Hochschule **RheinMain**
Fachbereich Design Informatik Medien
Studiengang Medieninformatik

Dokumentation

Web Technologien Sommer 2017

Vacation Station

Vorgelegt von	Christian Kalig, Nadine Kreisel
am	13. Juli 2017
Referent	Prof. Dr. Schaible

Inhaltsverzeichnis

1 Einführung

Vacation Station bietet dem Nutzer die Möglichkeit seine Reisegruppen und damit verbundene finanzielle Ausgaben zu verwalten. Die grundlegenden Funktionen sind hierbei:

1. Gruppe erstellen und anzeigen
2. Benutzer anlegen
3. Benutzer hinzufügen
4. Rechnung hinzufügen

Die Sourcen zur Applikation sind unter der URL `https://github.com/VacationStation` zu finden. Um die Anbindung der einzelnen Services zu konfigurieren, müssen diese in der Datei `Config.ts` im Ordner conf im Mainbackend eingetragen sein.

2 Aufbau

Die Vacation Station Applikation besteht grundlegend aus dem Frontend 2.1 auf Client-Seite und verschiedenen Microservices auf Server-Seite. Als Schnittstelle zum Client dient das Mainbackend 2.2. Aus ihm heraus werden die unterschiedlichen Services per REST angesprochen. Die gebündelte Schnittstelle im Mainbackend ermöglicht das leichte Austauschen des Frontends. In der Schnittstellen-Dokumentation 5 wird die Funktionalität der REST-API im Detail festgehalten.

2.1 Frontend

Das Frontend ist in Angular2 und Typescript umgesetzt. Die Projektstruktur ergibt sich aus der Trennung der Businesslogik und der für die View verantwortlichen Komponenten. Im Ordner `_services` befinden sich, inhaltlich getrennt (vgl. 2.2) die unterschiedlichen Services. Jeder dieser Services realisiert REST-Calls an die definierte Schnittstelle im Mainbackend. Um auf die aus den einzelnen Services erhaltenen Daten zuzugreifen, werden diese mittels Dependency Injection in die Components eingebunden. Die einzelnen Views werden in separaten HTML-Dateien an das jeweilige Component gebunden und nicht direkt inline, im Component erzeugt. So ergibt sich für jede Komponente ein Set aus HTML-Template und Komponenten-Logik. Im Ordner `_guards` befindet sich der Authentication Guard, der jede Frontend Route sichert. Dazu überprüft er das Ablaufdatum des im Localstorage befindlichen Tokens. Ist das Token abgelaufen, wird die Anfrage auf die Login Seite weitergeleitet.

2.2 Mainbackend

Das Mainbackend dient als Hauptanlaufpunkt und Schnittstelle für den Client und ist als Express.js Applikation mit einer Anbindung an eine MongoDb umgesetzt. Auch hier ergibt sich die Projektstruktur aus den verschiedenen Zuständigkeiten. Im Ordner `router` werden Router definiert. Sie bilden, je nach Aufgabengebiet, die unterschiedlichen Routen der Applikation ab. Die Anfragen werden dann innerhalb der Route an den zuständigen Controller , der sich im Ordner `controller` befindet, delegiert. Diese übernehmen die Business Logik, sowie Kommunikation mit weiteren Service Schnittstellen oder der Datenbank. Um die Integrität der Daten zu gewährleisten werden Interfaces für das jeweilige Datenmodell (vgl. 3) eingebunden.

2.3 Image Service

Der Image Service ist mit Express.js realisiert und ermöglicht das Speichern von Bilddateien in einer MySql Datenbank. Hierzu wird eine REST Schnittstelle angeboten die ein Bild als Base64 kodierten String entgegen nimmt und es als BLOB (Binary Large Object) in der Datenbank speichert. Darüber hinaus bietet die Schnittstelle auch die Möglichkeit ein Bild anhand seiner ID als Base64 kodierten String zurück zu bekommen.

2.4 Authentication Service

Der Authentication Service basiert auf einem Express.js Projekt mit der Anbindung an eine MySql Datenbank. Hier werden authorisierte Nutzer verwaltet und neue Nutzer angelegt. Des weiteren ist der Authentication Service Hauptbestandteil des Sicherheitsmodells und verifiziert die im Mainbackend ankommenden Anfragen (vgl. 4).

