Entwurfsdokumentation

Inhaltsverzeichnis

Dieses Dokument beschreibt wesentliche Elemente der Softwarearchitektur, sowie andere übergreifende Aspekte des Systems für die Mitgliederdatenbank des StuRa. Hier wird im Folgenden auf die Ziele, Annahmen, die architektonische Bedeutung, unsere Entscheidungen bzw. Einschränkungen und weitere Dinge eingegangen und dokumentiert.

Mit Hilfe von verschiedenen Modellen und Entwürfen für die Architektur, soll die spätere Weiterentwicklung und Anpassung einfacher gemacht werden.

1. Architektonische Ziele und Philosophie

Das System ist eine Webanwendung zur Verwaltung der Kandidaten und Mitglieder des Stura der HTW Dresden durch einen Admin. Offizielle Mitglieder erhalten über einen Login Zugang zur Anwendung und können verschiedene Informationen einsehen, aus diesem Grund muss eine parallele Nutzung von 5 Personen gewährleistet werden (lesend). Des Weiteren ist nicht bekannt auf welchem Endgerät die Nutzung der Anwendung erfolgen wird, weshalb auf die Kompatibilität des Inhaltes mit deren Ansicht auf diversen Bildschirmgrößen geachtet werden muss.

Eine gute Bedienbarkeit wird durch eine übersichtliche und intuitive Benutzeroberfläche erzielt, welche zur Akzeptanz des Gesamtsystems durch die Mitglieder und des Admin beiträgt.

Das System soll auf einem Linux-Webserver eingerichtet werden, der ausreichende Ressourcen zur Verfügung stellt.

Die Anwendung ist eine weiterentwickelte Version des Projektes der Gruppe, von der wir das System übernommen haben. Andere Gruppen, aus zukünftigen Semestern, werden vermutlich ebenfalls an der Optimierung dieser Mitgliederdatenbank arbeiten, sodass das System einfach erweiterbar sein soll.

2. Annahmen und Abhängigkeiten

Annahmen

- Jeder Nutzer hat eine stabile Internetverbindung und nutzt einen aktuellen Browser (Firefox oder Chrome).
- Der Server auf dem die Webseite laufen soll, bietet ausreichende Ressourcen:
  - Arbeitsspeicher: Verbund aus 3 Servern mit jeweils 72 GB RAM
  - Massenspeicher: ausreichend groß
  - Betriebssystem: Linux
- Die bisher verwendeten Datenbankmodelle und Frameworks können weiter verwendet werden.
- Die Mitgliederdatenbank wird in Zukunft funktional erweitert.
- Das Django-Framework gibt bereits eine Aufbau-Architektur vor.

Abhängigkeiten

- Wir sind vom Laborbereich abhängig, der den Server mit seinen Ressourcen stellt.

3. Architektur-relevante Anforderungen

Anforderung	Systemkomponente	Auswirkung auf die Architektur
UC01	Webanwendung	Erweiterung der Anwendung um eine Kandidatenverwaltung
F1	Webanwendung	Zugriffsschutz
F2	Datenbank	Persistenz
R1	Webanwendung Datenbank	Archivierung

4. Entscheidungen, Einschränkungen und Begründungen

1. Wir nutzen **Python als Programmiersprache**, da die bestehenden Teile der Anwendung in dieser Sprache programmiert wurden und wir einen Mehraufwand im Sinne einer Umstrukturierung vermeiden wollten.
   Außerdem ist sie objektorientiert und besitzt eine verhältnismäßig leichte Syntax.
2. Ebenso wie unsere Vorgänger nutzen wir **Django als Framework**. Es ist ebenfalls in Python verfasst und folgt einem Model-View-Template-Schema, aus dem eine leichte Erweiterbarkeit resultiert.
3. **SQLite wird als Datenbanksystem** aufgegriffen, da es standartmäßig von Django unterstützt wird und von der Syntax sehr dem SQL-Standard ähnelt, mit dem einige Gruppenmitglieder schon intensiveren Kontakt hatten.
   Abgesehen davon ist SQLite mit verschiedenen Betriebssystemen kompatibel und unterstützt so das Erreichen von mehr Nutzern, die sich durch ihre verschiedenen Voraussetzungen auszeichnen.
4. Ebenso verwenden wir **Django-Simple-History und simplejson** weiter, da diese bereits von der Vorgängergruppe verwendet wurden und die bestehende Funktionalität diese nutzt.
5. Im Gegensatz zur Vorgängergruppe entscheiden wir uns dagegen, das Framework Selenium im Zusammenhang mit den Tests zu verwenden, da hierbei der Einarbeitungsaufwand unverhältnismäßig groß wäre, da kein Mitglied unseres Teams damit bis jetzt in Kontakt gekommen ist. Die Tests werden stattdessen manuell durchgeführt.
6. Die Erweiterung der Anwendung um eine Kandidaturenverwaltung findet statt über Hinzufügen einer neuen "App", also eines neuen Pakets in der Anwendung. Damit wird die Wartbarkeit vereinfacht, denn verschiedene Funktionalitäten sind von einander getrennt.

5. Architekturmechanismen

Archivierung

- Zweck: Daten dürfen bei Systemausfällen nicht verloren gehen
- Eigenschaften: Daten werden regelmäßig gesichert und sind nach Systemfehlern oder -ausfall wiederherstellbar
- Funktion: Backup der Datenbank wird auf dem Server angelegt, pro Woche wird ein Backup erstellt. Es werden nie mehr als 3 Backups vorhanden sein.

Persistenz

- Zweck: Daten müssen dauerhaft verfügbar bleiben für den Admin und die Mitglieder des Stura
- Eigenschaften: Speicherung der Daten in einer Datenbank
- Funktion: Die Datensätze werden noch während der Ausführung der Anwendung in einer Datenbank gespeichert und sind von dort auch wieder abrufbar.

Zugriffsschutz

- Zweck: Schutz der personenbezogenen Daten vor unberechtigtem Zugriff
- Eigenschaften: Nicht jeder Nutzer ist dazu berechtigt, alle Daten einzusehen. Bestimmte Nutzer haben bestimmte Rechte, um verschiedene Informationen einsehen zu können.
- Funktion: Anmeldung der Nutzer mit Logindaten, anhand derer ermittelt werden kann, welche Rechte der Nutzer hat. (Unterscheidung nach Mitglied Stura und Admin)

5.1. Datenspeicherung

Die Speicherung der Daten erfolg über die SQLite-Datenbank. Jeder Datensatz ist identifizierbar über einen Primärschlüssel in Form einer ID.

