Entwurf

Max Limmer, Lorenz Heinrich, Christoph Niederbudder, Paul Enciu, Noah Biwer

January 2023

Inhaltsverzeichnis

1 Einleitung

Dieses Dokument steht im Kontext des PSE-Projekts *e-lection* und soll einen Überblick über die Konzeption und den Entwurf der Web-Anwendung bieten. Es knüpft dabei unmittelbar an das vorangegangene Pflichtenheft an, in welchem die konkreten Ziele des Projekts spezifiziert wurden. Zum besseren Verständnis dieses Dokuments sollen noch einmal die wichtigsten Ziele des Projekts genannt werden.

1.1 *e-lection* - die Ziele

Mit dem System *e-lection* sollen Ende-zu-Ende verifizierbare Wahlen online durchgeführt werden können. Dabei soll darauf geachtet werden, dass die Teilnahme als Wähler an einer solchen Wahl ohne technischen Kenntnisse möglich ist. Um das Wahlgeheimnis zu schützen, wird für das Verschlüsseln der Ballots (Stimmzettel) wird *threshold homomorphic ElGamal* verwendet. Diese Verschlüsselungsmethode erlaubt das zusammensetzen eines öffentlichen Schlüssels zu welchem die privaten Schlüssel auf mehrere Personen verteilt sein können. Dadurch ist es nur durch Kollaboration aller beteiligter Schlüssel-Halter möglich, eine Nachricht zu entschlüsseln. Die Verschlüsselung der individuellen Ballots wird auf dem jeweiligen Endgerät des Wählers berechnet und anschließend an den Server zur Speicherung übermittelt. Da diese Endgeräte korrumpiert sein können, wird dem Wähler ermöglicht nach der Verschlüsseln, statt der Abgabe, die Informationen zur Verschlüsselung herunterzuladen um diese, bestenfalls mit einem anderen Gerät, mittels des *e-lection* Systems, auf Korrektheit zu prüfen. Das bedeutet, es kann nach der Verschlüsselung geprüft werden, ob die Verschlüsselung tatsächlich den vom Wähler ausgefüllten Stimmzettel als Nachricht enthält. Darüber hinaus soll nach der Vollendung einer Wahl allen Beteiligten einer Wahl ermöglicht werden diese mittels des von Microsoft entwickelten *ElectionGuard* Systems auf Korrektheit zu prüfen.

1.2 Struktur des Dokuments

Das Dokument leitet mit einer Übersicht über den Entwurf des Systems ein. Eingegangen wird an dieser Stelle vielmehr auf die Aufteilung des Systems in Schichten und deren Aufgaben als auf deren interne Struktur. Es soll vermittelt werden wie die einzelnen Schichten miteinander zusammenarbeiten um als ganzes die Funktionen des Projekts zu realisieren. Anschließend lässt sich das Dokument in zwei Teilbereiche untergliedern, Backend udn Frontend, welche jeweils ihren eigenen Entwurf beschreiben. Jeder Teilbereich enthält seine eigenen Komponenten-,Sequenz- und Klassendiagramme, welche beim Verständnis des Zusammenspiels der einzelnen Komponenten und Klassen Unterstützen sollen.

Code Dokumentation

Die öffentlichen Klassen und deren Methodensignaturen sowie die Attribute sind mit Standardisierten Kommentaren versehen. Da das Frontend sowie das Backend unabhängig voneinander zu implementieren sind, entstehen dadurch zwei verschiedene *Code-Bases*, mit ihrer eigenen Dokumentation. Diese Dokumentation ist in dem jeweiligen Abschnitt unter *Dokumentation* verlinkt. Ebenso werden die Dokumentationen zusammen mit diesem Dokument in dem Abgabe Commit des Git-Lab Repositories zu finden sein.

Vollständiges Klassendiagramm

Die in den jeweiligen Abschnitten vorgestellten Pakete mit ihren Klassendiagrammen sind auch als vollständiges Klassendiagramm zu finden. Dieses ist nicht in diesem Dokument enthalten und dient ausschließlich der Präsentation, da es nur auf großen Oberflächen wie einem Din A0 Plakat oder auf eine großen Leinwand *gebeamt* gut lesbar ist. Das vollständige Klassendiagramm ist ebenso im Abgabe Commit des Git-Lab Repositories zu finden.

1.3 UML - Ein paar Hinweise

Um die Verständlichkeit des Entwurfs zu verbessern, sind Komponenten, Klassen und Interfaces abhängig davon welchem Modul sie angehören. Hierfür nun eine Legende:

Modul	**Farbe**
Model	Blau
View	Lila
Controller	Rot
Services	Orange
Spring-Framework	Grün
Externe Bibliotheken	Grau

Auch bei den Struktur-Objekten innerhalb eines Moduls gibt es farbliche Hervorhebung.

Struktur-Objekt	**Farbe**
Klasse	Farbe des Modules
Interface	Farbe des zugehörigen Modules in heller
Enum	Gelb

Teile des Klassendiagramms werden an einigen Stellen gezeigt um erklärte Entwurfsentscheidungen zu illustrieren. Dabei wird oft auf die Nennung mancher Attribute oder Methoden verzichtet, die an dieser Stelle nichts wesentliches beitragen um die Lesbarkeit und Übersichtlichkeit zu verbessern. Für eine akkurate Darstellung der Klassen ist das vollständige Klassendiagramm vorgesehen. Gibt es weitere Besonderheiten in den Klassendiagrammen wird an den jeweiligen Stellen darauf hingewiesen.

2 Übersicht und Grobentwurf

Das System verwendet eine Vier-Schichten-Architektur welche je nach Betrachtung auch als Klient-Server-Architektur verstanden werden kann.

Hierbei bilden die drei Schichten *Web-API*, *e-lection* und *Datenbank* den Server, da diese auf dem selben System operieren und miteinander zusammenarbeiten. Das *User-Layer* bildet die Schicht, welche auf den jeweiligen Endgeräten der Benutzer ausgeführt wird, die Klienten des Servers. Abbildung 1 zeigt die Unterteilung des Systems in Schichten.

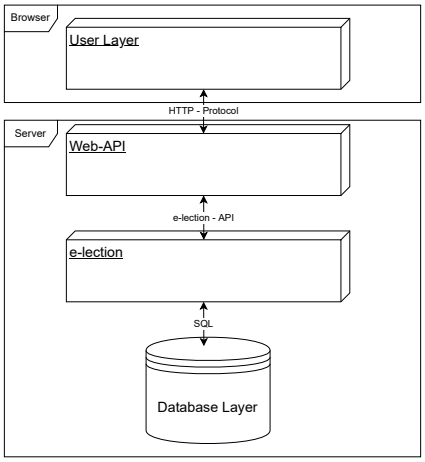

Abbildung 1: Schichtenarchitektur des *e-lection*- Systems

Die Aufteilung in vier Schichten ist deshalb von zentraler Bedeutung, da dadurch die einzelnen Aufgaben des Systems auf lose gekoppelte Schichten aufgeteilt werden kann.

Das *User-Layer* ist verantwortlich für die direkte Interaktion mit den Benutzern des Systems. Dabei müssen Daten angezeigt und eingelesen werden können.

Die *Web-API* Schicht bietet eine *HTTP*-Fassade über welche das *User-Layer* mit dem Server über das Internet kommunizieren kann. Diese kann ausgetauscht werden, wenn auf die *e-lection* Schicht über ein anderes Netzwerk zugegriffen werden soll. Diese Schicht beinhaltet auch die Authentifizierungskomponente.

Die *e-lection* Schicht implementiert die Anwendungslogik des Systems. In dieser wird beispielsweise geprüft, ob ein Stimmzettel zu einer bestimmten Wahl abgegeben werden darf, wie auch die *Key-Ceremony* und *Decryption* einer Wahl überwacht. Die Schicht bietet eine klar definierte Fassade über welche alle unterstützen Operationen an die Schicht angefragt werden können.

Das *Database-Layer* ist ein externes relationales *DBMS* welches beliebig ausgetauscht werden kann. Mit diesem wird über den *SQL Standard ISO 9075* kommuniziert.

3 Frontend

Die Frontend Schicht realisiert die Schnittstelle, mit welcher der Nutzer unmittelbar interagiert und wird als unabhängiges System auf dem Endgerät des Nutzers ausgeführt. Dabei ist vorgesehen dass diese Schicht in einem Browser funktional äquivalent zu Chromium 80.0.3987 ausgeführt wird. Diese Schicht muss hierbei mehrere Aufgaben übernehmen, wie das Anzeigen von Information, die Entgegennahme von Daten oder das Übermitteln von Daten an das Backend. Der Grobentwurf teilt diese Aufgaben einzelnen Modulen zu.

3.1 Grobentwurf

Die Aufteilung und das Zusammenspiel dieser Module folgt dem weiter verbreiten Model-View-Controller MVC Entwurfsmuster.

Die View ist dafür verantwortlich, vorliegende Daten im Browser anzuzeigen. Für das temporäre Speichern der anzuzeigenden Daten nutzt die View Klassen aus dem Model. Auf diesen Objekten kann die View unmittelbar Änderungen durchführen, welche durch Nutzereingaben angestoßen wurden. Müssen komplexe Berechnungen durchgeführt werden oder Daten mit dem Backend ausgetauscht werden, ruft die View dafür den Controller auf. Dieser übernimmt die asynchrone Kommunikation mit dem Backend und erhält die Kryptographische Fassade zum Verschlüsseln und Verifizieren von Ballots. Der Controller kann neue Instanzen von Model Klassen erzeugen und diese der View bereitstellen. Abbildung 2 zeigt die Kommunikation der drei Module.

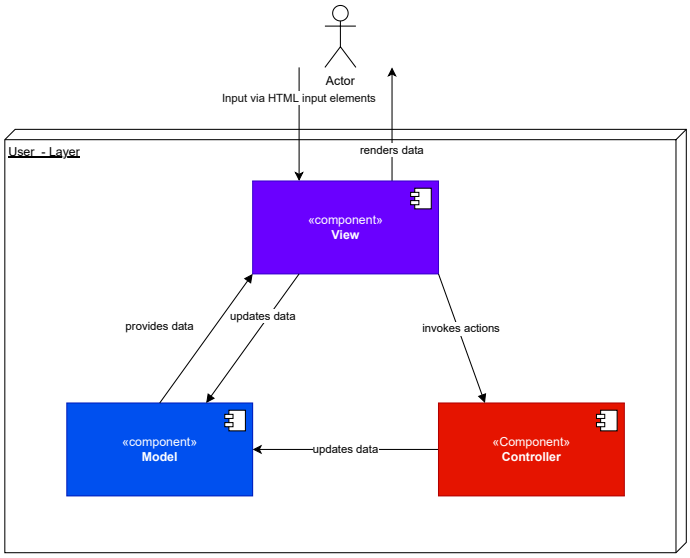

Abbildung 2: Zusammenspiel der Model-View-Controller Komponenten.

3.2 Model

Das Model enthält alle nötigen Klassen um Daten zu speichern, welche durch das Backend oder durch Nutzereingaben bereit gestellt werden. Auf oberster Ebene des Moduls werden zwei Interfaces definiert: `Mutable` und `Serializable`.

Anmerkung zu den UML-Klassendiagrammen: Alle Attribute der Model Klassen sind grundsätzlich als *public* markiert. Jedoch erfolgt der Zugriff über `getter` Methoden, die hier zu Gunsten der Übersicht verzichtet wird. Sofern kein `setter` angegeben, sind die Attribute *read-only*.

Mutable

Das Interface `Mutable` kann von Klassen implementiert werden, die eine bestimmte Model Klasse erzeugen. Im Prozess der Wahlerstellung müssen einige Klassen angelegt werden, welche allerdings an vielen anderen Stellen im Frontend als reine Datenhaltungsklassen gebraucht werden. Um diese Klassen von *immutable* zu halten, gibt es zu jeder dieser Klassen ein `Mutable`-Gegenstück. Das hier verwendete Entwurfsmuster ist eng verwandt mit dem bekannten Entwurfsmuster *Erbauer*. Die Methode `create(): T` kann einen `ConfigError` werfen, falls die gesetzten Attribute den Bedingungen der entsprechenden Model Klasse nicht genügen.

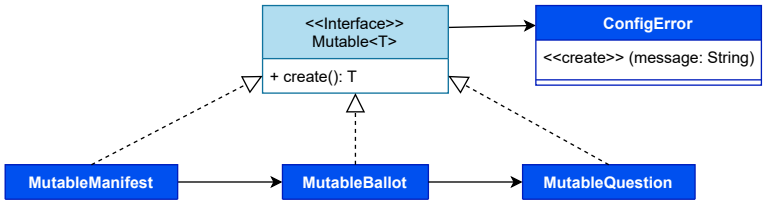

Abbildung 3: Das Interface `Mutable` und die implementierenden Klassen

Serializable

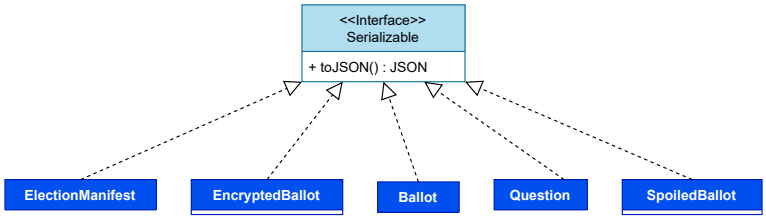

Abbildung 4: Das Interface `Serializable` und die implementierenden Klassen

Das Interface `Serializable` definiert die Methode `toJSON(): JSON` welche die implementierende Klasse in ein generisches Objekt verwandelt, welches anschließend in einen Byte-Strom umgewandelt werden kann. Implementiert wird dieses Interface von allen Klassen, die Datenmengen repräsentieren, die an das Backend übermittelt werden müssen.

3.2.1 Ballot

Das Paket *Model.Ballot* definiert alle Klassen, die Daten halten, die sich auf Ballots beziehen. Ein `Ballot` besteht aus einer `Map` von Frageindex auf eine `Question` welche alle Daten zu dieser Frage hält. Ein `Ballot` kann ein `PlaintextBallot` erzeugen, welches dann von einem Wähler ausgefüllt werden kann. Das `EncryptedBallot` wird verwendet um das Ergebnis der Verschlüsselung eines Ballots durch den Encryptor zu repräsentieren. Das `SpoiledBallot` wird als Datenobjekt für das herunter- und hochladen für das Verifizieren der Verschlüsselung verwendet. Siehe Abbildung 5.

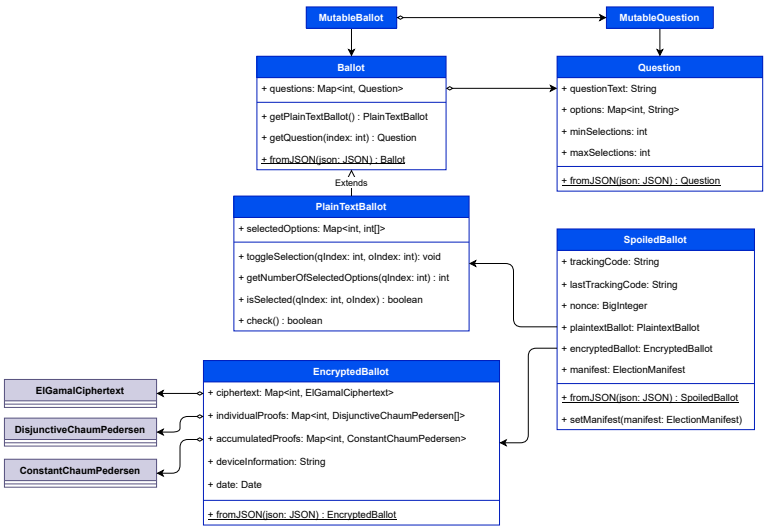

Abbildung 5: Klassendiagramm Paket Model.Ballot

3.2.2 Election

Das Paket *Model.Election* definiert die Klassen welche alle Daten zu einer Wahl halten können, die angezeigt werden oder nach Benutzereingaben an das Backend übermittelt werden sollen.

Die Art der Daten, die momentan zum Anzeigen einer Wahl nötig sind, ist abhängig von der Rolle des Nutzers und dem Zustand der Wahl. Um zu vermeiden, häufig Objekte zu instanziieren, in welchen ein überwiegender Teil der Attribute nicht initialisiert sind, werden stattdessen die Attribute, welche in jedem Fall initialisiert sind in einer der `Election`

Klasse zusammengefasst. Von dieser Klasse erben weitere Klassen welche spezifische Daten für einen bestimmten Zustand einer Wahl für eine bestimmte Rolle eines Nutzers halten kann. Dadurch kann genau dieses Objekt erzeugt werden, was zum Zustand der Wahl und Rolle des Nutzers passt ohne dabei Redundanz in Kauf zu nehmen.

Für die Daten der Konfiguration einer Wahl ergeben sich ähnliche Probleme. Zwar wäre es denkbar alle Konfigurationsdaten, also das ganze `ElectionManifest` einer Wahl, jedes mal vollständig vom Backend anzufordern, jedoch ist das in den meisten Fällen nicht nötig und würde zu unnötigem Datenverkehr führen. Eine mögliche Lösung ist auch hier Vererbung einzusetzen. Die Klasse `ElectionMeta` definiert ein leichtgewichtiges Manifest zu einer Wahl, welches von der Klasse `ElectionManifest` zu einem vollständigen Manifest erweitert werden kann. Für die Verbindung zwischen der `Election` Klasse und ihrem `ElectionManifest` wird *Komposition* anstatt *Vererbung* eingesetzt, da dies sonst zu komplexen Vererbungsstrukturen führen würde.

Abbildung 6 zeigt das Paket mit allen Klassen wie auch die oben beschriebenen Vererbungsstrukturen.

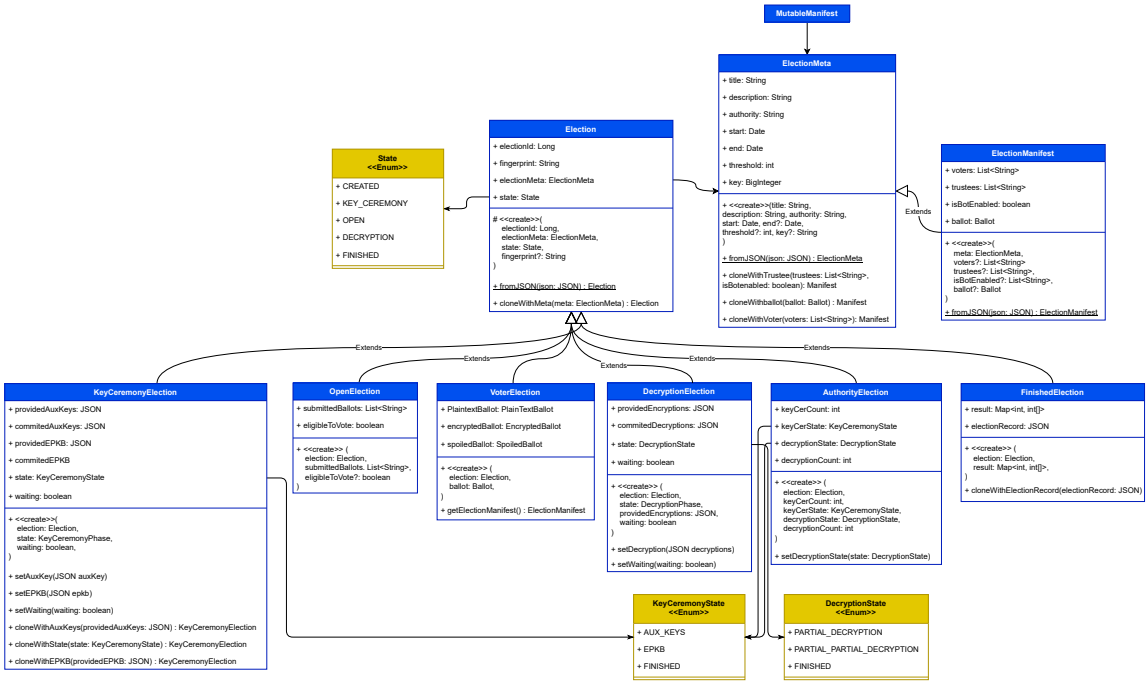

Abbildung 6: Klassendiagramm Paket Model.Election

3.3 View

Die View wird als *HTML5* Webseite entwickelt und verwendet *Cascading Style Sheets* zur Gestaltung. Da die Webseite eine Reihe an Interaktionen anbieten und dynamisch Informationen anzeigen muss, wird das Framework Vue.js.[1] verwendet. Vue ist ein performantes und reaktives Framework zur Entwicklung von Web-Benutzeroberflächen. Es wird die Composition Api von Vue verwendet, da diese u.A. eine bessere Typenkontrolle von Attributen ermöglicht. Das Vue Framework besitzt ein eigenes .vue Dateiformat. Diese Dateien können benutzt werden um einzelne Komponenten einer Webseite zu entwickeln und diese wiederum dynamisch aus Unterkomponenten zusammenzusetzen. In diesem Projekt soll es für jede Unterseite der Webseite eine eigene Vue Komponente geben, welche aus einer Reihe an Komponenten besteht um die entsprechende Seite darzustellen. Die Struktur der Komponenten orientiert sich hierbei stark an der im Pflichtenheft spezifizierten Seitenhierarchie der Webseite.