3 Datenmodell

Aus der bisher erläuterten Architektur und der Tatsache, dass die Reisegruppe an sich das Herzstück der Applikation bildet, ergibt sich das folgende Datenmodell. Hierbei sind alle relevanten Informationen über Buchungen und Gruppenmitglieder in der Gruppe selbst gespeichert (vgl. Abb. 1, Listing 1). Die daraus resultierenden Saldo für eine Gruppe oder einzelne Mitglieder werden ausschließlich in der Gruppe verwaltet und bei Bedarf anhand der einzelnen Buchungen berechnet.

```
export const GroupSchema: Schema = new Schema({
    createdAt: Date,
    name: String,
    description: String,
    destination: String,
    dateFrom: Date,
    dateTo: Date,
    owner: UserSchema,
    member: [UserSchema],
    bookings: [BookingSchema],
    invite: String
});
```

(a) GroupSchema

```
export const BookingSchema: Schema = new Schema({
    usage: String,
    biller: UserSchema,
    forAll: Boolean,
    recipients: [UserSchema],
    bill: BillSchema,
    type: {type: EBookingType},
    amount: Number, // in cents
    date: Date,
    active: Boolean
});
```

(b) BookingSchema

Abbildung 1: Eingebettete Schemata

```
{"group": {
      "creationDate": "",
      "name": "TestGruppe",
      "description": "Test Beschreibung",
      "destination": "Testgebiet",
      "dateFrom": "",
      "dateTo": "",
      "owner": {...
      },
      "member": [
            { "loginName": "user1",
              "firstName": "Kai",
              "lastName": "Ahnung",
              "mail": "kai@ahnung.de"
              },
            {..} ],
      "bookings": [
            { "usage": "Getraenke",
              "biller": {
                        "loginName": "user1",
                        "firstName": "Kai",
                        "lastName": "Ahnung",
                        "mail": "kai@ahnung.de"
                        },
              "forAll": "true",
              "recipients": [],
              "type": "BILL",
              "amount": "5678",
              "date": ""
              },
            {..} ],
      "invite" :[
            "AG52I8F",
            "59OSM40"
      ]
 }
}
```

Listing 1: Daten einer Gruppe im JSON Format

4 Sicherheit

Das Sicherheitsmodell sieht vor, dass nur autorisierte Calls ans Mainbackend erfolgen können. Die zentrale Aufgabe übernimmt der Authentication Service, kurz AS. Loggt sich ein bereits registrierter Nutzer ein, erhält er vom AS ein Token als Bestätigung der erfolgreichen Anmeldung. Bei jedem Call wird das Token in einem Authorization-Header mit gesendet. Bevor die Calls im Mainbackend ankommen, überprüfen AS und eine Middleware die Daten. Zuerst entnimmt der AS das im Header gesendete Token und überprüft es mit der `verify` Methode. Da es sich um ein JSON-Webtoken, kurz JWT handelt, kann er anhand der UserId entscheiden ob es sich wirklich um die zum Token zugehörigen UserId handelt. Dazu wird das JWT dekodiert. Im dekodierten Datenteil des JWT ist die ursprüngliche UserId enthalten. Stimmt diese mit der übergebenen UserId überein, ist die UserId verifiziert. Die extrahierte UserId wird nun an eine Middleware weitergeleitet. Diese versucht den User mittels der extrahierten ID aus der Datenbank zu laden. Erst wenn sichergestellt ist, dass der User existiert , wird die Anfrage an den zuständigen Router im Mainbackend geleitet (vgl. Abb. 7).

5 API Dokumentation

Die Schnittstelle zum Backend ist als REST API umgesetzt. Da sie so konzipiert ist, dass das Frontend auch ausgetauscht werden kann oder andere Applikationen daran angebunden werden können, bildet die API Dokumentation einen wichtigen Bestandteil. Zum Erstellen und Generieren der Dokumentation wird das Tool apidoc.js (`http://apidocjs.com`) verwendet. Hierzu wird eine Konfigurationsdatei `apidoc.json` benötigt, die im Ordner `routes` im Mainbackend zu finden ist. Dort können Meta-Daten der Dokumentation konfiguriert werden wie z.B. der Name, die Endpunkt-URL oder auch eine aktuelle Version. Alle Dokumente im Ordner `routes` können nun die von Apidoc.js vorgesehene Notation verwenden um daraus eine Dokumentation zu generieren.