5.2. Webschnittstelle

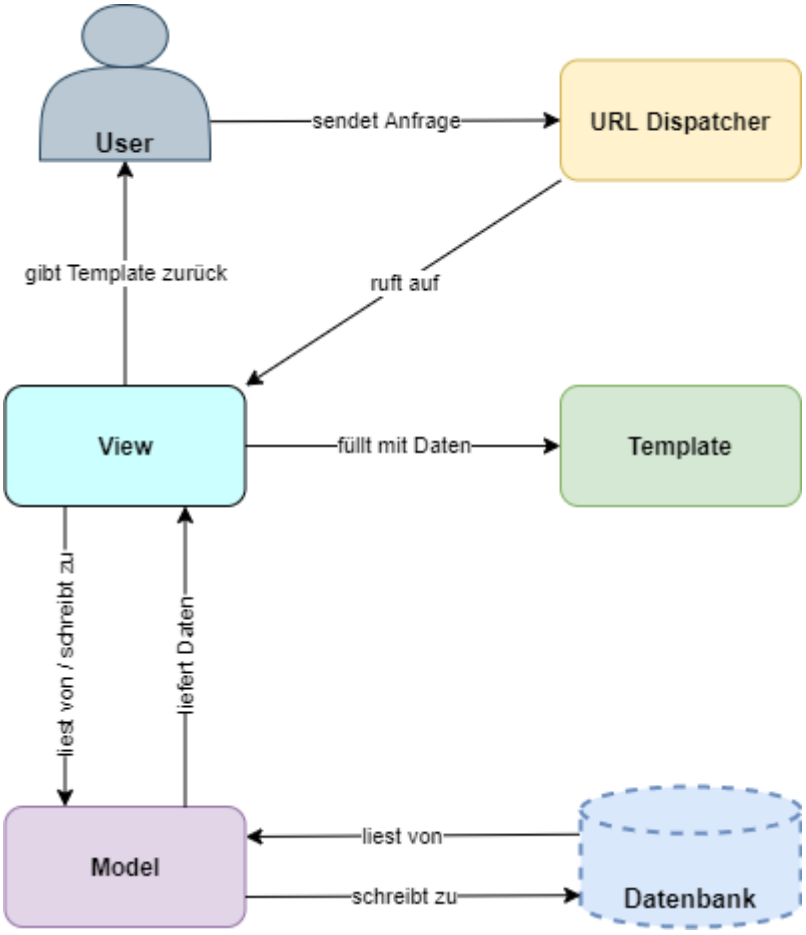
Die von Django bereitgestellte Schnittstelle wird verwendet um Daten über die Webseite zu organisieren und zu verwalten. Die Weboberfläche wurde mittels HTML/CSS erstellt.

6. Systemarchitektur

6.1. Architekturmuster

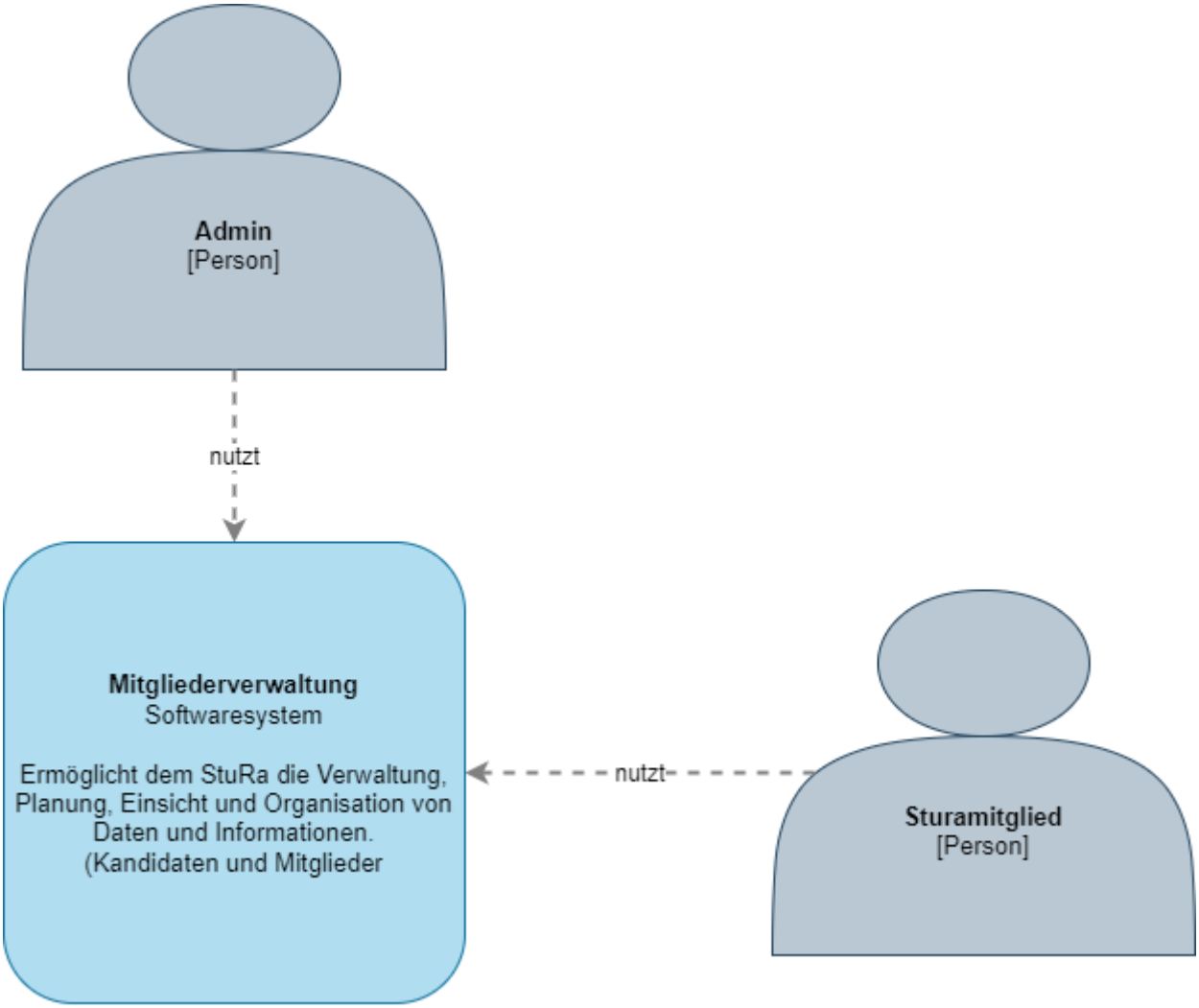
Model-View-Template (MVP)

