## **1. Einleitung**

## **1.1 Ziel des Projekts**

Das Ziel dieses Projekts ist die Entwicklung eines sicheren und benutzerfreundlichen Finanzverwaltungssystems. Das System ermöglicht Benutzern:

* Sich sicher anzumelden,
* Ihre Kontoeinstellungen zu verwalten (z. B. Passwortänderung),
* Finanztransaktionen durchzuführen,
* und diese in einer Historie nachverfolgen können.

Ein besonderer Fokus liegt auf der Implementierung moderner Sicherheitsmechanismen wie **risikobasierter Authentifizierung** und **Zwei-Faktor-Authentifizierung (2FA)**.

## **1.2 Motivation**

Die Software soll das Problem unzureichender Sicherheit bei der Verwaltung sensibler Finanzdaten lösen. Viele Benutzer verwenden einfache Passwörter oder keine zusätzlichen Sicherheitsmaßnahmen, was sie anfällig für Cyberangriffe macht. Durch die Implementierung von JWT, eines dynamischen Zugriffkontrollsystems und MFA wird ein höheres Maß an Sicherheit erreicht, das den Zugriff auf persönliche Finanzdaten schützt.

## **1.3 Zielgruppe**

Die Zielgruppe des Systems sind:

* Endbenutzer, die ihre Finanzen verwalten möchten,
* Administratoren, die das System warten und verwalten.

## **2. Systemarchitektur**

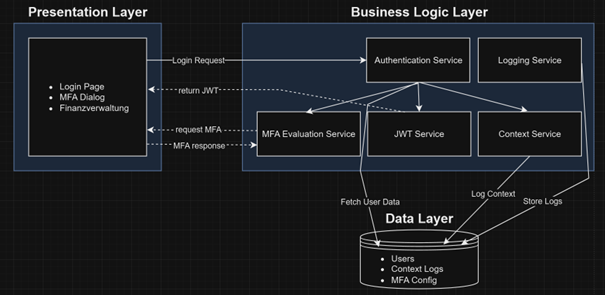
## **2.1 Überblick**

Das System basiert auf einer **Client-Server-Architektur**, bei der das Backend mit Spring Boot entwickelt wurde und das Frontend HTML, CSS, JavaScript sowie Thymeleaf verwendet.

## **2.2 Technologien**

| **Komponente** | **Technologie** | **Beschreibung** |
| --- | --- | --- |
| Backend | Kotlin, Spring Boot | RESTful API für Geschäftslogik |
| Frontend | Thymeleaf, JavaScript | Benutzeroberfläche mit serverseitigem Rendering |
| Sicherheit | Spring Security | Authentifizierungs- und Autorisierungsmechanismen |
| Datenbank | PostgreSQL | Speicherung der Benutzerdaten, Transaktionen und Login\_Events |

## **2.3 Architekturdiagramm**

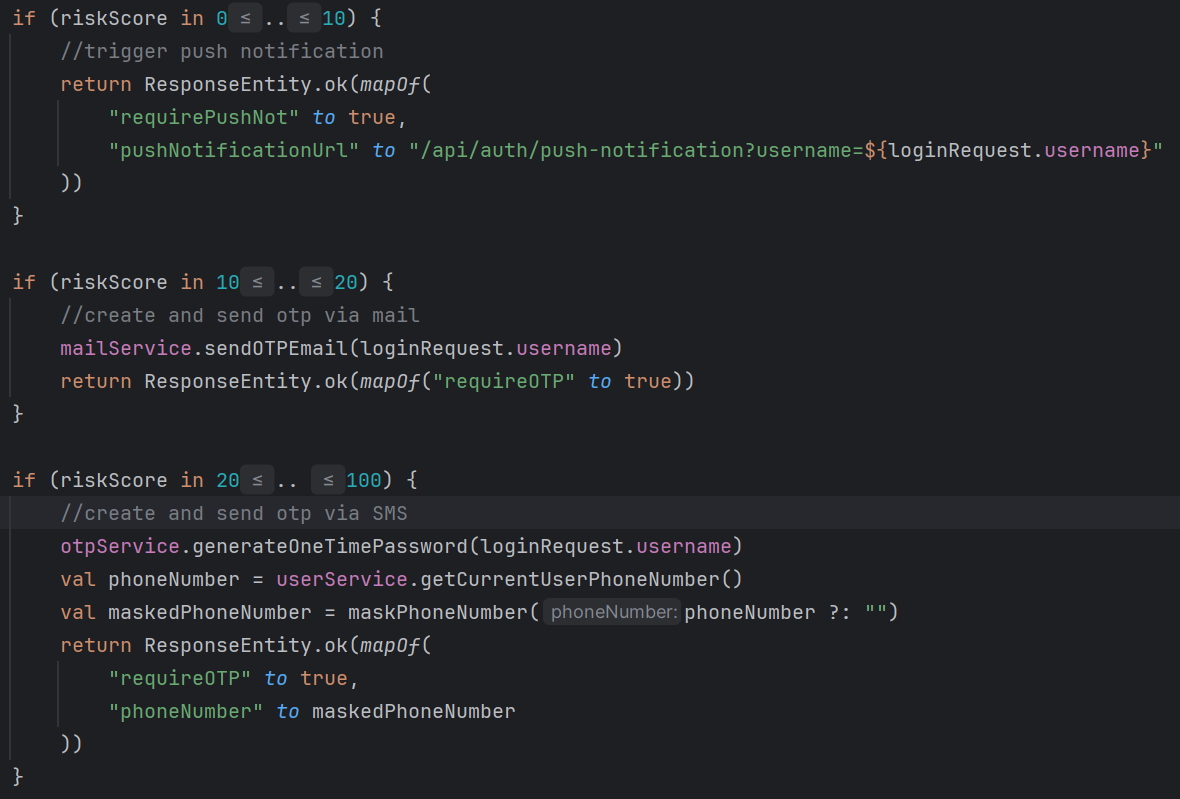


## **3. Hauptfunktionen**

## **3.1 Risikobasierte Benutzerauthentifizierung**

Die Authentifizierung basiert auf einem risikobasierten Ansatz:

* **Risikobewertung:** Ein Algorithmus berechnet einen Risiko-Score basierend auf Benutzerverhalten.
* **Dynamische Authentifizierungsmethoden:**
  + Score < 10: Push-Benachrichtigung.
  + Score 10–20: E-Mail OTP.
  + Score > 20: SMS OTP.