Die jeweiligen Komponenten halten ihre lokalen Model Objekte, die sie benötigen um Informationen anzuzeigen. Die Komponenten der View können durch Benutzereingaben in die entsprechenden HTML Input-Elemente Daten einlesen und anhand derer die Model Objekte verändern. Dafür hat jede Komponente eine Reihe an Methoden, welche den entsprechenden Eingabeelementen der Komponente zugewiesen werden können.

Die Abbildung 7 zeigt das Klassendiagramm der View Komponenten, wobei hier eine Vue Komponente als Klasse im UML Sinne zu betrachten ist. Die Attribute und Methoden sind alle als *protected* markiert, da lediglich die jeweiligen Unterkomponenten Zugriff auf die Attribute und Methoden ihrer jeweiligen übergeordneten Komponenten erhalten. Die View-Klassen halten keine Referenzen aufeinander, können sich aber gegenseitig durch Aufrufen von html-Links aktivieren. Die Pfeile in Abbildung 7 stellen somit keine *benutzt* Beziehung im Sinne eines klassischen UML-Klassendiagramms dar, sondern verdeutlichen, wie die Views sich gegenseitig aufrufen können

[1] `https://vuejs.org/`

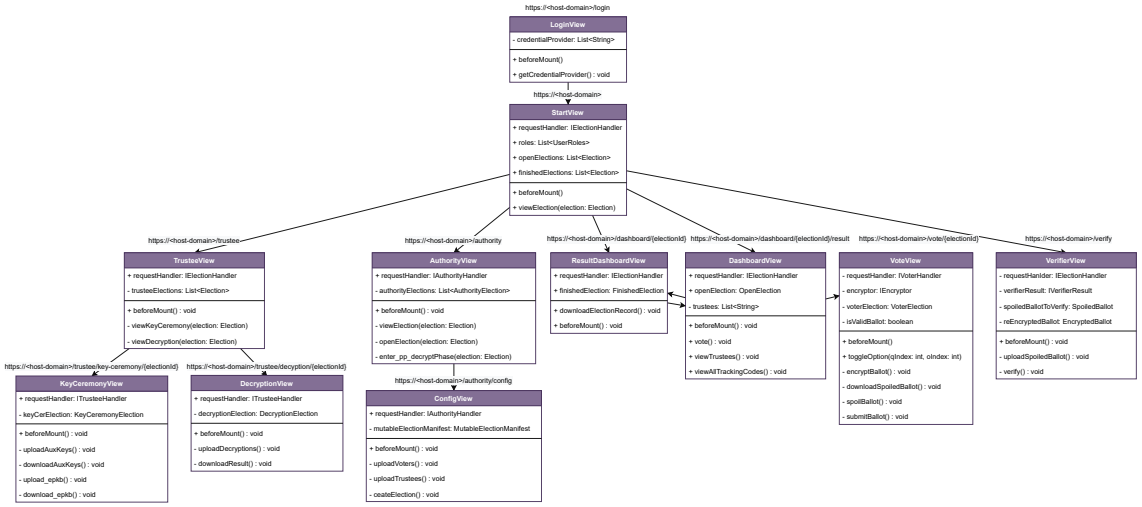

Abbildung 7: Klassendiagramm View

3.4 Controller

3.4.1 Handler

Das Paket *Controller.Handler* ist verantwortlich für die Kommunikation mit dem Backend. Die Klassen dieses Pakets rufen die Web Endpoints auf, um Informationen vom Backend anzufordern oder an das Backend zu senden. Diese Anfragen erfolgen asynchron. Die erhaltenen Informationen werden entweder direkt zurückgegeben oder in `Election`-Objekten gespeichert. Diese Objekte können dann im jeweils benötigten Typ abgefragt werden. Wird ein Typ abgefragt, für den noch nicht alle benötigten Informationen vorhanden sind, werden alle notwendigen Informationen direkt angefragt. Die Objekte sind oft nur teilweise initialisiert und werden nach Bedarf durch weitere Anfragen ergänzt. In den Klassen des Handler-Pakets wird im Allgemeinen nicht geprüft, ob die nötigen Anfragen im aktuellen Status zulässig sind. Bei unzulässigen Anfragen erhält der Handler einen Fehlerstatus vom Backend und wirft einen `RequestError`, der Informationen zur Ursache des Fehlers enthält. Da nicht alle Anfragen für alle Nutzer zur Verfügung stehen sollen, werden rollenspezifische Anfragen von den Klassen `AuthorityHandler`, `TrusteeHandler` und `VoterHandler` bereitgestellt. Da viele Anfragen auch für alle Nutzer relevant sind, erben die spezifischen Handler von der Klasse `ELectionHandler`, die allgemeinere Anfragen bereitstellt. Um die übrigen Module von den Details der Kommunikation zu entkoppeln und die verwendeten Web Endpoints leicht austauschbar zu machen, werden alle Handler-Klassen hinter einer Fassade aus entsprechenden Interfaces verborgen.

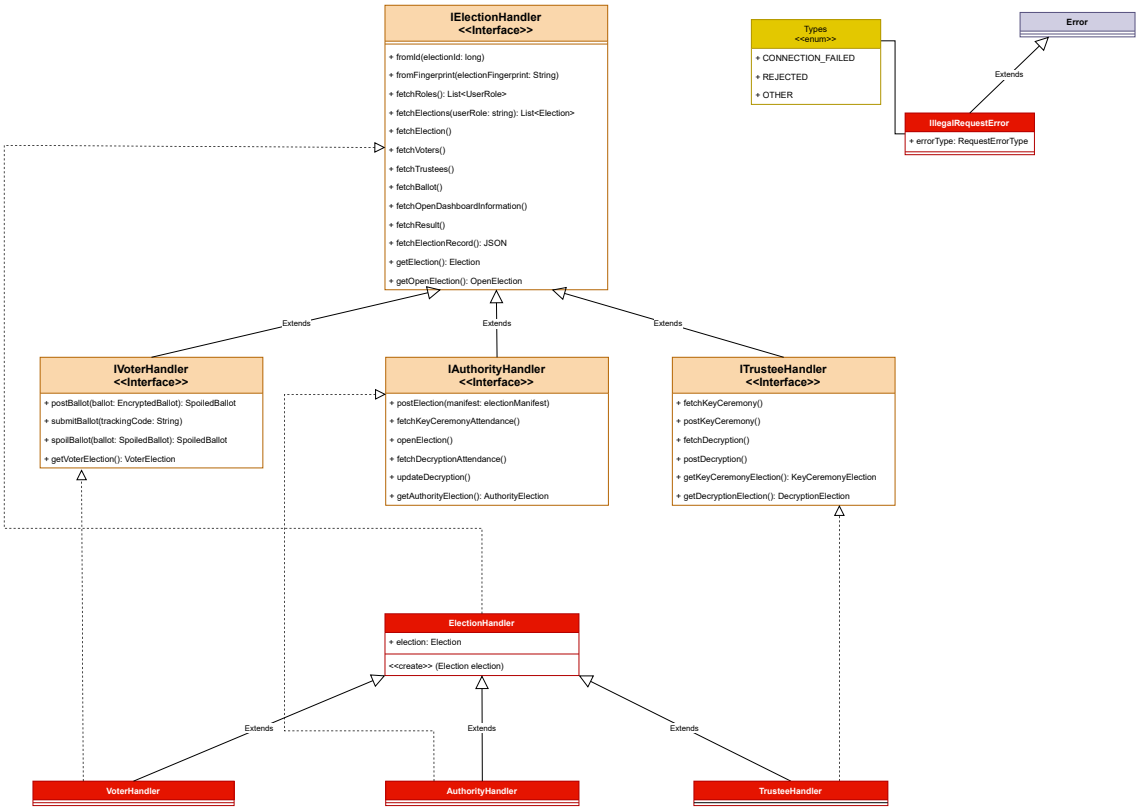

Abbildung 8: Klassendiagramm Handler

3.4.2 Encryption

Das Paket *Controller.Encryption* enthält alle Klassen, die zum Erstellen von verschlüsselten Ballots sowie zur Verifikation von Spoiled Ballots nötig sind. Die Klasse `Encryptor` erstellt aus einem `PlaintextBallot` ein `EncryptedBallot`, das die Verschlüsselung der Optionen und dazugehörige Korrektheitsbeweise enthält. Diese werden mithilfe einer TypeScript Implementierung [2] von ElectionGuard generiert. Die dabei verwendete Zufallszahl wird zunächst zwischengespeichert, da sie nicht im verschlüsselten Ballot enthalten sein darf. Die Klasse `Verifier` prüft ein `SpoiledBallot` auf Korrektheit der Verschlüsselung, des Tracking-Codes und des Election Fingerprints. Da dazu gemeinsame Funktionalität mit der Klasse `Encryptor` benötigt wird, die vom `Encryptor` nicht nach außen bereitgestellt wird, erweitert der `Verifier` die Klasse `Encryptor`. Die Ergebnisse der Überprüfung werden in einem `VerifierResult` zurückgegeben. Die Klassen `Encryptor` und `Verifier` werden hinter einer Fassade verborgen, um die Funktionalität leicht austauschen zu können.

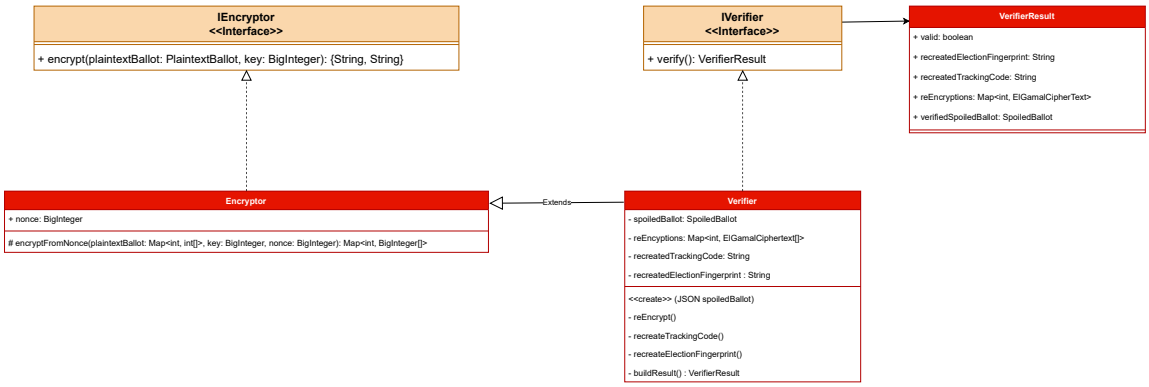

Abbildung 9: Klassendiagramm Controller-Encryption

[2] `https://github.com/danwallach/ElectionGuard-TypeScript`

3.5 Kommunikation mit dem Backend

Laden von Informationen

Im Sequenzdiagramm Dashboard wird am Beispiel des Ladens eines Dashboards dargestellt, wie Daten zu einer Wahl vom Backend angefordert werden. Zuerst wird immer eine Instanz einer der Klassen aus dem Paket Controller.Handler mit einem allgemeinen `Election`-Objekt benötigt. Diese `Handler` können mit einem `Election`-Objekt initialisiert werden, das von dem Singleton `ElectionStore` bereitgestellt wird. Falls dieses Objekt noch nicht initialisiert wird, kann auch ein neues `Election`-Objekt über den `Handler` angefordert werden. Anschließend können über den `Handler` weitere Anfragen an das Backend gerichtet werden, über die das `Election`-Objekt um Informationen ergänzt wird oder nach Bedarf auch eine speziellere Election angelegt wird. Bei jeder Anfrage muss aus dem alten E`Handler`-Objekt und den neuen Daten ein neues Objekt erstellt werden. Dies kann über die Konstruktoren der spezielleren `Handler`-Klassen erfolgen, wie man bei der Erstellung der `OpenElection` und `FinishedElection` 1 sieht. Alternativ können Objekte mit zusätzlichen Daten über die clone() Methoden der `Handler`-Klassen sowie der Klassen `ElectionMeta` und `ElectionManifest` erfolgen. Dies ist beispielsweise im Sequenzdiagramm Verifizierung zu sehen.

3.5.1 Posten von Informationen

Objekte, die im Frontend erstellt werden, können oft nicht vollständig initialisiert werden, da nicht alle Informationen im Frontend verfügbar sind. Meist werden zuerst die verfügbaren Informationen ans Backend geschickt und dort verarbeitet und vervollständigt. Anschließend können die Objekte entweder im Frontend durch die Antwort des Backend erweitert (siehe Wahlteilnahme) oder neu vom Backend angefordert werden. Dies ist beispielsweise bei der Wahlerstellung der Fall, die im Sequenzdiagramm Sequenzdiagramm Wahlerstellung dargestellt ist. Bei der Wahlerstellung fehlen z.B. Informationen wie die Wahlleitung, die erst im Backend hinzugefügt werden können. Damit die neu erstellte Wahl im Frontend sichtbar wird, muss erst die entsprechende Liste von Wahlen neu vom Backend angefordert werden.

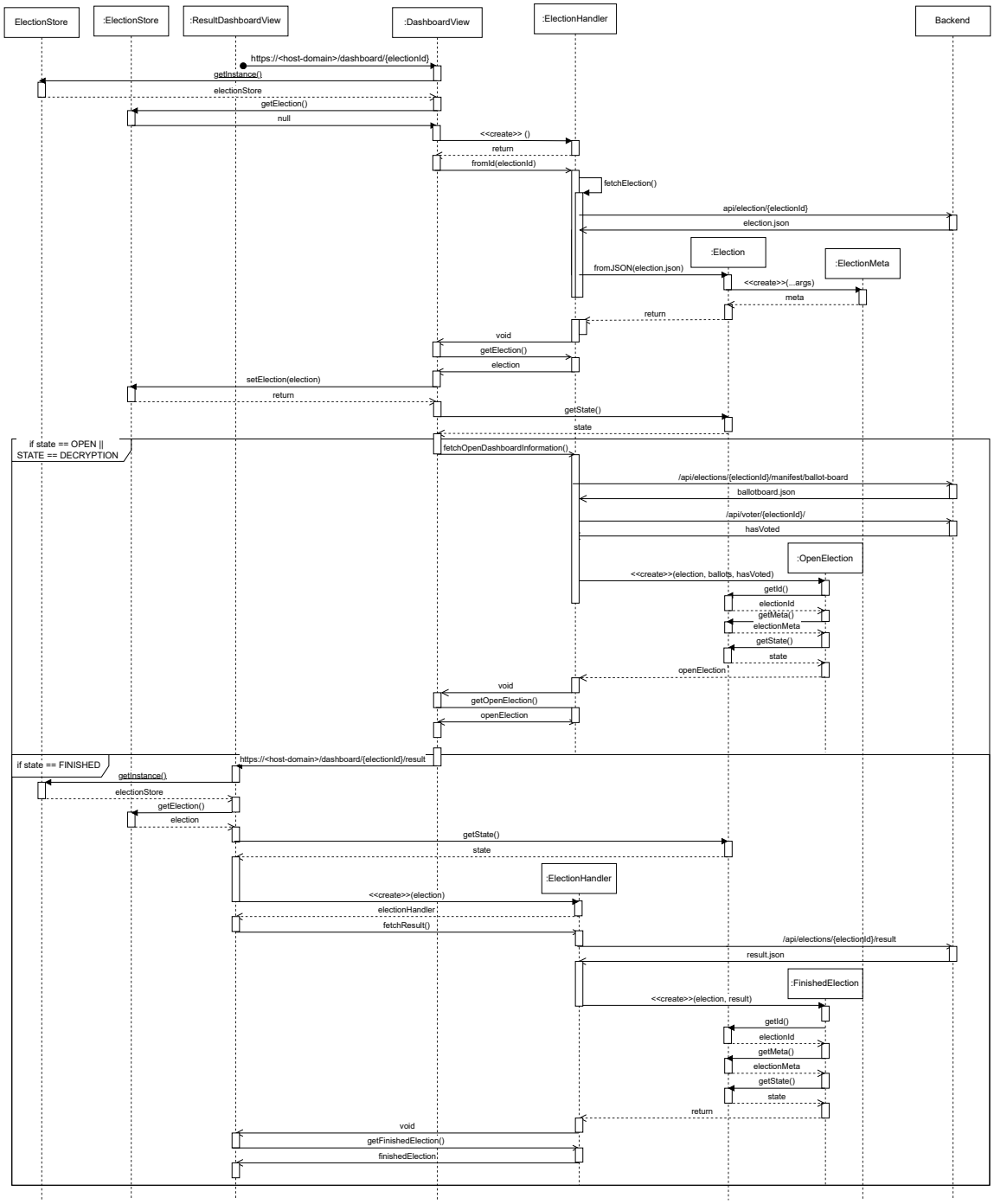

Abbildung 10: Sequenzdiagramm Dashboard

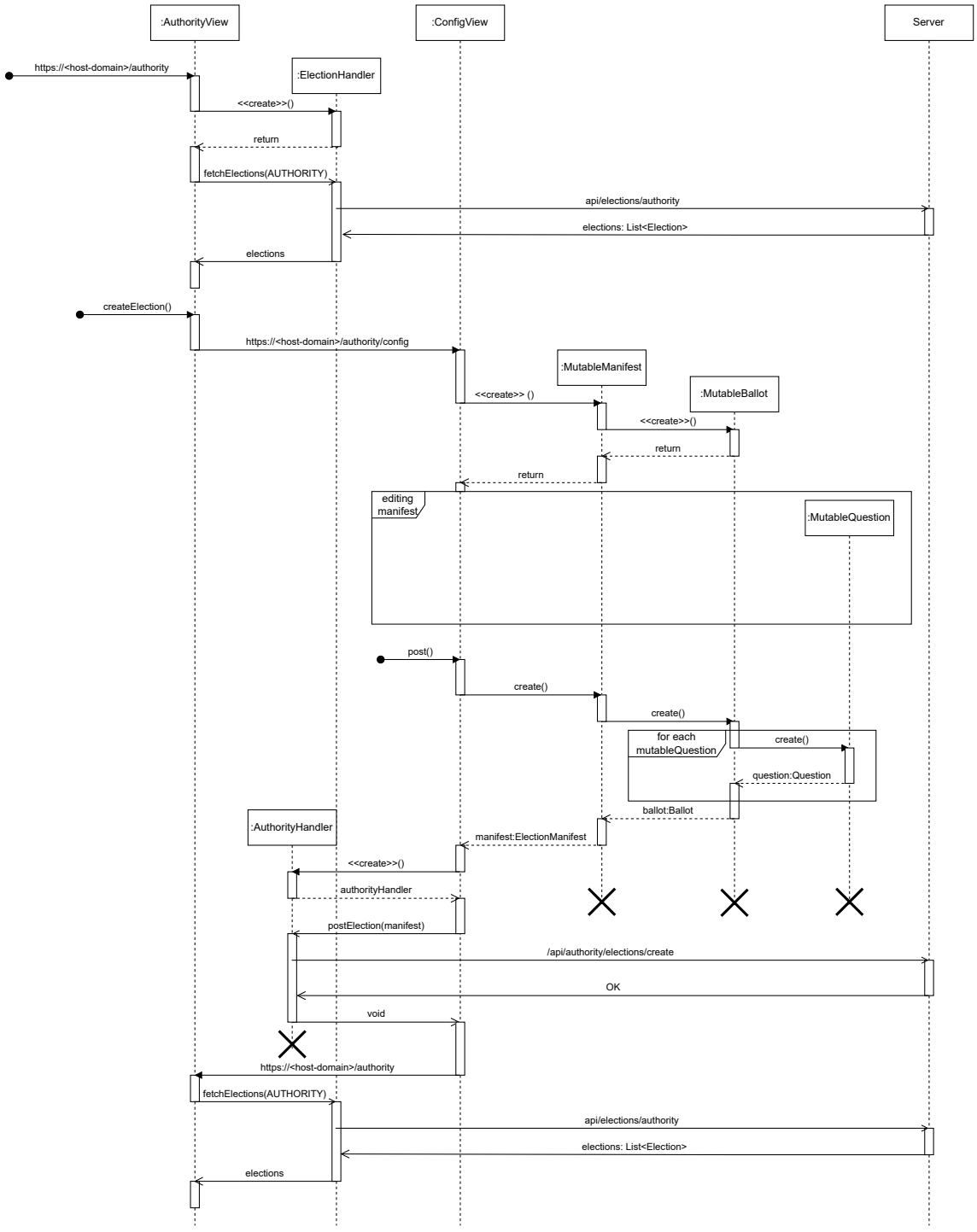

Abbildung 11: Wahlerstellung Dashboard

3.6 Key Ceremony und Entschlüsselung

Die Berechnungen zur Key Ceremony und Entschlüsselung werden von den Trustees lokal mit einer unabhängigen e-lection CLI durchgeführt. Die Ergebnisse werden im Frontend hochgeladen und dort in den entsprechenden `Election`-Objekten zwischengespeichert, bis sie an das Backend gesendet werden. Allerdings werden die Ergebnisse dort nicht verarbeitet oder überprüft. Aus diesem Grund werden sie auch nur als JSON-Objekte gespeichert und nicht deserialisiert.

