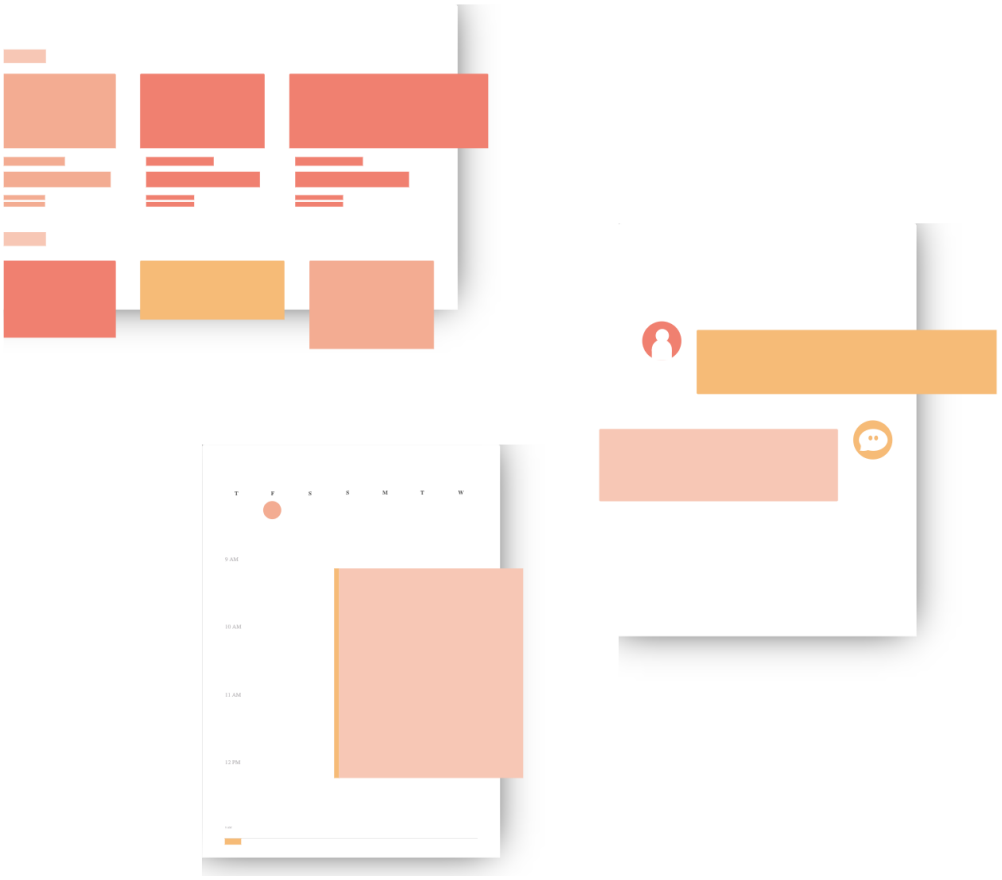
Eventing

Projektdokumentation



Rafail Antoniadis (ra022)	Matr.-Nr. 32982
Nathanael Paiement (np035)	Matr.-Nr. 33528
Prof. Walter Kriha	Betreuer

WS 2019/20

Allgemein

Eventing

Bei Eventing handelt es sich um einen Dienst zur Arrangierung von Veranstaltungen dient.
Eventing soll nicht nur eine Erleichterung für den Veranstalter, sondern auch für einen potentiellen Teilnehmer darstellen.

Zielsetzung

Ziel dieses Projektes war es einen Ansatz für das Verwalten von Veranstaltungen aus der Sicht eines potentiellen Teilnehmer, so wie aus der Sicht eines Veranstalters zu entwickeln.

Besonders wichtig war uns eine gering Hemmschwelle für die Nutzung zu schaffen, weshalb wir uns für einen Chatbot in Kombination einer Token Authentifizierung ohne zusätzlicher Registrierung entschieden haben.