Dazu werden im Code selbst Kommentare geschrieben die mit der Annotation `@api` beginnen. Das Tool bietet neben der Methode, URL, Übergabeparametern und Rückgabewerten eine Vielzahl an anderen nützlichen Befehlen an. Um dem Nutzer der API eine unkomplizierte Anbindung zu ermöglichen, ist es sinnvoll auch das Verhalten im Fehlerfall zu dokumentieren. Der Befehl `@apiError` erlaubt es Fehlerfälle genau zu beschreiben. Mit der Option `(group)` kann man die Fehlerfälle sinnvoll gruppieren und eigene Fehlertypen benennen die sich aus dem Errorhandling der Applikation ergeben (vgl. Abb. 2, Abb. 3). Um fehlerhafte Übergabeparameter zu vermeiden, kann man bei größeren Datenstrukturen ein Beispiel der erwarteten Parameter auch im JSON Format angeben.

Die Generierung der Dokumentation erfolgt über die Kommandozeile mit:

```
> apidoc -i /inputFolder -o /outputFolder -t /myTemplate
```

und kann so bequem in das Start Skript mit aufgenommen werden. Nach dem Start der Applikation ist die gesamte Dokumentation unter der Route `/api/doc` zu finden.

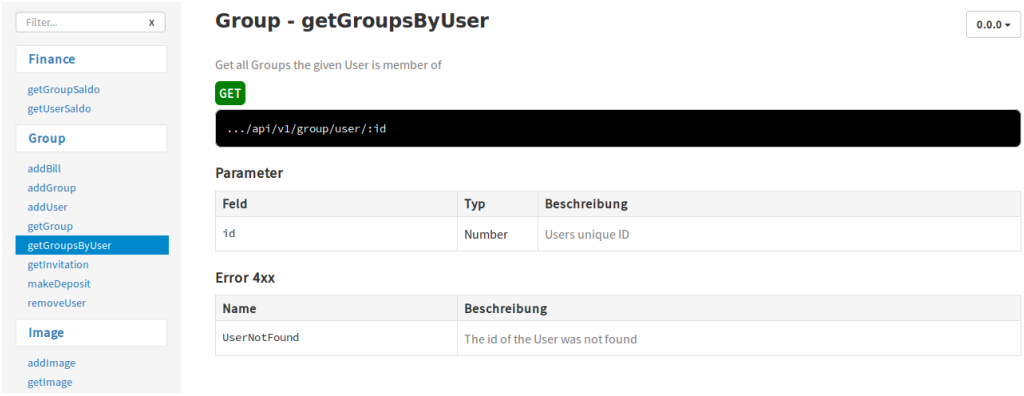

Abbildung 2: ApiDoc mit Fehlerfall UserNotFound

```
/**
 * @api {get} /group/user/:id getGroupsByUser
 * @apiName getGroupsByUser
 * @apiGroup Group
 * @apiDescription Get all Groups the given User is member of
 * @apiParam {Number} id Users unique ID
 *
 * @apiError UserNotFound The id of the User was not found
 *
 */
/**
 * get groups where user:id is member
 * @param req
 * @param res
 * @param next
 */
private getGroupByUser(req: Request, res: Response, next: NextFunction){
    let userId = req.params.id;
    GroupController.findGroupsWithUserId(userId).then(function(result){
        return res.status(200).json({success: true, groups: result});
    }).catch(function(err){
        return res.status(500).json({success: false, error: err});
    });
}
```