3.7 Wahlteilnahme

Das Sequenzdiagramm zur Wahlteilnahme zeigt das Ausfüllen, Verschlüsseln und Abschicken eines Ballots. Im ersten Teil 1 wird die Konstruktionen eines speziellen `Election`-Objekts durch einen Getter des `VoterHandler` gezeigt. Dadurch kann die View durch einen Methodenaufruf eine spezielle `Election`, in diesem Fall eine `VoterElection`, erhalten, ohne zuerst die nötigen Daten einzeln anzufordern. Im zweiten Teil fig:seq:vote:2]2 wird die Funktion der Klasse `Encryptor` und die Verwendung der ElectionGuard Bibliothek gezeigt.

3.8 Verifizierung

Zur Verifizierung einer Verschlüsselung werden die Informationen zu der Verschlüsselung und zur Berechnung des Fingerprints der Wahl in einem `SpoiledBallot` gesammelt und als .json Datei exportiert. Diese kann von dem Nutzer heruntergeladen und (am Besten auf einem neuen Gerät) auf der Verifikationsseite hochgeladen werden. Das Sequenzdiagramm Verifikation zeigt, wie dieses `SpoiledBallot` verifiziert wird. Die `VerifierView` erstellt aus dem `SpoiledBallot` einen `Verifier`, der die Verschlüsselung, den Tracking-Code und den Fingerprint der Wahl nachrechnet. Anschließend werden die Informationen zur Wahl über einen `VoterHandler` über den Fingerprint neu angefordert, damit geprüft werden kann, ob das BallSot zu einer gültigen Wahl gehört.

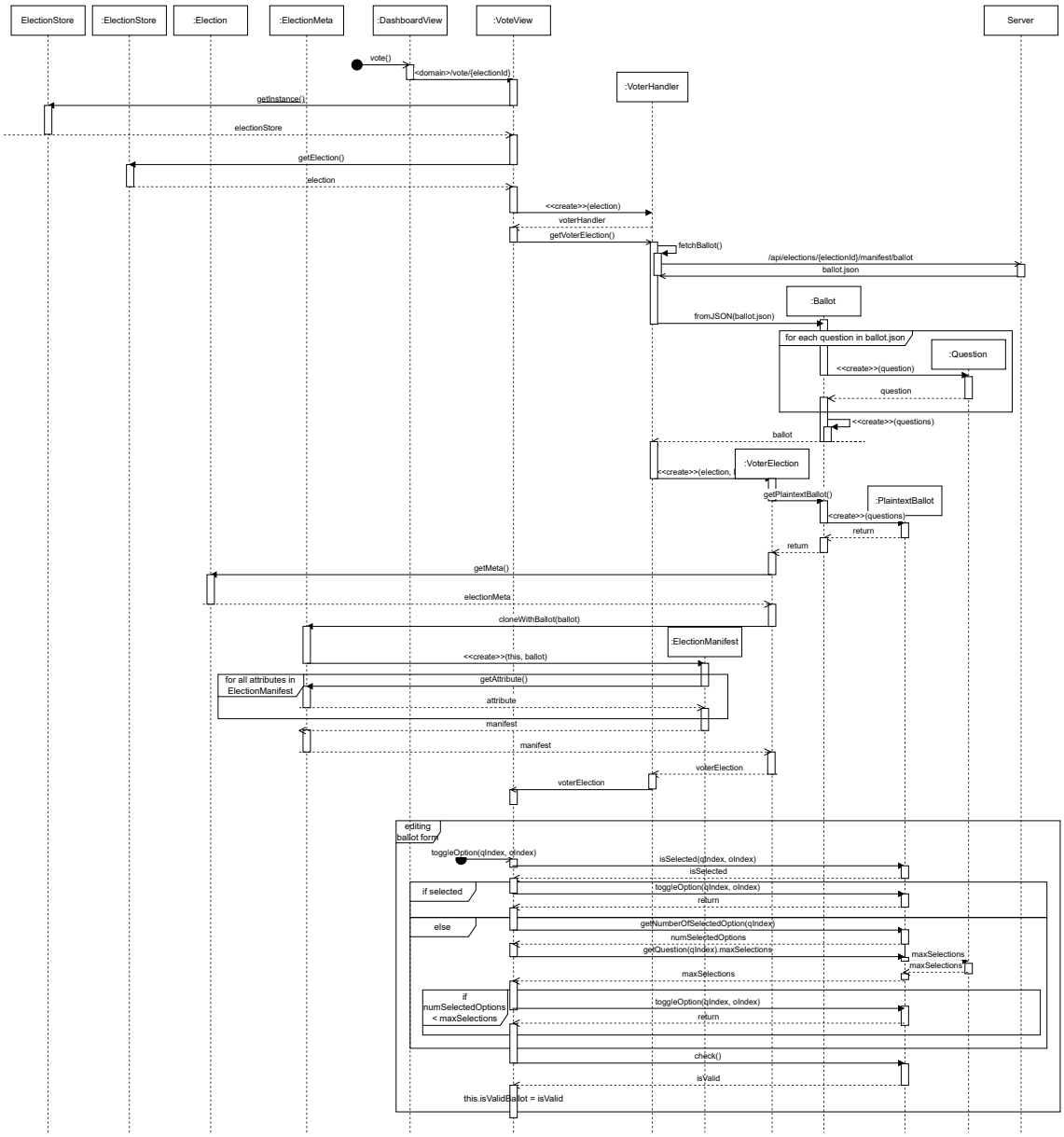

Abbildung 12: Sequenzdiagramm Wahlteilnahme(1)

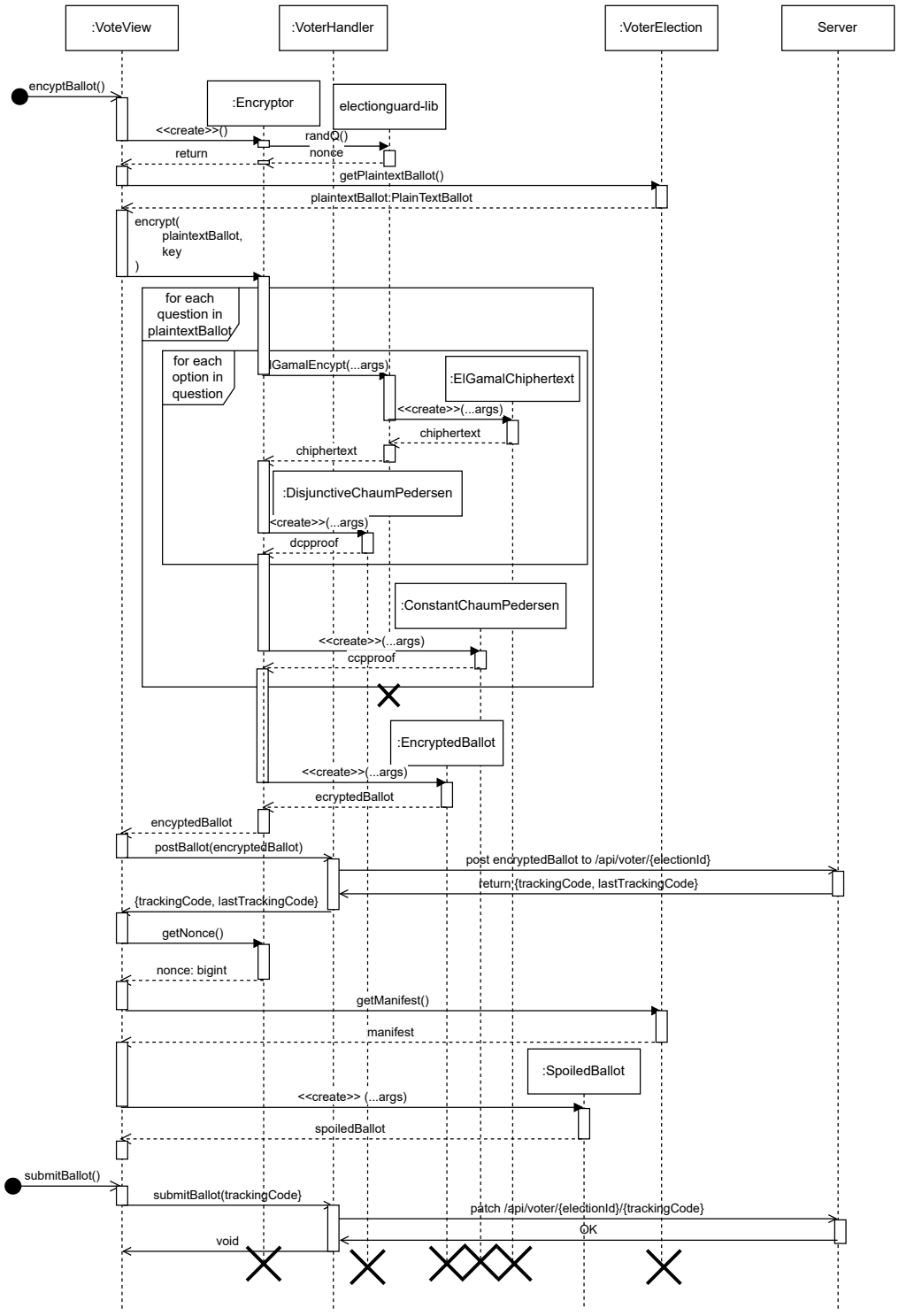

Abbildung 13: Sequenzdiagramm Wahlteilnahme(2)

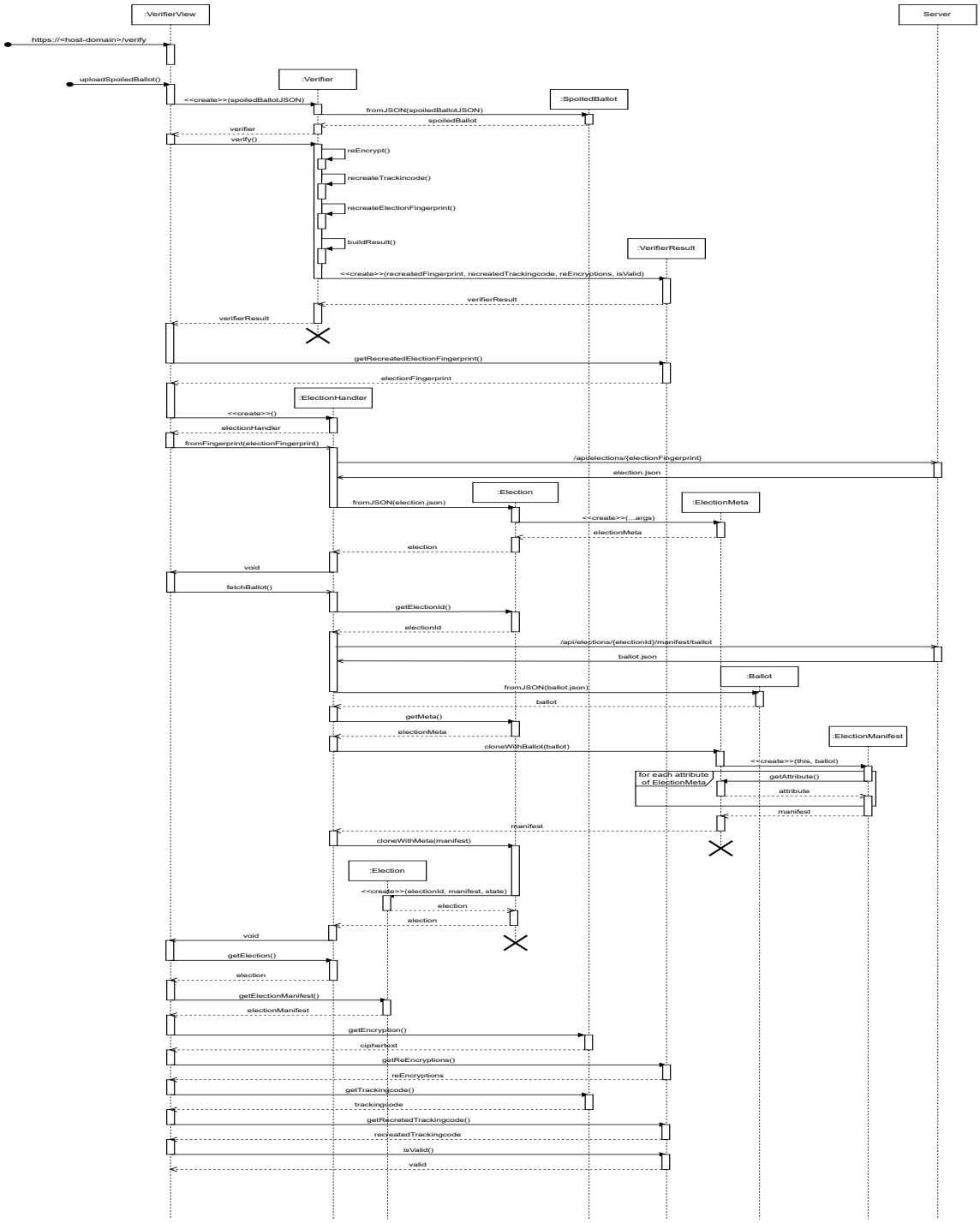

Abbildung 14: Sequenzdiagramm Verifikation

4 Backend

Das Backend von *e-lection* ist mit der Unterstützung von Spring Boot v3.0.1 entworfen worden. Spring ist ein Open-Source-Java-Framework, das Entwicklern dabei hilft, Anwendungen schnell und effizient zu entwickeln. Das Framework bietet aufgrund seiner umfangreichen Dependencies eine hohe Flexibilität und Sicherheit. Dafür kommuniziert es mit einer MySQL Datenbank, um Entitäten zu speichern und zu verwalten.
Die API verwendet folgende Spring Dependencies:

1. Spring Data JPA: Bietet die Möglichkeit über eine Java API und Hibernate, Daten bequem in einer SQL Datenbank zu speichern und zu verwalten. Mehr dazu im Kapitel der Datenbank.

2. Spring Web: Bietet die Möglichkeit mittels Spring MVC eine RESTful Web-Anwendung zu bauen. Mittels dieser Dependency können Endpunkte definiert werden.

3. Spring Security: Bietet die Möglichkeit, Authentifizierung und Zugriffssteuerung einzubringen.

4. OAuth2 Client: Wird für die Anmeldung mit OpenID Connect verwendet.

5. MySQL Driver: Bereitstellung von dem MySQL JDBC Treiber. Wird für die Kommunikation mit der Datenbank verwendet.

4.1 Aufbau des Backend

Um das Backend von *e-lection* unabhängig von der Umgebung der Außenwelt zu gestalten, wurde diese in 2 Teile aufgeteilt: Der Web-API und die eigentlichen *e-lection* Anwendung. Der Grund für diese Entscheidung ist, das so die Weichen gelegt werden, die API in einen grundlegend anderen Kontext bringen zu können. Abseits vom HTTP Protokoll oder in einem Wahllokal ohne Internetanschluss soll die Möglichkeit gegeben werden, *e-lection* verwenden zu können.

Abbildung 15 veranschaulicht den groben Aufbau der Schichten des Backends von *e-lection*. Die Web-API bildet dabei eine *HTTP-Fassade* und beinhaltet eine Authentifizierungskomponente. Die Authentifizierung des Nutzers erfolgt mit OpenID Connect 2.0. Von der Web-API gekapselt befindet des weiteren im Backend die *e-lection* Anwendung, die von der Web-API über ein *e-lection API* angesprochen wird. In der *e-lection* Anwendung befindet sich die Anwendungs-Logik, sowie die Datenbank und die Autorisierungs-Komponente.

