SSH (port 22), HTTP (port 80), and HTTPS (port 443)
    - Review and launch
    - Create or select an existing key pair
48. Connect to Your EC2 Instance:
    - Use SSH: `ssh -i /path/to/your-key.pem ubuntu@your-instance-public-ip`
49. Install Required Packages:

```
sudo apt update
sudo apt upgrade -y
sudo apt install openjdk-11-jdk maven nodejs npm mysql-server -y
```

50. Configure MySQL:

```
sudo mysql_secure_installation
```

Follow the prompts to set up a secure MySQL installation

15.2 Local Development Setup (Mac)

51. Install Homebrew:

```
/bin/ -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.s
h)"
```

52. Install Java 11:

```
brew install openjdk@11
```

53. Install Node.js and npm:

```
brew install node
```

54. Install Maven:

```
brew install maven
```

55. Install MySQL:

```
brew install mysql
```

56. Start MySQL Service:

```
brew services start mysql
```

57. Clone the Repository:

```
git clone https://github.com/dineshroxonn/personal-finance-management-app.git
cd personal-finance-management-app
```

58. Set Up Backend:

```
cd backend
mvn clean install
```

59. Set Up Frontend:

```
cd ../frontend
npm install
```

16. Running the Application

Backend:

60. Navigate to the backend directory:

```
cd backend
```

61. Run the Spring Boot application:

```
mvn spring-boot:run
```

Frontend:

62. Navigate to the frontend directory:

```
cd frontend
```

63. Start the React development server:

```
npm start
```

Access the application at `http://localhost:3000`

17. Deployment and Hosting

- **Production Environment**: t2.xlarge EC2 instance

- **Database**: MySQL server on the same EC2 instance (can be migrated to RDS for better scalability)
- **Domain and SSL**: Configured with a custom domain and SSL certificate for secure access
- Deployment steps as previously outlined

18. Future Enhancements and Unimplemented Features

The following features and aspects of the project have not been implemented yet:

64. **Testing and Quality Assurance**:
    - Comprehensive unit testing, integration testing, and end-to-end testing are yet to be implemented.
    - Performance testing and security testing are planned but not yet executed.
65. **CI/CD Pipeline**:
    - Continuous Integration and Continuous Deployment pipelines have not been set up.
66. **Monitoring and Logging**:
    - Centralized logging system and detailed application monitoring are not yet in place.
67. **Third-party Integrations**:
    - Integrations with payment processing systems, OAuth providers, and cloud storage services are planned but not implemented.
68. **Advanced Features**:
    - Mobile application development
    - Bank account integration for automatic transaction import
    - Machine learning-based expense predictions
    - Multi-currency support
69. **Code Quality Tools**:
    - Integration of static code analysis tools and automated code formatting is pending.

Conclusion

The Personal Finance Management Application provides a robust, secure, and user-friendly platform for individuals to take control of their financial lives. With its comprehensive feature set, scalable architecture, and focus on user experience, the application is well-positioned to meet the diverse needs of users in managing their personal finances effectively.

This project demonstrates the successful implementation of modern web technologies and best practices in software development, resulting in a high-quality, maintainable, and scalable application. The use of AWS for hosting ensures reliability and scalability, while the robust security measures protect user data and privacy.

As financial management continues to be a critical aspect of people's lives, this application stands ready to evolve and adapt to meet future needs and technological advancements.