

Dokumentation für das Softwaretechnik-Projekt AppCiMo (application for city movement)

Appcimo

Application for City Movement

Please enter your location and destination.

Dachauer Straße, Obersleißheim, München, Deutschl

Latitude: 48.2507798

Longitude: 11.528520200000003

Straße der DSF, Bergen auf Rügen, Deutschland

Latitude: 54.408382399999999

Longitude: 13.434692600000062

Timo Schwertfeger, Daniel Kaiser, Patrick Preuß

5. Juli 2017

Inhaltsverzeichnis

1	Einleitung	4
1.1	Ausgangssituation	4
1.2	Zielsetzung	4
2	Projektvorbereitung	6
2.1	Projektmanagementsoftware Taiga.io	6
2.1.1	Vorgehensmodell	6
2.1.2	Auswahl Taiga.io	7
2.1.3	Einrichtung des Projektteams	8
2.1.4	Integration HipChat	8
2.1.5	Definition von User-Stories	10
2.1.6	Definition von Tasks	10
2.2	Projektdurchführung	11
2.2.1	Sprintplanung	11
2.2.2	Retrospektive	12
3	Produktanforderungen	13
3.1	Funktionale Anforderungen	13
3.2	Nicht-funktionale Anforderungen	14
3.3	Projektanforderungen	14
3.4	User-Stories und Sprint-Tasks	14
4	Konfiguration und Einrichtung zur Softwareentwicklung	17
4.1	Entwicklungsumgebung	17
4.2	VueJS	18
4.3	Node.js und NPM	19
4.4	Github	22
4.5	CircleCI	23
4.6	Google	26
4.6.1	Google Services	26
5	Systementwurf und Umsetzung	27
5.1	Architektur	27
5.1.1	Systemarchitektur	27
5.1.2	Komponentendiagramm	27
5.1.3	Komponentenbeschreibung	28
5.1.4	Sequenzdiagramm	34
5.2	Qualitätsmanagement	35
5.2.1	Refactoring	35
5.2.2	Unit Tests	35
5.2.3	Build-Management	36
5.2.4	Continuous Integration	36

6 Zusammenfassung und Ausblick	37
7 Anhänge	38
Abbildungsverzeichnis	38
Tabellenverzeichnis	39

1 Einleitung

1.1 Ausgangssituation

In deutschen Großstädten gibt es eine Vielzahl von Möglichkeiten schnell von A nach B zu kommen. Dies ist vor allem der außerordentlichen Verkehrsinfrastruktur zu verdanken. Jedem Bürger ist die Wahl selbst überlassen, ob er mittels eines Privat-PKWs, mit den öffentlichen Verkehrsmitteln oder zum Beispiel mit dem Fahrrad seinen Bestimmungsort erreichen möchte. Doch in Großstädten, wie zum Beispiel Berlin, kann dieses Überangebot an Transportmöglichkeiten oft auf Situationen stoßen, in denen sich der/die Zielsuchende nicht sicher ist, welche Transportmöglichkeit die beste für ihn oder sie ist. Hier spielen Faktoren wie die innerstädtische Verkehrssituation, Verfügbarkeit von Car-Sharing Autos in der Nähe, Straßensperrungen oder Preise für die öffentlichen Verkehrsmittel eine Rolle. All diese Möglichkeiten abzuwägen, um möglichst schnell und günstig einen Zielort zu erreichen kann unter Umständen ein zu großer Aufwand sein.

Das Team AppCiMo möchte gerne etwas Licht in diesen Großstadtdschungel bringen und verschiedene Möglichkeiten für den eigenen Personentransport in (Groß-)Städten übersichtlich und ansprechend aufzeigen.

1.2 Zielsetzung

Ziel des Projektes ist die Planung und Entwicklung eines City-Movement-Prototypen in Form einer One-Page Webapplikation. Diese Webapplikation soll Nutzern als Entscheidungshilfe für ihre Weg-Zielerreichung in deutschen Großstädten dienen.

Den Nutzern sollen ausgehend von Start- und Bestimmungsort, drei unterschiedliche Transportmöglichkeiten für ihre Zielerreichung aufgezeigt werden. Diese Möglichkeiten umfassen Carsharing-Angebote in der Nähe, öffentliche Verkehrsmittel und das eigene Fahrrad. Zusätzlich sollen für eine sofortige Einschätzung der ermittelten Transportmöglichkeiten jeweils die Distanz, die voraussichtlich benötigte Zeit und die Kosten dargestellt werden.

Features:

- Mindestens drei verschiedene Transportmöglichkeiten um das Ziel zu erreichen
- Kurzansichten und detaillierte Ansichten
- Preisermittlung für jede Transportmöglichkeit
- Kartenvorschau in GoogleMaps
- Wegbeschreibung zu dem Ziel

Bei Appcimo handelt es sich um eine browserbasierte Webapplikation, d.h. für die Nutzung ist eine Internetverbindung notwendig. Außerdem ist eine automatische Standorterfassung von Vorteil. Diese Webapplikation soll auf den gängigen Browsern stabil laufen.

2 Projektvorbereitung

Für die Planung, Organisation, Aufteilung und Verfolgung des Projekts wird die Projekt Management Plattform **Taiga.io** verwendet. Taiga.io ist ein webbasiertes open-source Projekt-Management-Tool für agile Entwickler, Designer und Projektmanager. Die Stärken liegen vor allem in den individuellen Anpassungsmöglichkeiten, sowie in der einfachen und intuitiven Bedienung. Es wird von Start-Ups bevorzugt verwendet.

2.1 Projektmanagementsoftware Taiga.io

2.1.1 Vorgehensmodell

Das Projekt Appcimo wird anhand des Scrum Vorgehensmodells bearbeitet.

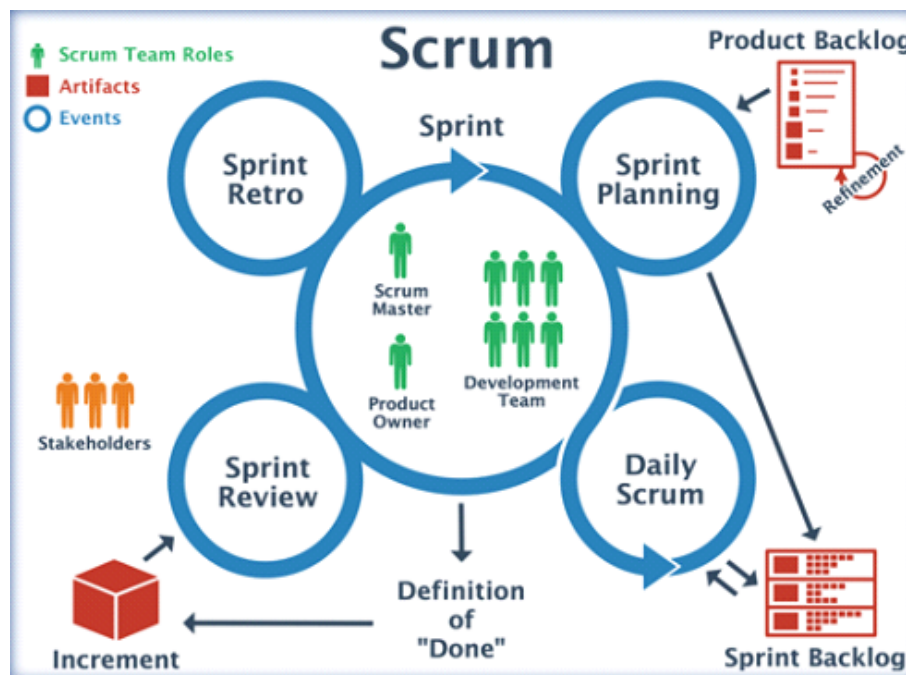


Abbildung 1: Scrum Infografik

Dabei werden vor allem folgende Vorzüge von Scrum in dem Projekt genutzt:

- wenige, leicht verständliche Regeln
- Kurze Kommunikationswege
- Hohe Flexibilität/Agilität durch adaptives Planen
- Hohe Effektivität durch Selbstorganisation
- Hohe Transparenz durch regelmäßige Meetings und Backlogs
- Zeitnahe Realisation neuer Produkteigenschaften bzw. Inkremente

- Kontinuierlicher Verbesserungsprozess
- Kurzfristige Problem-Identifikation
- Geringer Administrations- und Dokumentationsaufwand

Die Scrum-Rollen

Die Umstände in denen die Webapplikation Appcimo entwickelt wird, führen im Projektteam zu einer abgewandelten Form der klassischen Scrum-Vorgehensweise.

Somit gibt es im Projektteam keine festen Rollen wie Product Owner, Entwickler und Scrum Master. Jedes Projektmitglied ist Product owner und Scrum master. Persönliche Präferenzen lassen jedoch eine Spezialisierung wie Entwicklung, Dokumentation oder das Schaffen von wichtigen Voraussetzungen zu. Das Ziel ist hier eine ausbalancierte und effiziente Projektumgebung zu schaffen, in dem jedes Projektmitglied seine Stärken ausspielen kann und seine Wünsche berücksichtigt werden.

Die Scrum-Artefakte

Product Backlog: Darin sind die Anforderungen an die Webapplikation in Form eines vorläufigen Plans erfasst - dieser ist dynamisch und wird kontinuierlich weiterentwickelt.

Sprint Backlog: Basierend auf dem Product Backlog werden hier die im jeweiligen Sprint zu erledigenden Aufgaben für alle Projektbeteiligten einsehbar hinterlegt.

Product Increment: Das Produkt-Inkrement ist das erledigte Arbeitspaket, welches nach Ende eines Sprints als fertiges Teilprodukt geliefert wird.

Die Scrum-Aktivitäten

Sprint Planning: Für jeden Sprint muss geplant werden, welches neue Feature während eines Sprints realisiert werden kann.

Daily Scrum: Der Daily Scrum wird in einer abgewandelten Variante in Form eines Weekly Scrum realisiert. Das Ziel hier ist primär der Austausch untereinander im Projektteam bzgl. Probleme, Ideen, Lösungen und Entscheidungen.

Sprint Review: Eine nachträgliche Bewertung der zu umsetzenden Features, ob das im Sprint Backlog formulierte Entwicklungsziel aus Sicht des Projektteams zu 100 Prozent erreicht wurde.

Sprint-Retrospektive: Ein Kontrollmechanismus, ob die bisherige Arbeitsweise verbessert werden kann.

Product Backlog Refinement: Im Projektteam wird untersucht, inwieweit der im Product Backlog erfasste Plan bzw. die Produkt-Vision auf Basis neuen Wissens verbessert werden kann.

2.1.2 Auswahl Taiga.io

Für das Projektmanagement ist es wichtig eine geeignete Projektmanagementsoftware zu finden, die sich an die Bedürfnisse des Softwareprojektes anpassen lässt. Nach Recherchen und Austausch zwischen Kommilitonen und Arbeitskollegen wurde uns Taiga.io empfohlen. Taiga.io ist eine webbasierte Projektmanagement-Software mit der eine Projektdurchführung mit Scrum

ermöglicht. Integration mit Github ist ebenso möglich, wie die Einrichtung von Plugins, z.B. Chat-Programmen, wie HipChat und Slack.

2.1.3 Einrichtung des Projektteams

Sobald die Registrierung in Taiga.io erfolgt ist gelangt man auf das allgemeine Dashboard, worüber die Projekte verwaltet werden und die aktuellen Information eingesehen werden können.