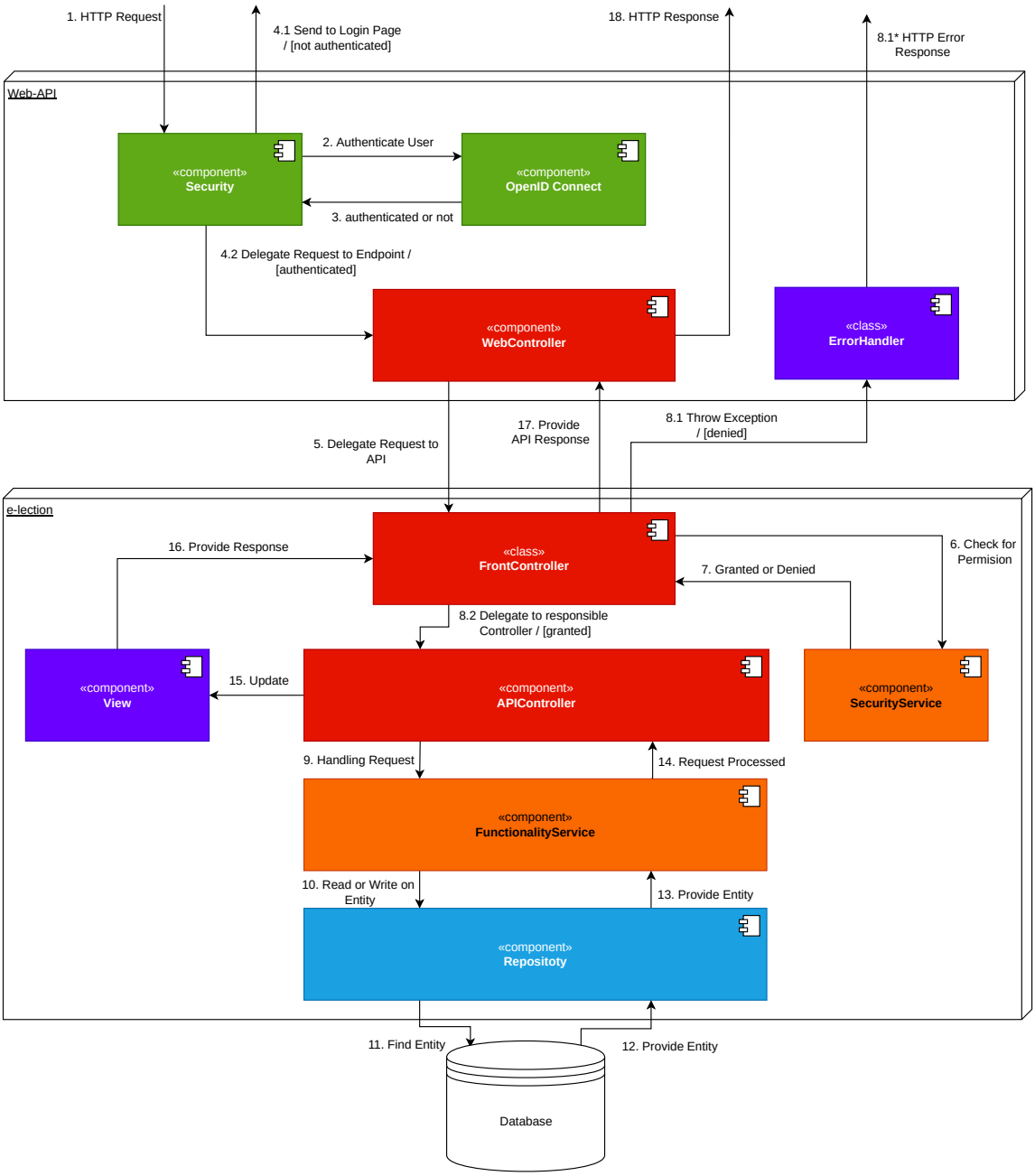

Abbildung 15: Überblick der Schichten des Backends

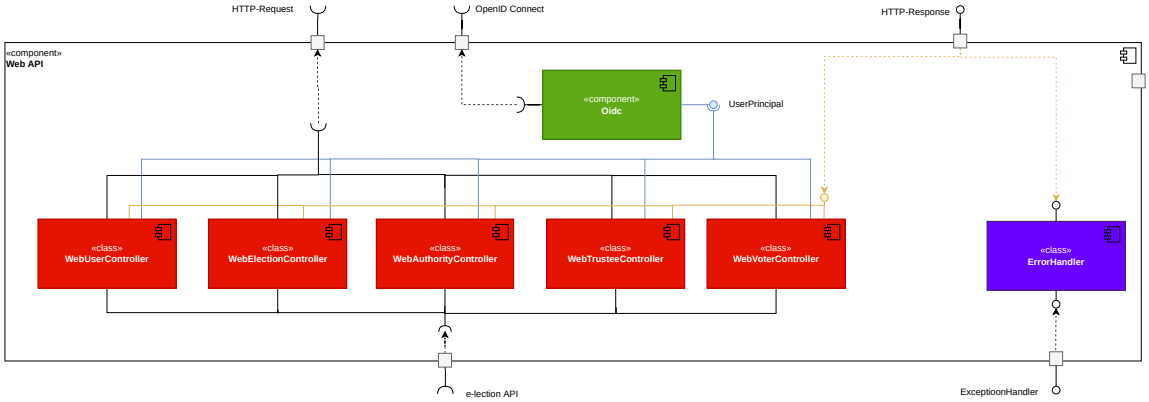

Abbildung 16: Aufbau der Web API

4.2 Aufbau der Web-API

Die Web-API beinhaltet sogenannte *Controller*, die über REST Endpunkte [3] angesprochen werden können. Diese Controller haben die Bezeichnung *WebController*. Damit ein Endpunkt angesprochen werden kann, muss der Nutzer im System eingeloggt sein. Für das Einloggen unterstützt die Web-API OpenID Connect 2.0. Dem Nutzer wird damit die Möglichkeit gegeben, sich mit einem bereits in einem anderen System existierenden Account bei *e-lection* anzumelden. Dafür erhält die Web-API Unterstützung von Spring Security. Über diese Dependency können Sicherheitsfilter definierten werden, die jede Anfrage durchlaufen muss, bevor diese einen Endpunkt erreicht. Diese Filter prüfen unter anderem die Authentifizierung des Nutzers. Mehr dazu im Abschnitt zu Sicherheit. Eine weitere Aufgabe der Web-API ist die Fehlerbehandlung. Geht in der *e-lection* Anwendung etwas schief, so muss die Web-API es dem Klient auch mitteilen können. Dafür gibt es die *ErrorHandler* Komponente. Diese fängt von der *e-lection* Anwendung geworfene *Exceptions* auf und übersetzt diese in passende HTTP-Fehlerstatus Codes. Abbildung 16 veranschaulicht die Komponenten der Web API. Über die *UserPrincipal* Schnittstelle, kann sich jeder *WebController* den authentifizierten Nutzer in den Endpunkt holen.

4.3 Aufbau von e-lection

Die *e-lection* Anwendung baut auf einer Model-View-Controller-Service Architektur auf. Bei dieser Architektur erhalten Controller eine Anfrage und Übersetzen diese falls nötig. Die übersetzte Anfrage wird an einen Service zur Bearbeitung übergeben. Die Services sind logisch getrennte Logik-Blöcke, die für Verarbeitung von Daten und Entitäten zuständig sind. Dafür greifen diese auf Repositories zu. Diese speichern und verwalten Entitäten und

[3] Spezifikationen der Endpunkte in backend-api-specification.md

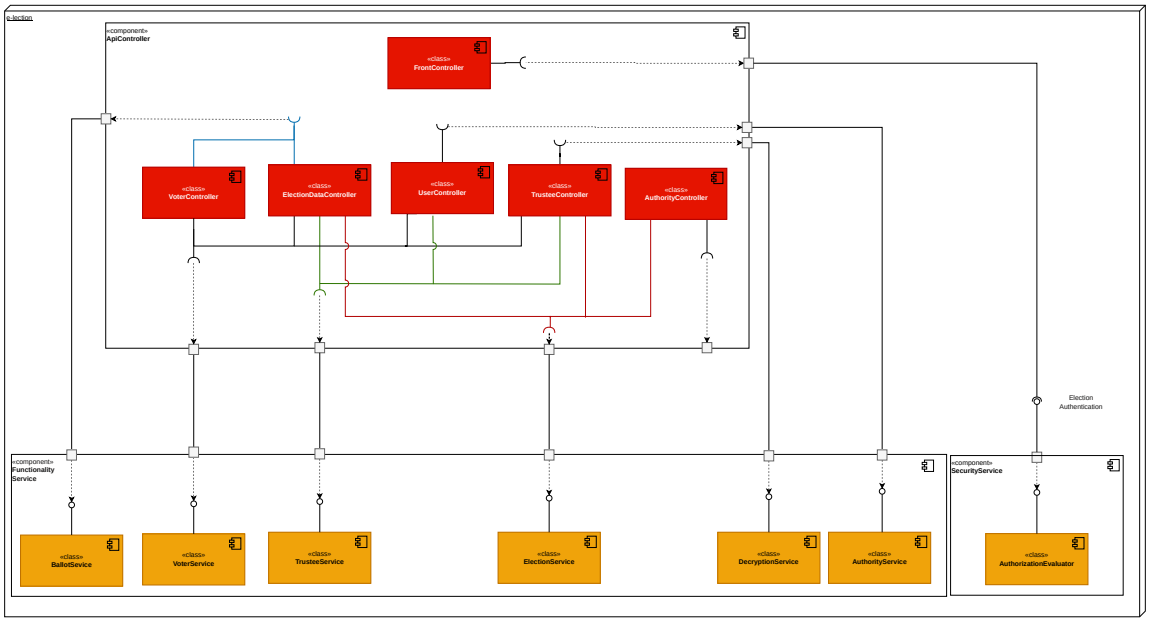

Abbildung 17: Interaktionen zwischen den Controllern der Anwendung und den Services

kommunizieren mit der MySQL Datenbank. Nach der Prozessierung der Daten und Entitäten, übergeben die Services dem Controller eine überarbeitete oder erstellte Entität. Diese übergibt er der View, um daraus eine Antwort zu konstruieren. Die View erhält dafür die notwendinen Informationen und Daten vom Controller und konstruiert daraus eine Antwort, die der Controller an die darüberliegende Schicht zurückgibt. Diese Sequenz ist auch in Abbildung 15 zu sehen. Die Besonderheit an der Anwendung von *e-lection* ist hierbei, dass die Controller und Services untereinander getrennt wurden. Bei den Controllern gibt es den `FrontController` und die *ApiController* und bei den Services gibt es den *SecurityService* und die *FunctionalityServices*. Der `FrontController` erhält jede eingehende Anfrage und überprüft, ob der Nutzer für die Prozessierung dieser Anfrage berechtigt ist. Dies prüft der *SecurityService*, indem er über eine *Authentication* Schnittstelle zur Web-API an die E-Mail des Nutzers gelangt. Dazu mehr im Kapitel zur Autorisierung des Benutzers. Erst nach der Überprüfung wird vom `FrontController` die Anfrage an den entsprechenden *API-Controller* weitergeleitet, wo sie von den *FunctionalityServices* verarbeitet werden.
In Abbildung 17 wird die Kommunikation zwischen den Controllern der Anwendung und den Services veranschaulicht. Die Farben der Pfade haben keine semantische Bedeutung und dienen rein der Übersicht. Zwischen den Services kommt es zu weiteren Interaktionen, die aus Gründen der Übersicht nicht in Abbildung 17 zu sehen sind. Dazu gibt es im Kapitel zu Services mehr.
Abbildung 18 veranschaulicht den Aufbau und die Unterteilung der Pakete des Backends.

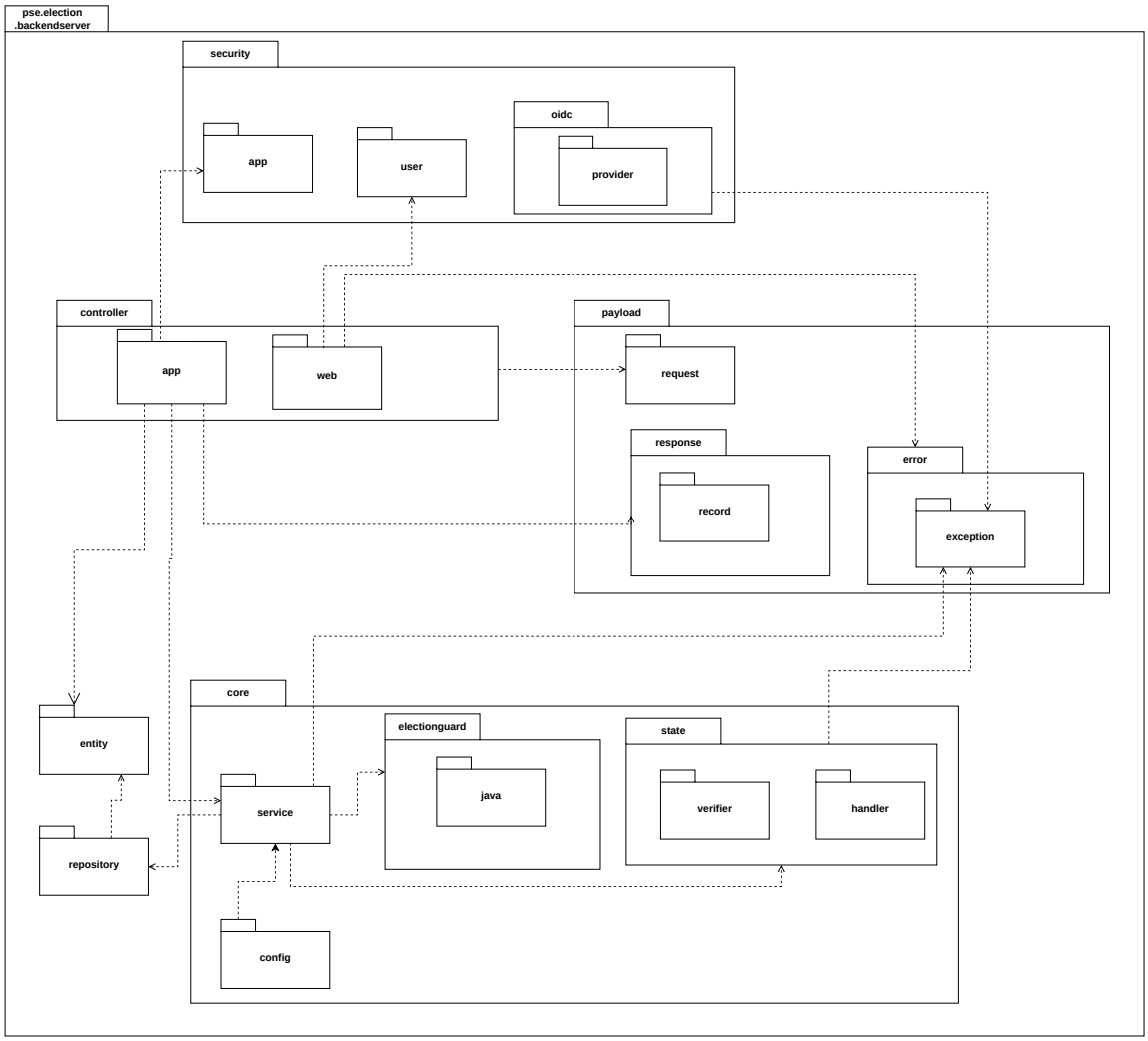

Abbildung 18: Paketdiagramm zum *e-lection* Backend

4.4 Interaktion mit dem Backend

In diesem Abschnitt wird auf die Interaktion mit dem Backend eingegangen. Dies beinhaltet die Art und Weise, die das Backend die Kommunikation nach außen umsetzt. Um ein näheres Verständnis dafür zu entwickeln, wird erst ein wenig darauf eingeganen, wie Spring Web das Interagieren mit REST-Endpunkten umsetzt.

4.4.1 Anfragen auf das Backend

Um mit dem Backend zu interagieren stellt dieser REST-Endpunkte bereit. Diese werden von *WebControllern* aufgefangen und dem System übergeben. Wie in Spring ein *WebController* erstellt werden kann und somit ein Endpunkt definiert werden kann, veranschaulicht folgendes Beispiel:

```
@RestController
@RequestMapping("/hier/kommt/ein/base/url")
public class BeispielController {

    @GetMapping("/hello")
    public String helloWorld() {
        return "Hello World";
    }
}
```

Die *@RestController*-Annotierung markiert die Klasse als einen *WebController* in Spring. Über *@RequestMapping(/hier/kommt/ein/base/url)* kann so der BASE URL definiert werden. Jede Web-Anfrage mit dem Aufbau: *{domain}/hier/kommt/ein/base/url* wird dann auf diesen Controller geleitet. Fügt man dieser Anfrage noch */hello* dazu, und führt eine GET-Request durch, so wird die Methode *helloWorld()* ausgeführt und der Benutzer bekommt *Hello World* in seinem Browser angezeigt.

Wenn ein Nutzer eine Entität im Backend erstellen oder verwalten möchte, so muss dieser eine Anfrage an das Backend schicken. Aufgrund der Nutzung der *WebController* kann die nötige Information als *PathVariable* - die Information ist mit in der URL verbaut, oder *RequestBody* - ein JSON-Body der bei der HTTP Request mitgeschickt wird, an das Backend übermittelt werden. Damit das Backend aber mit Java Objekten anstatt JSONs hantiert, bietet Spring Web die Möglichkeit, JSONs in Objekte zu übersetzen. Damit das klappt, müssen extra Klassen angefertigt werden, die den Aufbau der JSON in dem Body widerspiegeln. Das folgende Beispiel zeigt, wie das Backend eine Wahlerstellung entgegennimmt. Das JSON im *RequestBody* hat das folgende Format:

```
{
  electionMeta: {
    title: String,
    description: String,
    start: LocalDateTime / String,
    end: LocalDateTime / String,
    threshold: int,
  },
  ballot: {
    questions: [
      {
        questionText: String,
        options: String[],
        minSelections: int,
        maxSelections: int
      }
    ]
  },
  trustees: [
    {
      email: String
    }
  ],
  bot: boolean,
  voters: [
    {
      email: String
    }
  ]
}
```

Abbildung 19 zeigt den Auszug der *e-lection* Anwendung, der für das Entgegenehmen dieser Request zuständig ist. Sie veranschaulicht die Verwendung einer `ElectionCreationRequest`, um Informationen für eine neu erstelle Wahl aus der HTTP Request zu übersetzen. Die Request Klasse wird von den *WebControllern* an den `AuthoriryController` weitergeleitet und dieser erstellt eine `Election` Entität basierend auf den Informationen aus der `ElectionCreationRequest`. Dabei muss die `ElectionCreationRequest` genau das Format der JSON widerspiegeln. Der Controller ist dann für die Übersetzung der Request zuständig. Das bedeutet, dass dieser Entitäten in den Endpunkten aus den Informationen der Request erstellt und diese den entsprechenden Services übergibt. Wie genau der Controller die Services in diesem Beispiel anspricht, veranschaulicht Abbildung 35 in einem späteren Kapitel.

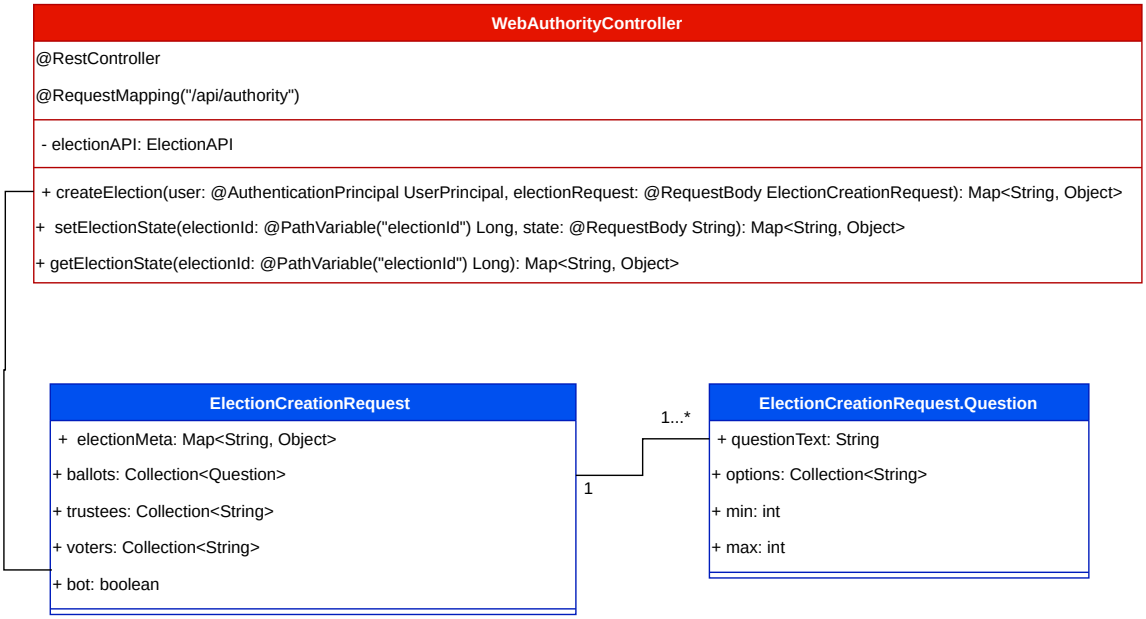

Abbildung 19: ElectionCreationRequest als Klasse die Informationen über eine neu erstelle Wahl beinhaltet

4.4.2 Antworten vom Backend

Um mit der Außenwelt kommunizieren zu können, erstellt die *e-lection* Anwendung antworten. Meist beinhalten diese Informationen über die Entitäten in der Datenbank. Da spezifische Anfragen nur spezifische Informationen von einer Entität benötigen, gilt es diese aus den Entitäten zu extrahieren. Dafür hat die *e-lection* Anwendung eine View Komponente verbaut. Diese besteht aus einem `ResponseBuilder`, der eine `Response` erstellt. Diese `Response` beinhaltet eine `Map<String, Object>`, die von dem `ResponseBuilder` konstruiert wurde. Der Grund für diese Map ist, das Spring Web eine Map in eine JSON umwandeln kann. Wie genau, illustriert das folgende Beispiel:

```java
@GetMappting("/url/to/get")
public Map<String, Object> mapToJsonExample() {
    Map<String, Object> map = new HashMap<>();

    List<Integer> list = new ArrayList<>();
    list.add(42);
    list.add(2023);

    map.put("hello", "world");
    map.put("listing", list);
    return map;
}
```

Wenn nun die Methode aufgerufen wird, ist die Antwort eine JSON mit dem folgenden Format:

```
{
    hello: "world"
    listing:
        0:  42
        1:  2023
}
```

Abbildung 20 zeigt den Aufbau der View und wie diese mit den *ApiControllern* interagiert. Die `ResponseBuilder` wird von dem aufgerufenen Controller aufgerufen und erhält die notwendigen Entitäten und Daten, um eine Antwort zu konstruieren. Der `ResponseBuilder` erstellt dann eine formatierte `Response`. Falls ein *Record* erstellt werden muss, so gibt es die `RecordBuilder` Schnittstelle, über die der `ResponseBuilder` einen Record mittels dem Strategy-Pattern erzeugen kann. Abbildung 21 zeigt den Aufbau der View Komponente der *e-lection* Anwendung. Details dazu stehen in den JavaDocs.