- wird von Django Framework bereitgestellt
- **Model:** ist für Zugriff auf die Datenbank zuständig, liest und schreibt Daten
- **View** (Ansicht)**:** für Verarbeitung der Daten verantwortlich
- **Template:** Leere HTML-Seite, die durch View mit Daten gefüllt wird

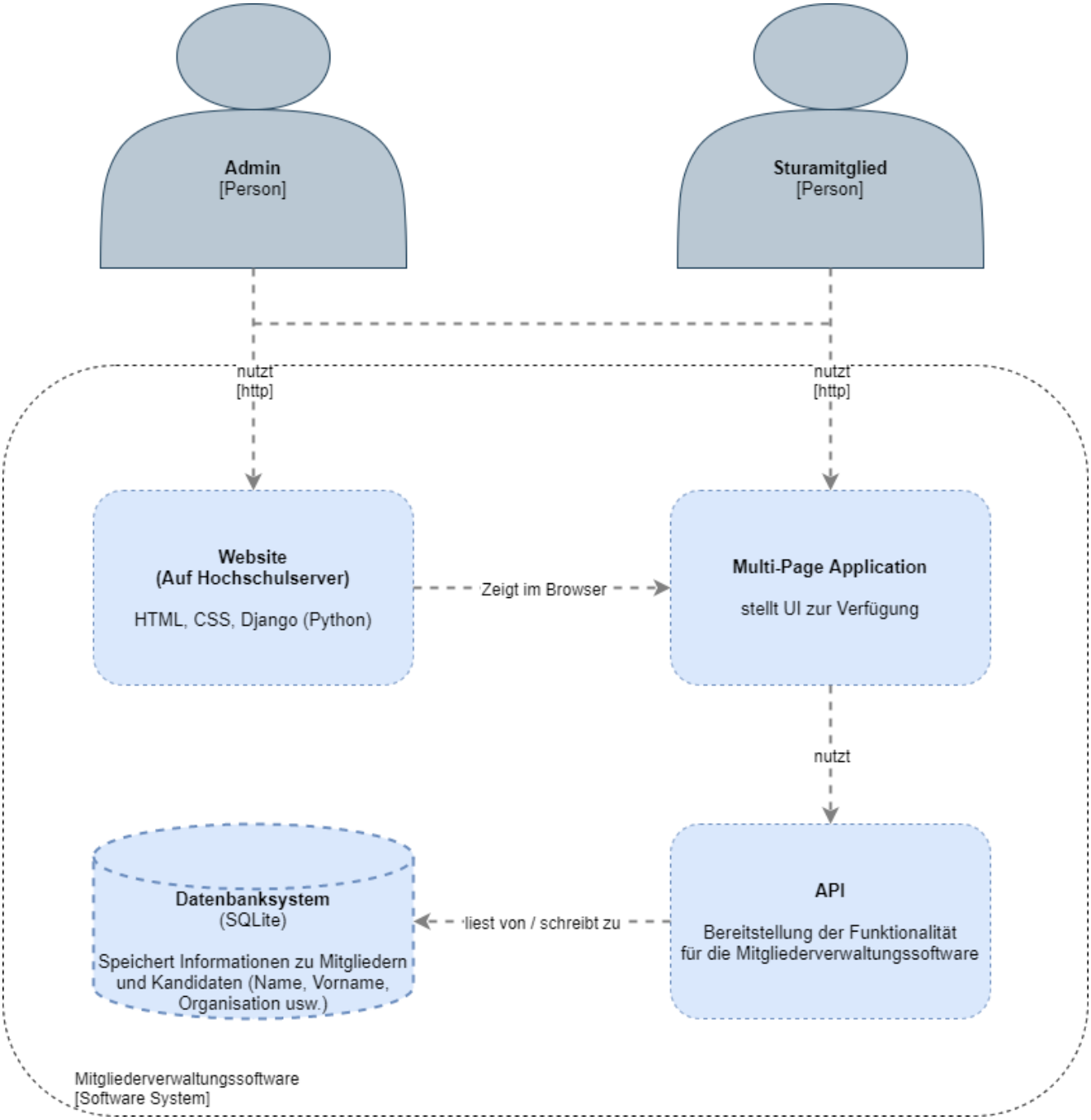


6.2. Logische Sicht

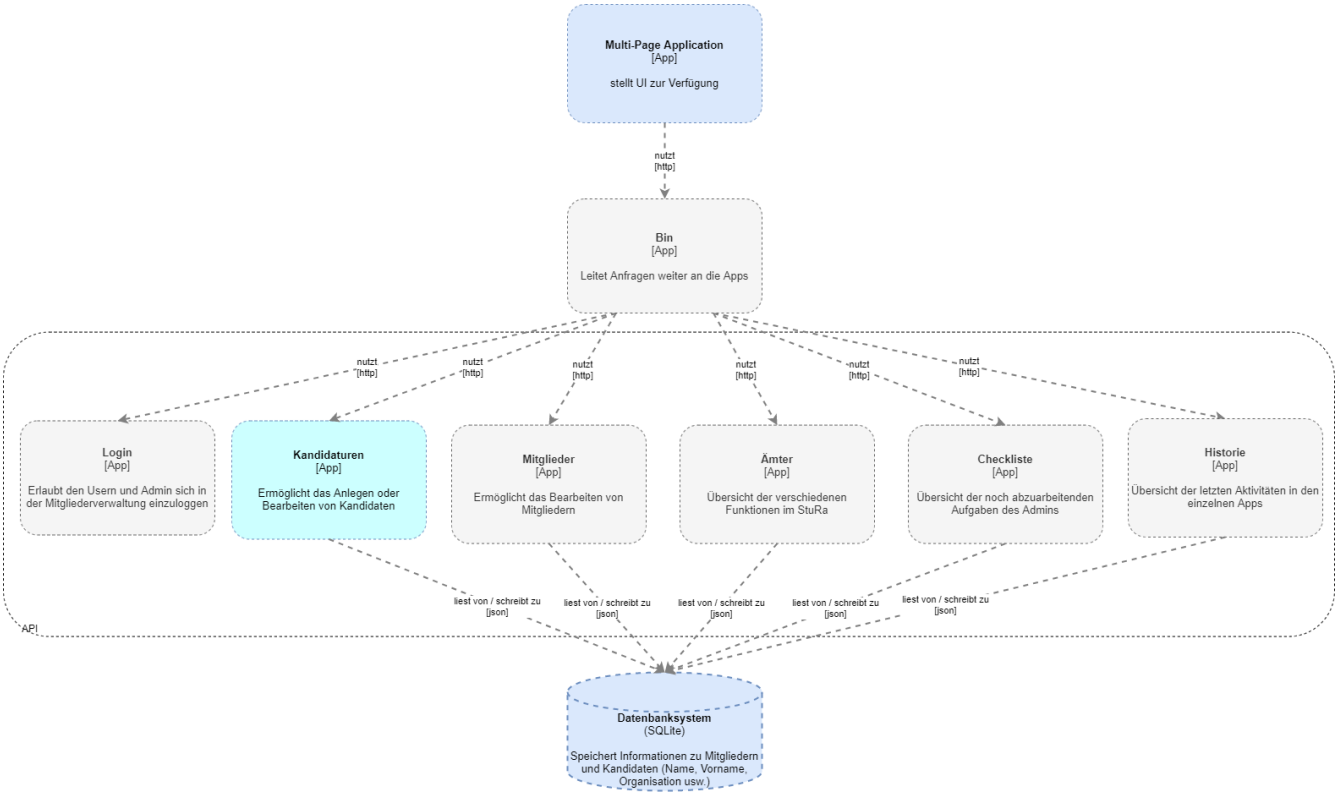
6.2.1. C4 Modelle


Admin
[Person]
nutzt
Mitgliederverwaltung
Softwaresystem
Ermöglicht dem StuRa die Verwaltung, Planung, Einsicht und Organisation von Daten und Informationen.
