BankApp

BankApp is a full-stack banking application that enables users to manage accounts, perform transactions, and view balances through a secure and user-friendly interface. This documentation provides a comprehensive overview of the repository, covering setup, architecture, APIs, and best practices.

Table of Contents

Overview

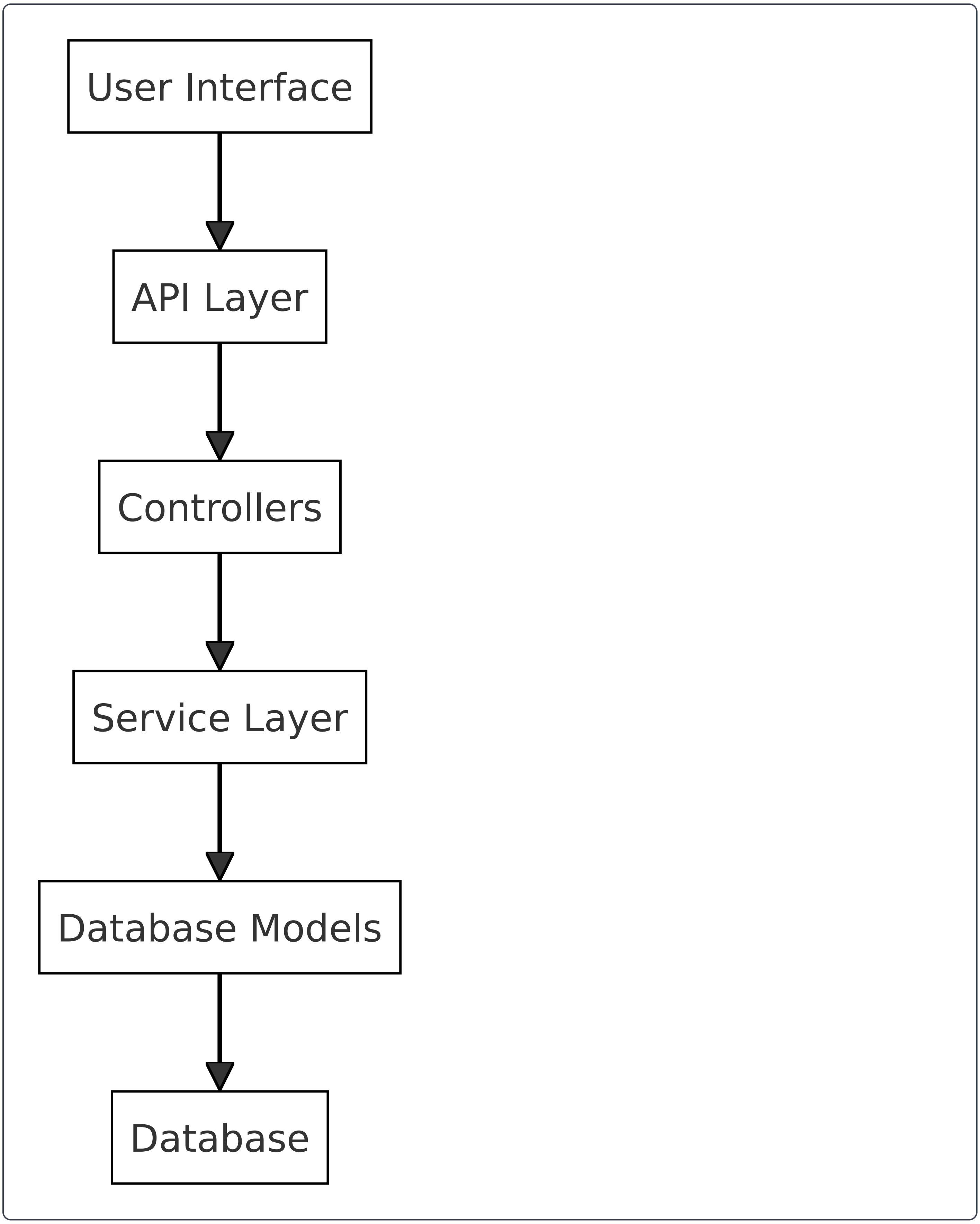
BankApp is designed for educational and demonstration purposes. It provides core functionalities of a banking system, such as account management, fund transfers, transaction history, and authentication.