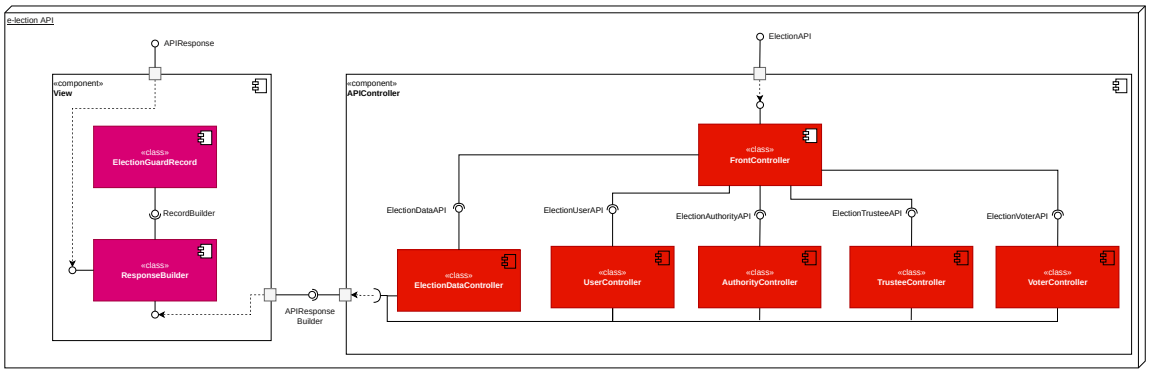

Abbildung 20: Zusammensetzung der Interaktion zwischen den APIControllern un der View

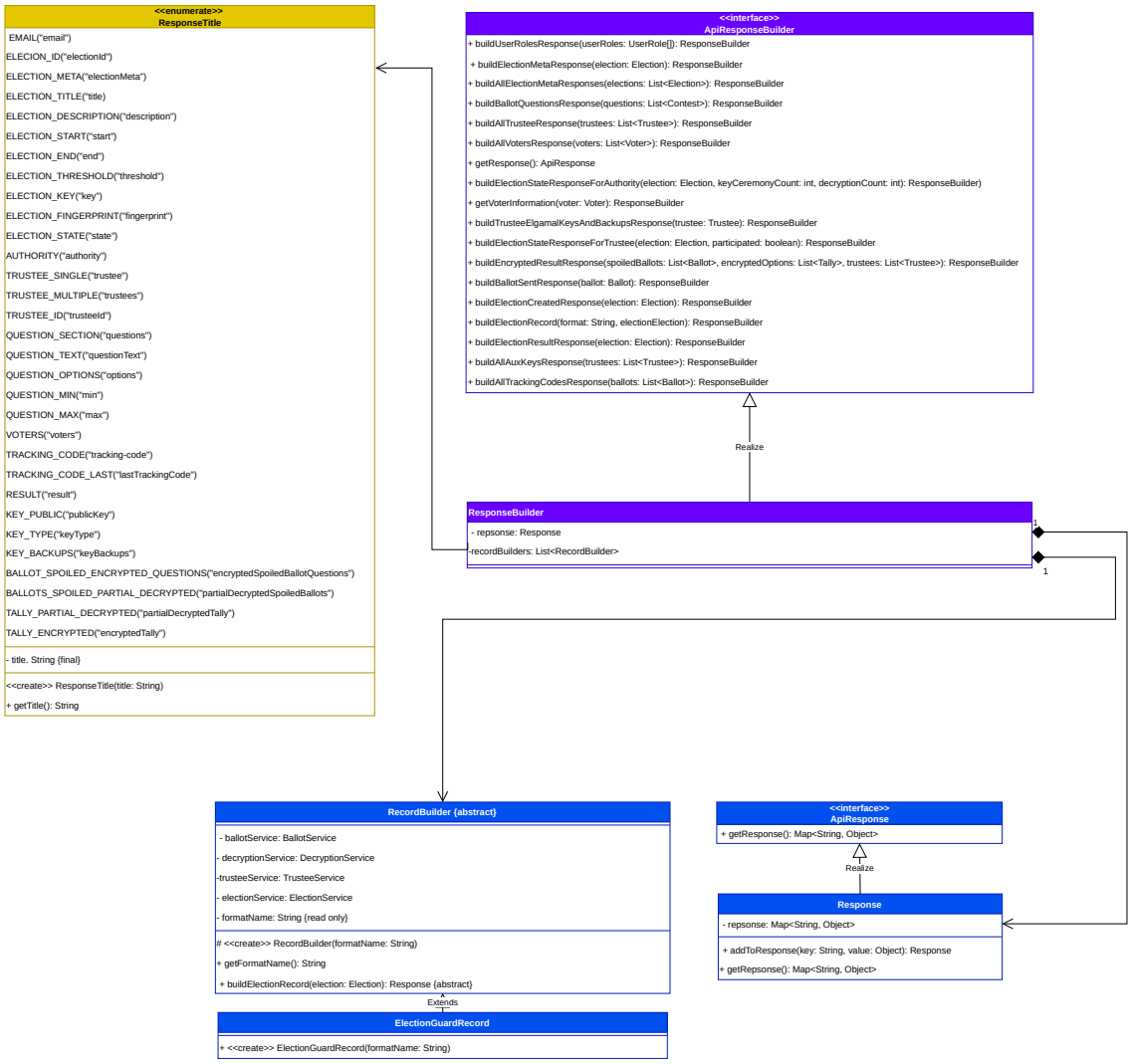

Abbildung 21: Aufbau der View-Komponente der *e-lection* Anwendung

Abbildung 22 zeigt den Aufbau der *ErrorHandler* Komponente. Dieser erbt aus Spring's `ResponseEntityExceptionHandler` und wird somit als *Exception Handler* von Spring markiert. Die Methoden im `ErrorHandler` geben eine `ResponseEntity` zurück, dessen HTTP Status Code und *ResponseBody* den Fehler näher beschreibt.

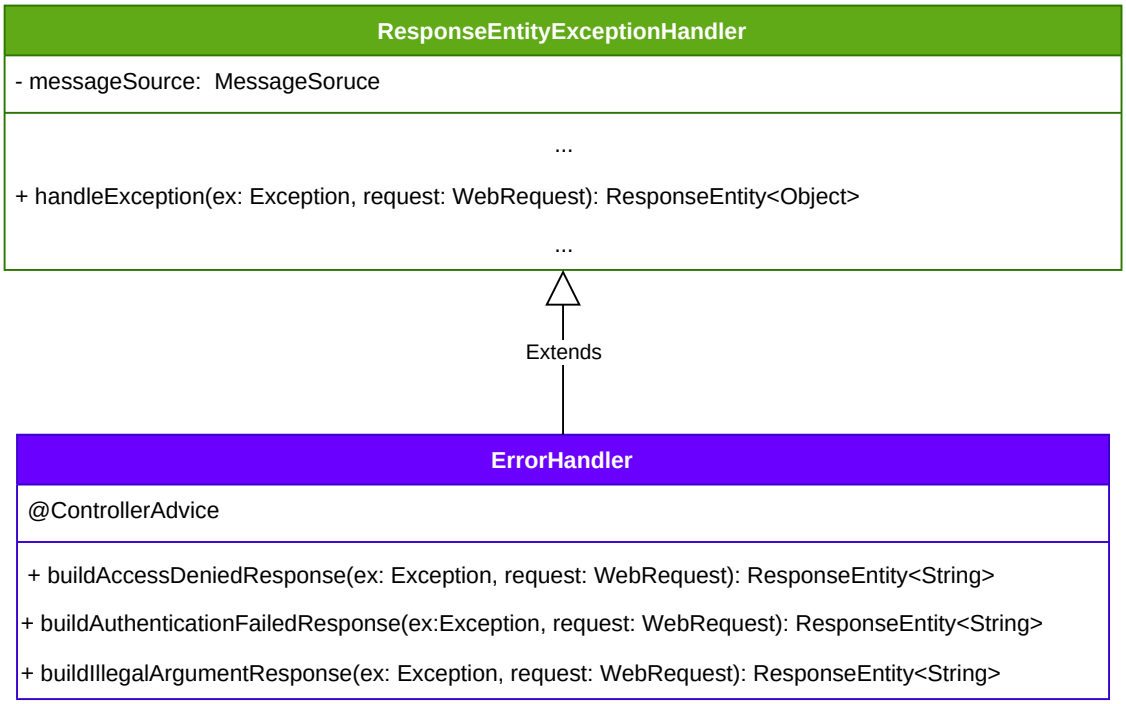

Abbildung 22: Aufbau der *ErrorHandlers* der Exceptions in HTTP Fehlercodes umwandelt

4.5 Sicherheit

Die Web-Anwendung von *e-lection* ermöglicht es seinen Nutzern sich mit ihrem Shibboleth Account des Karlsruher Institut für Technologie oder ihrem Google Account anzumelden. Dies wird durch eine OpenID Connect Schnittstelle ermöglicht, die mithilfe des Spring Security Frameworks und der OAuth2.0 Unterstützung erweiterbar für neue Provider ist und der Anwendung die nötige Sicherheit liefert. Die Sicherheit in der Anwendung lässt sich auf 3 Komponenten reduzieren: OpenID Connect Authentifizierung und Autorisierung des Benutzers, sowie dem Schutz der Wahlen vor Änderungen außerhalb des erlaubten Wahl-Zustands.

4.5.1 Authentifizierung mit OpenID Connect

Für die OpenID Connect Schnittstelle verwendet die Web-API größtenteils die von der OAuth2.0 mitgebrachten Komponenten. Der Aufwand für die Entwickler liegt hierbei beim integrieren einer Benutzerinformations-Schnittstelle, in welcher festgelegt wird, welche Benutzerinformationen für seine Anwendung benötigt werden. Die Benutzerinformationen werden bei einer erfolgreichen Authentifizierung bei einem Provider von diesem an unsere Web-Anwendung geschickt. Da das Format dieser Antwort zwischen den Providern unterschiedlich ist, muss auch bei der Extrahierung der Benutzerinformationen geholfen werden. Die Validierung von ID Tokens, sowie das Speichern der Benutzerinformationen wird von Spring unterstützt.
Spring setzt dies durch eine sogenannte *SecurityFilterChain* um. Dies ist eine Anreihung von Filtern, die jede Anfrage auf einen Endpunkt durchlaufen muss, bevor sie diesen erreicht. Diese Filter können in der *SecurityConfig* Klasse der Web-Anwendung definiert werden.

```
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity httpSecurity) {
        //Hier koennen auf httpSecurity die gewuenschten Filter definiert werden
        return httpSecurity.build();
    }

}
```

Die *@EnableWebSecurity*-Annotierung markiert für Spring die Klasse als eine Web-Sicherheits-Konfigurationsklasse und erlaubt es *@Bean*-annotierte Methoden zu definieren. Die mit *@Bean*-annotierten Methoden werden von Spring erkannt und verwaltet, sodass diese zur „richtigen“ Zeit aufgerufen werden. In der Methode *filterChain(httpSecurity)* können zum Beispiel die URLs festgelegt werden, auf die der Nutzer im Falle eines auslaufenden *Access-Tokens* - ein Token der bei erfolgreicher Authentifizierung erstellt wird und den Zugriff auf

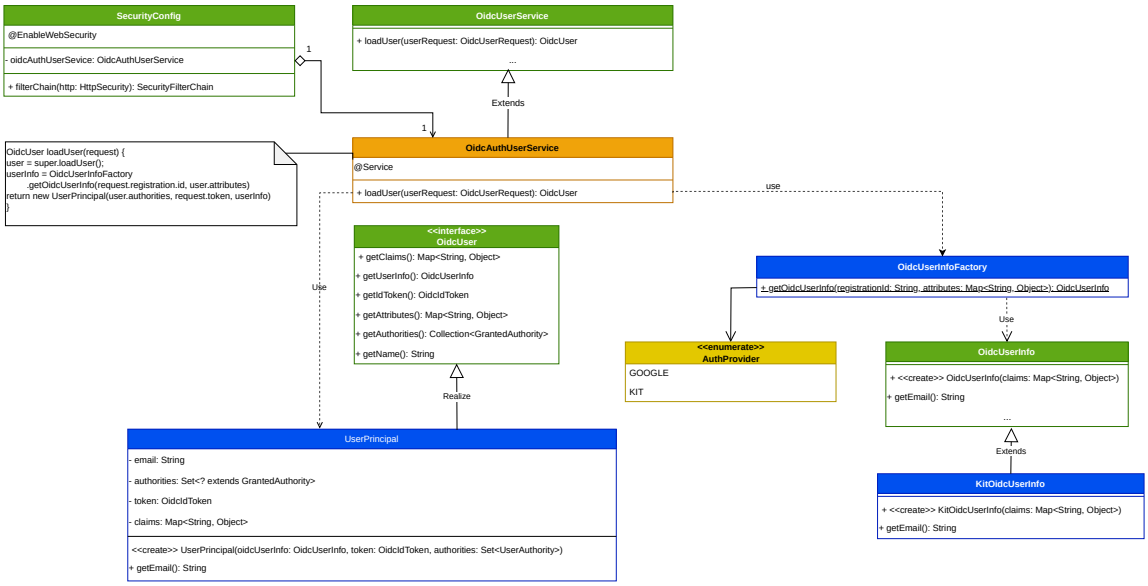

Abbildung 23: Klassen für die Anmeldung mit OpenID Connect

die API erlaubt, weitergeleitet werden kann.
Des weiteren wird ein `UserPrincipal` in Spring's *SecurityContext* gespeichert. Dies ist ein Bereich, der Informationen über den authentifizierten Nutzer in einem lokalen Thread speichert, sodass diese jederzeit abrufbar sind. Der `UserPrincipal` ist ein Nutzer-Objekt der für die Speicherung der Email-Adresse des eingeloggten Nutzers verwendet wird. Dieser wird aus der Antwort des Providers beim Einloggen erstellt. Abbildung 23 zeigt die Klassen der Web-Anwendung, um aus der Antwort des Providers den `UserPrincipal` zu erstellen.

In der `SecurityConfig` wird der *SecurityFilterChain* der `OidcAuthUserService` übergeben, sodass er diesen zur Authentifizierung verwendet. Der Service ist dafür zuständig, einen `UserPrincipal` zu erstellen, der als Namens-Attribut die Email-Adresse des Nutzers speichert. Dafür wird eine Factory verwendet, die je nach Provider die dazugehörige `OidcUserInfo` Klasse wählt. Für Google reicht die Standard `OidcUserInfo` Implementierung aus, für den Shibboleth Provider des KITs muss jedoch eine extra Klasse dafür erstellt werden. Dies ist nötig, da der Shibboleth Provider die E-Mail Adresse nicht unter dem Attribut *email* bereitstellt (wie bei Google), sondern unter dem Attribut *eduperson-principal-name*. Nachdem der `UserPrincipal` erstellt wurde, wird dieser in dem *SecurityContext* im lokalen Thread abgestellt. Danach kann dieser in eine Methode, welche als Parameter die *@AuthenticationPrincipal* Annotierung in Kombination mit dem `UserPrincipal` Objekt beinhaltet, durch Spring's *Dependency Injection* übergeben werden. Abbildung 24 veranschaulicht den gerade beschriebenen Prozess. Hierbei passiert die Anfrage auf den Server verschiedene Filter, bis sie den `AuthenticationFiler` erreicht. Dieser erkennt, das es sich um eine OAuth2.0

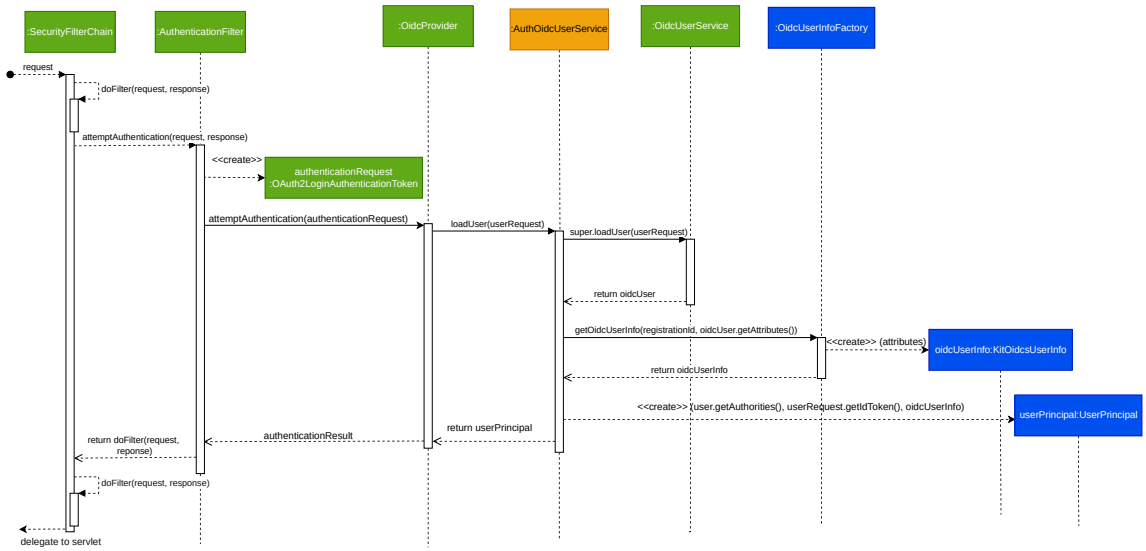

Abbildung 24: Authentifizierung eines Benutzers mit dem Shibboleth Identity Provider des KITs

Authentifizierung handelt (OpenID Connect Protokoll benutzt OAuth2.0 zur Authentifizierung) und erstellt einen Token, der die Informationen des `UserPrincipal` speichert.

4.5.2 Autorisierung der Benutzer

Für die Autorisierung gibt es einen zuständigen *SecurityService* in der *e-lection* Anwendung. Dieser ist von der Web-API getrennt und schützt die Entitäten der Anwendung vor unautorisierte Änderungen. Dies erreicht er durch das Prüfen, ob der Benutzer, der eine Funktionalität ausführen möchte, dazu berechtigt ist. Die Berechtigungen werden im System durch Rollen definiert. Es gibt die Rollen *Authority*, *Trustee* und *Voter*. Die Festlegung der Befugnisse der einzelnen Rollen ist auf die Controller geteilt. Die Verteilung der Zugriffsrechte ist dementsprechend folgendermaßen.