Abbildung 3: ApiDoc Annotation im Code

6 UML Diagramme

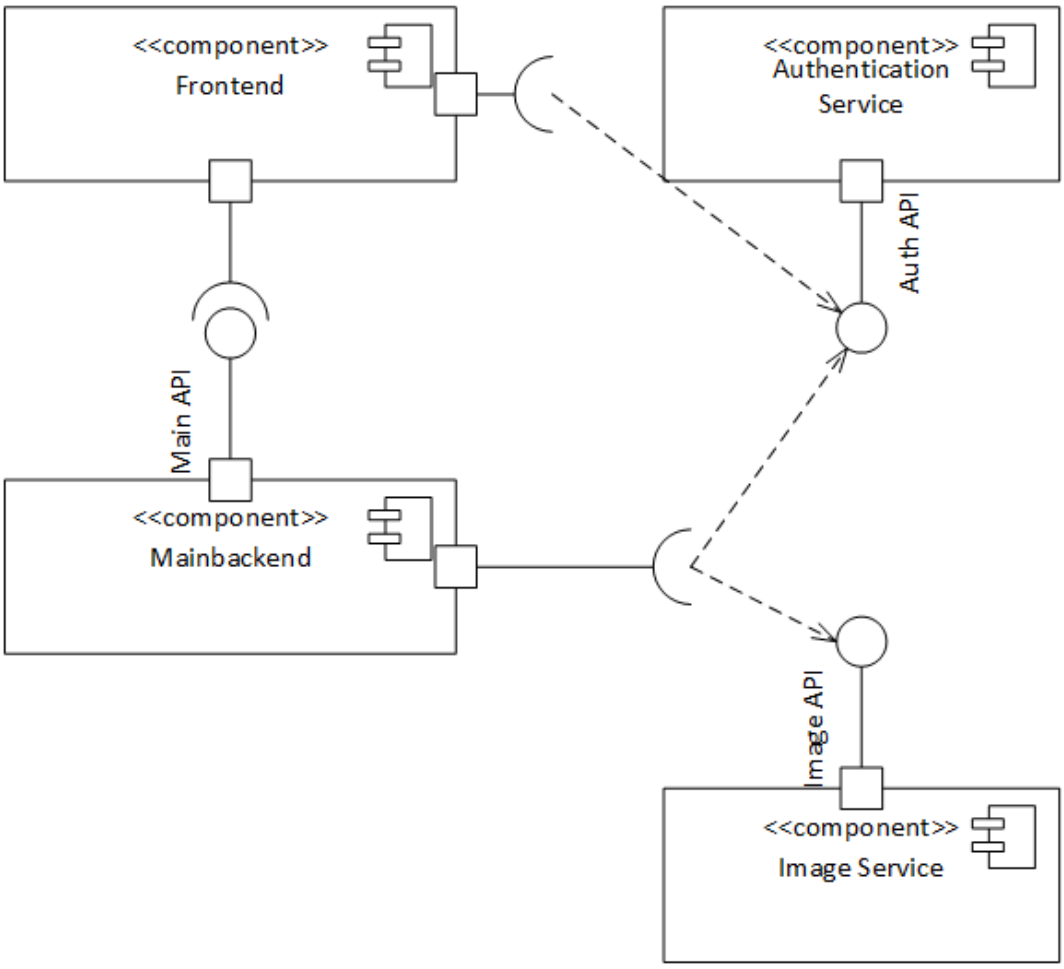

Abbildung 4: Kommunikation der einzelnen Komponenten