Features

- User registration and authentication
- Secure login with session management
- Account creation and management
- Fund transfers between accounts
- Transaction history and balance tracking
- RESTful API endpoints
- Error handling and input validation

Architecture

BankApp follows a layered architecture with clear separation between models, controllers, services, and routes. It uses an MVC (Model-View-Controller) pattern to promote maintainability and scalability.



Installation

Follow these steps to install BankApp locally:

1. Clone the repository:

```
git clone https://github.com/bhaskarsingamshetty/bankapp.git
cd bankapp
```

2. Install dependencies:

```
npm install
```

3. Set up environment variables as described in the Configuration section.

Configuration

BankApp requires environment variables for database connection, authentication, and server configuration. Create a `.env` file in the root directory with the following:

```
PORT=3000
DB_HOST=localhost
DB_PORT=5432
DB_USER=your_db_user
DB_PASSWORD=your_db_password
DB_NAME=bankapp
JWT_SECRET=your_jwt_secret
```

Running the Application

Start the application with:

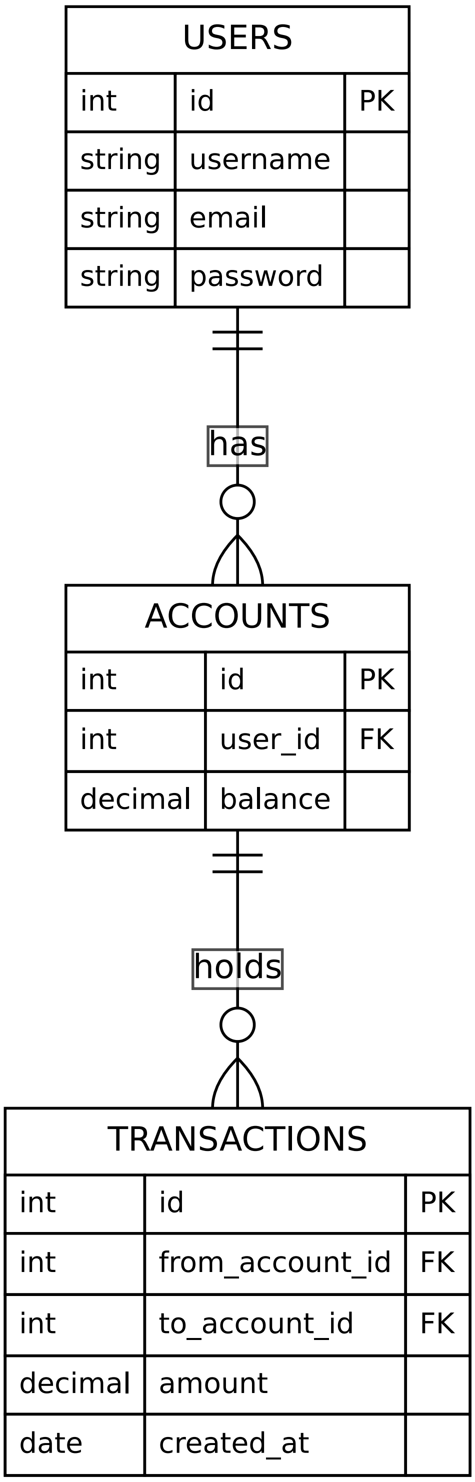
```
npm start
```

For development with hot-reloads:

```
npm run dev
```

Database Schema

The database consists of users, accounts, and transactions tables. Relationships are managed via foreign keys.



API Documentation

BankApp exposes a set of RESTful endpoints for core banking operations. Each endpoint requires proper authentication.

Register New User (POST /api/register)

Endpoint

User Registration

Export to Postman

Register a new user with username, email, and password.

POST `https://api.bankapp.com/api/register`

Request body

JSON payload required for this request.

```
{
  "username": "jane_doe",
  "email": "jane@example.com",
  "password": "securepass123"
}
```

Code examples

```
curl -X POST "https://api.bankapp.com/api/register" \
  -H "Content-Type: application/json" \
  -d '{
  \"username\": \"jane_doe\",
  \"email\": \"jane@example.com\",
  \"password\": \"securepass123\"
}'
```

Responses

201 400

User created successfully

```
{
  "message": "User registered successfully."
}
```

Validation error

```
{
  "error": { "message": "Missing required fields." }
}
```

User Login (POST /api/login)

Endpoint

User Login

Export to Postman

Authenticate user and issue JWT token.