1. AuthorityController: Nutzer mit der Rolle *Authority* zu einer erstellten Wahl. Ausnahme ist die Wahlerstellung, dafür reicht die Rolle aus.

2. TrusteeController: Nutzer mit der Rolle *Trustee* zu einer zugewiesenen Wahl

3. VoterController: Nutzer mit der Rolle *Voter* zu einer zugewiesenen Wahl

4. UserController: Jeder Nutzer, da dieser nur Informationen über den Nutzer selbst verwaltet

5. ElectionDataController: Nutzer muss der angefragten Wahl zugeordnet sein, unabhängig der Rolle

Der `AuthorizationEvaluator` prüft dann, ob der Nutzer die entsprechende Rolle hat. Dieser kann ebenfalls prüfen, ob der Nutzer in einer bestimmten Wahl über diese Rolle verfügt. Um an die entsprechende E-Mail Adresse des Nutzers zu gelangen, gibt es einen `ElectionAuthenticationAdapter`, dessen einzige Aufgabe es ist, die E-Mail Adresse des Nutzers zu holen. Durch die Verwendung von Spring Web gibt es eine *Authentication* Schnittstelle, durch welche der authentifizierte Nutzer aus dem *SecurityContext* geladen werden kann. Dies ist die generischste Authentifizierungs-Schnittstelle, denn diese hat nur eine *getName()* Methode. Dies ist auch der Grund, wieso der `UserPrincipal` die E-Mail als Namens-Attribute haben muss. Mit der E-Mail Adresse des Nutzer kann nun die Rolle des Nutzers identifiziert werden und damit kann der `AuthorizationEvaluator` über die Befugnis entscheiden.
Der `FrontController` ist dafür zuständig, den `AuthorizationEvaluator` anzusprechen. Bei Erfolg, delegiert er auf den verantwortlichen Controller weiter. Abbildung 25 veranschaulicht den Prozess zur Autorisierung mit einem Beispiel, in der ein Wahlleiter den Zustand seiner Wahl ändern möchte.

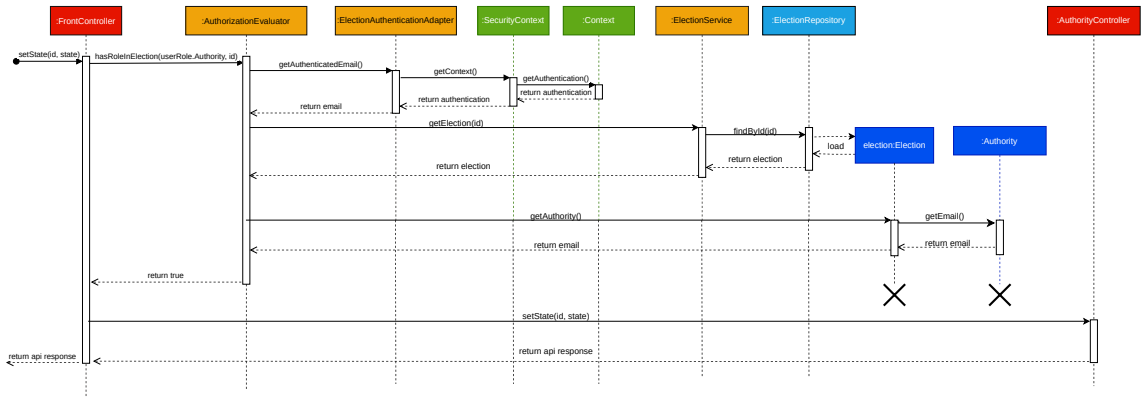

Abbildung 25: Autorisierung bei einer Anfrage

4.5.3 Sicherung der Wahlen

Für die Überprüfung, ob eine Wahl in einem Zustand ist, in der eine erfragte Funktionalität erlaubt ist, gibt es einen `ElectionStateEvaluator`. Welche Funktionalität in welchem Zustand durchgeführt werden darf, entscheiden die verantwortlichen Services selbst. Dies hat einerseits den Grund, dass die Services für ihre Entitäten verantwortlich sind. Des Weiteren ist die Aufteilung fix, sodass eine separate, dafür zuständige Komponente zur Festlegung der Funktionalitäten zu erhöhter Komplexität ohne spürbaren Nutzen erzielen würde. Ein Service, der eine Anfrage prozessiert, spricht daher als erstes den `ElectionStateEvaluator` an und übergibt ihm die Zustände, in der die zu bearbeitende Wahl sein muss. Dieser überprüft dann, ob die Wahl den Zustand hat und gibt dementsprechend eine Rückmeldung.

4.6 Controller

Die Controller haben die Aufgabe, Anfragen entgegenzunehmen und diese auszuführen. Sie bilden bilden eine Schnittstelle zwischen der Außenwelt und der Anwendungs-Logik. Das Backend von *e-lection* hat eine 3-schichtige Controller-Architektur. Die Web-API verfügt über *WebController*, über die HTTP Anfragen entgegengenommen werden können. Diese sprechen dann eine *e-lection API* an. Diese Anfrage wird von dem `FrontController` entgegengenommen und bezüglich der Befugnis geprüft. Bei erteilter Befugnis leitet der `FrontController` die Anfrage auf einen *ApiController* zur Ausführung weiter. Nach Verarbeitung erhält dieser eine Entität, über die eine Antwort generiert werden kann. Diese Antwort wird über den `FrontController` an den *WebController* übergeben, welcher daraus eine JSON baut und dem Anfrager übergibt.

4.6.1 Überblick über die Controller

Abbildung 26 zeigt den Aufbau der Controller in der gesamten Anwendung. Die *WebController* kommunizieren dabei mit einer *OpenID Connect* Komponente, über die sie den aktuellen Nutzer - dem *UserPrincipal*, in ihre Endpunkte einbinden können. Dies ermöglicht es der API, die Benutzer bei der Ausführung der Funktionalitäten zuzuordnen. Aus Abbildung 26 wird auch die Trennung der *Web-API* und der eigentlichen *e-lection* API verdeutlicht. Zudem zeigt diese, wie der *SecurityService* mit den *ApiControllern* und Spring's Security kommuniziert. Aus Gründen der Übersicht wurde hier nicht dargestellt, welche Services für die Repositories zur Autorisierung angesprochen werden.

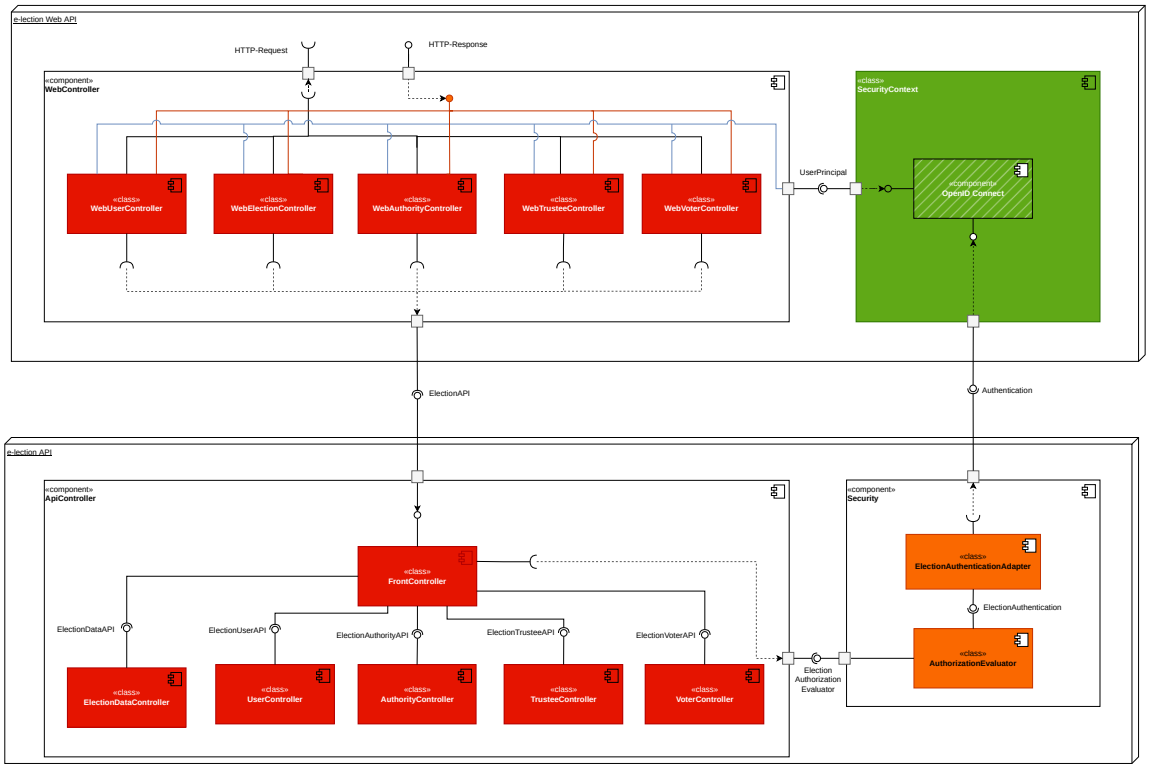

Abbildung 26: Übersicht über die Controller Komponenten und dem SecurityService

4.6.2 Aufbau der WebController

Die *WebController* definieren die REST Endpunkte der Anwendung. Auf die Umsetzung in Spring wurde bereits im Kapitel Anfragen an das Backend eingegangen. In der Anwendung gibt es pro URL Base einen zuständigen *WebController*. Die Zuteilung dabei ist folgendermaßen:

1. WebAuthorityController: {domain}/api/authority/elections
2. WebElectionController: {domain}/api/electuins
3. WebTrusteeController: {domain}/api/elections/{electionId}
4. WebUserControlelr: {domain}/api/user
5. WebVoterController: {domain}/api/voter/{electionId}

Jede von denen spricht die *e-lection API* auf und erhält eine `ApiResponse` zurück. Auf diese kann man mittels *getResponse()* eine `Map<String, Object>` erhalten, welche die Antwort der

API in eine JSON umwandelt. Für eine genauere Beschreibung zu den *WebControllern* und den Endpunkten verweise ich auf die JavaDocs.

4.6.3 Aufbau der APIController

Der `FrontController` ist Bestandteil der *APIController*, bildet aber eine eigene Schicht, da dieser keine Funktionalitäten der API ausführt. Er dient nur als Sicherheits-Komponente. Um alle Anfragen auf die API entgegenzunehmen, implementiert er die *e-lection API*. Abbildung 26 veranschaulicht dabei, wie der `FrontController` die Kommunikation mit dem *SecurityService* durchführt.
Nach der Überprüfung gelangt die Anfrage an einen für die Verarbeitung verantwortlichen Controller. Dazu hat jeder *WebController* einen *API-Controller* Zwilling. Um die Anfrage zu prozessieren, sprechen die Controller mit den zuständigen Services. Dies ist in der Abbildung 17 aus dem Kapitel e-lection Aufbau zu erkennen. Die Services antworten immer mit einer Entität, welcher dann dem `ResponseBuilder` übergeben werden kann. Dies veranschaulichte Abbildung 20 aus dem Kapitel Antworten des Backends. Abbildung 27 ist ein Ausschnitt aus den *ApiControllern* und ist für die Trustee Funktionalität zuständig. Dazu mehr in den JavaDocs.

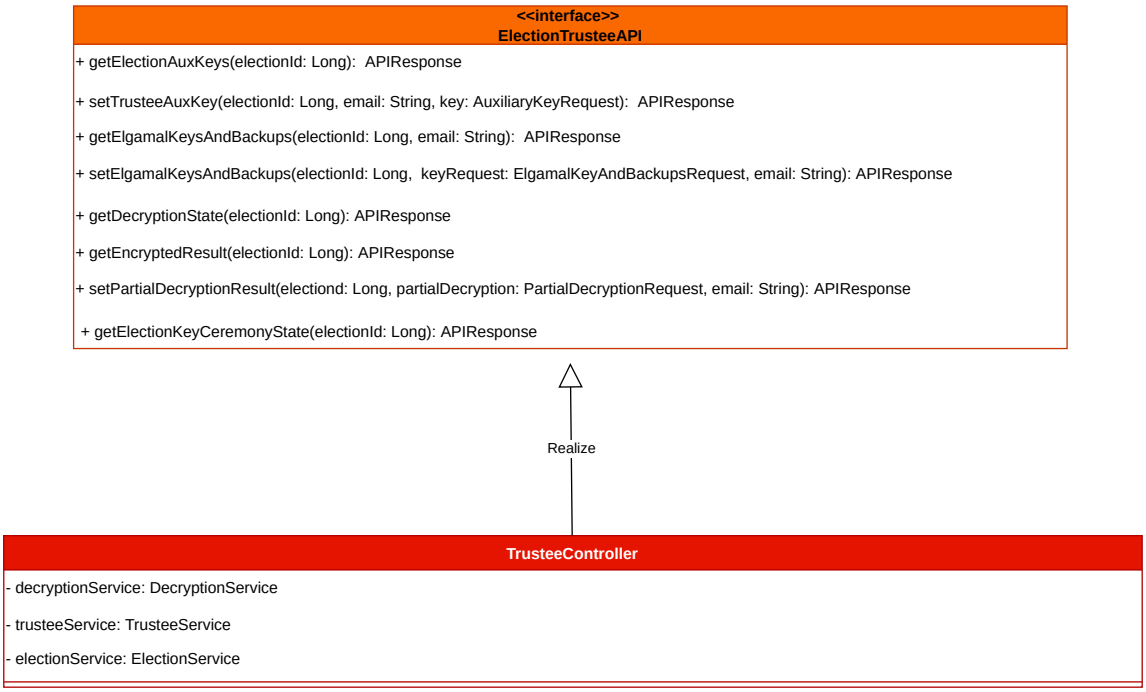

Abbildung 27: TrusteeController als Beispiel eines *APIControllers*

4.7 State Paket

Das *state* Paket übernimmt zwei Aufgaben. Die erste ist die Verifizierung, dass Endpunkte, die Wahlen verändern, nur angesprochen werden können, wenn sich die Wahl auch im richtigen ElectionState befindet. Dieser Teil liegt im *verifier* Paket. Die zweite Aufgabe ist zu überprüfen, ob der Wechsel in eine anderen ElectionState möglich oder sogar notwendig ist. Auf die zweite Aufgabe, die im *handler* Paket liegt, wird im Folgenden genauer eingegangen.

4.7.1 Grundidee des Service

Ein wichtiger Teil des E-lection Systems ist es, zu prüfen, ob sich durch gewisse Aktionen ein Wechsel des *ElectionState* einer Wahl ergeben hat. Dabei gibt es drei verschiedene Ereignisse, die so einen Wechsel auslösen könne.

1. Durch das Hinzufügen von Trusteeinformationen wurde ein *ElectionState* abgeschlossen und die angesprochene Strategie entscheidet automatisch in den Nächten *ElectionState* zu wechseln. Hierbei ist die Strategie eine Art Nachbedingung.

2. Die Wahlleitung entscheidet, in den nächsten *ElectionState* überzugehen und es wird geprüft, ob dies erlaubt ist.

3. Durch das Abgeben einer Stimme wird die Strategie als Vorbedingung aufgerufen, um zu überprüfen, ob das Enddatum der Wahl nicht bereits erreicht wurde.

Um dies umzusetzen, muss man zwischen zwei Funktionen unterscheiden. Die Erste überprüft, ob der Wechsel in den nächsten ElectionState möglich und zu machen ist, die zweite initialisiert den neuen ElectionState nach dem ein State Wechsel umgesetzt wurde. Hierbei kann die Methode des *stateSwitches()* entweder als Nachbedingung genutzt werden, um zu überprüfen, ob sich durch eine Eingabe ein Wechsel ergeben hat oder auch als Funktion, die aufgerufen wird, wenn wie in 2 beschrieben ein Wechsel von der Wahlleitung vorgegeben wird. Im Folgenden wird der grundlegende Aufbau des *handler* Pakets beschrieben.

4.7.2 Aufbau des handler Pakets

Der Aufbau des *handler* Pakets basiert auf dem Entwurfsmuster Strategie. Dieses ermöglicht bei Laufzeit zu entscheiden, welche Art der StateStrategy aufgerufen werden soll. Die Kontrolle des Pakets liegt bei dem StateHandler der zwischen den Strategien unterscheiden kann. Die einzelnen Strategien implementieren, wie man im folgenden Klassendiagramm sieht, die Funktionen des *StateSwitch*-Interfaces. Der Zugriff auf diese Funktionalitäten wird durch das *Upgradable*-Interface gekapselt, wodurch nur ein Anstoß kommt, ein Update auf eine Wahl durchzuführen, und der *ElectionHandler* selbstständig entscheidet, wie er dieses Update durchführt.

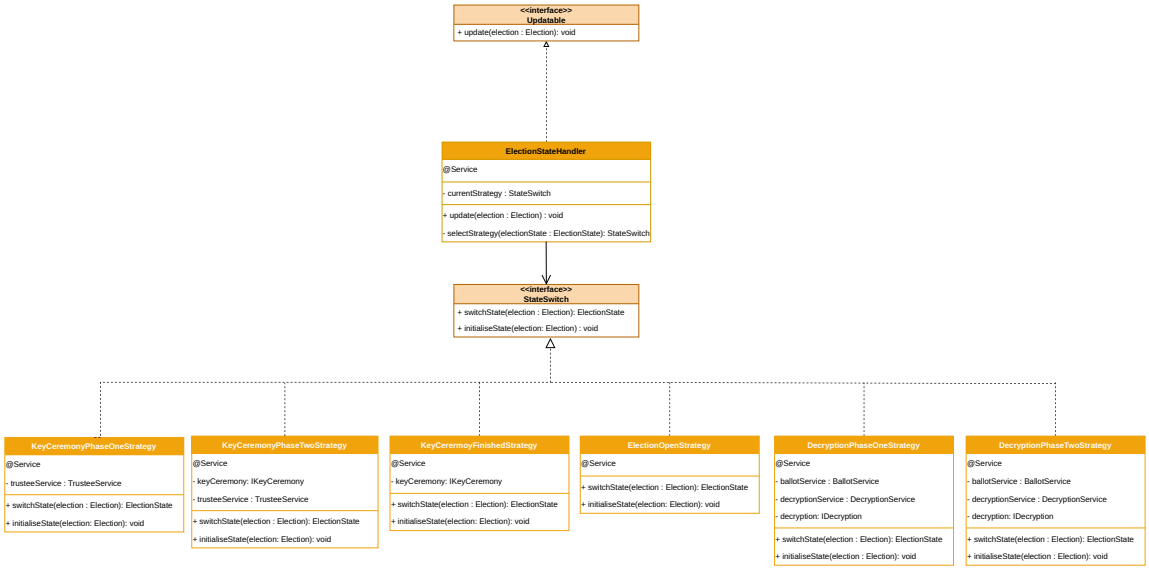

Abbildung 28: Aufbau der State Strategy, sowie deren Zusammenhang

4.7.3 Beispiel: Letzter Trustee gibt seinen Auxiliary Key ab

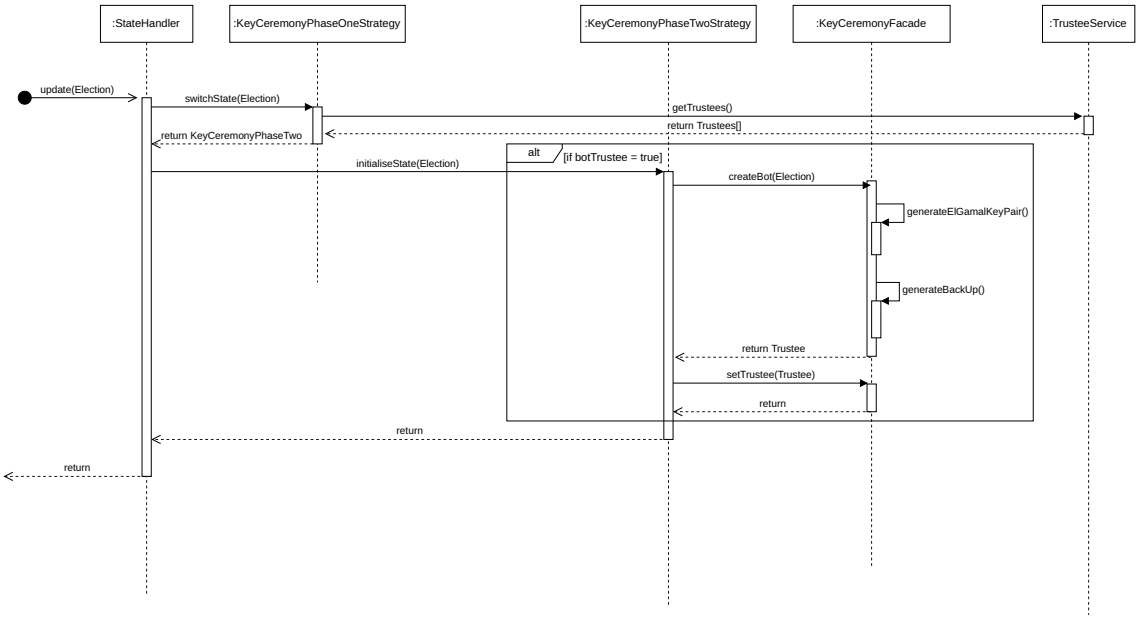

Abbildung 29: Sequenzdiagramm zur Abgabe des letzten Auxiliary Key

In diesem Sequenzdiagramm wird der Vorgang beschrieben, der eintritt, falls der letzte Trustee seine Auxiliary Key hochlädt und die Wahl außerdem einen System Bot besitzt. Zuerst wird in der *KeyCeremonyPhaseOneStrategy* überprüft, ob eine wechsel des ElectionState nötig ist. Da dies der letzte Trustee ist, der seinen Auxiliary Key hochladen muss, damit alle Auxiliary Keys im System angekommen sind, ist der Wechsel erfolgreich und die Strategie der *KeyCeremonyPhaseTwo* wird initialisiert. Dabei wird der System Bot erstellt und in die Datenbank eingefügt. Damit ist der State Wechsel abgeschlossen.