Funktionen

Authentifizierung

Mit dem Projekt wollten wir die Anmeldehürde des Benutzers so klein wie möglich halten. Unsere Plattform verknüpft existierende Profile von unterstützten Kommunikationsdienste (aktuell Telegram) mit der eigenen Benutzerverwaltung. Das heißt sobald ein Benutzer zum ersten Mal mit unserem Bot kommuniziert und sich auf der Webseite einloggen will, wird für ihn automatisch bei uns diese Verknüpfung zu seinem Profil des Kommunikationsdienstes erzeugt. Es wird dann für jeden Login einen einmaligen Session-Token erzeugt, den der Benutzer innerhalb einer Stunde auf der Webseite verifizieren muss. Sobald dieser Token erfolgreich eingegeben wurde, verschickt das Backend einen signierten Cookie an dem Browser zurück. Bei API-Anfragen können Handler verifizieren, ob sich eine gültige Session mit einem gültigen Benutzer hinter diesem Cookie verbirgt und entsprechend die Anfrage akzeptieren, ablehnen oder die Rückmeldung anpassen.

Interaktionen mit Veranstaltungen

Das Herzstück des Projektes sind die Veranstaltungen und die Möglichkeit für Benutzer sich für diese anmelden oder abmelden zu können. Um eine passende Veranstaltung zu finden muss der Benutzer die Eventing Webseite nutzen. Die Anmeldung wird in zwei Stufen aufgeteilt, einmal die sichere Anmeldung "Teilnehmen" und einmal "Teilnehmen mit Vorbehalt". Solange eine Veranstaltung noch nicht vergangen ist, kann ein Benutzer jederzeit, sowohl auf der Webseite als auch im Chatbot, seine Teilnahme anpassen. Außerdem kann der Benutzer im Chatbot für die Veranstaltungen jeweils eine Erinnerung setzen.

Persönlicher Kalender

Um seine Veranstaltungen immer im Blick zu haben kann über den Bot einen Link zu einer ics-Datei angefordert werden. Diese baut sich aus den Events, die ein potentieller Teilnehmer als “maybe” oder als “accepted” markiert hat.

Abonnieren kann man diese Kalender über Kalender-Dienste die ics-Dateien aus dem Netzwerk beziehen und veranschaulichen können.

Veranstalter und Abonnements

Diese Funktionen sind zwar nirgendwo für den Benutzer sichtbar, da diese doch nicht mehr unverzichtbar für den MVP (minimum viable product) waren, aber es wurde bereits eine Grundlage geschaffen. Theoretisch können Benutzer über API-Aufrufe Veranstalterprofile anlegen oder löschen, sowie weitere Administratoren hinzufügen oder entfernen und Veranstalterprofile abonnieren. Dabei war die Idee, dass Nutzer unter mehreren öffentlichen Profilen Veranstaltungen veröffentlichen und Anhänger finden können.

Zeitlich abhängige Tasks

Für Aufgaben, die meistens einmalig auffallen, haben wir ein JobsService entwickelt. Eingesetzt wird es aktuell nur für die Veranstaltungserinnerungen, die manchmal Wochen im Voraus eingerichtet werden könnten. Damit diese Aufgaben nicht einfach verschwinden, falls der Node-Server zum Opfer eines Crashes fällt, mussten wir diese irgendwo persistent speichern. Diesen Service haben wir letztendlich so entwickelt, dass er in regulären Zeitintervallen in der Datenbank nach Aufgaben nachschaut, die bis vor dem nächsten Zeitintervall ausgeführt werden sollen, reserviert sich diese, damit diese auf jeden Fall nicht mehrfach ausgeführt werden und packt sie in den Node-Scheduler.

Für die Ausführung und Weiterverarbeitung werden die Argumente de-serialisiert und einen Event mit dem definierten Hook ausgelöst. Funktionen, die auf diesem Hook lauschen werden ausgeführt und schlussendlich wird die Aufgabe aus der Datenbank gelöscht.

Benachrichtigungen

Eine der wichtigsten Aufgaben des Chatbots ist es den Benutzer auf dem Laufenden zu halten, wenn beispielsweise eine Erinnerung ausgelöst wurde oder wenn die Teilnahme an Veranstaltungen auf der Webseite geändert wurde.

In der von uns abgegebenen Version wurde das Grundgerüst für einen "Veranstalter Broadcasts" erstellt, konnte aber leider nicht fertig entwickelt werden, da diese Funktion auch von den Veranstalter-Funktionen abhängig war und wir uns im Laufe des Projektes immer mehr auf die Nutzersicht fokussiert haben. Dabei war die Idee Veranstalter zu ermöglichen an Teilnehmern Nachrichten zu verschicken, ähnlich zu einem Newsletter.