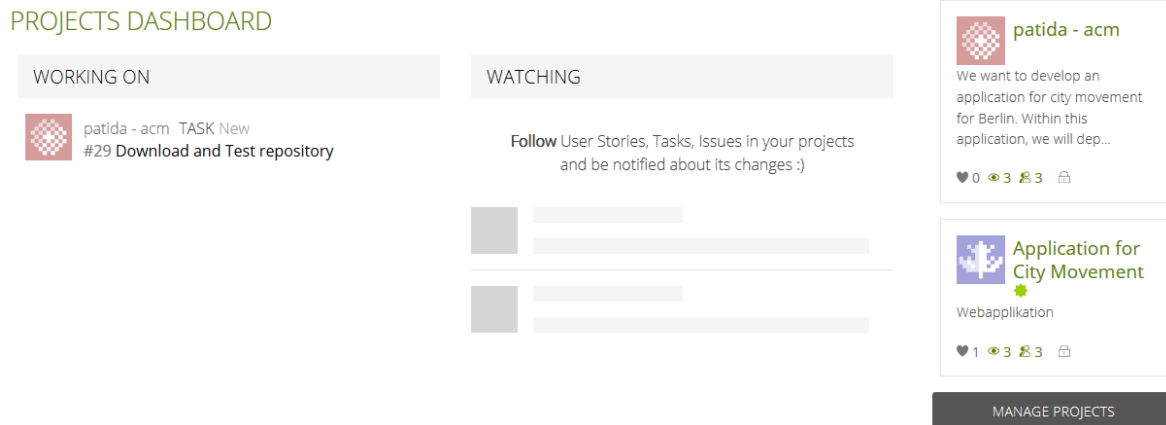


Abbildung 2: Taiga.io Dashboard

Über das Menü „MANAGE PROJECTS“ ist es möglich, alle Projekte zu verwalten und neue zu erstellen. Zur kostenlosen Nutzung ist die Erstellung eines privaten Projekts mit maximal vier Usern möglich oder beliebig viele öffentliche Projekte. Sobald das Projekt erstellt ist, können im Admin-Menü User hinzugefügt und die entsprechenden Rollen im Scrum-Vorgehensmodell zugewiesen werden.

PROJECT	APPLICATION FOR CITY MOVEMENT MANAGE MEMBERS				+ NEW MEMBER
ATTRIBUTES					
MEMBERS	Member	Admin	Role	Status	
PERMISSIONS	Daniel Kaiser arc.mereeti@gmail.com	Yes	Product Owner	Active	
INTEGRATIONS	Patrick Preuss patrick.preuss@posteo.de		Product Owner	Active	
PLUGINS	Timo Schwertfeger timo.schwertfeger@gmail.com	Yes	Product Owner	Active	

Abbildung 3: Taiga.io Admin-Members

Wir haben uns entschieden, dass alle Projektbeteiligten die Rolle Product Owner zugewiesen bekommen, da eine strikte Durchführung von Scrum nicht möglich war.

2.1.4 Integration HipChat

Um eine schnelle Kommunikation innerhalb des Projekts entschieden wir uns für eine Chat-Lösung die sowohl als Windows-Applikation als auch als Smartphone-App verfügbar ist. Für die Durchführung des Projektes nutzten wir HipChat von Atlassian. Die Integration in Taiga.io

funktioniert mit Hilfe eines Webhooks, der mit HipChat erzeugt wird und in Taiga.io integriert wird. Somit wird über sämtliche Aktivitäten im HipChat benachrichtigt.

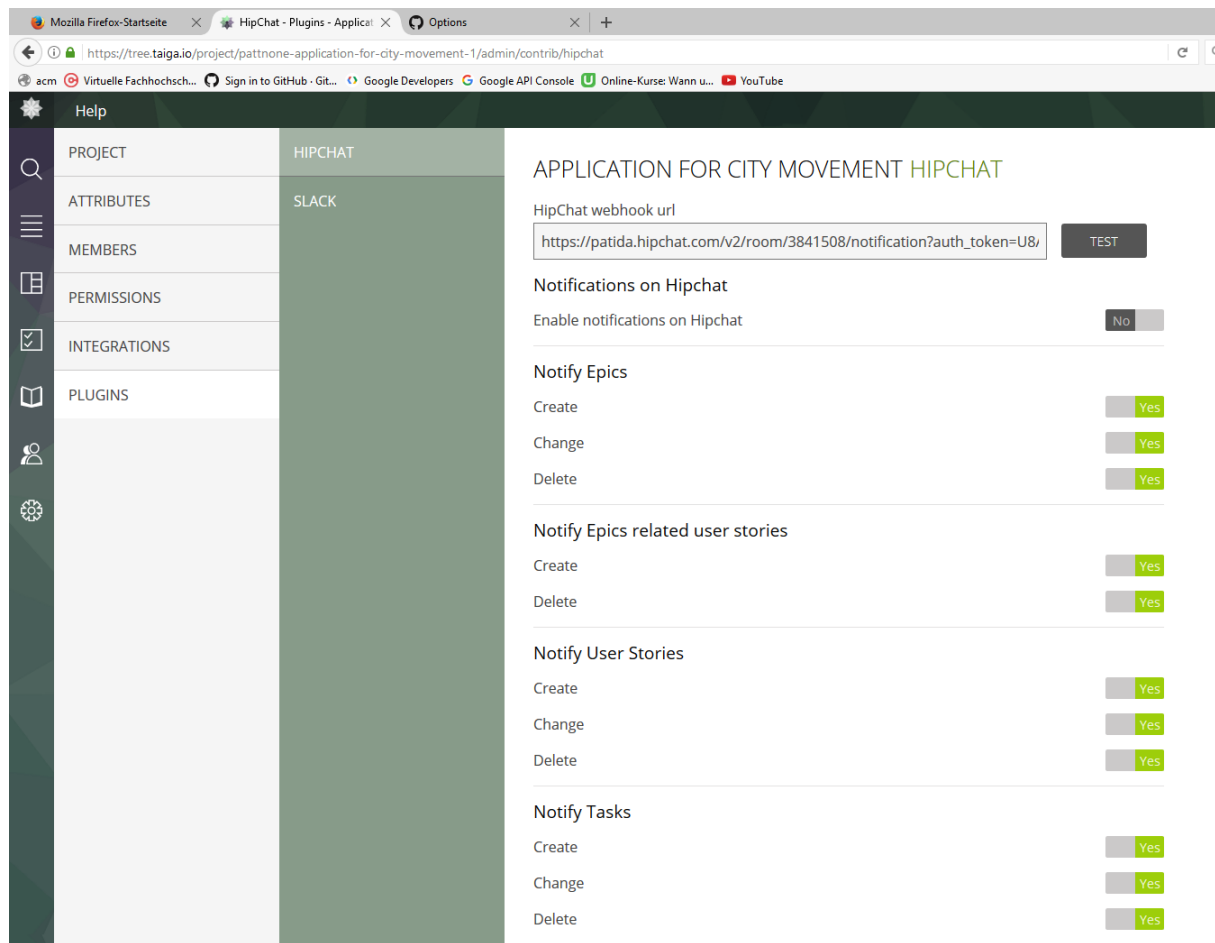


Abbildung 4: Taiga.io HipChat-Integration

Zusätzlich integrierten wir Github in den HipChat, um Aktivitäten im GitHub als Information in den HipChat zu bringen. Dies ist nützlich, um über Commits oder Pushes im GitHub und über den aktualisierten Head benachrichtigt zu werden.

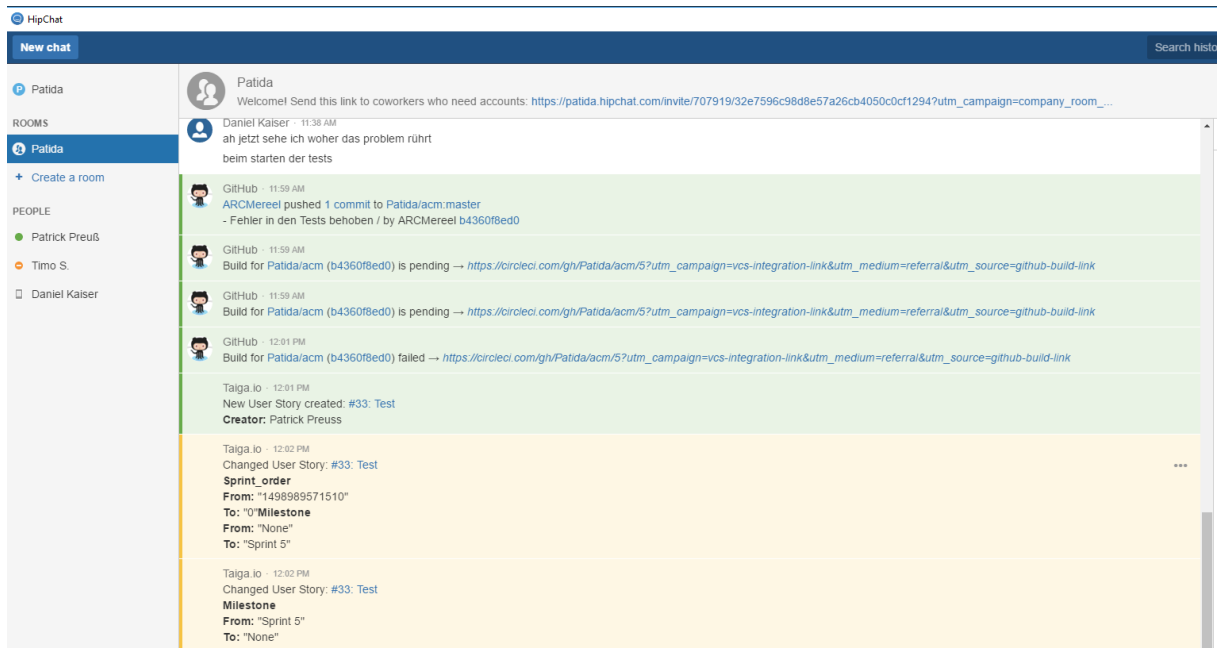


Abbildung 5: Taiga.io HipChat-Integration

2.1.5 Definition von User-Stories

User-Stories werden mit taiga.io im Backlog über „NEW USER STORY“ erstellt. Nachdem die User-Story erstellt ist, kann die User-Story einem geplanten Sprint zugeordnet werden.

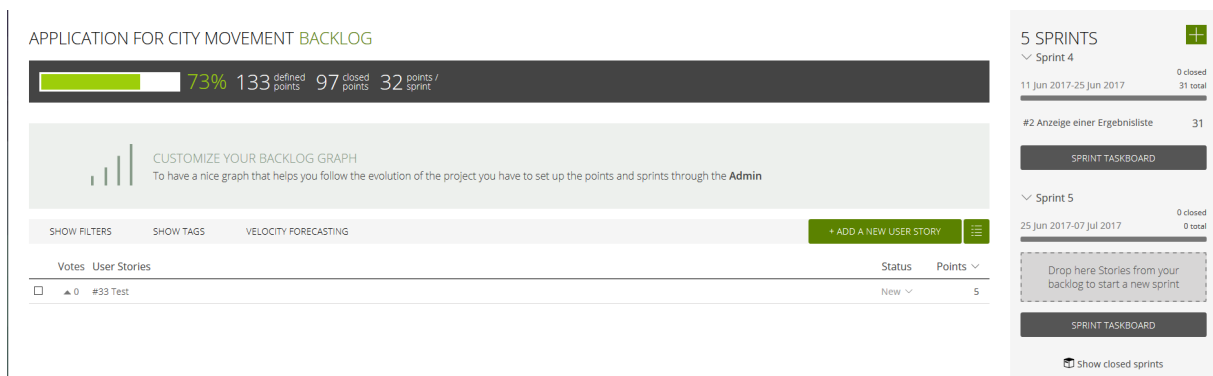


Abbildung 6: Taiga.io Backlog

Die definierten User-Stories sind unter dem Punkt 3.4 aufgeführt.