4.8 ElectionGuard Paket

Das *Electionguard*-Paket kümmert sich um das Einbinden des Java ElectionGuard Projekts, welches vor allem kryptografische Funktionen zur Verfügung stellt. Dazu zählt unter anderem das Verifizieren von Beweisen, das erstellen von Trackingcodes und Fingerprints sowie das entschlüsseln von Texten. Dabei wird jeder Teilbereich von einer Fassade zusammengefasst und von dieser zur Verfügung gestellt. Der Zugriff auf die verschiedenen Fassaden wird dabei von den im folgenden beschriebenen Schnittstellen kontrolliert, die immer nur Teilbereiche des ElectionGuard Java Projektes abdecken, womit kein Bereich die volle Kontrolle über die Funktionen des ElectionGuard Java Projekts erhält.

4.8.1 Entschlüsselung

Das *IDecryption*-Interface kapselt die Funktionen, die in den Phasen der Entschlüsselung einer Wahl benötigt werden. Diese Funktion wird ausschließlich von den StateStrategies genutzt, die sich um die Phasen der Entschlüsselung einer Wahl und deren Initialisierung kümmern.

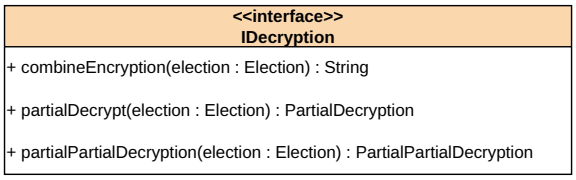

Abbildung 30: IDecryption Interface

4.8.2 KeyCeremony

Das *IKeyCeremony*-Interface kapselt die Funktionen, die in den Phasen der KeyCeremony Das *IKeyCeremony*-Interface kapselt die Funktionen, die in den Phasen der KeyCeremony einer Wahl benötigt werden. Es bietet die Möglichkeit die KeyCeremony zu Initialisieren sowie eine System Bot hinzuzufügen. Diese Funktionen werden ausschließlich von den StateStrategies angesprochen, die sich um die KeyCeremony kümmern.

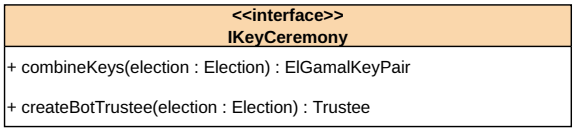

Abbildung 31: IKeyCeremony Interface

4.8.3 Hash

Das *Hashable*-Interface bietet die Möglichkeit, eine unbestimmte Anzahl an Argumenten zu hashen und wird verwendet, um den Fingerprint einer Wahl und den Trackingcode eines Ballots zu generieren.

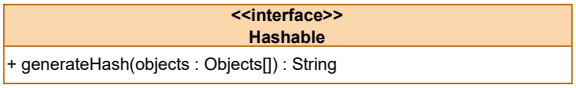

Abbildung 32: Hashable Interface

4.8.4 Verifikation

Die *IVerifiable*-Interface bietet die Möglichkeit, bestimmte Elemente einer Wahl zu verifizieren. Dazu gehören die Schlüssel eines Trustees, die Abgabe einer Stimme und deren Beweise und die Entschlüsselungen der Trustees.

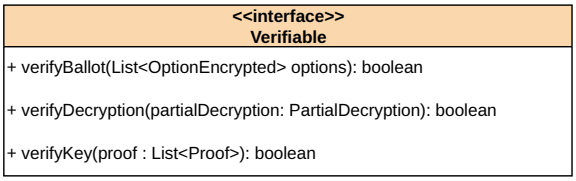

Abbildung 33: Verifiable Interface

4.8.5 Beispiel: Erstellung eines Ballots

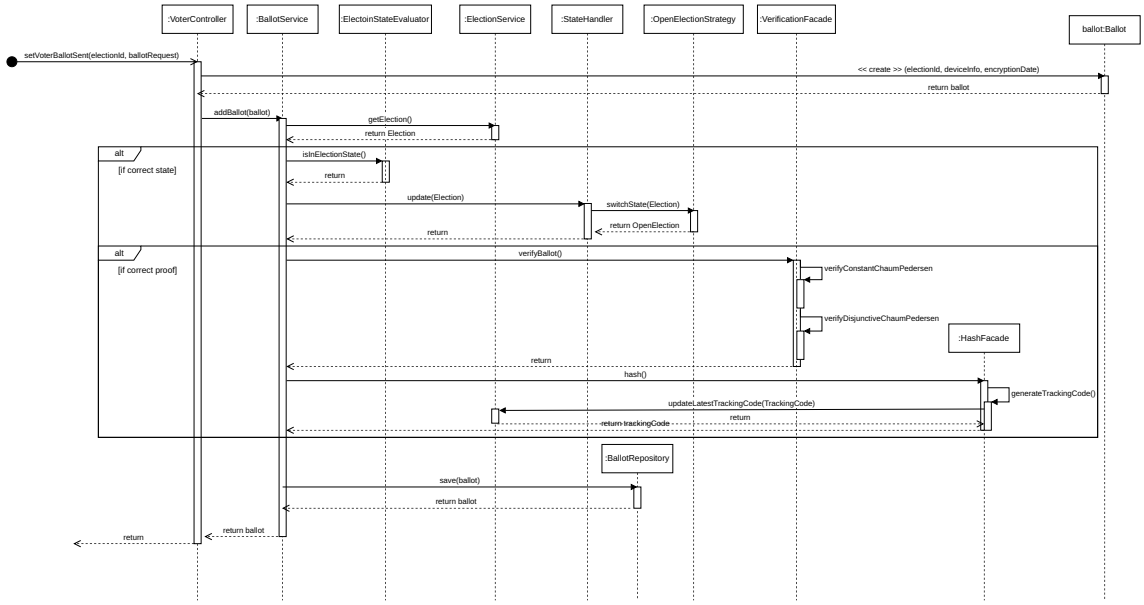

Abbildung 34: Sequenzdiagramm der Erstellung eines Ballots

In diesem Sequenzdiagramm wird der Ablauf des verarbeiten eines verschlüsselten Ballots dargestellt, der in der Datenbank gespeichert wird. Dabei ist der erste Schritt zu überprüfen, ob die Wahl noch im Zustand *ElectionState.OPEN* ist oder ob das Enddatum der Wahl bereits erreicht wurde. Der zweite Punkt ist die Verifizierung des Ballots, wobei die ChaumPedersen Beweise verifiziert werden. Der letzte Vorgang ist das Generieren des TrackingCodes. Die Methode *generateTrackingCode()*, die diesen Vorgang ausführt, ist synchronized. In ihr wird zuerst eine neuer TrackingCode für die HashChain erstellt. Außerdem wird atomar in der gleichen Methode die Datenbank um diesen TrackingCode geupdated, um sicherzustellen, das jede neue Generierung eines TrackingCodes immer mit dem aktuellsten TrackingCode arbeitet, um eine Race Condition auszuschließen. Abschließend wird das verifizierte Ballot in der Datenbank gespeichert und der Client, der das Ballot abgegeben hat, erhält seinen TrackingCode vom System.

4.9 ElectionService

Der ElectionService ist ein Service, welcher in direktem Kontakt mit der Datenbank steht. Außerdem ist dieser Service die einzige Schnittstelle im Backend, um mit der Datenbank, bezüglich der Election-Entitäten, zu kommunizieren. Somit wird der ElectionService auch von mehreren anderen Services angesprochen, wie zum Beispiel hier und hier zu sehen ist.

Dementsprechend wird hier die Speicher Logik, also z.B ob eine neue Wahl auch wirklich berechtigt ist in die Datenbank aufgenommen zu werden, gekapselt. Des Weiteren kümmert sich der ElectionService nicht nur um die korrekte Speicherung von Wahlen, sondern auch um die korrekte Rückgabe von Wahlen, oder Bestandteilen von Wahlen. Diese werden nämlich entweder zu anderen Services zur Verarbeitung, oder zum Controller zum Versenden ans Frontend, gereicht.

Der ElectionRepository und der ContestRepository werden nur durch den ElectionService angesprochen, denn Contest-Entitäten und Election-Entitäten werden nur in einem gemeinsamen Kontext benötigt.

4.9.1 Beispiel: Erstellung einer Wahl

Dieses Sequenzdiagramm zeigt den Verlauf zur Speicherung einer neuen Wahl. Dabei kommt über den Endpunkt */api/authority/elections/create* die Anfrage über die Erstellung einer Wahl. Der ElectionService kapselt in der *intialSave(Election election)*-Methode die Überprüfung, ob die zu erstellende Wahl berechtigt ist in die Datenbank aufgenommen zu werden. Dabei wird lediglich geprüft, ob es in der Datenbank nicht bereits eine Wahl existiert mit gleichem Titel, da Dieser laut Pflichtenheft einzigartig sein muss.

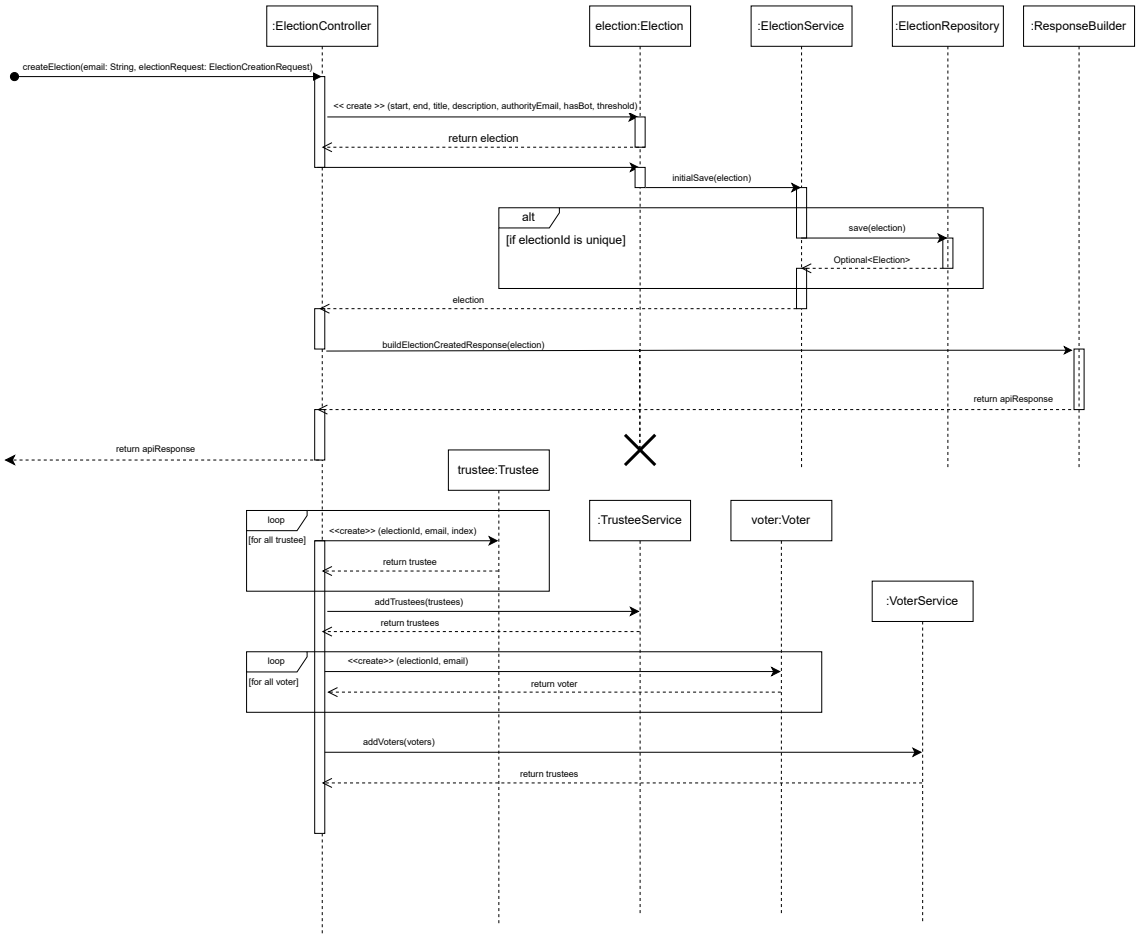

Abbildung 35: Ablauf bei Erstellung einer Wahl

4.10 VoterService

Der VoterService ist ein Service, welcher das VoterRepository direkt anspricht und somit in direktem Kontakt mit der Datenbank ist. Folgende Verantwortungen besitzt der VoterService:

- Korrekte Hinzufügen von Voter-Entitäten im Kontext der Wahlerstellung
- Korrektes Laden von Voter-Entitäten

Dies kann im Kontext einer Wahl durch die Angabe der *electionId* oder durch die Angabe der E-Mail eines Wählers geschehen.

4.11 TrusteeService

Der TrusteeService bietet die Möglichkeit, Trustees und deren Daten, wie zum Beispiel die Public Elgamal Keys und die Election Partial Key Backups, der Datenbank korrekt hinzuzufügen. Außerdem stößt er auch die Verifikation dieser Daten an. So spricht er das *Verifiable*-Interface an um die ElGamal Keys zu verifizieren.

4.12 AuthorityService

Der AuthorityService gewährleistet die Datenkonsistenz in der Datenbank bezüglich der Wahlleiter. Dies nimmt eine wichtige Rolle im System ein, da die korrekte Funktionalität des Systems von dieser Datenkonsistenz abhängt.

Dabei wird der AuthorityService nur vom ConfigService angesprochen. Genauer wird durch die *onModified()*-Methode in der *AuthorityListConfigStrategy* ein Impuls an den AuthorityService gesendet, damit dieser alle Einträge in der Datenbank bezüglich der Wahlleiter mit den neuen Wahlleitern aus der Konfigurationsdatei abgleichen und aktualisieren kann. Die Konfigurationsdatei wird dabie vom Host festglegt und kann auch nur von ihm verändert werden.

4.13 BallotService

Der BallotService bietet die Möglichkeit, Ballot-Entitäten, sowie auch die dazugehörigen OptionEncrypted-Entitäten der Datenbank korrekt hinzuzufügen. Allerdings werden die OptionEncrypted-Entitäten über das *Verifiable*-Interface zuerst auf Korrektheit geprüft. Man beachte, dass das korrekte Hinzufügen von Ballot-Entitäten einen gewissen Prozess mit mehreren weiteren Services startet, siehe dazu das Sequenzdiagramm.

Weiter kapselt das BallotService die Funktionalität die Spoiled Ballots in Submitted Ballots um zu wandeln. Dafür wird einfach *isSubmitted* in der Ballot-Entität auf true gesetzt.

4.14 TallyService

Der TallyService ist für die korrekte Speicherung und korrekte Verwaltung der verschlüsselten zusammengezählten Optionen aller Ballots zu einer Wahl zuständig.

Die Option eines Contest wird über die *ElectionGuard Fassade* nach Abschluss einer Wahl homomorph verschlüsselt und im direkten Anschluss zusammengezählt, wodurch eine Tally-Entität zu dieser Option entsteht. Diese Tally-Entität wird durch das TallyRepository in die Datenbank gespeichert.

Weiter spielt der TallyService bei der Entschlüsselung des Wahlergebnisses ein Rolle, denn über den TallyService werden die vorher zusammengezählten Optionen an die Trustees, zur Entschlüsselung, übermittelt.

4.15 DecryptionService

Der DecryptionService ist für die korrekte Speicherung und korrekte Verwaltung von Teil- und Teil-Teil-Entschlüsselungen zuständig. Der TrusteeController wird bei einer hochgeladenen Teil-Entschlüsselung diese in eine PartialDecryption-Entität übersetzen und diese dem DecryptionService übergeben.

Dieser überprüft erst einmal, ob die Wahl den korrekten Zustand zur Entschlüsselung hat. Ist die Wahl im richtigen Zustand, so prüft der Service die PartialDecryption-Entitäten auf Korrektheit, indem das Verifiable-Interface des Pakets *ElectionGuard* anspricht. Bei korrekter Teil-Entschlüsselung wird diese dem PartialDecryptionRepository zur Speicherung übergeben. Analog ist das Vorgehen zu Teil-Teil-Entschlüsselungen.

4.16 ConfigService

Dieser Service ist keinem Repository zugeordnet und ist nach dem Entwurfsmuster Strategy konzipiert, damit weitere zukünftige Konfigurationen einfach und bequem zu erweitern sind, in dem eine neue Strategy-Klasse angelegt wird, wobei die *onModified()*-Methode neu definiert wird. Der Service kann nur durch den *ConfigHandler*, welcher dem Entwurfsmuster Facade entspricht, angesprochen werden.