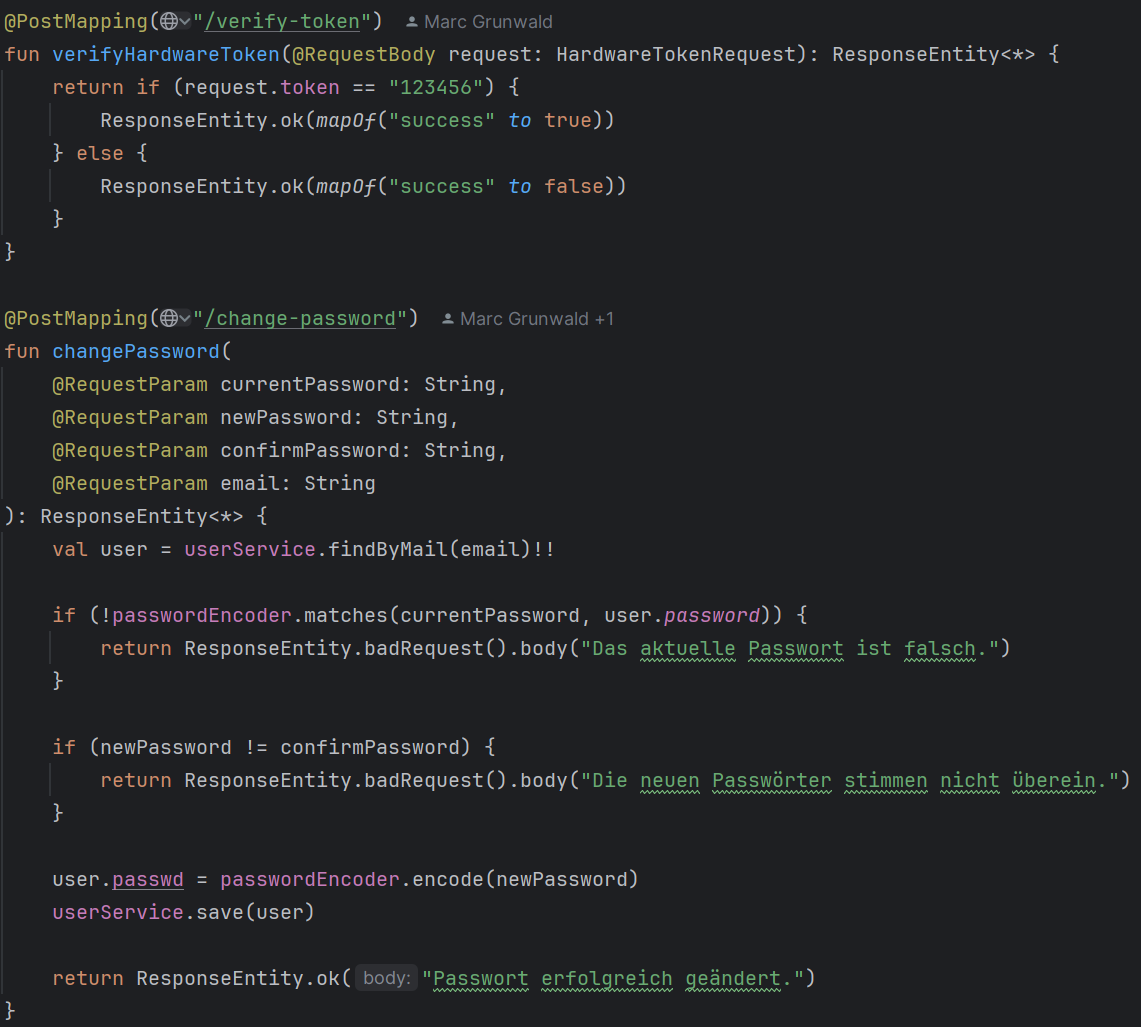


## **3.2 Kontoeinstellungen**

Benutzer können ihre Kontoeinstellungen (z. B. Passwort) ändern. Dabei wird ein **Hardware-Token** simuliert, um zusätzliche Sicherheit zu gewährleisten.

## **Workflow:**

1. Benutzer füllt das Formular zur Passwortänderung aus.
2. System fordert den Benutzer auf, seinen Hardware-Token einzugeben.
3. Nach erfolgreicher Verifizierung wird das Passwort geändert.



## **3.4 Geldtransfer**

* Implementiert eine zusätzliche Sicherheitsüberprüfung für Überweisungen über 500 Einheiten.
* Nutzer ohne "AUTHORIZED" Rolle müssen einen OTP für solche Überweisungen eingeben.

## **3.5 Rollenbasierte Zugriffskontrolle**

* Implementiert ein System zur Verwaltung von Benutzerrollen.
* Rollen können dynamisch aktualisiert werden, z.B. nach erfolgreicher OTP-Verifizierung bei Überweisungen.

## **4. Sicherheitskonzept**

## **4.1 Risiko-basierte Authentifizierung**

Die Authentifizierungsmethode wird basierend auf einem Risiko-Score ausgewählt:

* **Niedriges Risiko (<10):** Push-Benachrichtigung.
* **Mittleres Risiko (10–20):** E-Mail OTP.
* **Hohes Risiko (>20):** SMS OTP.

## **Vorteile:**

* Reduziert unnötige Sicherheitsmaßnahmen für vertrauenswürdige Anmeldungen.
* Erhöht die Sicherheit bei verdächtigen Aktivitäten.

## **4.2 Multi-Faktor-Authentifizierung (MFA)**

* Für verschiedene Szenarien wird MFA verwendet:

1. Anmelden von einem neuen Gerät: SMS/E-Mail OTP (SMS simuliert)
2. Regelmäßige Anmeldung: Push Bestätigung
3. **SICHERHEITSFRAGEN: Wann nochmal genau?**
4. Hohe Transaktion: E-Mail OTP
5. Kontoänderungen: Hardware-Token (simuliert)

## **Workflow:**

1. Benutzer gibt Anmeldedaten ein.
2. System entscheidet anhand des Risiko-Scores über die MFA-Methode.
3. Benutzer führt zusätzliche Verifizierung durch.