2.1.6 Definition von Tasks

Die Tasks zu den User-Story werden im Sprinttaskboard erstellt und können via Drag & Drop den Spalten „New“, „In Progress“, „Ready for Test“, „Closed“ oder „Need Information“ zugewiesen werden. Somit ist eine Übersicht des Sprint im allgemeinen und des Fortschritt der Tasks ersichtlich.

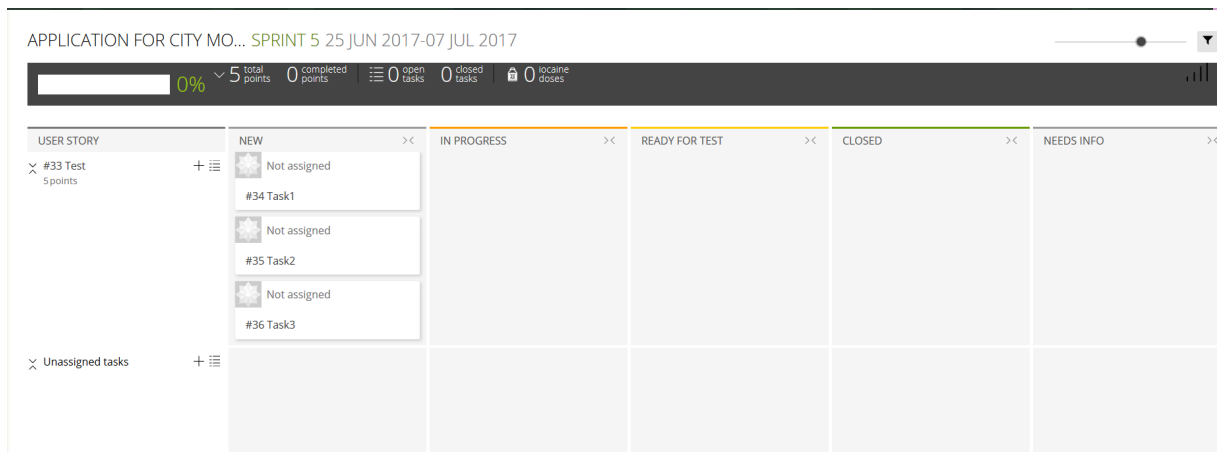


Abbildung 7: Taiga.io Sprinttasks

2.2 Projektdurchführung

2.2.1 Sprintplanung

Zur Planung des Projektes haben wir sechs Sprint den Gesamtzeitraum des Projekts definiert. Jeder Sprint dauert zwei Wochen, dadurch soll gewährleistet werden, dass der Sprint mit allen Vorgehensweisen des Scrum durchgeführt werden kann.

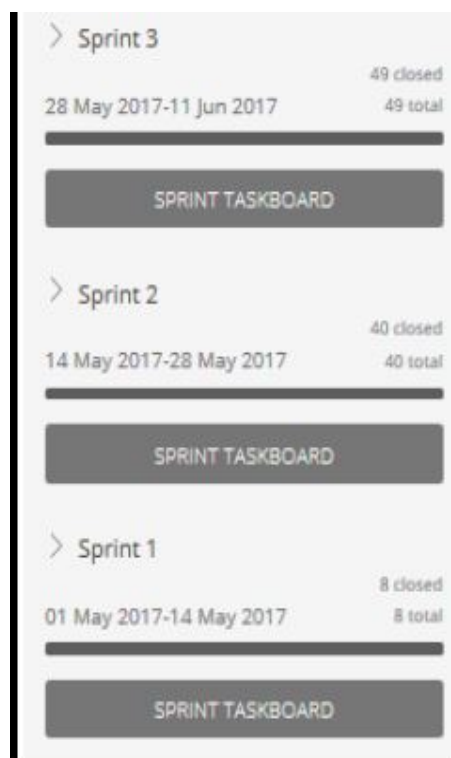


Abbildung 8: Taiga.io Sprints 1-3

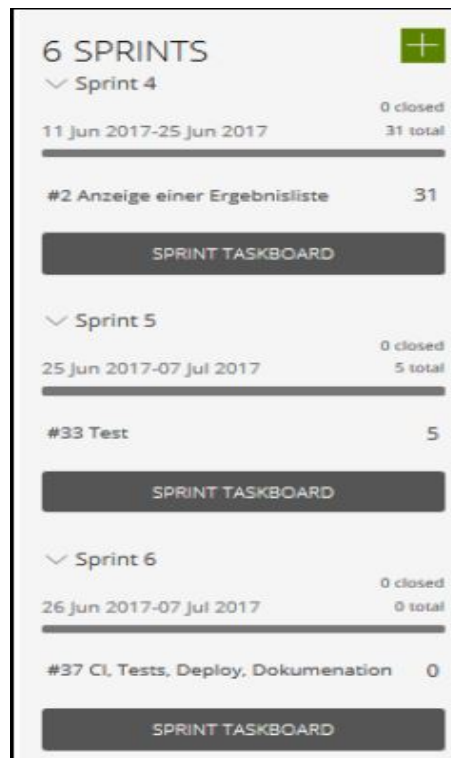


Abbildung 9: Taiga.io Sprints 4-6

2.2.2 Retrospektive

Nachdem Abschluss eines Sprints fand ein Austausch bezüglich der Ergebnisse der geplanten Tasks statt. Aus der Retrospektive ergaben sich Verbesserungsvorschläge für das weitere Vorgehen, sowie für die Planung weiterer Sprint. Der Austausch ist wichtig, um Einsicht in die Aufgaben jedes einzelnen Teammitgliedes zu erhalten und dadurch auf Stand zu sein. Da jedes Teammitglied eine wichtige Rolle in der Retrospektive spielt, führte dies zu einem guten Teamwork. Durch das Besprechen von Problemen wurde Frust innerhalb des Team vermieden.

3 Produktanforderungen

3.1 Funktionale Anforderungen

Tabelle 1: funktionale Anforderungen

Nr.	Beschreibung
FU01	Das System muss fähig sein JSON-Objekte aus den Anfragen an die Google API zu verarbeiten.
FU02	Sobald der Benutzer eine Verbindung sucht, muss das System dem Benutzer die Möglichkeit bieten einen Start- und Zielort einzugeben.
FU03	Sobald der Benutzer den Start- und Zielort eingibt, muss das System die Möglichkeit bieten, dem Benutzer eine Vorschlagsliste während der Eingabe anzuzeigen.
FU03.01	Die Vorschlagsliste muss bei der Eingabe des ersten Zeichens angezeigt werden.
FU03.02	Die vorgeschlagenen Tupel sollen mit folgender Reihenfolge angezeigt werden, 1. Straße, 2. Ort, 3. Postleitzahl
FU04	Sobald der Benutzer Start und Zielort eingegeben hat, muss das System die Möglichkeit bieten, die gesuchte Verbindung mit unterschiedlichen Transportmitteln anzuzeigen.
FU04.01	Das Transportmittel zu Fuß muss auswählbar sein.
FU04.02	Das Transportmittel Auto muss auswählbar sein.
FU04.03	Das Transportmittel öffentliche Verkehrsmittel muss auswählbar sein.
FU05	Falls der Benutzer die Suchanfrage ändert, muss das System die Möglichkeit bieten, die gesuchte Verbindung und die Karte zu aktualisieren.
FU05.01	Die Vorschlagsliste muss bei Neueingabe des Start- und Zielorts angezeigt werden.
FU05.02	Die Verbindungskarte muss mit neuem Start- und Zielort die gesuchte Verbindung anzeigen.
FU05.03	Die Distanz der neuen Verbindung muss aktualisiert werden.
FU05.04	Die Dauer der neuen Verbindung muss aktualisiert werden.
FU05.05	Der Preis der neuen Verbindung muss angezeigt werden.
FU06	Falls der Benutzer eine Verbindung sucht, muss das System die Möglichkeit bieten, mehrere Transportmittel auszuwählen.
FU07	Sobald der Benutzer eine Verbindung sucht, muss das System die Möglichkeit bieten, eine Ergebnisliste der gesuchten Verbindung anzuzeigen.
FU07.01	In der Ergebnisliste muss der aktuelle Preis angezeigt werden.
FU07.02	In der Ergebnisliste muss die Dauer angezeigt werden.
FU07.03	In der Ergebnisliste muss die Distanz angezeigt werden.

3.2 Nicht-funktionale Anforderungen

Tabelle 2: Projektanforderungen

Nr.	Beschreibung
NFU01	Das System muss plattformunabhängig und webbasiert sein.
NFU02	Das System muss mit einem Entwicklungs-Framework umgesetzt werden.
NFU03	Das System soll als Single-Page Anwendung umgesetzt werden.
NFU04	Das System soll getestet werden.
NFU05	Das System soll über ein Deployment verfügen.
NFU06	Für Das System soll Continious Integration implementiert werden.

3.3 Projektanforderungen

Tabelle 3: Nicht-funktionale Anforderungen

Nr.	Beschreibung
PRJ01	Für die Umsetzung des Systems soll ein modernes Entwicklungs-Framework für die Softwareerstellung genutzt werden.
PRJ02	Es muss eine Projektmanagementsoftware, für die mit der Softwareerstellung einhergehende Projektarbeit, genutzt werden.
PRJ03	Der entwickelte Programmcode muss auf Github als Master-Branch hochgeladen werden.
PRJ04	Für das Software-Projekt muss eine Projektdokumentation erstellt werden.

3.4 User-Stories und Sprint-Tasks

Tabelle 4: Sprint 1

US01 - Eingabe des Start- und Zielorts	
Task #7	Felder erstellen für die Eingabe eines Start- und Zielorts
US03 - Vorschlagsliste für Eingabe des Start- und Zielorts	
Task #8	Einarbeitung in Google API Dokumentation für Autocomplete-Funktion
Task #9	Autocomplete-Funktion in Felder des Start- und Zielorts implementieren

Tabelle 5: Sprint 2

US06 - Auswahl verschiedener Transportmittel	
Task #10	Auswahl für öffentliche Verkehrsmittel hinzufügen
Task #11	Auswahl für Car2Go/Auto hinzufügen
Task #12	Auswahl für Fahrrad hinzufügen
Task #31	Mockup für Car2Go Objekte erstellen und einbinden

US05 - Aktualisierung der Verbindungen nach Änderung des Start- und/oder Zielorts	
Task #13	Aktualisierung der Google Karte bei Änderungen implementieren
Task #32	Aktualisierung der Wegbeschreibung bei Änderung implementieren

Tabelle 6: Sprint 3

US04 - Anzeige von Verbindungen auf der Karte	
Task #14	Google map für Autofahrt implementieren
Task #15	Google Map für Fahrt mit öffentlichen Verkehrsmitteln implementieren
Task #16	Google Map für Fahrradfahrt implementieren
Task #17	Marker-Funktion bereitstellen
Task #18	Gmap Marker- und Pfad-Funktionen auf Transportmittel und Autocompletefelder anwenden
Task #20	Wegbeschreibung für Autofahrt implementieren
Task #29	Zwischenstopp für Car2Go auf Google Map erstellen
Task #30	Einbindung des Zwischenstopps für das Car2Go Auto in der Wegbeschreibung

US07 - Anzeigen einer Wegbeschreibung	
Task #21	Wegbeschreibung für Transit-Fahrt implementieren
Task #22	Wegbeschreibung für Fahrradfahrt implementieren
Task #23	Webbeschreibung von Map abkoppeln

Tabelle 7: Sprint 4

US02 - Anzeige einer Ergebnisliste	
Task #24	Autofahrt Zusammenfassung
Task #25	Transit Zusammenfassung
Task #26	Fahrrad Zusammenfassung
Task #27	Aufklappfunktion für Ergebnisleisten implementieren
Task #28	Erstellen von Unterabschnitten zu den jeweiligen Ergebnisleisten

Tabelle 8: Sprint 5

US08 - Programmcode-Refactoring	
Task #34	Komponenten umstrukturieren
Task #35	Methoden umschreiben, damit diese testbar sind
Task #36	Konfiguration der Testumgebung

Tabelle 9: Sprint 6

US09 - CI, Tests, Deploy, Dokumentation	
Task #38	Continuous Integration konfigurieren (CircleCI)
Task #39	Unit Tests schreiben
Task #40	Projektdokumentation schreiben
Task #41	Deployment AppCiMo nach AWS (Amazon Web Service)

4 Konfiguration und Einrichtung zur Softwareentwicklung

4.1 Entwicklungsumgebung

Webbasierte Javascript-Anwendungen lassen sich mit Hilfe einer Vielzahl von Entwicklungsumgebungen (IDEs) programmieren. Das Entwicklungsteam hat sich, nach Abwägung der jeweiligen Vor- und Nachteile, für die Verwendung von zwei unterschiedlichen IDEs entschlossen.