(Kandidaten und Mitglieder
nutzt
Sturamitglied
[Person]

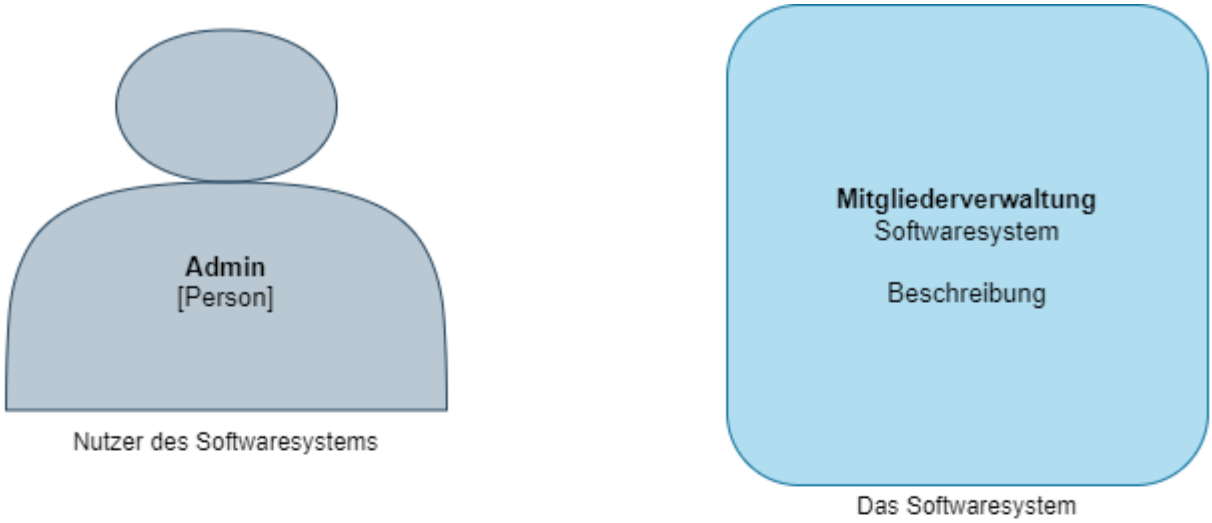
Level 1


Admin
[Person]
Sturamitglied
[Person]
nutzt
[http]
nutzt
[http]
Website
(Auf Hochschulserver)
HTML, CSS, Django (Python)
Zeigt im Browser
Multi-Page Application
stellt UI zur Verfügung
nutzt
Datenbanksystem
(SQLite)
Speichert Informationen zu Mitgliedern und Kandidaten (Name, Vorname, Organisation usw.)
liest von / schreibt zu
API
Bereitstellung der Funktionalität für die Mitgliederverwaltungssoftware
Mitgliederverwaltungssoftware
[Software System]