## **4.3 JWT-basierte Authentifizierung**

* Verwendung von JWTs für zustandslose Authentifizierung.
* Struktur des JWT:
  + Header: Algorithmus und Typ
  + Payload: Benutzerinformationen
  + Signature: Gehashte Kombination aus Header, Payload und Secret

## **4.4 Dynamisches Zugriffskontrollsystem**

* Analyse des Benutzerverhaltens (z.B. IP-Adresse, Standort).
* Anpassung der Sicherheitsmaßnahmen basierend auf dem Risiko.

## **4.5 Passwort-Sicherheit**

* Verwendung von PasswordEncoder für sicheres Hashing.
* Verifizierung des aktuellen Passworts vor Änderungen.

## **4.6 Sicherheitsfragen**

* Implementiert als zusätzliche Sicherheitsmaßnahme bei mittlerem Risiko-Score.
* Benutzer müssen eine vordefinierte Sicherheitsfrage korrekt beantworten.
* Nach erfolgreicher Eingabe der Antwort, wird der Benutzer automatisch eingeloggt.

## **4.7 Push-Benachrichtigungen**

* Verwendet für Authentifizierung bei niedrigem Risiko-Score.
* Benutzer erhalten eine Push-Benachrichtigung zur Bestätigung des Login-Versuchs.
* Push Benachrichtigung wird im Browser simuliert. Nach Bestätigung durch einen Button wird der Benutzer automatisch eingeloggt.

**4.8 OTP Generierung**

* Verwendung von JSON Web Tokens (JWT) für sichere, zeitlich begrenzte Tokens
* Signierung mit HMAC-SHA256-Algorithmus
* Versand des OTP per E-Mail oder SMS
* Integration in den risikobasierten Authentifizierungsprozess
* Zeitliche Begrenzung der Tokens zur Minimierung von Replay-Angriffen
* Maskierung sensibler Daten (z.B. Telefonnummern) beim Versand

**4.9 Hardware-Token-Anfrage**

* Verwendung einer Hardware-Token Anfrage beim Ändern von Einstellungen des Benutzers.
* Diese Hardware-Token Anfrage ist simuliert. Der Hardware-Token (“123456”) ist im Code implementiert.
* Bei erfolgreicher Eingabe des Hardware-Token werden die Daten des Benutzers angepasst.

**5. Persistence-Kompontenten**

**5.1 User**

**Funktionen:**

* Registrierung und Anmeldung in der Applikation
* Tätigen einer Überweisung
* Fähigkeit seine Daten (Passwort) anzupassen

**Komponenten:**

* **User\_Id**: Eindeutige Identifizierung des Benutzers
* **mail**: E-Mail Adresse des Benutzers (erforderlich zur Authentifizierung)
* **password**: Passwort des Benutzers (erforderlich zur Authentifizierung)
* **role**: Rolle des Benutzers
* **enabled**: Status des Benutzers, ob dieser freigeschaltet ist oder nicht
* **budget**: Das derzeitige Budget des Benutzers
* **one\_time\_password**: One Time Passwort des Benutzers, falls erforderlich
* **otp\_request\_time**: Zeitpunkt der Anfrage des OTPs
* **otp\_expiry\_date**: Zeitpunkt des Ablaufs des OTPs (20 Minuten)
* **phone\_number**: Telefonnummer des Benutzers

**5.2 Transaktion**

**Funktionen:**

* Speichern aller Transaktionen, die zwischen Benutzern stattfinden
* Erforderlich, um die Historie von Transaktionen nachvollziehen zu können

**Komponenten:**

* **id**: Eindeutige Identifizierung einer Transaktion
* **sender\_id**: Eindeutige Id des sendenden Benutzers
* **recipient\_id**: Eindeutige Id des empfangenen Benutzers
* **amount**: Betrag, der überwiesen wird
* **timestamp**: Zeitpunkt der Überweisung

**5.3 Login Events**

**Funktionen:**

* Berechnung des Risk-Scores

**Komponenten:**

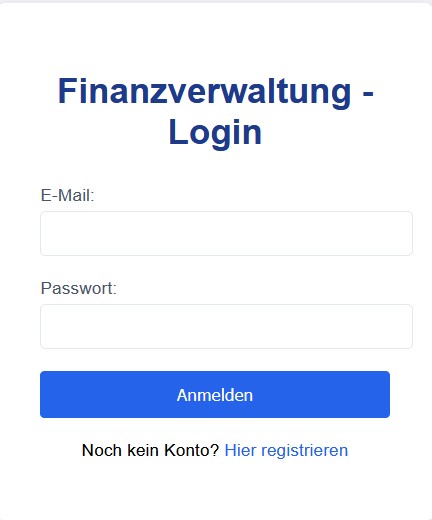
* **id**: Eindeutige Identifizierung eines Login Events
* **user\_id**: Verweis auf den Benutzer des Login Events
* **mail**: E-Mail Adresse des Benutzers
* **login\_time**: Zeitpunkt des Logins
* **ip\_adress**: IP-Adresse des Benutzers
* **location**: Standort des Benutzers
* **browser**: Browser des Benutzers während des Login Prozesses
* **browser\_version**: Derzeitige Browserversion
* **operating\_system**: Betriebssystem des Benutzers
* **status**: Status des Benutzers, ob dessen Authentifizierung erfolgreich war (Ja/Nein)

## **6. Frontend-Komponenten**

## **6.1 Login-Seite**

## **Funktionen:**

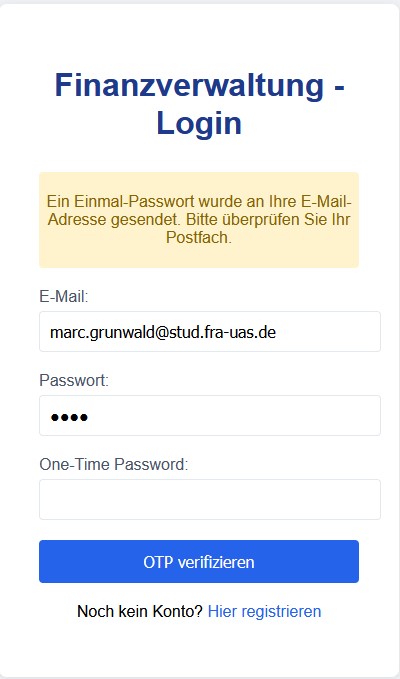
* Eingabefelder für E-Mail und Passwort.
* Dynamische Anzeige von OTP-Feldern basierend auf der Serverantwort.