Eine dieser IDEs ist **WEBSTORM**, das kostenlos von **JETBRAINS** angeboten wird.



Abbildung 10: IDE Webstorm Logo

In der aktuellen Version 2017.1 wird WEBSTORM mit einer Reihe an nützlichen Tools und Kontrollmechanismen zur Verfügung gestellt. Dies beinhaltet unter anderem die Unterstützung aktueller Javascript Frameworks wie Angular, React, Vue.js und Meteor. Code-completion, Navigations- und Übersichtshilfen, Error-Erkennung und eingebaute Refactoring-Funktionen.

Dartüber hinaus lassen sich npm-Befehle direkt in Webstorm ausführen und in einer eigenen Konsole anzeigen lassen. Weitere nützliche Funktionen sind die Anbindung gängiger Version Control Systems wie Github oder das Erstellen und Ausführen von Testszenarien, z.B. mittels Karma, Mocha, Jest and Protractor.

— Relevante Webstorm-Aktionen/Libraries/Addons zu unserem Projekt einfügen

Bei der anderen IDE, die für die Webapp verwendet wurde, handelt es sich um Atom, ein moderner und anpassbarer Text-Editor.

Ausgeliefert wird Atom in einem simplen Design mit wenigen Tools und Werkzeugen. Das Motto lautet hier: Weniger ist Mehr. Konzentriertes und ablenkungsfreies Programmieren soll somit ermöglicht werden.

Über den integrierten Package-Manager lassen sich jedoch noch weitere Pakete installieren, die für das Softwareprojekt benötigt werden. Diese Pakete sind, wie Atom auch, Open-Source Pakete. Es lassen sich aus tausenden von Paketen die benötigten Werkzeuge und Tools installieren. Unter anderem sind das GUI-Themes, Folder-Management Tools, Overview-Tools, Error-Detection und Tools, um die Arbeit am Code visuell zu verbessern.



Abbildung 11: Texteditor Atom Logo

Für die Erstellung der Applikation Appcimo wurde das community-Package **language-vue** installiert, dass Error-Handling und visuelle Unterstützung beim Programmieren zur Verfügung stellt.

4.2 VueJS

Appcimo wird mittels des clientseitigen Javascript-Frameworks **Vue.JS 2.0** erstellt.

Vue.js ist eine Library für interaktive User-Interfaces. Technisch gesehen ist Vue.js auf den ViewModel-Layer des MVVM-Pattern fokussiert: Sie verbindet die View und das Model über Two-Way-Data-Bindings. Es wird bevorzugt bei der Erstellung von Single-Page-Anwendungen verwendet.

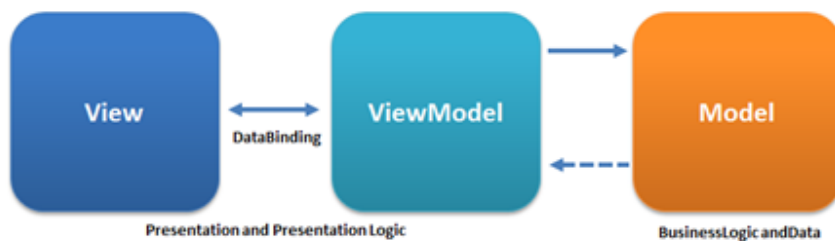


Abbildung 12: MVVM Schema

Vue.js verbindet die sichtbaren Elemente und die Datensicht eines Systems selbstständig. Damit reagiert es automatisch bei Änderung von Variablen und stellt diese mittels DOM-Manipulationen und Output Formatting dar.

Das Herzstück der Vue.js-Bibliothek sind jedoch die Komponenten, mit denen sich komplexe Strukturen abbilden lassen. Wie bei anderen Systemen können sie weitere Komponenten enthalten.

Die Parent-Komponenten bestimmen dabei die Eigenschaften der Child-Komponenten. Die Kommunikation zwischen den Komponenten untereinander wird mit einem Event-System realisiert.

Außerdem bietet Vue.js Möglichkeiten beim Einfügen von neuen Elementen, diese mit animierten Effekten oder Übergängen zu verschönern.

Bei der Erstellung der Webapplikation Appcimo sorgen vor allem **Direktiven** für übersichtliche und leicht-verständliche Code-Abschnitte. Damit lassen sich zum Beispiel Schleifen durch ein

Array iterieren, HTML-Knoten optional einbinden (v-if) und ausblenden (v-show), Klickevents abfangen (v-on) und Attribute an Variablen binden (v-bind).

4.3 Node.js und NPM

Appcimo wird mit Hilfe der open-source JavaScript Runtime **Node.JS** erstellt. Es wird von der **Node.js Foundation** entwickelt und vertrieben.



Abbildung 13: Node.JS Logo

Node.JS wird benötigt, um dem Javascript-Programm eine Laufzeitumgebung zur Verfügung zu stellen und den Quellcode mittels des Node.JS Interpreters einzulesen, zu analysieren und auszuführen. Dadurch kann Javascript-Code auf einem Server lauffähig gemacht werden.

Die aktuelle Node.JS Version kann auf der offiziellen Seite <https://nodejs.org/en/> gratis heruntergeladen und installiert werden.

Mit der Installation von Node.JS wird auch der **Node.JS Package Manager** zur Verfügung gestellt.



Abbildung 14: Node.JS Package Manager Logo

Der npm Package Manager ist eine Plattform für Javascript-Entwickler, die Ihren Code oder Teile Ihres Codes für andere Entwickler zur Verfügung stellen möchten. Dieser Code wird in Packages oder Module gebündelt, die JavaScript-Bibliotheken enthalten. Ein Package ist ein Verzeichnis, das eine oder mehrere Dateien enthält. In ihnen gibt es typischerweise die Datei **package.json**, die einige Meta-Daten zu dem Package enthält.

Packages sind relativ klein und meistens nur für einen bestimmten Zweck gedacht. Die Idee hier ist, mittels mehrerer kleiner Packages maßgeschneiderte Lösungen für die eigene Applikation zu

konstruieren.

Die Packages und Module werden über die offizielle npm Seite www.npmjs.com/ zum Download angeboten. Sie werden üblicherweise über den Bash/Cmd Konsolenbefehl `npm install [...]` installiert. Sie werden dann als Komponenten in die Applikation eingefügt und in dem Projekt-Verzeichnis **node_modules** abgelegt.

Mittels des Package-Managers lassen sich die für das eigene Programm verwendeten Packages leicht verwalten und auch updaten.

Durch die Verwendung des Webpack Templates werden eine Vielzahl an Tools und Modulen für die Webseitenerstellung bereitgestellt. Neben dieser Grundinstallation wurden folgende npm Packages nachträglich installiert:

Name	vue-cli
Version	2.8.2
Entwickler	yyx990803 (https://www.npmjs.com/~yyx990803)
Installation	<code>\$ npm install -g vue-cli</code>
Benutzung	Git-Bash <code>\$ vue init <template-name> <project-name></code> Beispiel: <code>\$ vue init webpack my-project</code>
Zweck	Das Package vue-cli eine Art grundlegendes Gerüst für Vue.JS Applikationen. Für Appcimo wurde dank vue-cli in die Installation das Template Webpack ausgewählt, dass einige wichtige Module für Vue-Webentwickler zur Verfügung stellt, unter anderem asset management, minification, module bundling, linting, testing, ... <code>vue init <template> <name></code> Beispiel: <code>Vue init webpack Appcimo</code>

Name	vue-google-autocomplete
Version	1.0.10
Entwickler	Olefirenko (https://www.npmjs.com/~olefirenko)
Installation	<code>\$ npm install vue-google-autocomplete --save</code>
Benutzung	Das Package wird in das Template mit folgender Syntax eingefügt: <code>import VueGoogleAutocomplete from 'vue-google-autocomplete'</code> <code><vue-google-autocomplete id="map" classname="form-control" placeholder="Start typing" v-on:placechanged="getAddressData" > </vue-google-autocomplete></code>
Zweck	Das Package stellt eine „Vue Google Autocomplete“ - Komponente zur Verfügung, dass der offiziellen Google Eingabemaske in GoogleMaps ähnelt. Mit der Eingabe einer Adresse oder eines bestimmten Ortes in das Eingabefeld können unter anderem Positionsdaten (Breitengrad, Längengrad) zusammen mit anderen Daten (Land, Stadt, Bundesland, Straße, Hausnummer, PLZ) zu diesem speziellen ort ermittelt werden. Außerdem lassen sich ermittelte Resultate auf Länderebene beschränken und es enthält die Funktion die Position des benutzers automatisch zu bestimmen.

Name	promise
Version	8.0.0
Entwickler	forbeslindesay (https://www.npmjs.com/~forbeslindesay)
Installation	\$ npm install promise
Benutzung	<pre>var Promise = require('promise'); var promise = new Promise(function (resolve, reject) { get('http://www.google.com', function (err, res) { if (err) reject(err); else resolve(res); }); });</pre>
Zweck	<p>Sobald der Zustand fulfilled oder rejected erreicht sind, können die Ergebnisse für von Promise abhängige Funktionen genutzt werden. Diese Funktionen können abhängig von dem promise platziert werden. Sie werden somit erst ausgeführt sobald einer der beiden Zustände eintritt.</p> <p>Der Kernidee hinter 'promises' ist, dass ein promise ein Resultat einer asynchronen Operation enthält. Ein promise kann drei verschiedene Zustände haben: pending (Anfangsstatus), fulfilled (Erfüllte Operation) und rejected (Fehlgeschlagene Operation).</p> <p>Promises werden bei Appcimo unter anderem dazu verwendet Abfragen mittels Google-API-Diensten an die Google Server zu stellen und sicherzustellen, dass die Daten korrekt eingetragen und verwendet werden können.</p>

Name	vue-material
Version	0.7.4
Entwickler	pablohpsilva (https://www.npmjs.com/~pablohpsilva)
Installation	\$ npm install --save vue-material
Benutzung	<pre>import Vue from 'vue' import VueMaterial from 'vue-material' import 'vue-material/dist/vue-material.css'</pre> <p>Zur Verwendung: Vue.use(VueMaterial)</p>
Zweck	<p>Vue Material ist ein leichtgewichtiges Framework, das darauf abzielt, wiederverwendbare Komponenten in Form von ansprechenden UI Elementen zur Verfügung zu stellen.</p> <p>In Appcimo wurde die Komponente <tabs> von vue-material verwendet, um die Wegbeschreibung und die GoogleMap einzubinden. Die Inhalte können dadurch strukturiert und ein- und ausgeblendet werden.</p>