Projektverlauf

Als das Team feststand und das Projekt angemeldet war, gab es in den ersten Vorlesungswochen regelmäßige Treffen an der Hochschule. Dabei wurden erste Pfeiler für eine gute Zusammenarbeit errichtet. Wir haben unsere Ziele besprochen, Technologien festgelegt, die Projektarchitektur entworfen und die ersten Aufgaben aufgeteilt.

Anfang November haben wir Notion.so als zentraler Ort für die Verwaltung unserer Aufgaben, für die Planung sowie für die Pflege unserer internen Dokumentation (bsp. das Datenmodell und ein Überblick über die wichtigsten Funktionen und ihre Abhängigkeiten) eingesetzt.

Ab dem Punkt haben wir das Projekt in wöchentlichen Sprints aufgeteilt. Die Schwerpunkte dieser Sprints sahen folgendermaßen aus:

- Sprint 1: Repositories initialisieren, Datenmodell entwerfen
- Sprint 2: Chatbot, Backend aufsetzen
- Sprint 3: Abschicken von Benachrichtigungen aus dem Bot ermöglichen, Frontend aufsetzen
- Sprint 4-5: Session- und Benutzerverwaltung
- Sprint 6-7: Refactoring auf das NestJS framework, CalDav (sabre/dav) Server testen

- Sprint 8: Benutzer/Session-API fertig entwickeln, Eventverwaltung
- Sprint 9-10: Backend und Bot MVP, Veranstalterfunktionen, Autorisierung/Authentifizierung
- Sprint 11: Bot und Backend Merge, Benutzerkalender, Erinnerungen, Testing, Frontend MVP
- Sprint 12: Feature-Freeze, Polishing, Deployment, Testing

Zwischen zwei Sprints haben wir uns gegenseitig unser Fortschritt, unsere Schwierigkeiten sowie unsere Ziele für den nächsten Sprint vorgestellt. Um uns Zeit und Wege zu ersparen hielten wir diese Meetings über Google Hangouts.

Zu guter letzt wurden nach der Prüfungsphase, die seit der Medianight übrig gebliebenen Bugs beseitigt, sowie diese Dokumentation erstellt.

Schwierigkeiten

sabre/dav

Bei sabre/dav handelt es sich um ein Framework, welches zur erstellen WebDAV-, CalDAV- und CardDAV-Server genutzt werden kann.

Wir hatten uns zunächst für einen WebDAV-Server ausgesprochen. Bei der Recherche nach möglichen WebDAV-Frameworks sind wir auf unterschiedliche Ansätze gestoßen. Letztendlich hatten wir uns für sabre/dav entschieden, da es im Vergleich zu anderen Frameworks in unsere Datenbank eingepflegt werden konnte.
Damit nicht jeder alles manuell einstellen muss, haben wir das ganze Dockerized, wo das Problem bestand, dass wird die grafische Oberfläche auf Grund von falsch konfigurierten Container nicht nutzen konnten. Für Aushilfe sorgte ein PHP Container und ein NGINX Reverse Proxy. Der Reverse-Proxy hat alle PHP-Dateianfragen an den PHP-Container und alle anderen an den sabre/dav Container weitergeleitet.
Leider hatten wir zunächst nicht bedacht, dass wir für den Zugriff auf ICS-Dateien über sabre/dav einen Benutzer und ein Passwort benötigen, da sabre/dav keine Möglichkeit bietet öffentliche Kalender zu erstellen.
Nach längerer Überlegung und Rücksprache haben wir die Entwicklung von mehreren Wochen verworfen und uns für einen eigens geschriebenen Service entschieden, der aus den Events des Benutzers eine ICS-Datei erstellt und zurück gibt.

Typeorm Frontend und Backend Queries

Um beispielsweise für das Filtern von Events aus der Datenbank nicht mehrere Endpunkte zu erstellen, brauchten wir eine Möglichkeit dies über URL-Parameter in Kombination mit dem TypeORM Querybuilder zu tun. TypeORM bietet die Möglichkeit das ganze als SQL WHERE Bedingung, als Objekt (z.B. {name: 'Max Mustermann'}) oder als Array zu übergeben. Ersteres wäre vermutlich der Beste Ansatz gewesen, um einen eigens konstruierten QueryBuilder zu nutzen.
Zunächst haben wir jedoch den zweiten Ansatz verfolgt, welcher sich als Problematisch bei OR verknüpfungen erwies.
Umgesetzt wurde der dritte Ansatz, bei dem die zu filternden Eigenschaften mit beispielsweise '__like' erweitert werden und mit OR verknüpft werden.