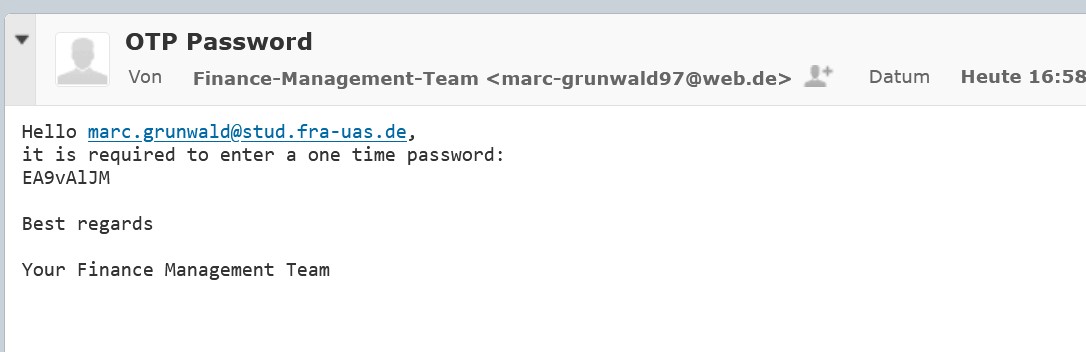


**6.1.1 Login-Seite mit Anforderung einer Push-Bestätgigung**

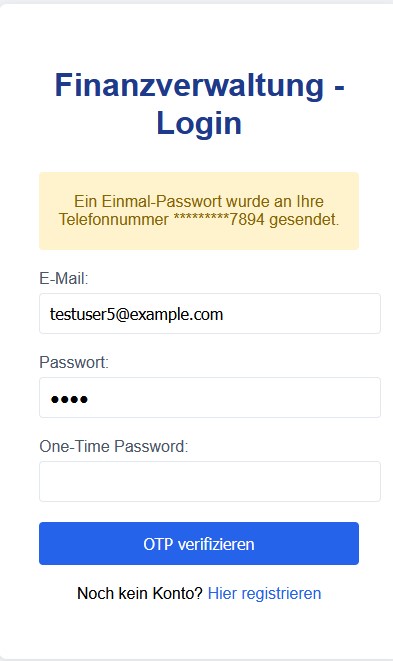


**6.1.2 Login-Seite mit Anforderung eines E-Mail OTP**





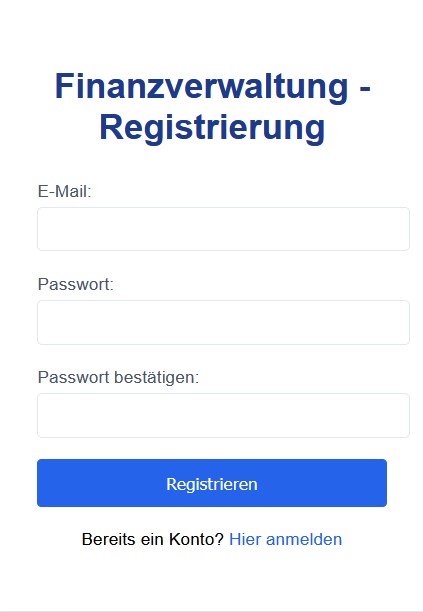
**6.1.3 Login-Seite mit Anforderung eines SMS OTP**



**6.2 Registierungs-Seite**

**Funktionen:**

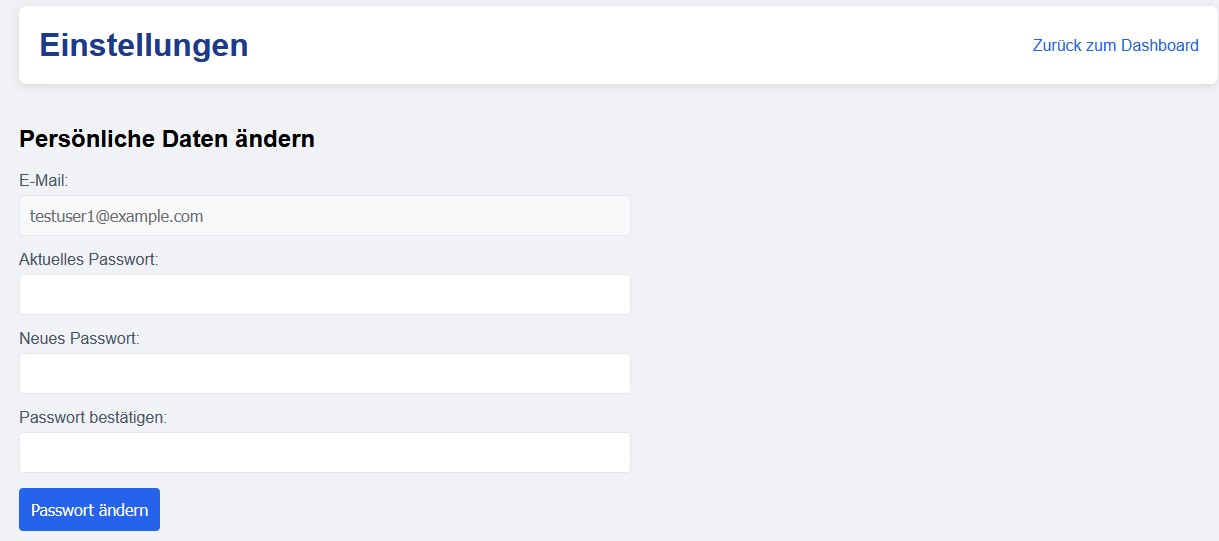
* Registrierung eines neuen Benutzers durch Angabe seiner E-Mailadresse und eines Passworts