Name	vue-router
Version	2.7.0
Entwickler	yyx990803 (https://www.npmjs.com/~yyx990803)
Installation	\$ npm install vue-router
Benutzung	<pre><script src="/path/to/vue.js"></script> <script src="/path/to/vue-router.js"></script></pre> <p>Vue.use(VueRouter)</p>
Zweck	<p>Vue-router hilft den Entwicklern Single-Page Applikationen zu erstellen.</p> <p>In Vue.JS werden Projekte in Komponenten strukturiert. Mit dem vue-router lassen sich die einzelnen Komponenten an bestimmte Routen koppeln. Vue-Router kontrolliert demnach, wo welche Komponenten gerendert werden sollen.</p> <p>Da Appcimo eine Single-Page Applikation ist, werden die Komponenten zunächst nicht auf verschiedenen Seiten der Applikation gerendert, sondern nur auf der Startseite (path: '/'). Es entfallen routen, die auf Seiten wie Benutzerprofile oder Dateiuploads verweisen.</p>

Name	karma
Version	1.7.0
Entwickler	dignifiedquire (https://www.npmjs.com/~dignifiedquire)
Installation	\$ npm install karma --save
Benutzung	Karma wird mit dem Webpack-Template von vue-cli installiert. Änderungen in der Konfiguration von Karma werden durch Anpassungen der Datei test/unit/karma.conf.js ermöglicht.
Zweck	Karma ist Entwicklungstool, das verwendet wird, um Abschnitte des JavaScripts-Codes zu testen. Die Tests können dabei lokal und auf verschiedenen Browsern mittels der Konsole (z.B. Bash) ausgeführt werden. Da Karma auf einer Javascript Konfigurationsdatei basiert, kann es von vielen Continuous Integration Lösungen ausgelesen und importiert werden. Weitere Packages für die Tests: Mocha ist ein JavaScript test framework für node.JS. Es führt Tests seriell aus und ermöglicht flexibles und genaues Reporting der Testresultate. Sinon-Chai liefert Assertions in einer eigenen Assertions-Library. Dadurch lassen sich Testfälle generieren und gegen erwartete Werte gegentesten.

4.4 Github

Appcimo wird mit dem Verison Control System **Git** auf **Github** gespeichert. Dadurch wird eine transparente und sichere Erstellung der Applikation gewährleistet.



Abbildung 15: GitHub Logo

<https://github.com/Patida/acm>

Hierfür wurden Entwickler des Teams als Contributor für das Projekt freigeschaltet.

Git-Hub Accounts

<https://github.com/schwetim>

<https://github.com/ARCMereel>

<https://github.com/PattnOne>

Das **Repository** enthält den gesamten Code des Projekts, zuzüglich der Tests und der Dokumentation und wird an einem Master Branch erstellt.

änderungen an dem Programm-Code werden von den Projektmitgliedern commitet und gepusht, so dass alle Entwickler dauerhaft Zugriff auf die aktuellste Programm-Version haben.

Durch Kommentare bei jedem Commit, kennen alle Entwickler den aktuellen Stand und die Gründe des Pushs.

Bei Push-Überschneidungen werden die Branches gemerged, so dass alle Änderungen und Features im neuen Versionsstand enthalten sind.

4.5 CircleCI

CircleCI ist eine Continuous Integration und Release Platform.

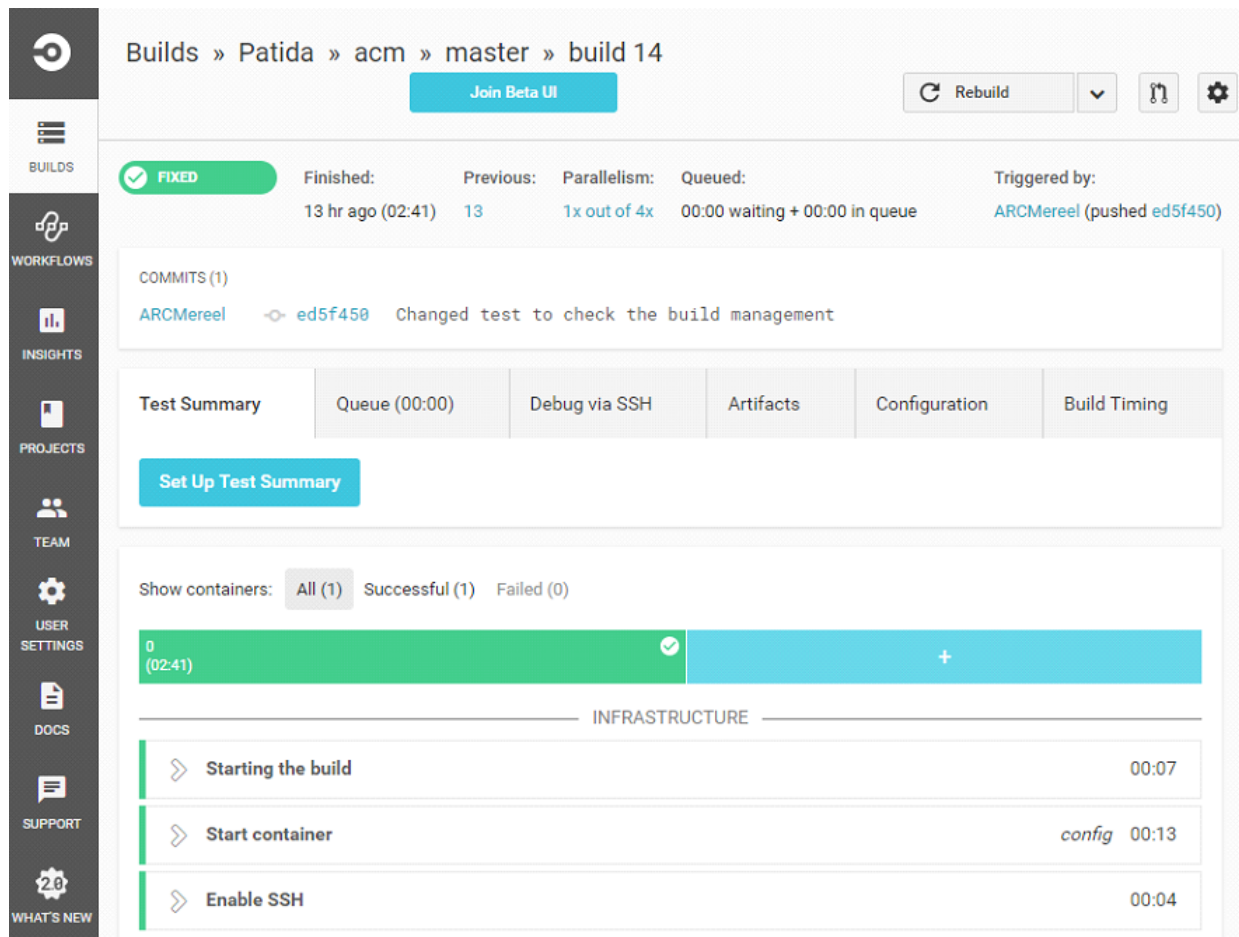


Abbildung 16: CircleCI Logo

CircleCI automatisiert den Build-, Test-, und Deploy-Prozess von Applikationen. Bei der Einrichtung des CircleCI accounts, wird das Github Repository über eine Schnittstelle mit CircleCI verknüpft.

Somit kann CircleCI auf den kompletten Code zugreifen und Tools und Services zur Verfügung stellen.

Der Software-Build kann über ein modernes UI gesteuert und beobachtet werden.



BUILDS » Patida » acm » master » build 14

[Join Beta UI](#)

[Rebuild](#) [v](#) [🔗](#) [⚙️](#)

FIXED Finished: 13 hr ago (02:41) Previous: 13 Parallelism: 1x out of 4x Queued: 00:00 waiting + 00:00 in queue Triggered by: ARCMereel (pushed ed5f450)

COMMITTS (1)
ARCMereel [ed5f450](#) Changed test to check the build management

Test Summary Queue (00:00) Debug via SSH Artifacts Configuration Build Timing

[Set Up Test Summary](#)

Show containers: **All (1)** Successful (1) Failed (0)

0 (02:41) [+](#)

INFRASTRUCTURE

- Starting the build 00:07
- Start container config 00:13
- Enable SSH 00:04

Abbildung 17: CircleCI Starteseite

Da der Software-Build inklusive aller Tests logisch ausgeführt wird, kann auf das Konsolen-Reporting der Testfälle innerhalb der CircleCI UI zugegriffen werden.


```

Appcimo
  ✓ should render correct contents
  ✓ Car Adresse Mocking work
  ✓ Message works
  ✓ Subline works
  ✓ Headline works

GetRoutes
  ✓ Price works
  ✓ Duration at mounting works
  ✓ function duration work
  ✓ function transportmethod works for DRIVING
  ✓ function transportmethod works for WALKING
  ✓ function transportmethod works for TRANSIT
  ✓ function transportmethod works for BICYCLING

SubResultDescription
  ✓ Total works
  ✓ WayDescriptor works

SubResultMap
  ✓ Total works
  ✓ isCarSearch works
  ✓ isTrainSearch works

PhantomJS 2.1.1 (Linux 0.0.0): Executed 17 of 17 SUCCESS (0.01 secs / 0.002 secs)
TOTAL: 17 SUCCESS

===== Coverage summary =====
Statements : 32.56% ( 42/129 )
Branches   : 20% ( 10/50 )
Functions  : 33.33% ( 10/30 )
Lines      : 32.56% ( 42/129 )
=====

```

Abbildung 18: Ausschnitt der Appcimo-Testauswertung

AWS S3

Ein weiteres Feature von CircleCI ist die Koppelung des CircleCI-Accounts mit einem Amazon AWS Account.

Über eine Schnittstelle zu Amazons Web Services ist CircleCI in der Lage, nach erfolgreichem Test, die Applikation oder den Build der Applikation zu einem Amazon AWS S3 Speicherort hochzuladen (deployen).

```

$ aws s3 sync /home/ubuntu/acm/dist s3://appcimo --delete

upload: dist/index.html to s3://appcimo/index.html
upload: dist/static/js/app.6d944625cf82fcdcfdf6.js to s3://appcimo/static/js/app.6d944625cf82fcdcfdf6.js
upload: dist/static/js/manifest.27de37d567170b0365ce.js to s3://appcimo/static/js/manifest.27de37d567170b0365ce.js
upload: dist/static/js/manifest.27de37d567170b0365ce.js.map to s3://appcimo/static/js/manifest.27de37d567170b0365ce.js.map
upload: dist/static/js/app.6d944625cf82fcdcfdf6.js.map to s3://appcimo/static/js/app.6d944625cf82fcdcfdf6.js.map
upload: dist/static/css/app.d6a2f8ca9a7005ee9f3536d4f6aaed0f.css to s3://appcimo/static/css/app.d6a2f8ca9a7005ee9f3536d4f6aaed0f.css
upload: dist/static/css/app.d6a2f8ca9a7005ee9f3536d4f6aaed0f.css.map to s3://appcimo/static/css/app.d6a2f8ca9a7005ee9f3536d4f6aaed0f.css.map
upload: dist/static/img/appcimo-background-pic.b99153d.png to s3://appcimo/static/img/appcimo-background-pic.b99153d.png
upload: dist/static/js/vendor.29e576132a4888eb6085.js to s3://appcimo/static/js/vendor.29e576132a4888eb6085.js
upload: dist/static/js/vendor.29e576132a4888eb6085.js.map to s3://appcimo/static/js/vendor.29e576132a4888eb6085.js.map

```

Abbildung 19: Application Build - CircleCI

Link zur Applikation: <http://appcimo.s3-website-eu-west-1.amazonaws.com/>

4.6 Google

4.6.1 Google Services

Für die Einbindung von Google-Services wie die Ermittlung von Positionsdaten, GoogleMaps und Routenberechnungen wird die Einbindung einer Google-API mit gültigem Key in die Applikation vorausgesetzt.