Level 2

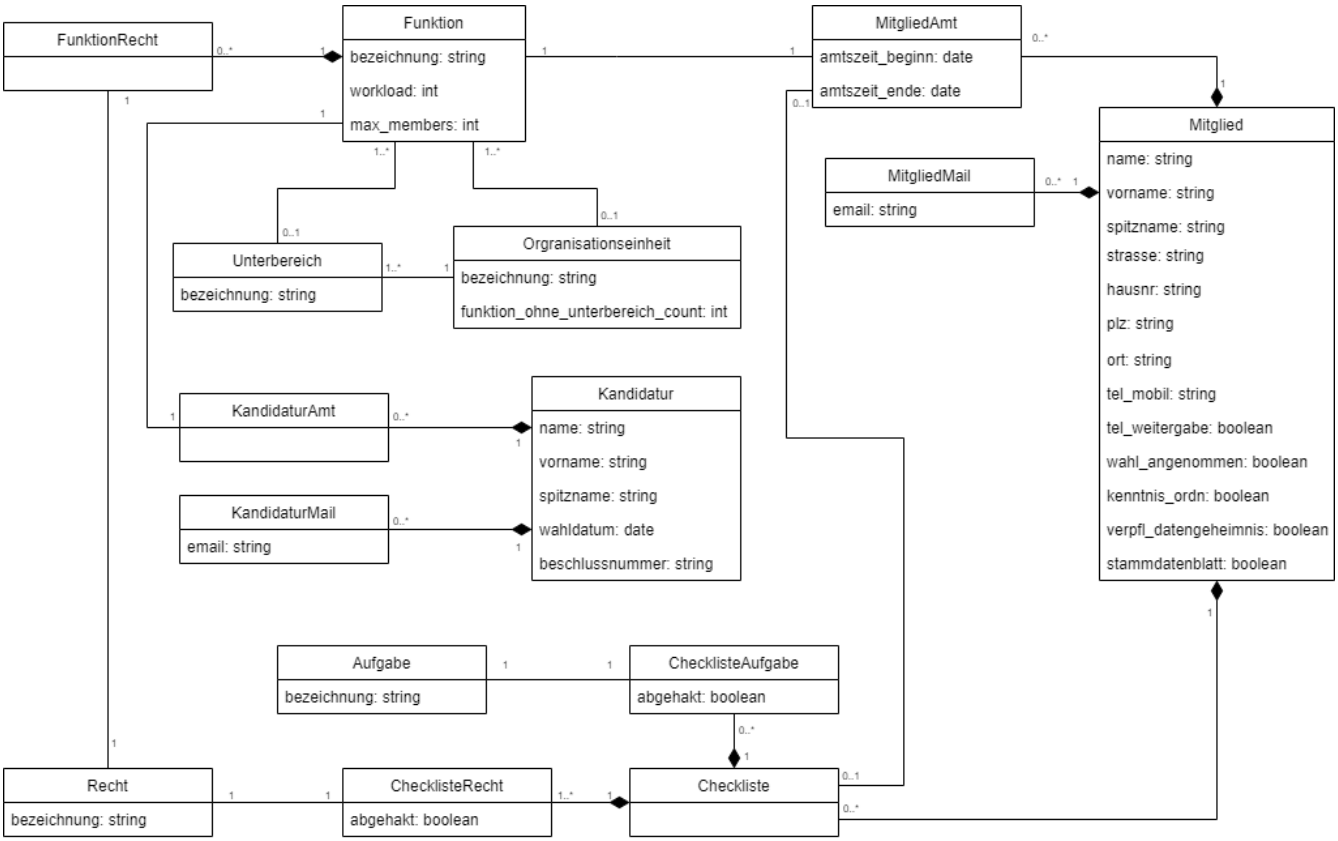

Multi-Page Application
[App]
stellt UI zur Verfügung
nutzt
[http]
Bin
[App]
Leitet Anfragen weiter an die Apps
nutzt
[http]
Login
[App]
Erlaubt den Usern und Admin sich in der Mitgliederverwaltung einzuloggen
Kandidaturen
[App]
Ermöglicht das Anlegen oder Bearbeiten von Kandidaten
Mitglieder
[App]
Ermöglicht das Bearbeiten von Mitgliedern
Ämter
[App]
Übersicht der verschiedenen Funktionen im StuRa
Checkliste
[App]
Übersicht der noch abzuarbeitenden Aufgaben des Admins
Historie
[App]
Übersicht der letzten Aktivitäten in den einzelnen Apps
liest von / schreibt zu
[json]
API
Datenbanksystem
(SQLite)
Speichert Informationen zu Mitgliedern und Kandidaten (Name, Vorname, Organisation usw.)

Level 3


Admin
[Person]
Nutzer des Softwaresystems
Mitgliederverwaltung
Softwaresystem
Beschreibung
Das Softwaresystem
Website
(Auf Hochschulserver)
HTML, CSS, Django (Python)
Komponente
Datenbanksystem
(SQLite)
Speichert Informationen zu Mitgliedern und Kandidaten (Name, Vorname, Organisation usw.)
Datenbank
Mitgliederverwaltungssoftware
[Software System]
Zoom in Einzelteile
Kandidaturen
[App]
Beschreibung
neu implementierte Django App
Historie
[App]
Beschreibung
schon vorhandene Django App
nutzt
Beziehung