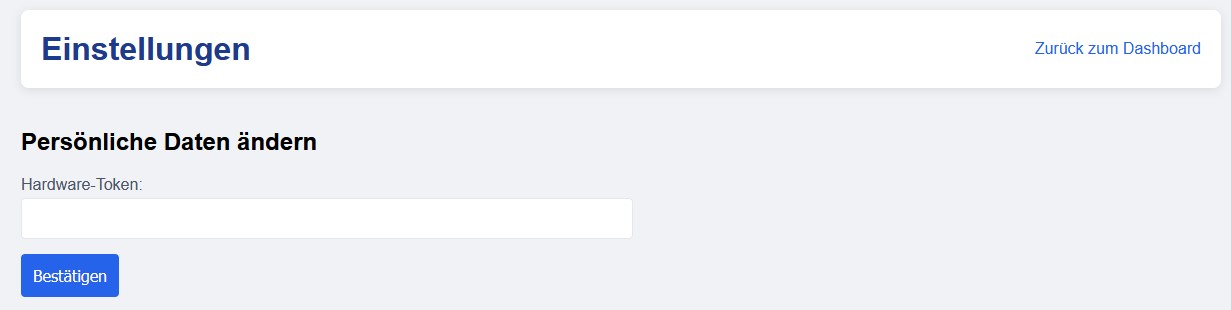


## **6.3 Einstellungsseite**

## **Funktionen:**

* Formular zur Änderung des Passworts.
* Eingabefeld für Hardware-Token bei sicherheitsrelevanten Aktionen.