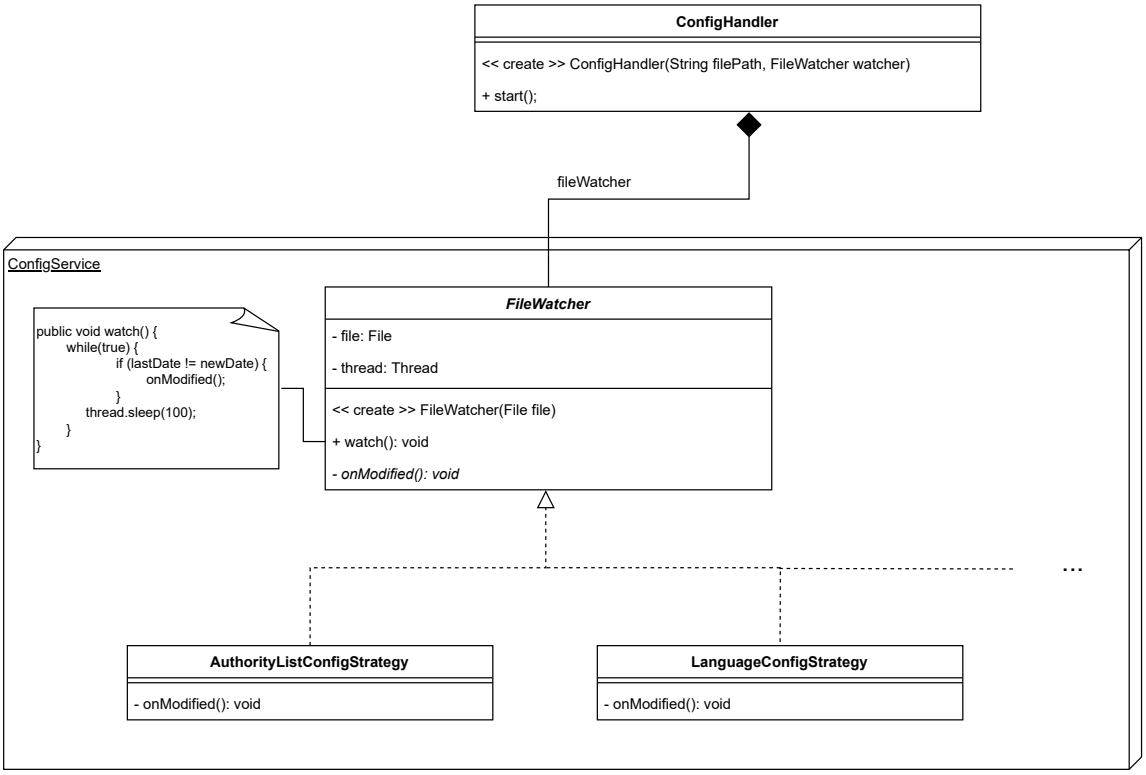

Abbildung 36: Service, welcher sich um die Konfiguration des Systems kümmert

Eines der Kernaufgaben des Hosts des Systems ist es, die Wahlleiter festzulegen. Wahlleiter haben die Berechtigung Wahlen zu erstellen, zu konfigurieren und zu verwalten. Für weitere Informationen zum Aufgabenbereich von Wahlleitern, siehe Pflichtenheft. Da eine korrekte Datenkonsistenz in der Datenbank eine entscheidende Rolle für die Funktionalität des Systems spielt, so gibt es einen *FileWatcher*, welcher dafür zuständig ist, Änderungen der Konfigurationsdatei festzustellen, damit neue Wahlleiter in die Datenbank aufgenommen werden.

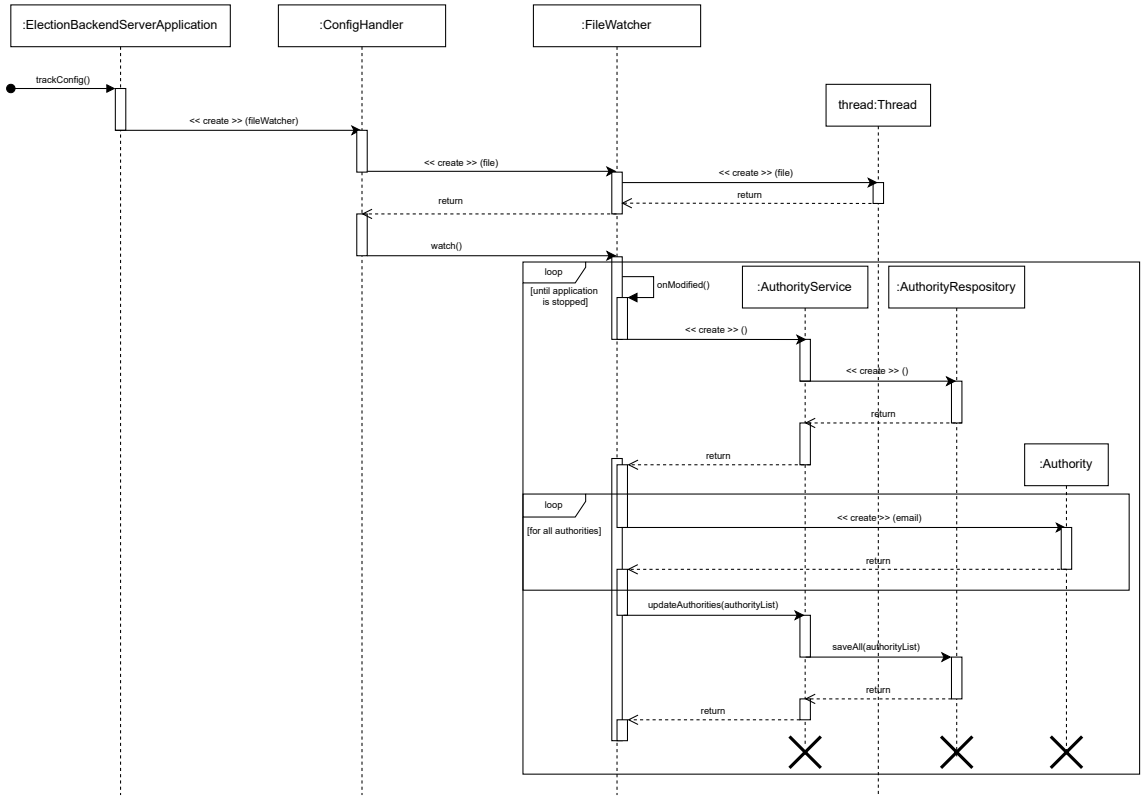

Abbildung 37: Prozess zur Aktualisierung der Authority

4.17 Repository

Mittels Spring Data JPA gibt es bereits eine Hierarchie von Interfaces, welche die wichtigsten und üblichsten Funktionalitäten zu Datenbankzugriffen bereitstellen. Dabei nutzen wir das CrudRepository, wobei Dieses die Operationen CREATE, READ, UPDATE und DELETE *(kz. CRUD)* bereitstellt.

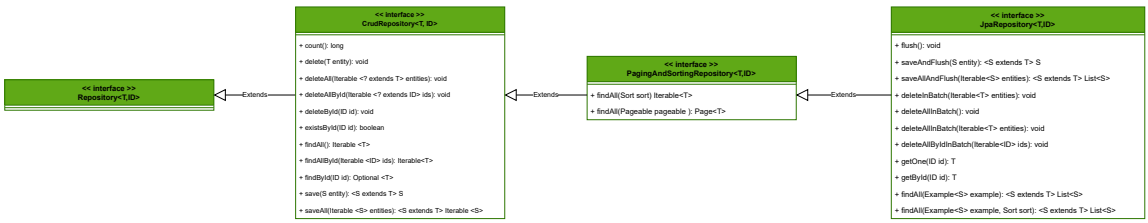

Abbildung 38: Hierarchie der, mit Spring Boot kommenden, Interfaces

Repositories werden durch die Spring Boot Annotation *@Repository* gekennzeichnet und in den Services, die die Repositories instanziieren, werden die Repository-Instanzen mit der Spring Boot Annotation *@autowired* markiert. Durch Spring Data JPA wird zur Laufzeit eine Implementierung der Repositories automatisch generiert. Dadurch wird dem Entwickler das Implementieren von Repositories erspart.

Weiter lässt sich mit Spring Data JPA, Datenbankanfragen vom Methodennamen ableiten. Dafür gibt es eine von Spring Boot festgelegte Liste an Schlüsselbegriffen, siehe dazu `https://docs.spring.io/spring-data/jpa/docs/1.3.0.RELEASE/reference/html/jpa.repositories.html` (unter Table 2.2).

Im folgender Abbildung erkenne man, wie diese Funktionalität der Datenbankanfragen zu unserem Vorteil ausgenutzt werden.

Für detailliertere Informationen bezüglich der vereinzelten Repositories, siehe Javadoc.

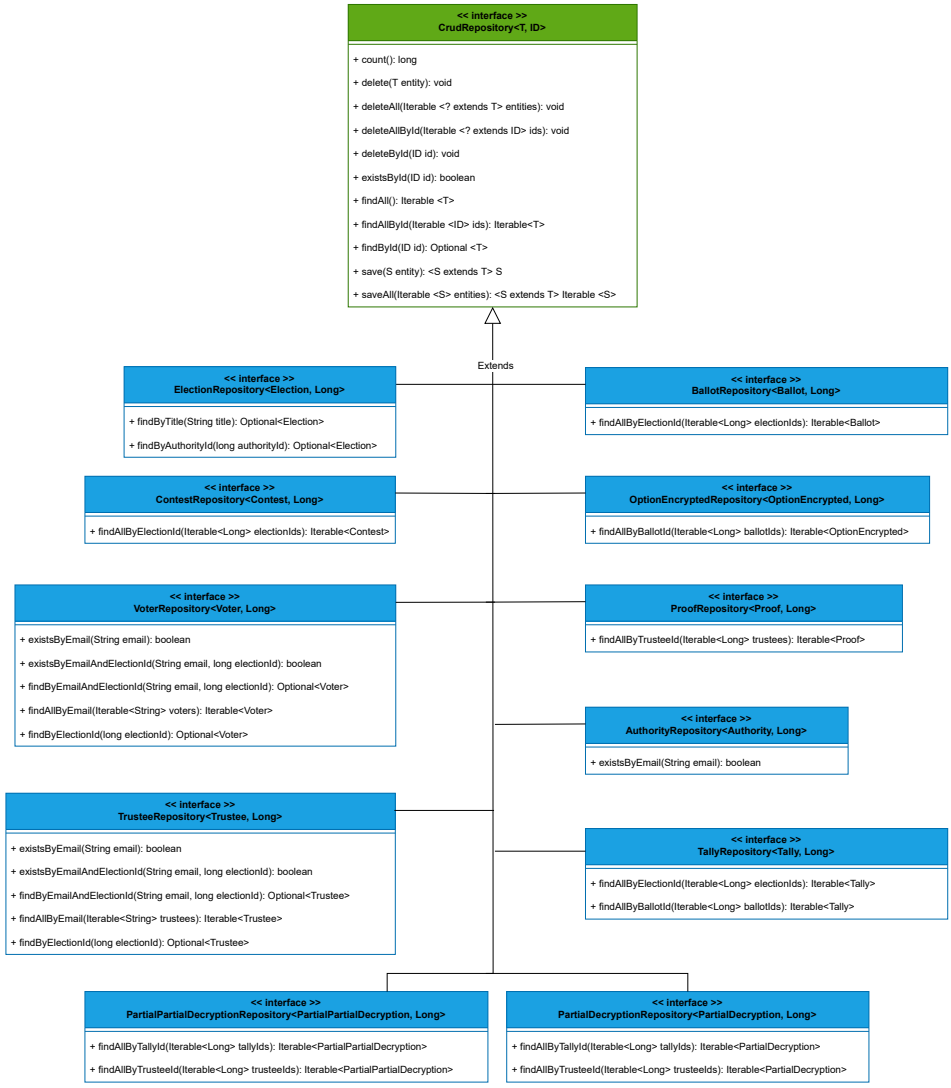

Abbildung 39: Repositories

4.18 Kommunikation mit der Datenbank

4.18.1 Datenbank

Als Datenbanksystem wird MySQL verwendet, welches eines der weltweit verbreitesten relationalen Datenbankverwaltungssysteme ist. Des Weiteren liegen die Stärken bei MySQL auf der Schnelligkeit, Zuverlässigkeit, Skalierbarkeit und schließlich der einfachen Benutzung.

Mehr zu MySQL: `https://de.wikipedia.org/wiki/MySQL`

4.18.2 Entity

Dank Spring Data JPA, welches nur einen Teil der großen Spring Data Familie ausmacht, können die Relationen der Datenbank auf sogenannte Entity *(dt. Entitäten)* eindeutig zugeordnet werden. Entitäten sind POJO-Klassen, welche mit der Spring Boot Annotation *@Entity* gekennzeichnet sind. Dadurch wird erreicht, dass außerhalb der Datenbank nur mit POJOs gearbeitet wird und somit eine vollständige objektorientierte Programmstruktur gewährleistet werden kann.

Mehr zu Entitäten: `https://docs.spring.io/spring-batch/docs/current/api/org/springframework/batch/core/Entity.html`

Unterstrichene Attribute in den Entitäten repräsentieren den Primärschlüssel der jeweiligen Relation. Weiter erkennt man an den gerichteten Assoziationen die Fremdschlüssel der jeweiligen Relation. Mittels Primär- und Fremdschlüssel können Relationen in Relation gesetzt werden um aus Beiden eindeutig zusammenhängende Informationen zu gewinnen. Für nähere Informationen bezüglich der einzelnen Entitäten, siehe im Javadoc nach.

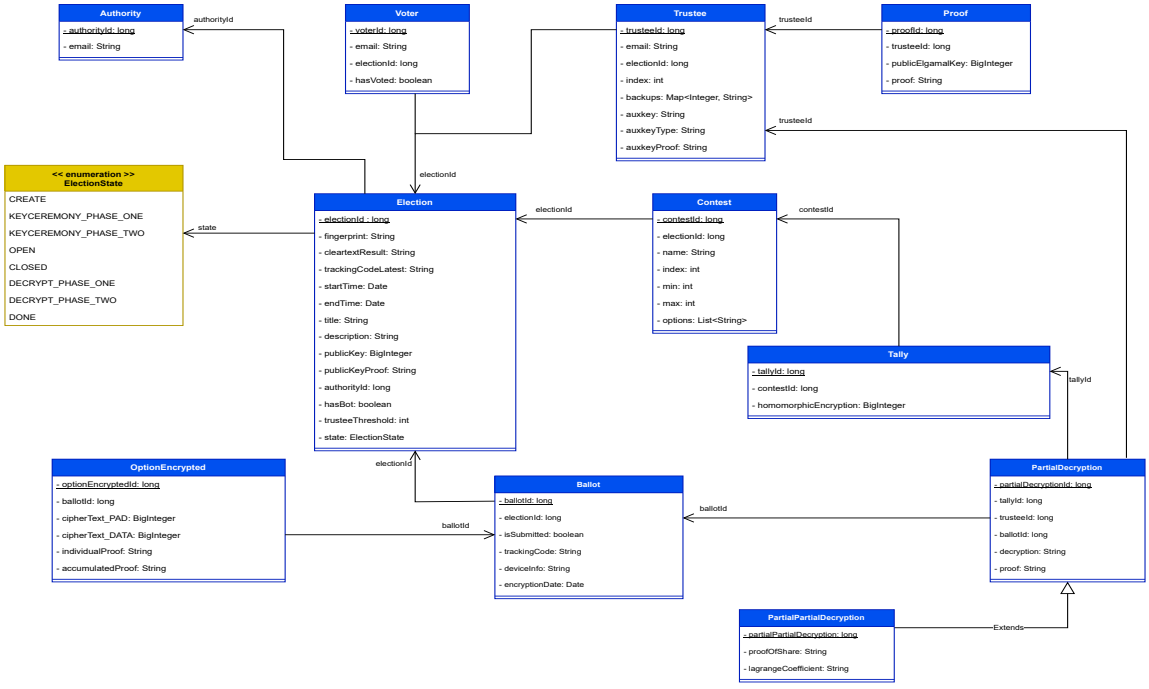

Abbildung 40: Aufbau der Entitäten als POJO-Klassen

4.18.3 Entity-Relationship-Model

Dieses Modell verdeutlicht die logischen Beziehungen zwischen einzelnen Entitäten und zeigt ebenfalls die Kardinalitäten der Beziehungen, welches aus der obigen Abbildung nicht ablesbar wäre.

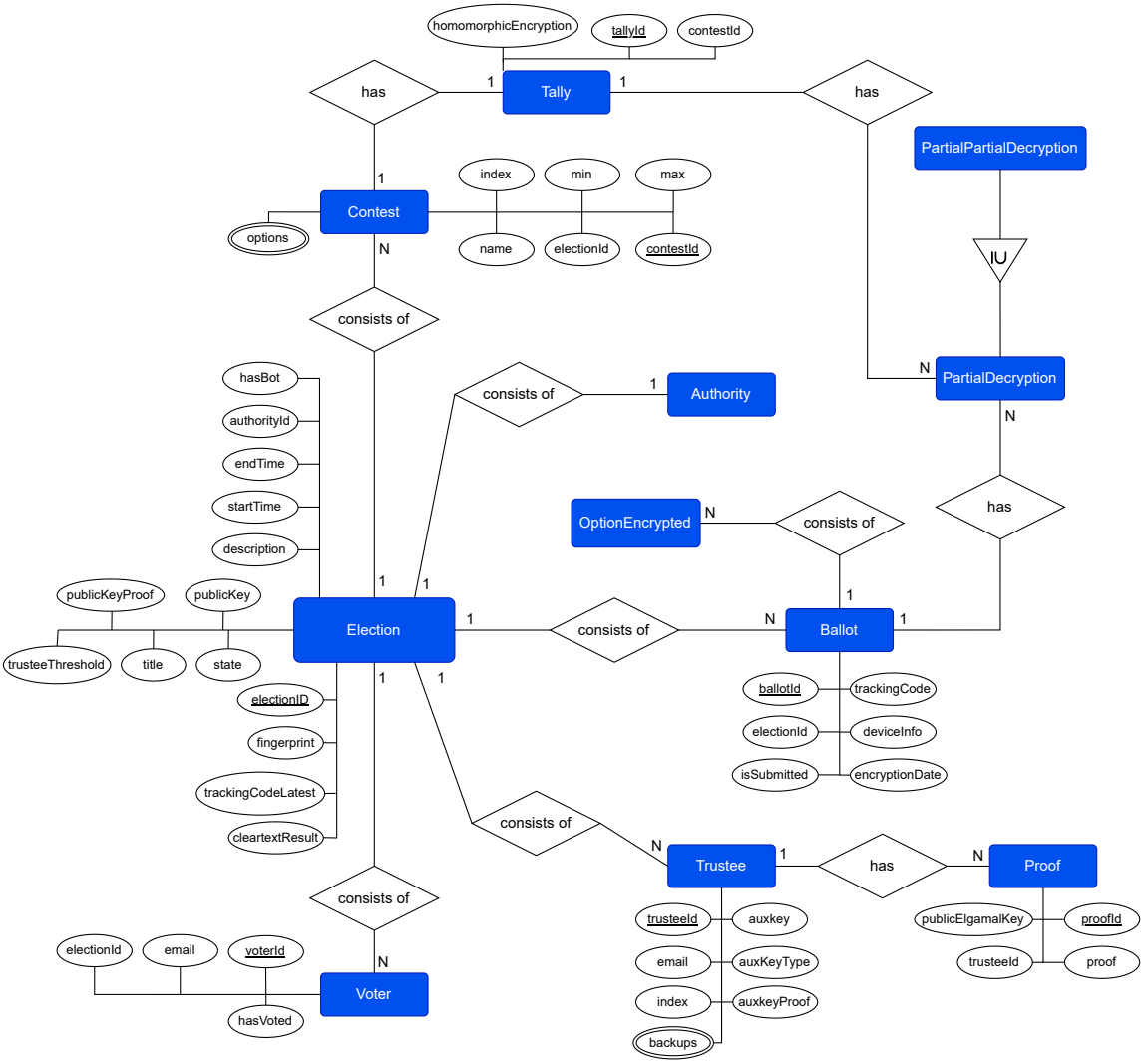

Abbildung 41: Entity-Relationship-Model der Datenbank

5 Änderungen am Pflichtenheft

Berechnung des Election Fingerprint

Der Election Fingerprint wird nun als SSH 256 Hash über das vollständige `ElectionManifest` (siehe Model.Election im Abschnitt Frontend oder die entsprechenden TypeDocs). Damit benötigt man zum Berechnen des Fingerprints zusätzlich zu den im Pflichtenheft genannten Informationen auch die Beschreibung, den Startzeitpunkt und den Threshold der Wahl sowie die liste der Wähler. Der Grund dafür ist, dass über den Fingerprint alle Parameter einer Wahl gesichert werden sollen.