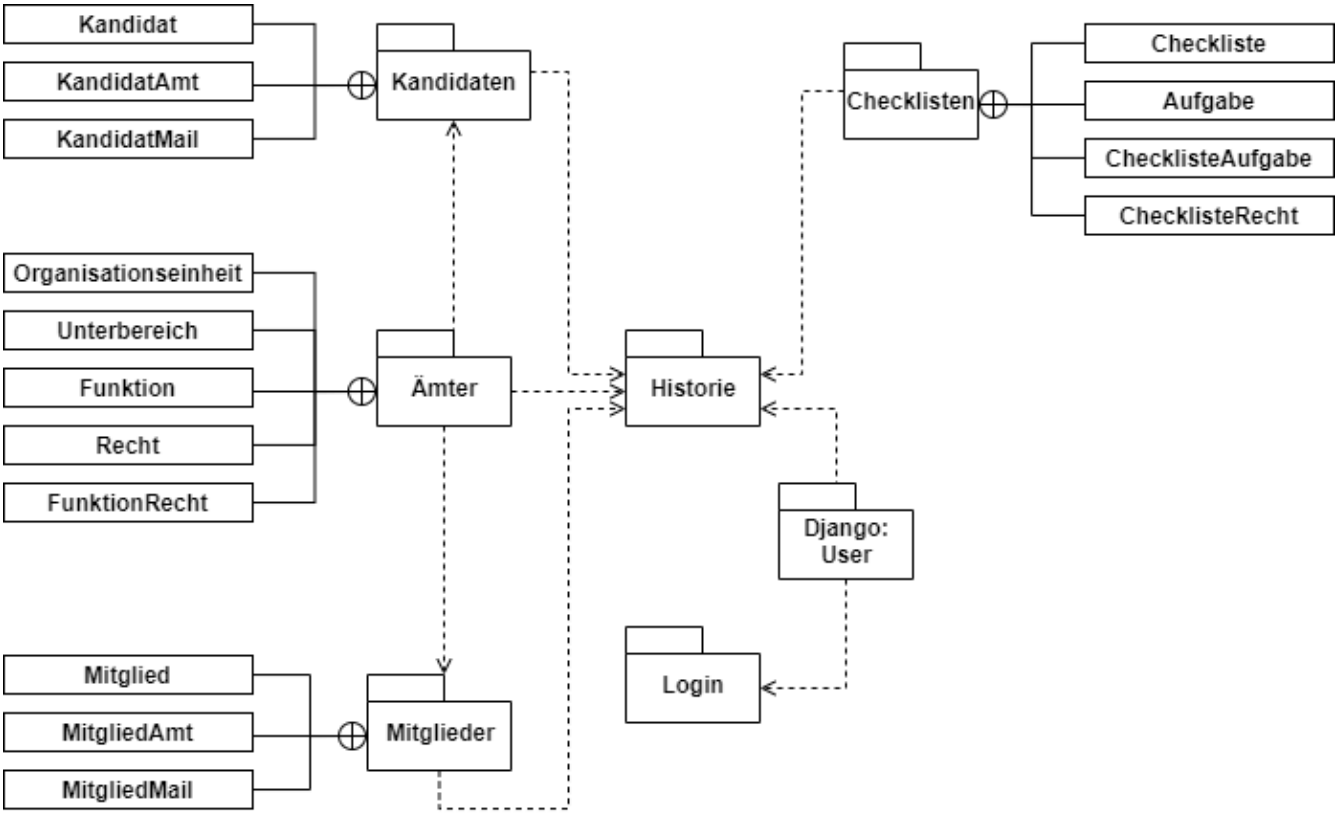
Legende

6.2.2. Klassendiagramm

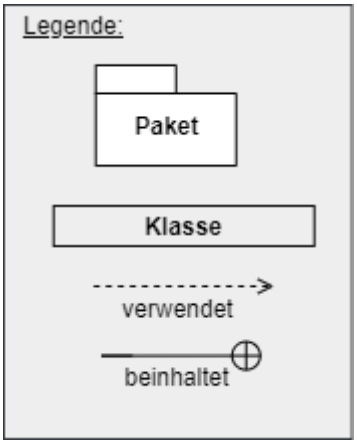


Klassendiagramm

6.2.3. Paketdiagramm

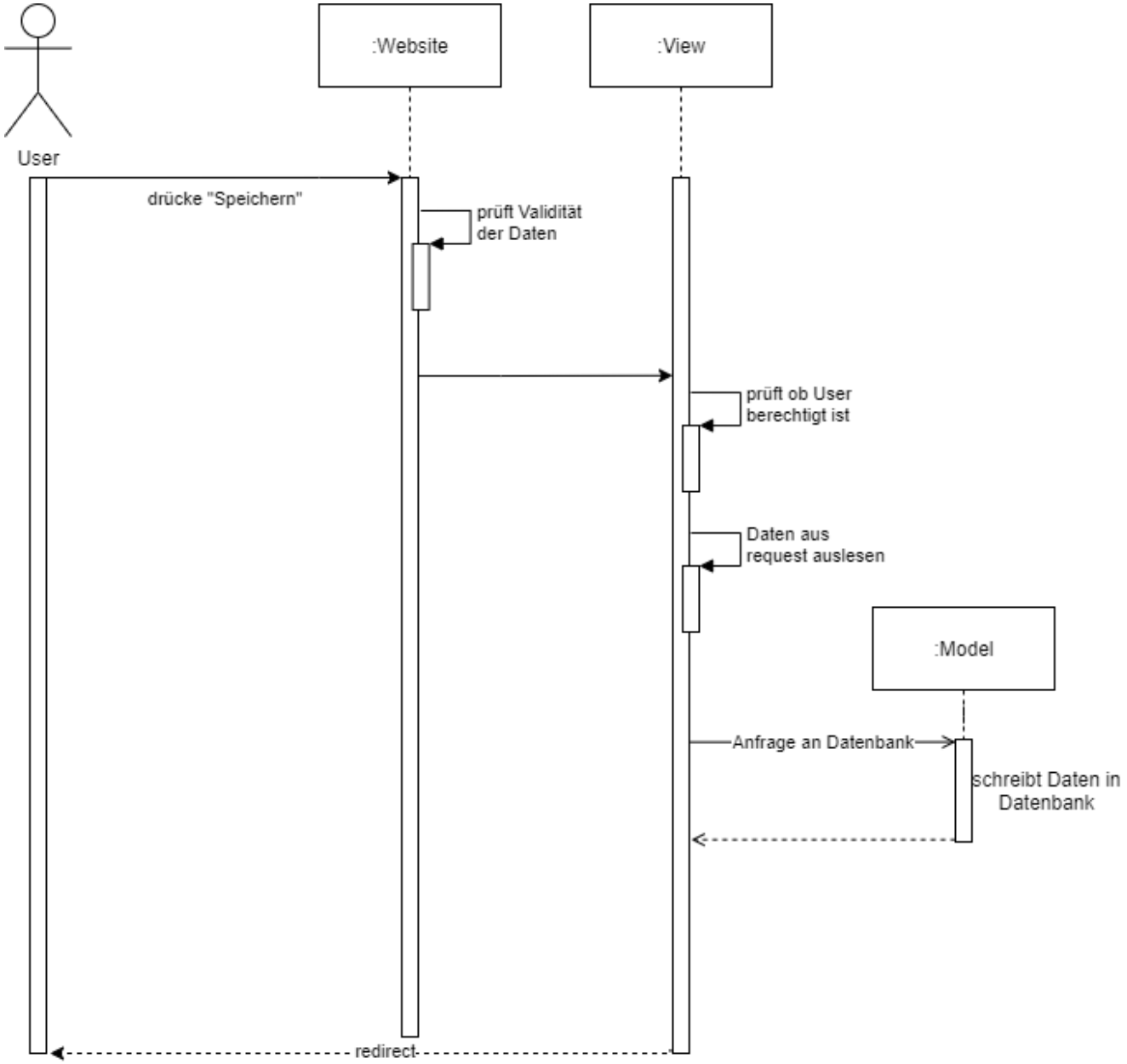