Die Beantragung eines Keys erfolgt über den Google API Manager. Voraussetzung hierfür ist ein Google-Konto.

Folgende Google-Services werden für Appcimo benötigt.



Abbildung 20: Google API's

Google stellt dafür ein Tageskontingent von 1.000 kostenlosen Anfragen an die Places Library pro Tag zur Verfügung.

Die Überwachung der API-Nutzung, eventuelle Latenzen und Übertragungsfehler lassen sich auch im Google-API Dashboard aufrufen.

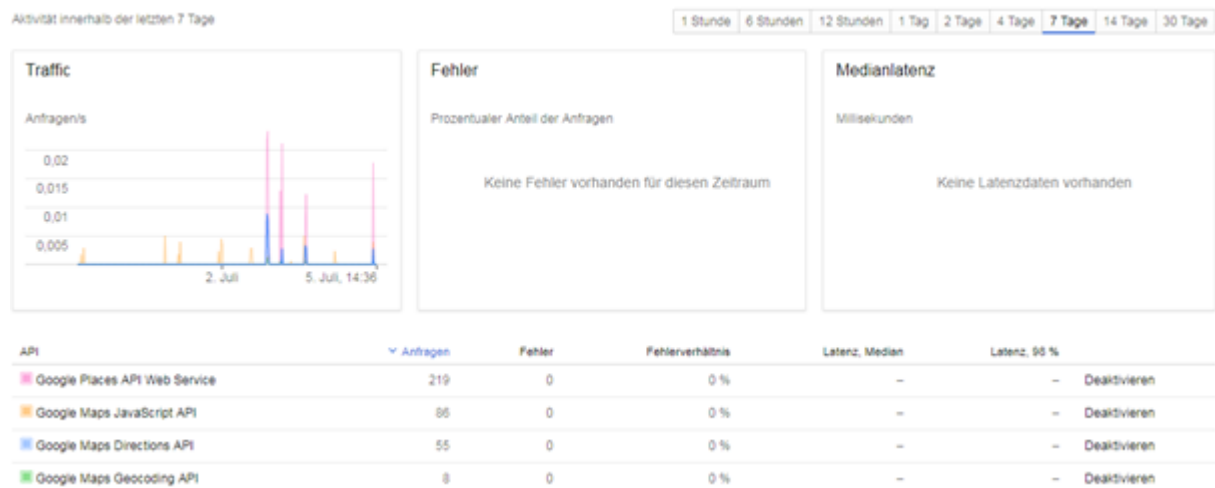


Abbildung 21: Google API Dashboard

5 Systementwurf und Umsetzung

5.1 Architektur

5.1.1 Systemarchitektur

Die Entwicklung der Webapplikation richtet sich entsprechend VueJS nach der MVVM Model-View-ViewModel Architektur.

Um diese simple Architektur auf die Applikation anzuwenden, wird das Komponentendiagramm abstrakt genutzt. Das folgende Diagramm veranschaulicht, dass das View- und Viewmodell-Layer in der VueJS-Applikation selbst zu finden ist. In den einzelnen Vue-Komponenten sind sowohl die View, als auch die Steuerung und die Business Logic implementiert.

Das Backend/ Modell wird hierbei durch die Google-API und die Car2go-API (Car2go-API wird momentan nur simuliert) abgebildet. In ihnen werden die Routenberechnungen durchgeführt und an das ViewModell zurückgegeben. Im ViewModell werden diese Routen dann weiter für die View aufgearbeitet.

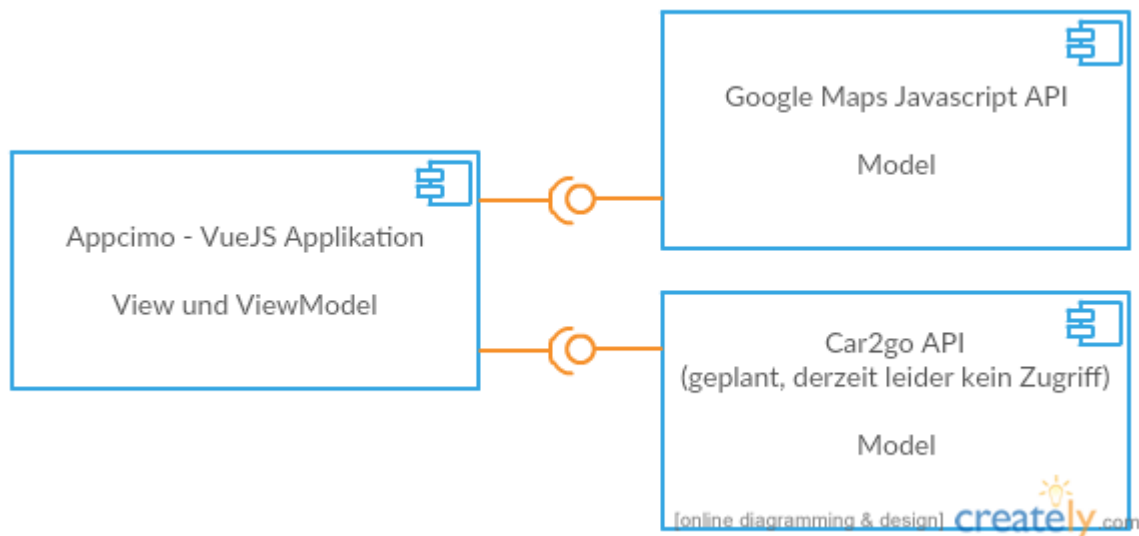


Abbildung 22: MVVM Appcimo

5.1.2 Komponentendiagramm

Im Komponentendiagramm sind die einzelnen Vue-Komponenten sowie ihre Zugriffe untereinander abgebildet. Das Interface der Komponente stellen hierbei die Properties von diesen dar. Durch Properties, und nur durch diese, können Daten in der Applikation von einer Komponente zur nächsten gereicht werden.

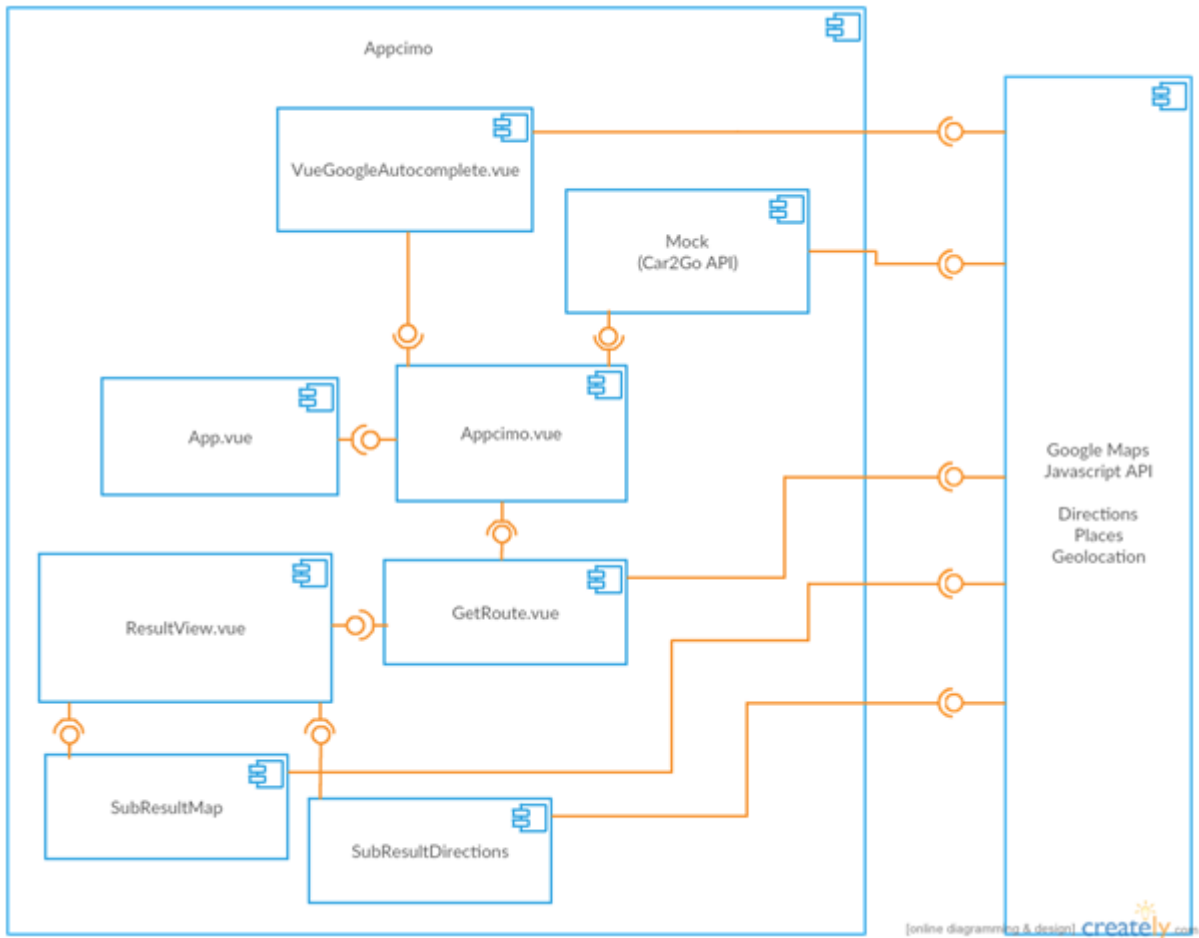


Abbildung 23: Komponentendiagramm

5.1.3 Komponentenbeschreibung

Komponenten sind eines der mächtigsten Features in Vue. Durch sie ist es möglich, HTML-Elemente und die damit verbundenen Script- und Style-Elemente wiederverwendbar zu machen. Diese Komponenten können auf das Nutzerverhalten oder Events reagieren und Werte und Attribute aktiv modifizieren und bereitstellen.

Komponenten

1. Appcimo.vue

Appcimo.vue ist die Hauptkomponente der Webapplikation. Alle weiteren Komponenten sind entweder in die Appcimo-Komponente eingesetzt oder in Unterkomponenten von Appcimo.vue eingesetzt.



Abbildung 24: Komponente Appcimo.vue

Appcimo.vue enthält die grundsätzlichen Style-Elemente für das Logo der Seite, die Top-Navigation, den Hintergrund und die einzelnen Untermenüs. Außerdem enthält diese Komponente den Button, der die Methode `getRoutes()` startet: Finde meinen Weg!

Enthaltene Komponenten: `VueGoogleAutocomplete`, `GetRoutes`.

Funktionen

`getOrigin()`: Ermittelt durch Auswertung der `VueGoogleAutocomplete`-Komponente die Geoinformationen vom Startpunkt und über die GoogleMaps-API den nächstgelegenen Car2Go Standort.

`getDestination()`: Ermittelt durch Auswertung der `VueGoogleAutocomplete`-Komponente die Geoinformationen vom Zielort.

`getRoutes()`: Startet den Prozess die Komponente `GetRoutes.vue` zu rendern und sichtbar zu machen.

`mockCarAddress()`: Simuliert zu Testzwecken einen Standort eines Car2Go-Autos.

`getCarLocation()`: Ermittelt über den asynchronen Google geocode service den Standort des nächstgelegenen Car2Go-Autos als ein Google geocode Objekt.

