Pflichtenheft – PlanPago

1. Einführung

1.1 Projektbeschreibung

PlanPago ist eine moderne, browserbasierte Webanwendung zur digitalen Vertragsverwaltung. Sie richtet sich an Privatpersonen, die ihre wiederkehrenden Zahlungsverpflichtungen wie Miete, Abos oder Versicherungen strukturiert organisieren und verwalten möchten.

Die Applikation ermöglicht:

- die strukturierte Erfassung von Verträgen mittels dynamischer Formulare,
- das Hochladen von Dokumenten,
- die automatische Erinnerung an Fristen und Zahlungen per E-Mail,
- eine grafische Auswertung der finanziellen Situation.

1.2 Zielsetzung

Ziel ist es, eine benutzerfreundliche, sichere und datenschutzkonforme Weblösung zu schaffen, die:

- Fristversäumnisse verhindert,
- die Übersicht über laufende Kosten verbessert,
- und langfristig als persönlicher Vertrags- und Finanzassistent dient.

---

2. Allgemeine Anforderungen

2.1 Zielgruppe

- Berufstätige und Studierende
- Haushalte mit mehreren Fixkosten
- Menschen mit geringem technischen Vorwissen

2.2 Usability

- Plattformunabhängig via Browser (Responsive Design)
- Desktop-First-Ansatz mit intuitivem UI

3. Funktionale Anforderungen

3.1 Vertragsverwaltung

- CRUD-Funktionen für Verträge (Create, Read, Update, Delete)
- Dynamische Eingabeformulare je nach Vertragsart (z. B. Gehaltsvertrag = Brutto/Nettoberechnung)
- Vertragsarten: Miete, Versicherung, Streaming, Gehalt, Leasing, etc.
- Upload-Funktion für Verträge (PDF, JPEG, PNG)
- Vertrags-Metadaten: Betrag, Zahlungsintervall, Fälligkeit, Status, Notizen
- Automatisierte Statusberechnung (aktiv, gekündigt, abgelaufen, kündbar)
- Anzeige von nächsten Fälligkeiten

3.2 Dashboard

- Übersicht aller Verträge mit Filter-/Suchfunktionen
- Farbliche Statusanzeige (z. B. rot = bald fällig)
- Diagramme:
  - Fixkostenanalyse
  - Monatsübersicht
  - Vertragsartenverteilung
- Berechnung des verbleibenden Gehalts nach Fixkosten

3.3 Erinnerungsfunktionen

- Automatisierte E-Mail-Reminder mit anpassbaren Inhalten
- Erinnerung an Kündigungsfristen & Zahlungstermine
- Vorschau der anstehenden Fristen im Dashboard

3.4 Benutzerverwaltung

- Registrierung und Login via E-Mail und Passwort
- Passwort-Hashing (z. B. bcrypt)
- Passwort-Reset via Mail-Link
- Nutzerbezogene Datenspeicherung

4. Nicht-funktionale Anforderungen

4.1 Technikstack

- Frontend: React.js + Tailwind CSS
- Backend: FastAPI (Python)
- Datenbank: SQLite
- Mailversand: SMTP (Gmail, Mailgun etc.)

4.2 Sicherheit & Datenschutz

- HTTPS-Verschlüsselung
- Zugriffsschutz und Authentifizierung
- Verschlüsselte Speicherung sensibler Daten

---

5. Erweiterungen & optionale Features (Nice-to-have)

5.1 Exporte

- Datenexport (CSV, PDF)
- Monatsberichte per Mail

5.2 Mobile App

- Native App (Flutter / React Native)
- Offline-Modus & Synchronisation

5.3 Finanzplanung & Prognose

- Prognose zukünftiger Kosten
- Erkennung & Vorschläge zu kündbaren Verträgen

5.4 Tags & Kategorisierung

- Eigene Tags (z. B. "Haushalt", "Streaming", "Auto")
- Statistiken je Kategorie

5.5 Kalender-Integration

- Export von Fristen in iCal / Google Calendar

- Automatische Synchronisierung

5.6 Zwei-Faktor-Authentifizierung (2FA)

- E-Mail-Code oder TOTP via Authenticator-App

5.7 Mehrbenutzer-Unterstützung

- Gemeinsame Vertragsverwaltung (z. B. Familienkonto)
- Rollen: Admin, Editor, Viewer

5.8 Browser-Erweiterung

- Verträge direkt beim Surfen speichern (z. B. bei Online-Abos)

5.9 Dark Mode

- Umschaltbar über die UI

---

6. Schnittstellen

6.1 Frontend

- Responsives UI
- Validierung direkt im Formular
- Fehlermeldungen und Erfolgsanzeigen

6.2 Backend

- RESTful API (JWT-Authentifizierung)
- Endpunkte: `/contracts`, `/users`, `/reminders`, `/files`
- OpenAPI/Swagger-Dokumentation

---

7. Datenbankmodell

7.1 Tabellen

- Users (ID, E-Mail, Passwort, 2FA-Status)

- Contracts (Name, Typ, Status, Fristen)
- Payments (Betrag, Zahlungsintervall, Zahlungshistorie)
- Reminders (Datum, Status, Nachrichtentext)
- Files (Dateiname, Pfad, UserID)

7.2 Backup & Restore

- Tägliche automatische Backups
- 30 Tage Archiv
- Restore-Funktion

---

8. Qualitätsanforderungen

- Ladezeiten: <1 Sekunde
- Verfügbarkeit: >99 %

---

9. Wartung & Erweiterbarkeit

- Modularer Code mit Clean Architecture
- Dokumentation (Code, API, User Guide)

---

10. Abnahmekriterien

- Alle Kernfunktionen umgesetzt
- Positive Testresultate (≥90 %)
- UX-Test mit min. 5 Personen
- Erfolgreicher E-Mail-Versand
- Deployment auf Testsystem