Paketdiagramm mit Klassen

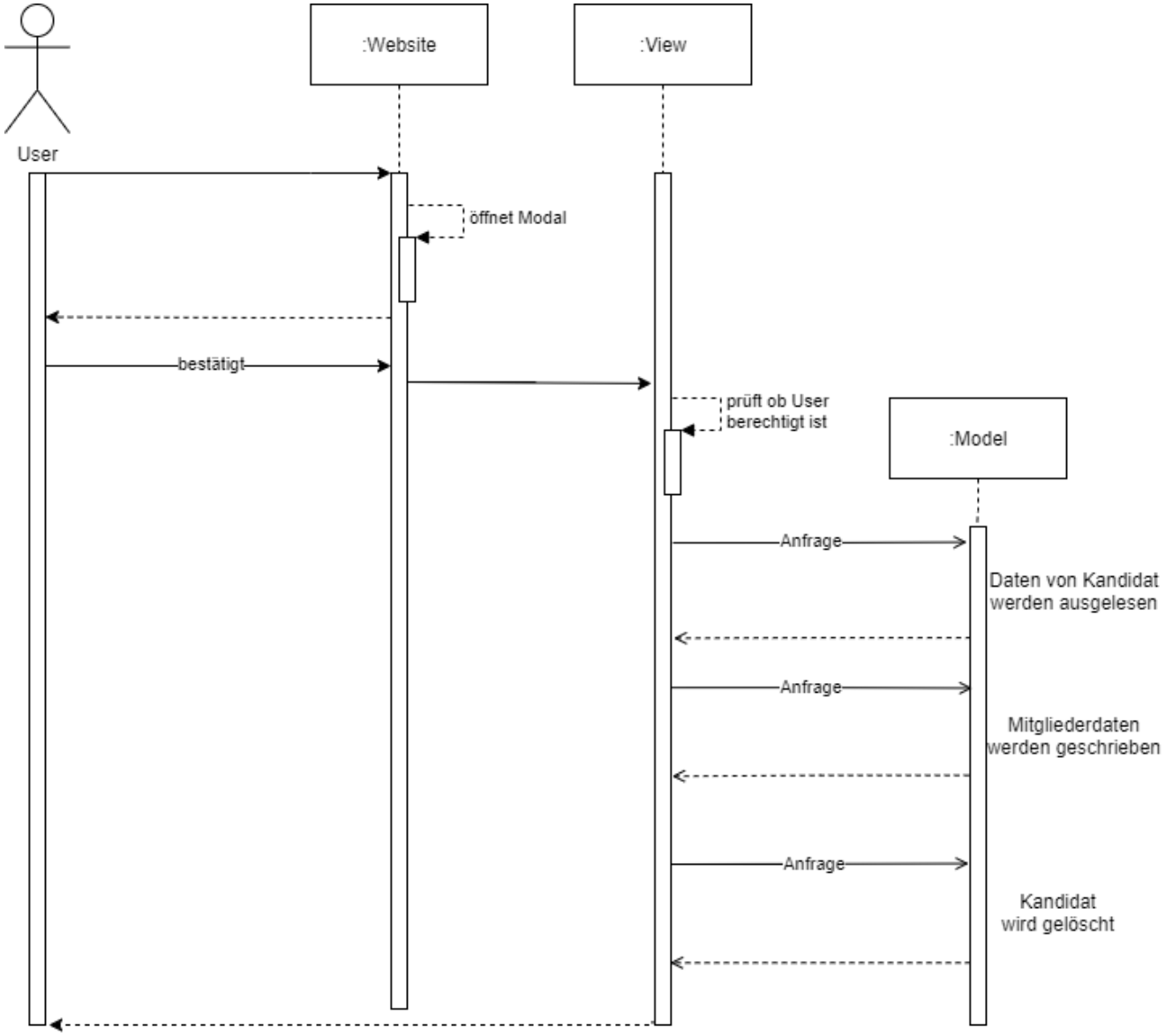


6.3. Ablaufsicht

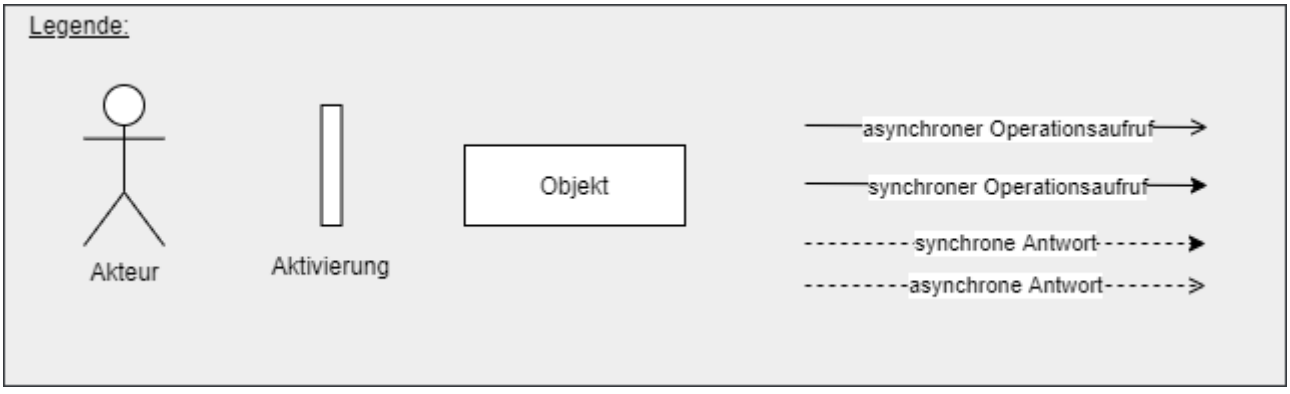
6.3.1. Sequenzdiagramme



UC01: Kandidat hinzufügen

