BankBackend Service Overview

BankBackend is a Java-based backend application providing core banking services. It exposes RESTful APIs for managing customer **accounts**, **loans**, **KYC (Know Your Customer)** data, and **transactions**, along with authentication and security features. The service is built with Spring Boot and MongoDB [1]. It follows a layered architecture: controllers handle HTTP requests, services implement business logic, and repositories manage data persistence [2] [3]. Key modules include Accounts, Loans, KYC, Transactions, and Security (authentication/authorization, OTP).

Architecture and Modules

The application uses the standard Spring Boot project structure and modular design [2] [3]. Each banking domain is a separate module/package:
- **Accounts** – manages customer bank accounts (open, update, close accounts).
- **Loans** – handles loan applications, approval, disbursement, and repayments.
- **KYC** – processes customer identity documents and verification status.
- **Transactions** – processes financial transactions (deposits, withdrawals, transfers) and maintains transaction history.
- **Security** – handles user registration/login, role-based access, and OTP (One-Time Password) verification.

Controllers in each module define the REST endpoints (paths and HTTP methods), services contain business operations, and repository classes (using Spring Data MongoDB) perform database interactions [2] [3]. The application also includes configuration and utility components (e.g. JWT security config). Scheduled tasks are used for periodic jobs like interest calculation and auto-debit (Spring's `@EnableScheduling` enables this capability) [4].

Technologies Used

- **Java** (core language)
- **Spring Boot** (framework for REST API, dependency management) [1]
- **Spring MVC** (REST controllers)
- **Spring Data MongoDB** (NoSQL persistence) [1]
- **Spring Security** (authentication/authorization)
- **Spring Scheduling** (`@EnableScheduling` for periodic tasks) [4]
- **Maven** (build and dependency management)
- **JUnit** (unit testing framework)
- **JWT (JSON Web Tokens)** for stateless auth (if implemented)

Features

- **Account Management:** CRUD operations for bank accounts, balance inquiry, and account status (active/closed).

- **Loan Processing:** Apply for loans; handle approval, **loan disbursement**, and repayment tracking.
- **KYC Workflow:** Upload and verify KYC documents; retrieve verification status.
- **Transactions:** Create and record financial transactions (deposits, withdrawals, transfers); maintain transaction history.
- **Scheduled Tasks:** Automatic **interest calculation** and **auto-debit** of loan EMIs using scheduled jobs (Spring's scheduling support is enabled) [4] . For example, interest may be computed daily or monthly, and loan payments may be debited automatically on schedule.
- **Security & OTP:** User registration/login with role-based access; **OTP (One-Time Password)** generation and verification for sensitive operations (e.g. login or high-value transactions).

API Endpoints

Accounts

- `GET /accounts` – Retrieve a list of all accounts.
- `GET /account/{id}` – Retrieve details of a specific account by its ID.
- `POST /account` – Create a new bank account.
- `PUT /account/{id}` – Update an existing account (e.g. modify customer info or status).
- `DELETE /account/{id}` – Close or delete an account.

KYC (Know Your Customer)

- `POST /kyc` – Submit KYC details (e.g. documents or form data).
- `GET /kyc/{customerId}` – Retrieve KYC verification status for a given customer.
- `PUT /kyc/{customerId}` – Update or approve KYC status after review.

Loans

- `GET /loans` – List all loan accounts (or query by customer).
- `GET /loan/{id}` – Get details of a specific loan by its ID.
- `POST /loan` – Apply for a new loan (create loan account).
- `PUT /loan/{id}` – Update a loan (e.g. approve, disburse funds, or modify terms).
- `DELETE /loan/{id}` – Close or cancel a loan (if supported).

Transactions

- `GET /transactions` – List transactions (optionally by account).
- `GET /transaction/{id}` – Retrieve a specific transaction by ID.
- `POST /transaction` – Create a new transaction (deposit, withdrawal, or transfer).
- `GET /transactions/account/{accountId}` – List all transactions for a specific account.

Security (Authentication / Users / OTP)

- `POST /register` – Register a new user/customer account.
- `POST /login` – Authenticate a user and return an access token.
- `POST /logout` – Invalidate user session or token (if implemented).
- `POST /otp/send` – Send/generate a One-Time Password for verification (e.g. to email or SMS).
- `POST /otp/verify` – Verify a submitted OTP for authentication or transaction confirmation.

- `GET /user/{id}` – Retrieve user profile or role information by user ID (if applicable).

Each endpoint follows standard HTTP methods (GET for retrieval, POST for creation, PUT for updates, DELETE for removals) and exchanges JSON data. The above endpoints cover the major operations; additional paths (e.g. query endpoints or admin actions) may be included as needed.

Sources: The service is implemented using Spring Boot with Spring Data MongoDB [1] and follows a typical Spring layered architecture (controllers/services/repositories) [2] [3]. Spring's scheduling support is used for periodic tasks like interest and auto-debit calculations [4].

---

[1] Building a core banking microservice with Spring boot and MongoDB | by Page against the machine | Medium
https://medium.com/@johnlpage/introduction-359bedda58b1

[2] [3] Spring Boot Folder Structure (Best Practices) | by Malshani Wijekoon | Medium
https://malshani-wijekoon.medium.com/spring-boot-folder-structure-best-practices-18ef78a81819

[4] Getting Started | Scheduling Tasks
https://spring.io/guides/gs/scheduling-tasks/