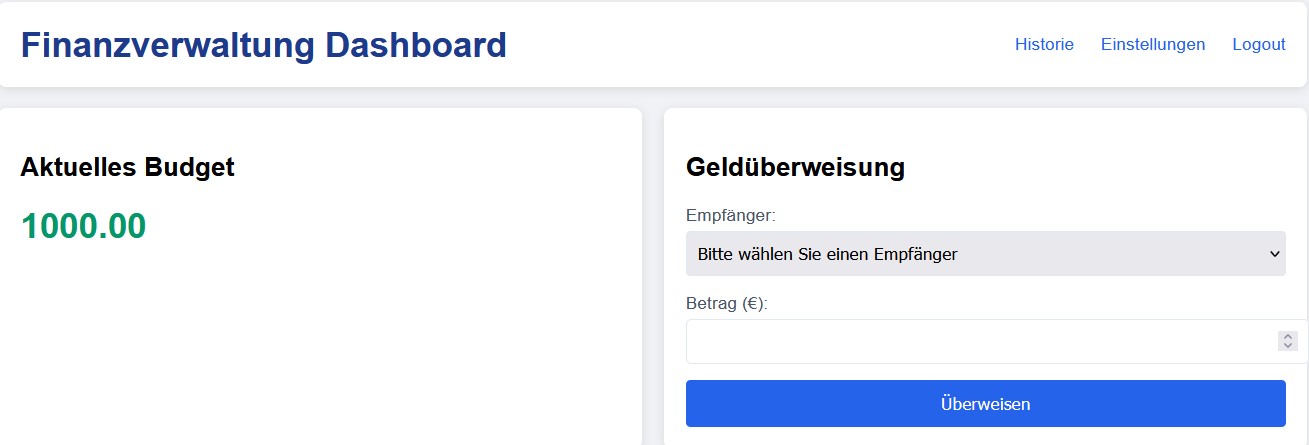




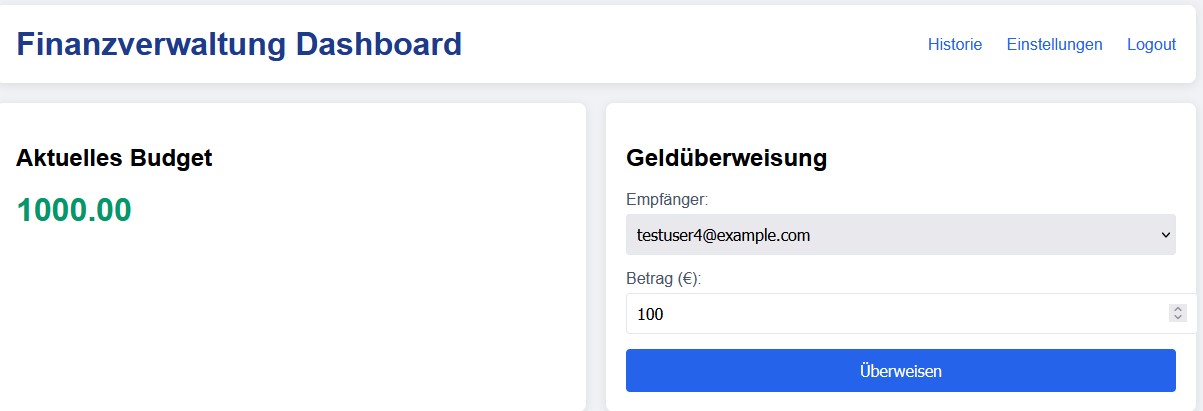
**6.4 Dashboard-Seite**

**Funktionen:**

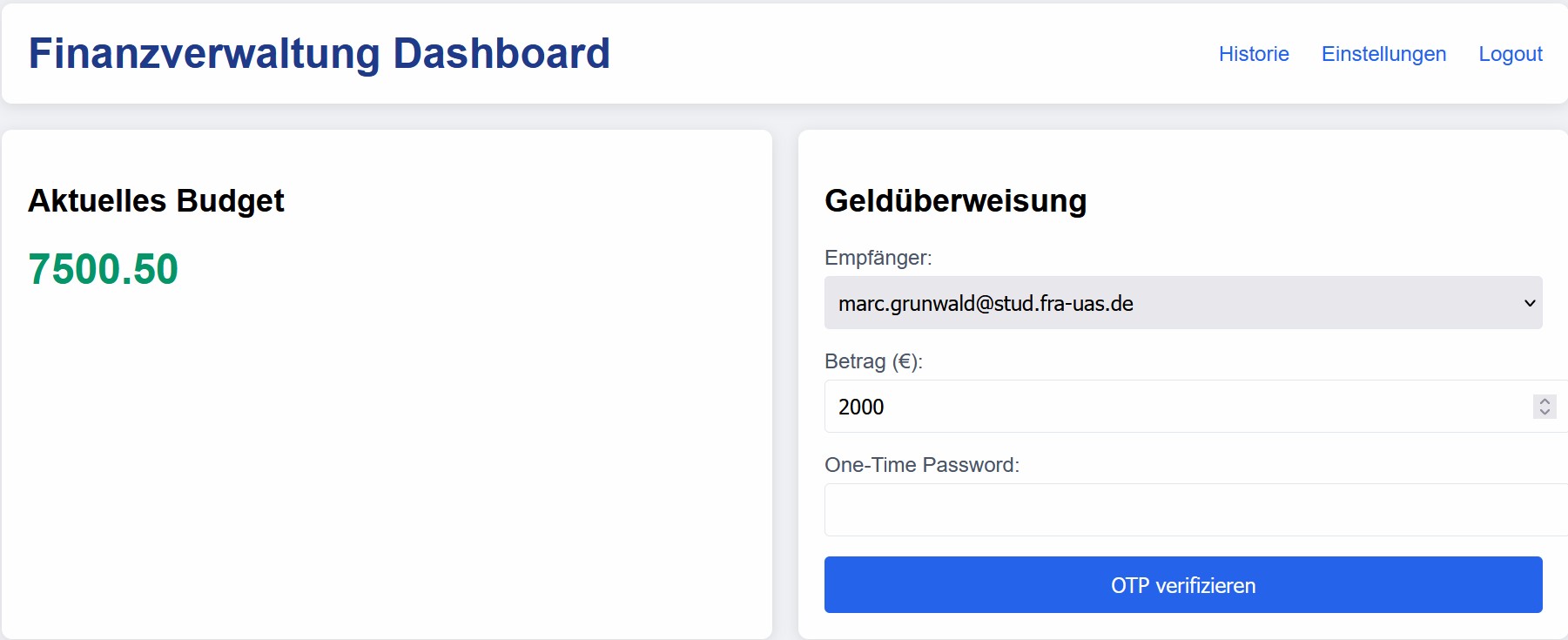
* Formular zur Überweisung an einen Benutzer



* Überweisung eines Betrags an einen Benutzer



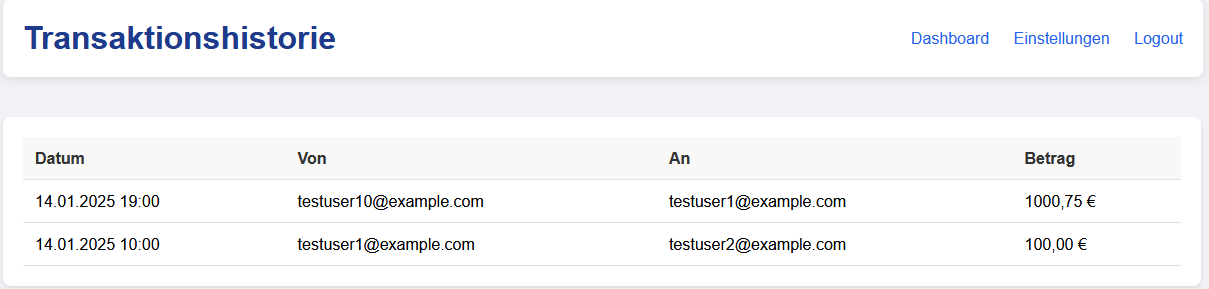
* Überweisung eines hohen Betrags an einen Benutzer mit anschließender OTP-Verifizierung



**6.5. Historie**

**Funktionen:**

* Übersicht der Transaktionshistorie bestehend aus:
  + Datum
  + Sender und Empfänger
  + Betrag



**7 Frameworks**

**7.1 Backend**

* Ktor: Framework für die Erstellung der RESTful API in Kotlin
* Maven: Projektmanagementwerkzeug

**7.2 Frontend**

* Thymeleaf: Für serverseitiges Rendering von HTML-Templates

**7.3 Sicherheit**

* Spring Security: Unterstützung für Authentifizierung und Autorisierung (diese wurde eigenständig umgesetzt, Spring liefert dabei geeignete Module zur Umsetzung dieser Themen)

**7.4 Datenbank**

* MySQL: Für die Speicherung von Benutzerdaten, Finanzinformationen und Login-Events

**7.5 JWT**

* jsonwebtoken: Bibliothek zur Erstellung und Verifizierung von JWTs

**7.6 MFA**

* SendGrid: Versenden von E-Mails

**8 Nutzer-/System-Schnittstellen**

## **8.1 Login-Seite**

* **URL:** /login
* **Funktionalität:**
  + Benutzer können sich mit E-Mail-Adresse und Passwort anmelden.
  + Implementiert risikobasierte Authentifizierung mit verschiedenen MFA-Methoden:
    - Push-Benachrichtigung (Risiko-Score 10-20)
    - Sicherheitsfrage (Risiko-Score 20-30)
    - E-Mail OTP (Risiko-Score 30-40)
    - SMS OTP (Risiko-Score 40-100)
  + Dynamische Anzeige zusätzlicher Authentifizierungsfelder basierend auf dem Risiko-Score.
  + Weiterleitung zum Dashboard nach erfolgreicher Anmeldung.