Beispiel URL: `.../?title__like=test&OR&description__like=test`

Auch bei diesem Ansatz gibt es noch einige Schwierigkeiten mit allgemeinen Filtern, weshalb das erwünschte Ergebnis eines universellen Endpunktes zur Abfrage von Daten nicht ganz erfüllt wird.

Testing

Testing gehört eigentlich zu den wichtigen Bestandteilen einer gut funktionierenden Anwendung. Leider haben wir bei diesem Projekt auf Grund von fehlenden Kenntnissen oder schlechten Gewohnheiten diesen Teil nicht nach empfohlener Vorgehensweise umgesetzt. Trotz modularer Aufbau des Projektes, hielten wir Unit-Tests für zu aufwändig. Die komplexen Abhängigkeiten hätten vermutlich zu einem zu hohen Anteil an Mocking geführt, weshalb das Testing hauptsächlich mit Hilfe einer End-to-End Test-Suite mit fester Reihenfolge und direkter Anbindung auf einer Datenbankinstanz durchgeführt wurde.

Bei diesem Lösungsansatz gab es einige Herausforderungen wie man beispielsweise die Test-Suite übersichtlich hält ohne die feste Reihenfolge loswerden zu müssen oder wie man leicht produktive und test Datenbank-Anbindung austauschen kann ohne zu feste Anforderungen an der Laufzeitumgebung zu stellen (z.B. die Verwendung von Docker Container).

Circular Dependencies

Durch die Umstellung auf das NestJS Framework, was unter anderem Dependency-Injection als Design-Pattern anbietet, war das Projekt regelmäßig von "Circular Dependencies" geplagt. Da hätte eine bessere Planung der REST-Routen und der dahinterstehenden Services sicherlich geholfen dieses Phänomen zu verhindern.

Lernerfolge

Mit diesem Projekt hatten wir von Anfang an vor neue Technologien und Architekturen auszuprobieren. Natürlich wollten wir trotzdem eine gute Balance zwischen bekannten und neuen Technologien treffen. Damit wir von einem gewissen Spielraum profitieren konnten, haben wir uns für eine agile Herangehensweise entschieden. Auch wenn wir bereits jeweils Vorerfahrung in agilen Methoden gesammelt hatten, haben wir die Gelegenheit bei diesem Projekt genutzt neue Werkzeuge auszuprobieren, die uns unter anderem bei der Organisation sehr geholfen haben.

Auf der technologischen Seite hatten wir uns anfangs überlegt Sprachen und Werkzeuge einzusetzen, mit denen wir bisher keine Berührungspunkte hatten. Letztendlich haben wir uns entschieden bekannte Sprachen zu verwenden aber dafür gerne neue Frameworks einzusetzen. Damit wollten wir einigermaßen sicher gehen können, dass wir in der Lage sind in der uns zur Verfügung stehenden Zeit wichtige, aber auch umfangreiche Funktionen umzusetzen.

Als wir einen Dienst für die Verwaltung von Veranstaltungen und persönliche Kalender einsetzen wollten, haben wir gemerkt, dass das Entwickeln von Konnektoren dafür viel zu aufwändig wäre, als etwas eigenes zu entwickeln, was nur die nötigen Funktionen hätte. Wir haben die Idee diesen viel zu umfangreichen Dienst zu verwenden schließlich verworfen (siehe sabre/dav).

Auf der architektonischen Seite haben wir uns ursprünglich für eine stark entkoppelte Microservice-Architektur entschieden gehabt, in dem unsere Benutzerschnittstellen Bot und Webseite komplett getrennt von dem API-Server arbeiten. Dies führte natürlich dazu, dass der API-Server vielmehr Schnittstellen nach außen anbieten musste. Als wir lernten, dass man für die Telegram-Anbindung unseres Chatbots nur einen einzelnen HTTP Endpunkt einrichten und freigeben musste, anstatt den ganzen Service bei Microsoft Azure hosten zu müssen, haben wir uns dafür entschieden zumindest bei diesem Service, aus Zeit- sowie Komplexitätsgründen für unser kleines Team die Microservice-Architektur-Idee vorerst aufzugeben.