2. `VueGoogleAutocomplete.vue`

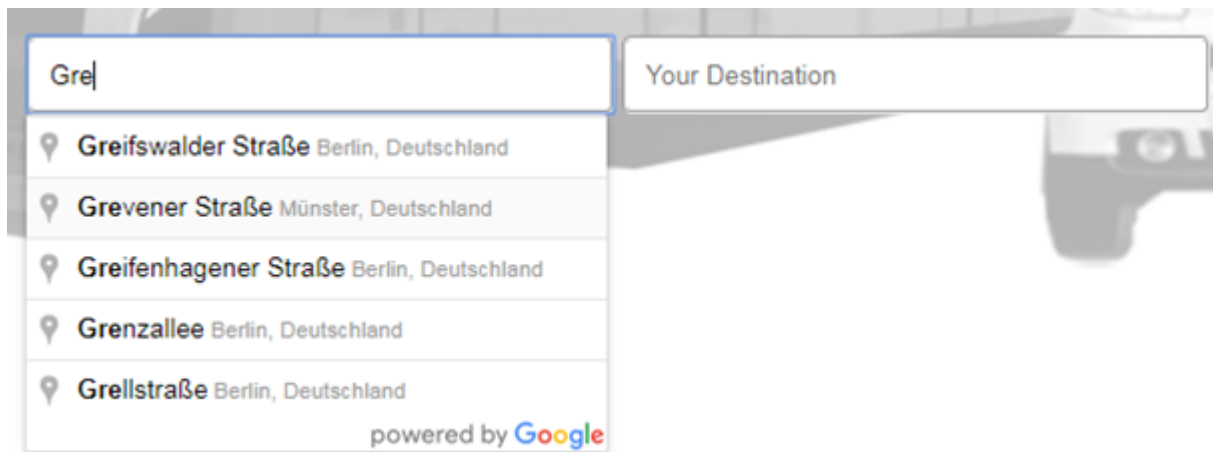


Abbildung 25: Komponente VueGoogleAutocomplete.vue

VueGoogleAutocomplete wurde über den npm-Manager installiert. Diese Komponente stellt eine Inputbox, entsprechende Styleelemente und Google-API-Funktionalitäten zur Verfügung. Durch diese Komponente können beliebige Adressen den entsprechenden Geoinformationen (Breitengrad, Längengrad) zugeordnet werden.

Es werden zwei VueGoogleAutocomplete-Komponenten verwendet. Eine für den Startpunkt, bzw. den aktuellen Standort und eine Komponente für den Zielort bzw. die Zieladresse. Die ermittelten Geoinformationen werden in den entsprechenden Properties von Appcimo.vue zwischengespeichert.

3. GetRoutes.vue

GetRoutes.vue enthält die Business-Logic des Programms.

Enthaltene Komponenten: ResultView

Funktionen:

getRoutes(): Ermittlung und Speicherung der zur Auswahl stehenden Transportmöglichkeiten mittels Promise-Funktionen. Die Promise-Funktionen dienen dabei als Zusicherung eines zurückerhaltenen Wertes durch die Google-API Calls.

GetRoute(): Gibt alle Promise-Resultate der Funktionen wieder, die den Parameter ppromises enthalten. Diese Funktion wird benötigt, um mehrere zeitgleiche asynchrone google javascript api Calls auszuwerten. Dadurch wartet die getRoutes-Funktion auf Ergebnisse, bevor die Berechnungen der erhaltenen Werte starten können.

GoogleRouteQuery(): Ermöglicht asynchrone Benutzung der Google Javascript API, um Routen zu ermitteln. Diese Funktion gibt ein Promise zurück, das den Empfang von Werten durch API Calls signalisiert.

getShortinfo(): Berechnet die Werte, die in der Shortview ausgegeben werden. Dies beinhaltet die Startzeit und die Ankunftszeit für jede der Transportmöglichkeiten. Diese Werte werden in Form eines JSON-Objekts zurückgegeben.

getDescription(): Gibt die Google-Ergebnisse der SubResultDescription wieder, um Wegbeschreibungen auf der Seiten anzeigen zu können. Außerdem werden Ergebnisse für den Fußweg anhand der Wegpunkte aussortiert, da diese falsch sein würden.

getMap(): über Sortiermechanismen werden die Ergebnisse des Google-API Calls gefiltert und zur Speicherung wiedergegeben.

transportmethod(): Formatiert die Transportmethode und gibt sie zurück.



Abbildung 26: Transportmethod in Appcimo

duration(): Die Fahrtdauer wird addiert, um ein Endergebnis für die Dauer zu erhalten. Für CarSharing-Transportmethoden werden 4 Minuten zusätzlich berechnet.



Abbildung 27: Duration in Appcimo

end(): Berechnung der Ankunftszeit. Diese Funktion wird nur für CarSharing-Angebote benötigt. Es werden 4 Minuten zusätzlich berechnet.

Ankunftszeit
15:17
15:12
15:08

Abbildung 28: Ankunftszeit in Appcimo

price(): Berechnung des Preises für den Transport. Der Preis für die öffentlichen Verkehrsmittel ist momentan noch fix.

Preis
5.25 €
2,70 €
Its free and healthy!

Abbildung 29: Preisberechnung in Appcimo

4. ResultView.vue

Die Komponente **ResultView.vue** stellt eine Zusammenfassung der ermittelten Transportmöglichkeiten in Form von ausklappbaren Leisten dar.

Transportmittel	Startzeit	Ankunftszeit	Dauer	Preis
Carsharing !	14:57	15:17	20:10min	5.25 €
Öffis	15:00	15:12	12:13min	2,70 €
Fahrrad	14:57	15:08	11:00min	Its free and healthy!

Abbildung 30: Komponente ResultView.vue

Die Ergebnisleisten werden bei einem Klick mittels eines Transition-Effekts ausgeklappt und

die Details zur jeweiligen Transportmöglichkeit abgerufen und ausgegeben. Diese Komponente enthält eine Watcher-Funktionalität, um bei mehrmaliger Benutzung der Routenermittlung stets die aktuellen Ergebnisse anzuzeigen.

Zum jetzigen Stand ist die CarSharing-Option nicht vollständig implementiert, so dass die Car2Go-Standorte simuliert werden. Ein entsprechender Hinweis wird auf der Webseite abgebildet.

Enthaltene Komponenten: SubResultMap, SubResultDescription

5. SubResultDescription.vue

Diese Komponente enthält die Wegbeschreibung einer Route vom Startpunkt bis zum Zielpunkt.

Transportmittel	Startzeit	Ankunftszeit	Dauer	Preis
Carsharing	14:57	15:17	20:10min	5.25 €

ZUSAMMENFASSUNG

MAP

Wegbeschreibung zum Ziel

Der Routenplaner für Fußgänger ist noch im Beta-Stadium. Seien Sie vorsichtig! – Auf dieser Route gibt es eventuell keine Bürgersteige oder Fußwege.

Brandenburgische Str. 57, 10707 Berlin, Deutschland

0,7 km. ca. 9 Minuten

- Auf Brandenburgische Str. nach Südosten Richtung Westfälische Str. 0,1 km
- Weiter auf Fehrbelliner Pl. 0,2 km
- Weiter auf Brandenburgische Str. 0,3 km
- Links abbiegen auf Gieselerstraße
Das Ziel befindet sich auf der rechten Seite. 13 m

Gieselerstraße 30, 10713 Berlin, Deutschland

Kartendaten © 2017 GeoBasis-DE/BKG (©2009), Google

Gieselerstraße 30, 10713 Berlin, Deutschland

2,6 km. ca. 8 Minuten

- Auf Gieselerstraße nach Südwesten Richtung Brandenburgische Str. starten 13 m
- Rechts abbiegen auf Brandenburgische Str. 0,2 km
- Rechts abbiegen auf Wegenerstraße 32 m
- Links abbiegen auf Sächsische Str. 0,2 km

Abbildung 31: Komponente SubResultDescription.vue

Funktionen:

updateDescription(): Ermittelt über einen API-Call an den Google-Direction-Service die Wegbeschreibung einer angefragten Route und rendert diese innerhalb der Komponente. Diese Funktion wird ausgeführt, sobald die Komponente aufgerufen wird.

6. SubResultMap.vue

Die GoogleMap wird in dieser Komponente dargestellt.

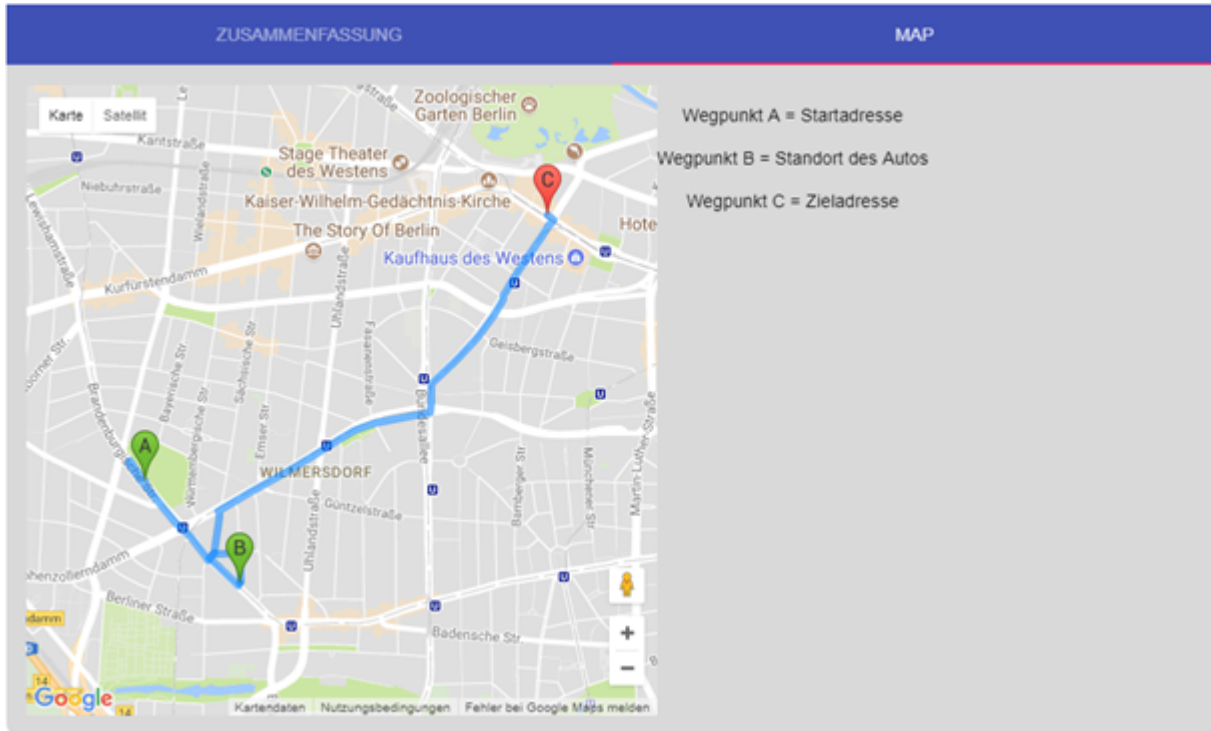


Abbildung 32: Komponente SubResultMap.vue

Funktionen:

initMap(): Die GoogleMap wird beim Start der Komponente gerendert und anhand von vorher festgelegten Parametern (Zoom-Level der Karte, Initialstandort, ziehbare Karte) dargestellt. Die direkte Einbindung des `directionDisplay` ermöglicht es, die Route anhand von farbigen Pfaden und Wegpunkten darzustellen.