## **8.2 Dashboard**

* **URL:** /dashboard
* **Funktionalität:**
  + Zentrale Übersicht über die Finanzdaten des Benutzers.
  + Anzeige von Ausgaben, Budgets und Berichten.
  + Grafische Darstellung der finanziellen Aktivitäten.
  + Zugriff auf weitere Funktionen wie Kontoeinstellungen und Transaktionshistorie.
  + Anzeige von Benachrichtigungen über wichtige Ereignisse (z.B. Budgetgrenzen)

## **8.3 Kontoeinstellungen**

* **URL:** /settings
* **Funktionalität:**
  + Benutzer können persönliche Daten einsehen und bearbeiten.
  + Option zur Änderung des Passworts mit zusätzlicher Sicherheitsverifizierung.
  + Verwaltung von MFA-Einstellungen (z.B. Hinzufügen/Entfernen von Authentifizierungsmethoden).
  + Einrichtung von Sicherheitsfragen für zusätzliche Verifizierung.

**8.4 Transaktionsseite**

* URL: /transactions
* Funktionalität:
  + Benutzer können Geldüberweisungen durchführen.
  + Implementierung von zusätzlichen Sicherheitsmaßnahmen für hohe Beträge:
    - OTP-Anforderung für Überweisungen über 500€.
    - Biometrische Authentifizierung für große Transaktionen.

## **8.5 Passwortänderung**

* **URL:** /api/settings/change-password (via Formular in /settings)
* **Funktionalität:**
  + Benutzer können ihr Passwort ändern, nachdem sie ihr aktuelles Passwort eingegeben haben.
  + Die Änderung erfordert eine zusätzliche Verifizierung durch einen simulierten Hardware-Token.

## **8.6 Hardware-Token-Verifizierung**

* **URL:** /api/settings/verify-token (via Formular in /settings)
* **Funktionalität:**
  + Überprüfung eines simulierten Hardware-Tokens, um sicherzustellen, dass der Benutzer berechtigt ist, kritische Änderungen vorzunehmen (z. B. Passwortänderung).

## **8.7 Push-Benachrichtigungsseite**

* **URL:** /api/auth/push-notification
* **Funktionalität:**
  + Anzeige einer Bestätigungsanfrage für den Benutzer.
  + Benutzer können die Anmeldung durch Klicken auf einen Button bestätigen.
  + Wird bei niedrigem Risiko-Score (10-20) verwendet.

## **8.8 OTP-Eingabeseite**

* **URL:** /api/auth/verify-otp (via dynamische Anzeige in /login)
* **Funktionalität:**
  + Benutzer geben ein Einmalpasswort (OTP) ein, das per E-Mail oder SMS gesendet wurde.
  + Wird bei mittlerem Risiko (30-100) verwendet.

**8.9 Sicherheitsfragen-Seite**

* **URL:** /verify-security-question
* **Funktionalität:**
  + Anzeige einer vordefinierten Sicherheitsfrage.
  + Eingabefeld für die Antwort des Benutzers.
  + Validierung der eingegebenen Antwort.
  + Wird bei mittlerem Risiko-Score (20-30) verwendet.

## **8.10 Historie**

* **URL:** /history
* **Funktionalität:**
  + Benutzer können ihre Überweisungshistorie einsehen und nachverfolgen.

## **8.11 Registrierung**

* **URL:** /register
* **Funktionalität:**
  + Benutzer können sich für das System registrieren.
  + Eingabe von E-Mail, Passwort und optionalen Sicherheitsfragen für spätere Verifizierungen.

**9 Umsetzung der Szenarien**

Im Rahmen dieses Projekts wurden verschiedene Multi-Faktor-Authentifizierung (MFA) Szenarien erfolgreich implementiert, um die Sicherheit der Finanzverwaltungs-App zu erhöhen. Diese Szenarien decken unterschiedliche Risikostufen ab und bieten maßgeschneiderte Sicherheitsmaßnahmen für verschiedene Benutzeraktivitäten.

## **9.1 Anmeldung von einem neuen Gerät**

* Implementierte MFA-Methode: E-Mail-Verifizierung
* Funktionsweise: Bei der ersten Anmeldung von einem unbekannten Gerät wird ein einmaliger Code (OTP) generiert und an den Benutzer gesendet.
* Sicherheitsaspekt: Verhindert unbefugten Zugriff, selbst wenn das Passwort kompromittiert wurde.

## **9.2 Durchführung hoher Transaktionen**

* Implementierte MFA-Methode: E-Mail-Verifizierung
* Funktionsweise: Bei Überweisungen großer Geldbeträge wird ein einmaliger Code (OTP) generiert und an den Benutzer gesendet.
* Sicherheitsaspekt: Stellt sicher, dass nur der autorisierte Kontoinhaber kritische Finanztransaktionen durchführen kann.

## **9.3 Verdächtige Anmeldeaktivität**

* Implementierte MFA-Methode: Adaptive MFA mit Sicherheitsfragen oder SMS-Verifizierung
* Funktionsweise: Bei Anmeldungen von unbekannten Standorten oder verdächtigen IP-Adressen werden zusätzliche Sicherheitsfragen gestellt oder ein einmaliger Code (OTP) generiert und per SMS (simuliert) an den Benutzer gesendet.
* Sicherheitsaspekt: Bietet eine zusätzliche Verteidigungslinie gegen potenzielle Hackerangriffe.

## **9.4 Regelmäßige Anmeldungen**

* Implementierte MFA-Methode: Push-Benachrichtigung über eine Authentifizierungs-App
* Funktionsweise: Bei regulären Anmeldungen von vertrauten Geräten erhält der Benutzer eine Push-Benachrichtigung zur Bestätigung.
* Sicherheitsaspekt: Bietet eine benutzerfreundliche, aber dennoch sichere Methode für häufige Anmeldungen.

## **9.5 Kontoänderungen**

* Implementierte MFA-Methode: Zwei-Faktor-Authentifizierung mit simuliertem Hardware-Token
* Funktionsweise: Bei kritischen Kontoänderungen muss der Benutzer einen Code von einem simulierten Hardware-Token eingeben.
* Sicherheitsaspekt: Gewährleistet höchste Sicherheit bei sensiblen Änderungen der Kontodaten.