POST `localhost:8080/api/login`

Request body

JSON payload required for this request.

```
{
  "email": "jane@example.com",
  "password": "securepass123"
}
```

Code examples

```
curl -X POST "localhost:8080/api/login" \
  -H "Content-Type: application/json" \
  -d '{
  \"email\": \"jane@example.com\",
  \"password\": \"securepass123\"
}'
```

Responses

200 401

Login successful, returns JWT token

```
{
  "token": "<jwt_token>"
}
```

Authentication failed

```
{
  "error": { "message": "Invalid credentials." }
}
```

Get Account Balance (GET /api/accounts/:id/balance)

Endpoint

Get Account Balance

Export to Postman

Retrieve current balance for a specified account.

GET `https://api.bankapp.com/api/accounts/:id/balance`

Headers

`Authorization` string • header required

Bearer <token>

Path parameters

`id` string • path required

Account ID

Code examples

```
curl -X GET "https://api.bankapp.com/api/accounts/:id/balance" \
  -H "Authorization: Bearer <token>"
```

Responses

200 404

Balance retrieved

```
{
  "account_id": 1,
  "balance": 1000.00
}
```

Account not found

```
{
  "error": { "message": "Account does not exist." }
}
```

Transfer Funds (POST /api/accounts/transfer)

Endpoint

Transfer Funds

Export to Postman

Transfer funds between accounts.

POST `https://api.bankapp.com/api/accounts/transfer`

Headers

`Authorization` string • header required

Bearer <token>

Request body

JSON payload required for this request.

```
{
  "from_account_id": 1,
  "to_account_id": 2,
  "amount": 50.00
}
```

Code examples

```
curl -X POST "https://api.bankapp.com/api/accounts/transfer" \
  -H "Authorization: Bearer <token>" \
  -H "Content-Type: application/json" \
  -d '{
  \"from_account_id\": 1,
  \"to_account_id\": 2,
  \"amount\": 50.00
}'
```

Responses

200 400

Transfer successful

```
{
  "message": "Transfer completed successfully."
}
```

Insufficient funds

```
{
  "error": { "message": "Insufficient balance." }
}
```

View Transaction History (GET /api/accounts/:id/transactions)

Endpoint

View Transaction History

Export to Postman

Retrieve transaction history for a specific account.

GET `https://api.bankapp.com/api/accounts/:id/transactions`

Headers

`Authorization` string • header required

Bearer <token>

Path parameters

`id` string • path required

Account ID

Code examples

```
curl -X GET "https://api.bankapp.com/api/accounts/:id/transactions" \
  -H "Authorization: Bearer <token>"
```

Responses

200 404

Transaction history retrieved

```
{
  "transactions": [
    {
      "id": 1,
      "amount": 100.00,
      "type": "deposit",
      "date": "2024-05-10"
    }
  ]
}
```

Account not found

```
{
  "error": { "message": "Account does not exist." }
}
```

Testing

BankApp includes unit and integration tests for all major components. Run tests with:

```
npm test
```

Test coverage includes models, controllers, and API endpoints.

Contributing

We welcome contributions! Please follow these steps:

- Fork the repository
- Create a feature branch
- Commit your changes with clear messages
- Open a pull request describing your changes

Make sure to add tests and update documentation as needed.

Best Practices

- Validate all user input
- Use HTTPS in production
- Store sensitive data securely (never commit secrets)
- Follow consistent code style and naming conventions
- Write tests for new features and bug fixes

Troubleshooting

- Ensure database service is running and credentials are correct.
- Check `.env` configuration for correctness.
- Review application logs for error details.
- Restart the server after configuration changes.

License

This project is licensed under the MIT License.

Contact

For questions, open an issue or contact the maintainer via GitHub Issues.

Security Reminder
Never share your JWT tokens or sensitive credentials. Always use environment variables for secrets.