5.1.4 Sequenzdiagramm

Das Sequenzdiagramm stellt einen einfachen schematischen Ablauf der Routensuche dar. Die Funktionen sind nicht deckungsgleich zu den Funktionen der Komponenten benannt, da diese teils sehr verschachtelt sind und nicht zum einfachen Verständnis des Prozesses beitragen würden.

Vielmehr können die hier verwendeten Funktionsnamen als Funktionsbox gesehen werden, die alle Javascript-Funktionen beinhalten, die zur Realisierung des beabsichtigten Rückgabewertes dienen.

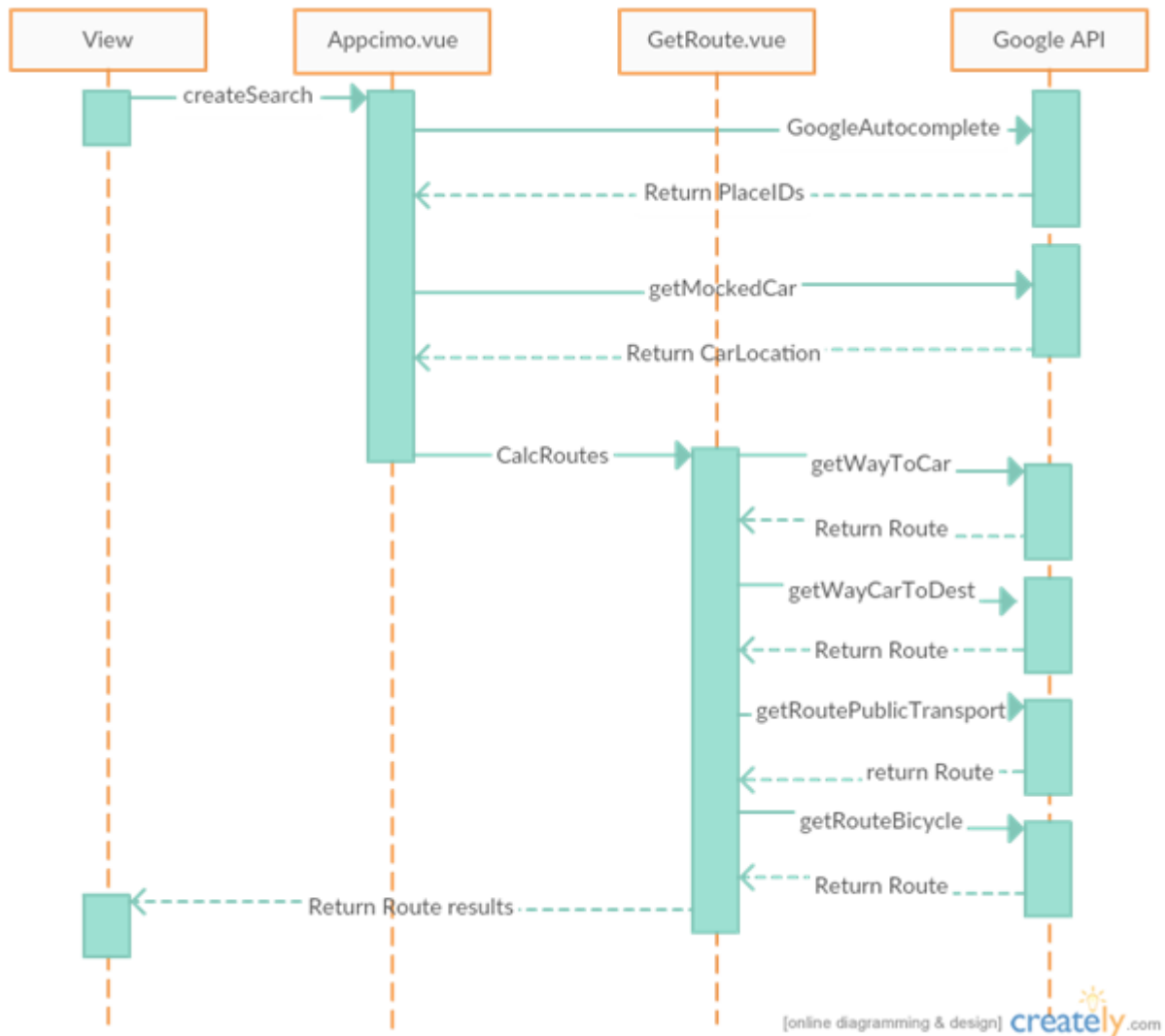


Abbildung 33: Sequenzdiagramm

5.2 Qualitätsmanagement

5.2.1 Refactoring

Um einen wartbaren Code zu schaffen, wurde regelmäßig ein Code-Refactoring betrieben. Als Beispiel dient hier der erste Prototyp. Er funktionierte als Feldtest, war aber nur sehr schwer testbar und deshalb qualitativ nicht gut genug. Daraufhin wurden zwei große Refactoring-Iterationen durchgeführt, um wart- und testbaren Code zu erhalten.

5.2.2 Unit Tests

Unit-Tests sind bei dem Programm ein wichtiger Bestandteil um die gewünschten Kernfunktionen der Applikation zu erhalten. Ein Refactoring sollte hierbei die Business Logic des Programms nicht negativ beeinträchtigen.

Deshalb wurden entsprechende Unit Tests für die Kernfunktionen der Applikation implementiert.

5.2.3 Build-Management

VueJS Applikationen sind in ihrem Entwicklungszustand nicht im Browser lauffähig. Sie müssen einem Build-Prozess unterzogen werden, bei dem die Vue-Dateien in HTML-, CSS- und Javascript-Dateien übersetzt werden.

Dies wird bei der Applikation Appcimo durch NodeJS und den entsprechenden npm Skripten gelöst. Der Build-Prozess wird in der Build-Konfigurationsdatei `build.js` definiert. Diese basiert auf weiteren Konfigurationsdateien.

Der Aufruf des Build-Prozesses wird durch npm scripts ausgelöst. Diese sind wiederum in einer Konfigurationsdatei `package.json` definiert. Hier werden die Stärken des einheitlichen JSON-Format genutzt. Es ist durch seinen strukturierten Aufbau für viele Applikationen gut nutzbar.

Die in der `package.json` hinterlegten Ziele starten z.B. die `build.js`-Datei und können auf eventuelle Abhängigkeiten verweisen. So ist es zum Beispiel möglich, die unit tests als Pre-Build Prozess zu konfigurieren (wie man es als Dependencies aus z.B. Ant kennt). Hierbei wird der Build erst ausgeführt wenn das Testskript eine positive Rückmeldung gibt.

5.2.4 Continuous Integration

Das Prinzip der Continuous Integration steuert einen großen Teil zur Qualität der Software bei. Hier wird das Online-Tool CircleCI genutzt, da es in Github integrierbar ist und die Build-Konfigurationen aus den `package.json` Konfigurationsdateien auslesen kann.

Wenn ein `git-push` zu dem Github-Repository durchgeführt wird, startet CircleCI den Continuous Integration Prozess.

Zunächst wird die Build-Umgebung abhängig der `package.json` Konfiguration-Datei hergestellt. Danach erfolgt der Test und Build der Software. Hier werden die definierten npm Skripte ausgeführt. Im Fall von Appcimo wird zunächst der Test und abhängig davon der Build ausgeführt. Sind der Test und der Build erfolgreich, wird der definierte Deploy Prozess zu Amazon S3 durchgeführt. Ist der Test hingegen nicht erfolgreich werden die App-Owner darüber benachrichtigt und der fehlerhafte Code wird nicht zur Live-Umgebung übertragen.

6 Zusammenfassung und Ausblick

Mit Abschluss des Semester-Projektes Appcimo für das Modul Softwaretechnik-Projekt wurde ein funktionierender und stabil-laufende Prototyp entwickelt. Durch die erstellten Testfälle wird die korrekte Ausführung der Applikation sichergestellt. Die Einbindung der Continuous Integration Platform CircleCI ermöglicht das Ausführen und Überwachen des Software-Builds inklusive der Tests und ermöglicht, das Programm auf einem Amazon AWS Server zu hosten. Der Software-Prototyp enthält die zuvor im Team abgestimmten Features.

Für die Zukunft ist jedoch noch eine Weiterentwicklung des Programms geplant. Unter anderem sollen weitere Carsharing-Anbieter als Transportmöglichkeit integriert werden. Auch die Routenanfrage der öffentlichen Verkehrsmittel soll erweitert werden, eine detaillierte Auswahl an Fahrzeugtypen anbieten können (U-bahn, S-bahn, Tram, Bus, Regionalbahn) und Preise exakt berechnen können.

Das Template der Wegbeschreibung wird momentan von Google gestellt. Da alle Informationen des Weges als Objekt geliefert werden, ist es jedoch möglich eine individuelle, stilistisch auf das Programm abgestimmte Wegbeschreibung zu erstellen.

Zur Positionsermittlung wird momentan die Startadresse abgefragt und ausgewertet. Es soll zukünftig - bei Einwilligung - eine automatische Standorterfassung erfolgen, die die Koordinaten der Person als Startadresse verwendet, um noch schneller, flexibler und genauer Routen abzufragen.

Der Grundstein für eine sinnvolle und innovative Webapplikation ist gelegt. Mit den festgelegten Maßnahmen für die Zukunft soll die Applikation ein großes Publikum ansprechen und den besten Service für die Zielerreichung in Großstädten liefern.

7 Anhänge

Abbildungsverzeichnis

Abb. 1:	Scrum Infografik	6
Abb. 2:	Taiga.io Dashboard	8
Abb. 3:	Taiga.io Admin-Members	8
Abb. 4:	Taiga.io HipChat-Integration	9
Abb. 5:	Taiga.io HipChat-Integration	10
Abb. 6:	Taiga.io Backlog	10
Abb. 7:	Taiga.io Sprinttasks	11
Abb. 8:	Taiga.io Sprints 1-3	11
Abb. 9:	Taiga.io Sprints 4-6	12
Abb. 10:	IDE Webstorm Logo	17
Abb. 11:	Texteditor Atom Logo	18
Abb. 12:	MVVM Schema	18
Abb. 13:	Node.JS Logo	19
Abb. 14:	Node.JS Package Manager Logo	19
Abb. 15:	GitHub Logo	22
Abb. 16:	CircleCI Logo	23
Abb. 17:	CircleCI Starteseite	24
Abb. 18:	Ausschnitt der Appcimo-Testauswertung	25
Abb. 19:	Application Build - CircleCI	25
Abb. 20:	Google API's	26
Abb. 21:	Google API Dashboard	26
Abb. 22:	MVVM Appcimo	27
Abb. 23:	Komponentendiagramm	28
Abb. 24:	Komponente Appcimo.vue	29
Abb. 25:	Komponente VueGoogleAutocomplete.vue	30
Abb. 26:	Transportmethod in Appcimo	31
Abb. 27:	Duration in Appcimo	31
Abb. 28:	Ankunftszeit in Appcimo	32
Abb. 29:	Preisberechnung in Appcimo	32
Abb. 30:	Komponente ResultView.vue	32
Abb. 31:	Komponente SubResultDescription.vue	33
Abb. 32:	Komponente SubResultMap.vue	34
Abb. 33:	Sequenzdiagramm	35

Tabellenverzeichnis

Tab. 1:	funktionale Anforderungen	13
Tab. 2:	Projektanforderungen	14
Tab. 3:	Nicht-funktionale Anforderungen	14
Tab. 4:	Sprint 1	14
Tab. 5:	Sprint 2	15
Tab. 6:	Sprint 3	15
Tab. 7:	Sprint 4	15
Tab. 8:	Sprint 5	16
Tab. 9:	Sprint 6	16