**10 Starten des Projektes**

1. Entpacken des Codes und Starten in einer der bevorzugten Tools wie IntelliJ
2. Starten der Docker Container
   1. DockerFile zu finden im Root-Verzeichnis des Projektes (Name: “docker-compose.dev.yaml”)
3. Nun kann das Projekt gestartet werden (per Startbutton im bevorzugten Tool)
   1. Die Anbindung der Datenbank etc. wird von Spring übernommen
4. Die Applikation ist erreichbar unter “localhost:8080/finance-management”

**11 Erkenntnisse über die Implementierung von MFA in JWT-basiereten Systemen**

Die Implementierung von Multi-Faktor-Authentifizierung (MFA) in JWT-basierten Systemen hat wertvolle Einsichten geliefert:

1. **Erhöhte Sicherheit**: Die Kombination von JWT und MFA bietet eine deutlich verbesserte Sicherheit gegenüber herkömmlichen Authentifizierungsmethoden.
2. **Flexibilität**: JWT ermöglicht eine flexible Integration verschiedener MFA-Methoden, von SMS-OTP bis hin zu biometrischer Authentifizierung.
3. **Benutzerfreundlichkeit vs. Sicherheit**: Es ist wichtig, eine Balance zwischen Sicherheit und Benutzerfreundlichkeit zu finden. Push-Benachrichtigungen für regelmäßige Anmeldungen haben sich als benutzerfreundlich und sicher erwiesen.
4. **Dynamische Anpassung**: Ein dynamisches Zugriffskontrollsystem ermöglicht die Anpassung der MFA-Methoden basierend auf dem Benutzerverhalten und Risikoniveau.
5. **Token-Management**: Die sichere Verwaltung und Validierung von JWTs ist entscheidend für die Gesamtsicherheit des Systems.
6. **Skalierbarkeit**: JWT-basierte Systeme mit MFA lassen sich gut skalieren, da sie zustandslos sind und keine serverseitige Sitzungsspeicherung erfordern.
7. **Implementierungskomplexität**: Die Integration verschiedener MFA-Methoden erhöht die Komplexität des Systems und erfordert sorgfältige Planung und Implementierung.
8. **Datenschutz**: Bei der Implementierung von MFA müssen datenschutzrechtliche Aspekte, insbesondere bei der Verwendung biometrischer Daten, berücksichtigt werden.
9. **Performance-Auswirkungen**: Die zusätzlichen Authentifizierungsschritte können die Systemleistung beeinflussen, was bei der Architekturplanung berücksichtigt werden muss.
10. **Benutzererziehung**: Die Einführung von MFA erfordert eine gute Kommunikation und Schulung der Benutzer, um Akzeptanz und korrekte Nutzung sicherzustellen.

**12 Abschlussbericht (AI generated):**

## **1. Einleitung**

Das Projekt "Sichere Finanzverwaltung" hatte zum Ziel, eine mobile Anwendung zu entwickeln, die durch die Kombination von JSON Web Tokens (JWT), einem dynamischen Zugriffskontrollsystem und Multi-Faktor-Authentifizierung (MFA) ein hohes Maß an Sicherheit für Finanzdaten gewährleistet. Die Entwicklung wurde erfolgreich abgeschlossen und die App bietet nun eine sichere und benutzerfreundliche Plattform zur Verwaltung persönlicher Finanzen.

## **2. Projektziele und Ergebnisse**

## **Hauptziele:**

* Entwicklung einer Finanzverwaltungs-App mit JWT-basierter Authentifizierung
* Implementation eines dynamischen Zugriffskontrollsystems
* Integration verschiedener MFA-Methoden

## **Ergebnisse:**

* Funktionierende App mit sicherer Benutzerauthentifizierung
* Erfolgreich implementiertes dynamisches Zugriffskontrollsystem
* Integration von fünf MFA-Szenarien für verschiedene Risikostufen

## **3. Implementierte Funktionen**

* Benutzerregistrierung und -anmeldung mit JWT
* Dynamisches Zugriffskontrollsystem basierend auf Benutzerverhalten
* Multi-Faktor-Authentifizierung (SMS-OTP, E-Mail-Bestätigung, biometrische Authentifizierung)
* Finanzverwaltungsfunktionen (Ausgabenverfolgung, Budgetierung, Berichterstellung)
* Datenvisualisierung für finanzielle Übersichten
* Benachrichtigungssystem für wichtige finanzielle Ereignisse

## **4. Technische Umsetzung**

* Backend: Kotlin mit Ktor Framework
* Frontend: HTML und CSS
* Datenbank: MySQL
* JWT-Bibliothek: jsonwebtoken
* MFA-Bibliotheken: auth0, Spring Security, Android Biometric API

## **5. Sicherheitsaspekte**

* JWT-basierte Authentifizierung für zustandslose Sitzungsverwaltung
* Dynamisches Zugriffskontrollsystem zur Risikobewertung
* Implementierung verschiedener MFA-Methoden je nach Risikostufe
* Sichere Passwort-Hashing-Verfahren

## **6. Herausforderungen und Lösungen**

* Herausforderung: Balance zwischen Sicherheit und Benutzerfreundlichkeit  
   Lösung: Implementierung eines adaptiven MFA-Systems basierend auf Risikoanalyse
* Herausforderung: Integration verschiedener MFA-Methoden  
   Lösung: Modularer Ansatz zur einfachen Erweiterung um zusätzliche Authentifizierungsmethoden

## **7. Erkenntnisse**

* Die Kombination von JWT und MFA bietet ein hohes Maß an Sicherheit
* Dynamische Anpassung der Sicherheitsmaßnahmen verbessert die Benutzererfahrung
* Implementierung von MFA in JWT-basierten Systemen erfordert sorgfältige Planung der Token-Verwaltung

## **8. Ausblick**

* Mögliche Erweiterungen: Integration eines echten Hardware-Token-Systems
* Verbesserung der Algorithmen zur Risikobewertung
* Erweiterung der Finanzverwaltungsfunktionen

## **9. Fazit**

Das Projekt wurde erfolgreich abgeschlossen und hat die gesetzten Ziele erreicht. Die entwickelte Finanzverwaltungs-App bietet ein hohes Maß an Sicherheit durch die innovative Kombination von JWT, dynamischer Zugriffskontrolle und MFA. Die gewonnenen Erkenntnisse liefern wertvolle Beiträge zur Verbesserung der Sicherheit in mobilen Finanzanwendungen.