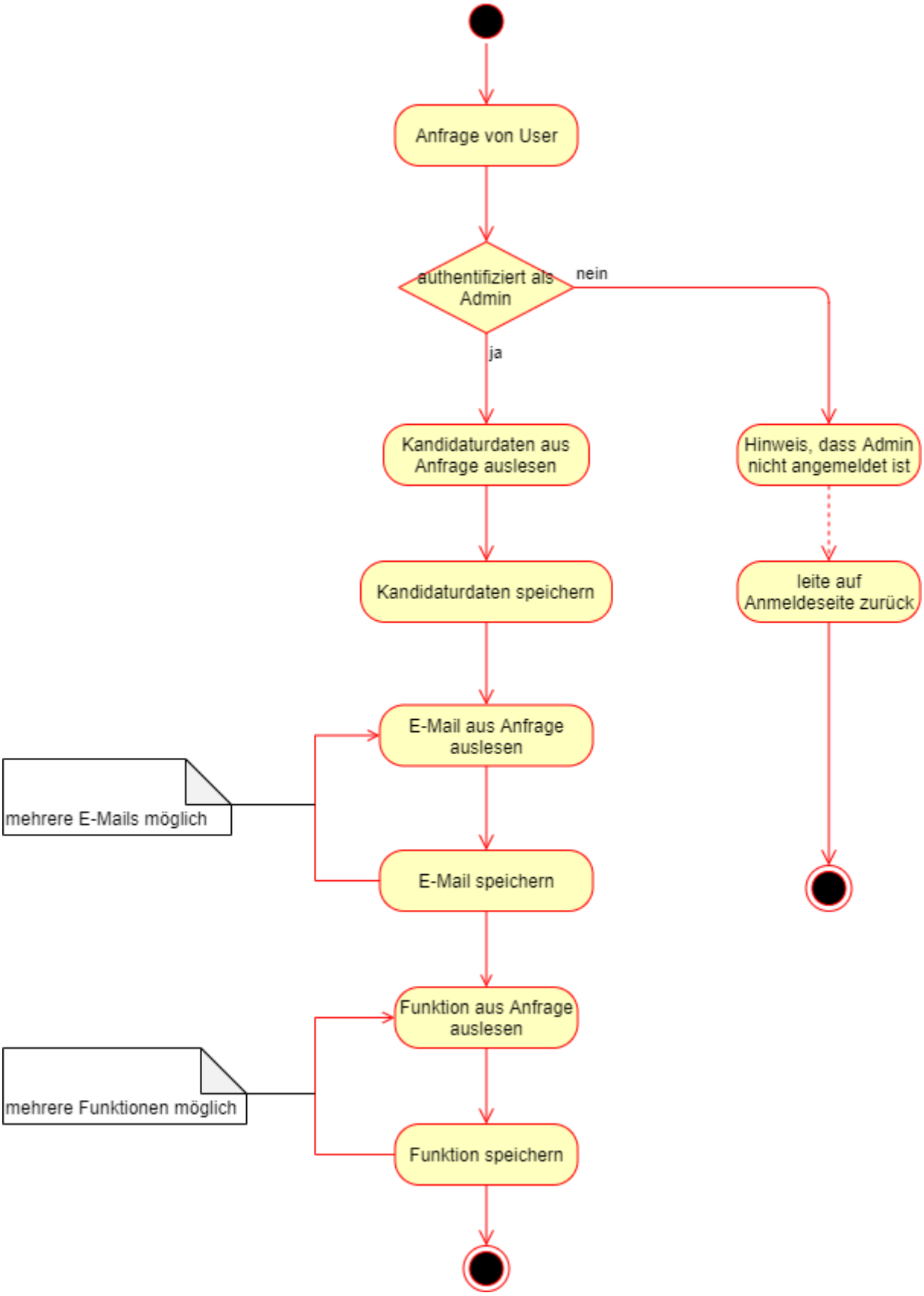


UC02: Kandidat aufnehmen

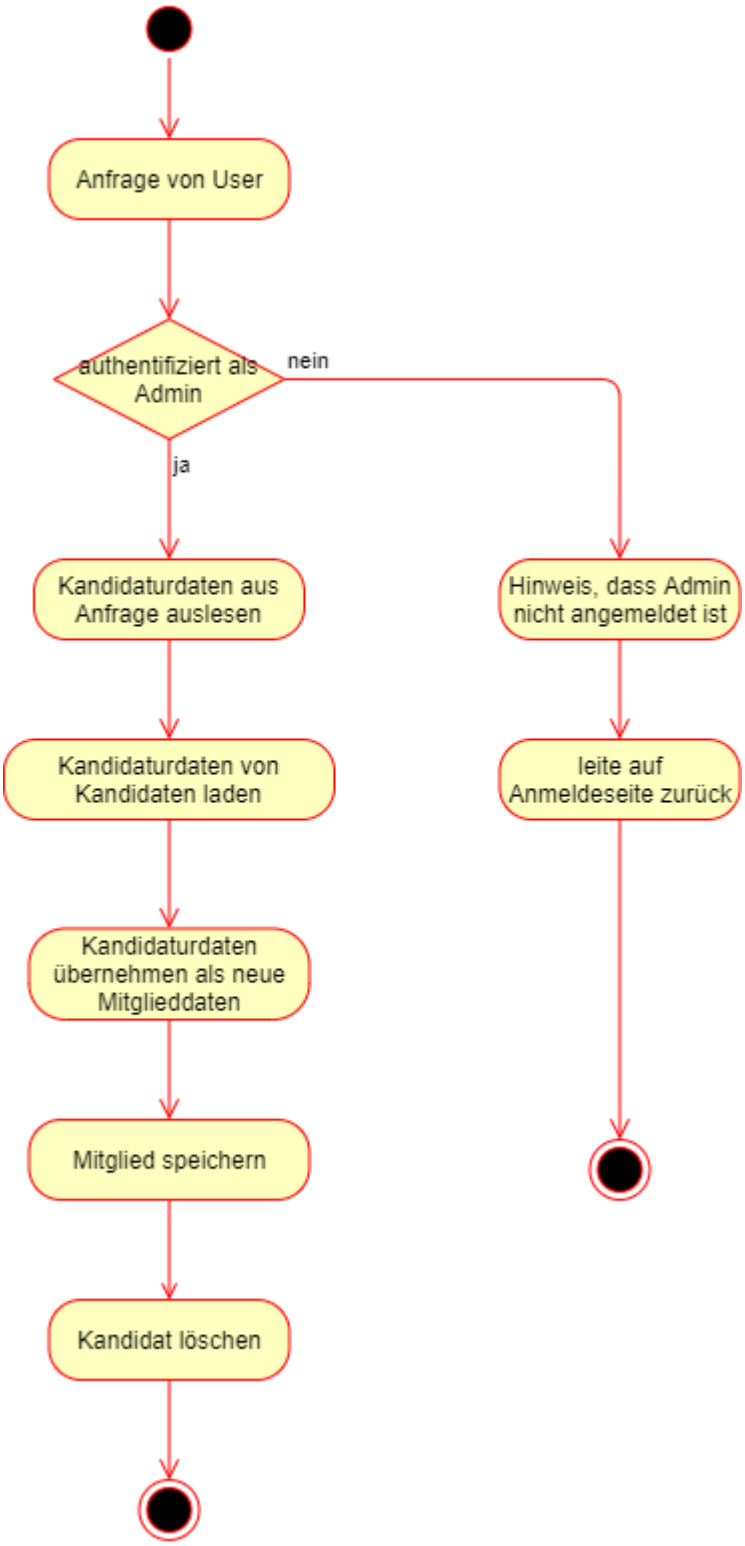


6.4. Szenarien

6.4.1. Aktivitätsdiagramme



UC01: Kandidat hinzufügen



UC02: Kandidat aufnehmen