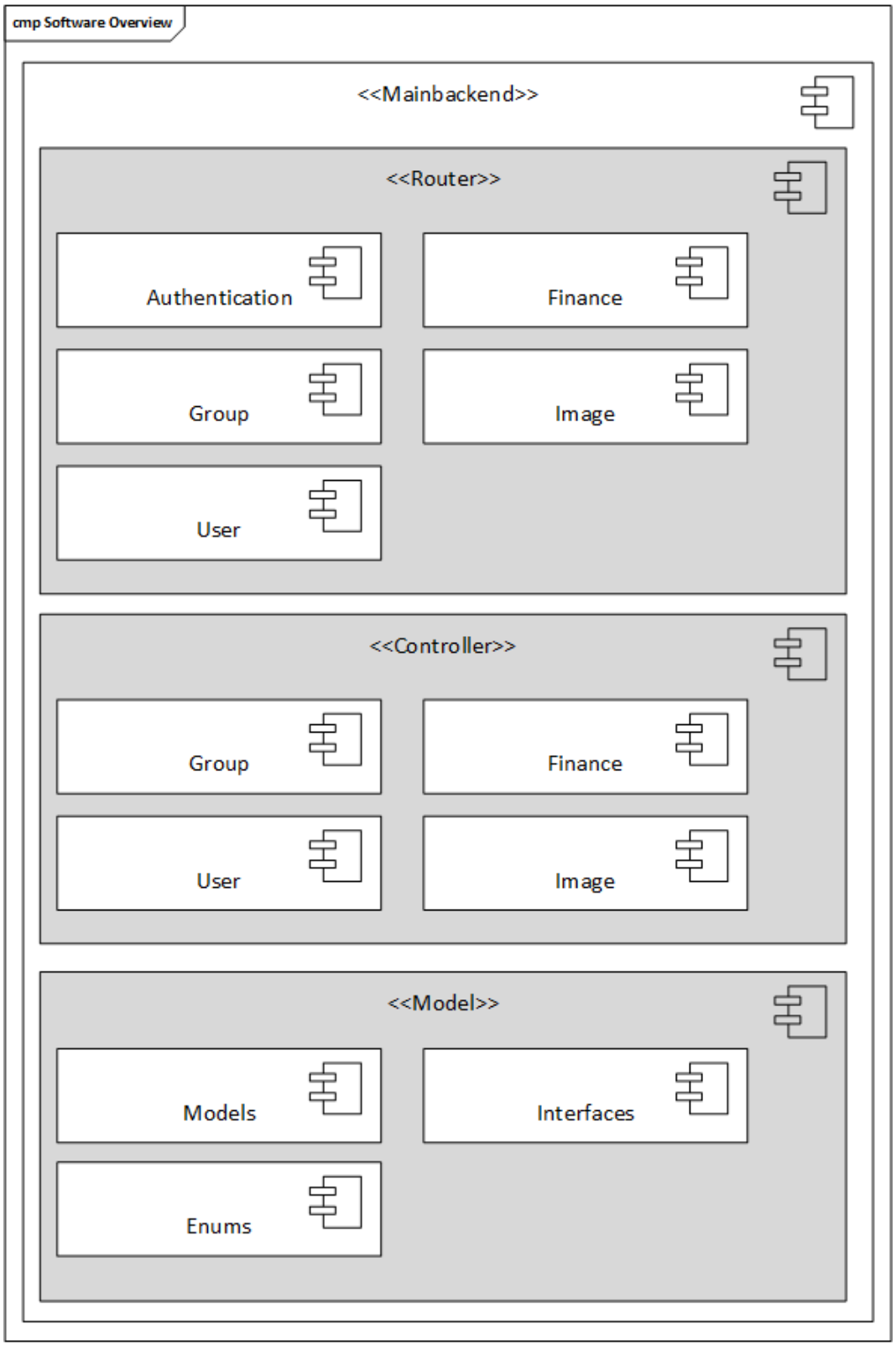

Abbildung 5: Komponenten Übersicht Mainbackend

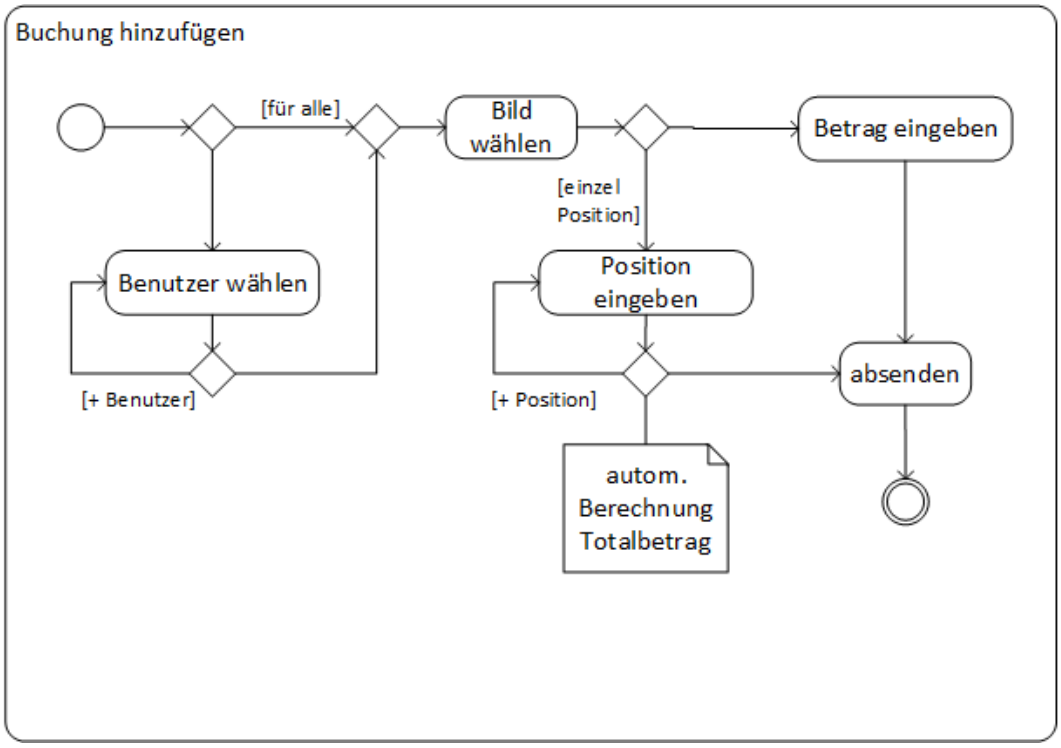

Abbildung 6: Ablauf Buchung hinzufügen

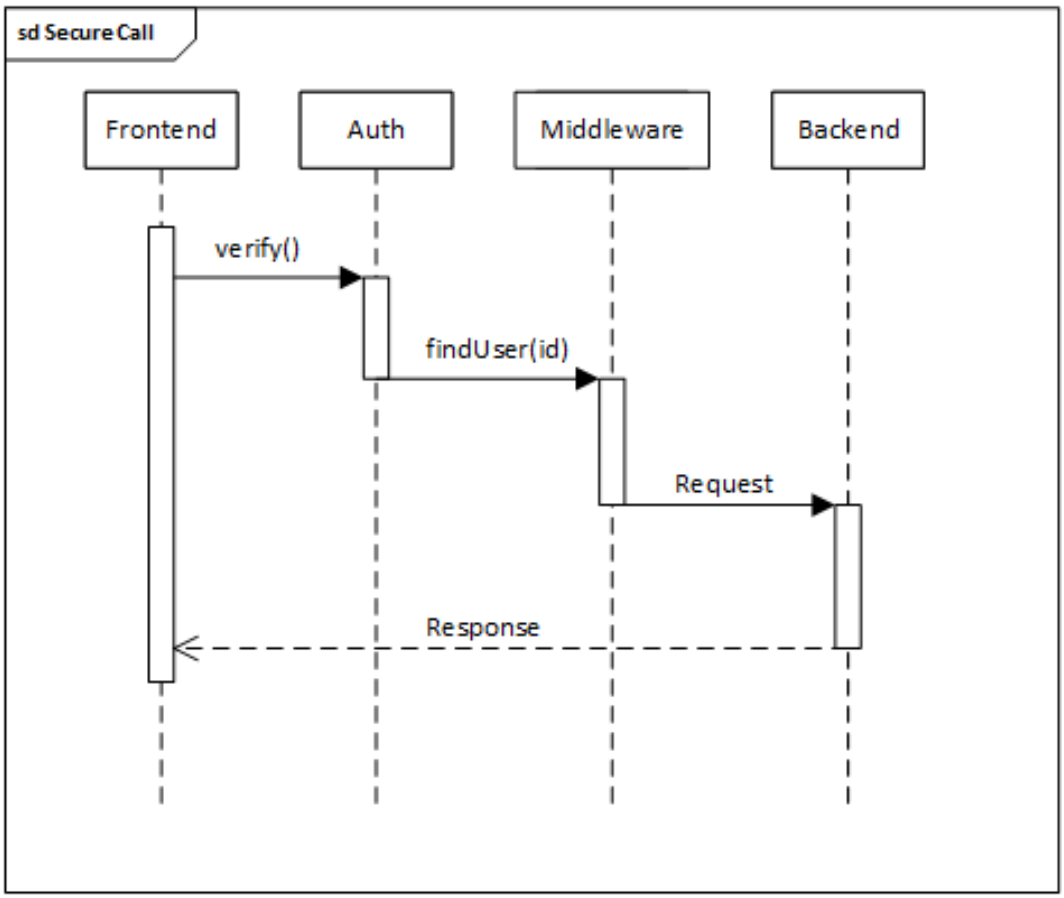

Abbildung 7: Sequenzdiagramm eines gesicherten Calls an das Mainbackend