Volkswagen

Infotainment - Navigation

Projektdokumentation

INTERNAL
INTERN

Inhalt

Abbildungsverzeichnis

Tabellenverzeichnis

Plagiatserklärung / Klausel

Mit der Abgabe dieser Dokumentation bestätigen wir, David Justo und Thies Bücker, dass das Projekt sowie die Dokumentation eigenständig von uns angefertigt wurde. Uns ist bewusst, dass unsere Arbeit bei Täuschungshandlungen bzw. Ordnungsverstößen mit „null" Punkten bewertet werden kann. Wir sind weiter darüber aufgeklärt worden, dass dies auch dann gilt, wenn festgestellt wird, dass unsere Arbeit im Ganzen oder zu Teilen mit der eines anderen Prüfungsteilnehmers übereinstimmt. Es ist uns bewusst, dass Kontrollen durchgeführt werden.

Ausgangslage

Die zukünftige Generation der Fahrzeuge der Volkswagen AG wird überwiegend elektrisch angetrieben. Dies verändert das grundlegende Konzept der Ladeinfrastruktur. Somit soll eine Applikation erstellt werden, welche die Ladestationen für Elektrofahrzeuge stärker berücksichtigt. Diese soll nicht nur Routen planen können, sondern auch in der Lage sein, diverse Ladepunkte einzusehen und zu erstellen.

Zielsetzung

1. Ein JavaFX-Client stellt einen innovativen Routenplaner bereit.
2. Der Anwender kann eine Route planen.
    a. Dabei kann dieser einen Startpunkt und einen Zielpunkt angeben.
    b. Für die geplante Route können diverse Ladepunkte in der Umgebung eingesehen werden.
3. Falls der Anwender neue, nicht-persistierte Ladestationen während der Reise findet, können diese im Client eingetragen werden.
4. Während des gesamten Anwendungszeitraums soll dem Benutzer die aktuelle Uhrzeit und das aktuelle Datum angezeigt werden sowie die Möglichkeit, das System an- bzw. auszuschalten, gegeben sein

Projektplanung

Zeitplanung

Die Umsetzung des Projektes erfolgte in Zweierteams im Zeitraum vom 26.01.2022 bis 04.02.22. Daraus ergibt sich bei einem siebenstündigen Arbeitstag ein Zeitbudget von 112 Stunden. Dies wurde wie folgt geplant:

Phase / Arbeitspaket	**Zeit in Stunden**
Vorbereitung	**3**
- Einrichten der Entwicklungsumgebung	3
Analyse	**8**
- Evaluierung der Stammdaten	3
- Soll-Analyse der technischen Anforderungen	2
- Identifikation von Arbeitspaketen der Implementierung	3
Entwurf	**17**
- Erstellen von Mockups der Ansichten	6
- Entwicklung eines Datenbankmodells	5
- Planung der Softwarearchitektur	6
Implementierung	**59**
- Arbeitspaket Hauptmenü	2
- Arbeitspaket Stammdaten	4

- Arbeitspaket Karte einbinden	5
- Arbeitspaket Routenplanung	13
- Arbeitspaket Stationen filtern und anzeigen	10
- Arbeitspaket CSV einlesen	10
- Arbeitspaket Uhr	2
- Arbeitspaket Standby Ansicht	3
- Arbeitspaket "Zurück" Knopf	2
- Styling der Applikation	4
- Finales Refactoring und Packaging	4
Dokumentation	**15**
Puffer	**10**
Gesamt	**112**

Tabelle 1: Geplante Zeit pro Arbeitspaket

Kollaboration

Für die Zusammenarbeit wurde einerseits der Ansatz des Pair-Programming, bei dem zwei Entwickler zusammen an einem Bildschirm arbeiten um wichtige Software-Komponenten direkt im Team abstimmen zu können. Andererseits wurde auch auf konzentrierte Einzelarbeit an dafür vorgesehenen Komponenten gesetzt. Allerdings standen die Entwickler auch hierbei in ständigem Austausch über einen offenen Kommunikationskanal. Für die Versionierung wurde BitBucket verwendet. Grundsätzlich wurde hierbei jedes Feature in einem eigenen Branch entwickelt und anschließend mit dem Master Branch zusammengeführt.

Am Anfang des Projektes wurde die Entwicklungsumgebung angepasst, sowie eine passende Softwarearchitektur erstellt, für die passende Bibliotheken und Komponenten identifiziert wurden. Der Großteil der Implementierung erfolgt in den ersten fünf Projekttagen. An den letzten beiden Tagen wurde das Augenmerk verstärkt auf die Dokumentation gelegt. Dies lässt sich auch aus der Übersicht der Commits ableiten. Der 3. Februar wurde mit einem abschließenden Refactoring verbracht.

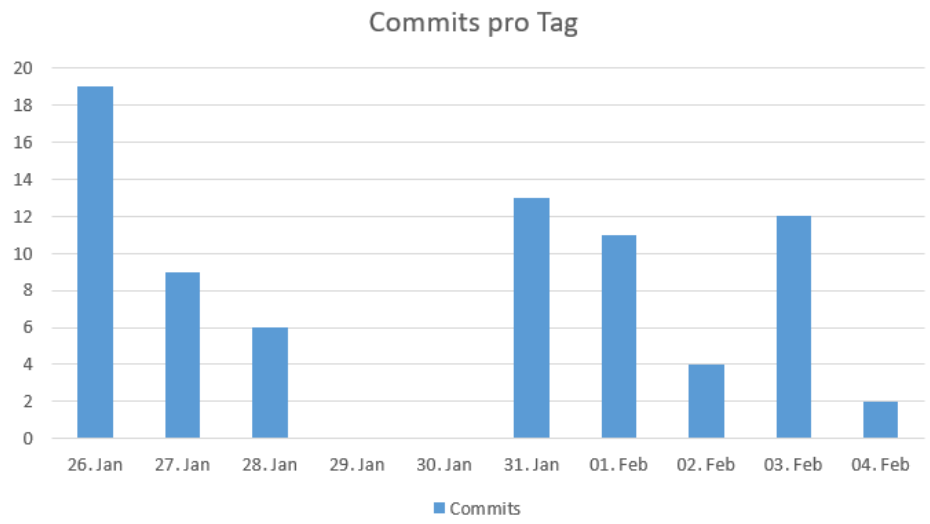

Abbildung 1: Zeitliche Verteilung der Commits

Für einen groben Überblick über den Stand des Projektes zu gewähren wurde der Projektantrag in eine Checkliste übersetzt, in der implementierte Features abgehakt wurden.

Durchführung

Design

Design Prototypen wurden als Mockups erstellt. Die hier veranschaulichten Mockups sind Entwürfe und können von dem aktuellen Stand der Applikation abweichen. Bei der Entwicklung des Designs wurde Farbtöne verwendet, die auch in anderen Volkswagen-Anwendungen Verwendung finden.

Die Applikation zeigt im Standby Modus das aktuelle Datum, die Uhrzeit, sowie einen Einschalter an. Daraufhin wird die zuletzt geladene Ansicht, oder, bei erstmaligem Anschalten das Hauptmenü angezeigt.

Abbildung 2: Mockup StandBy-Bildschirm

Nach dem Aufwecken der Applikation wird das Hauptmenü angezeigt, in welchem sich neben den implementierten Punkten Navigation und Data auch zwei Knöpfe für mögliche Erweiterungen für Telefon und Radio befinden.

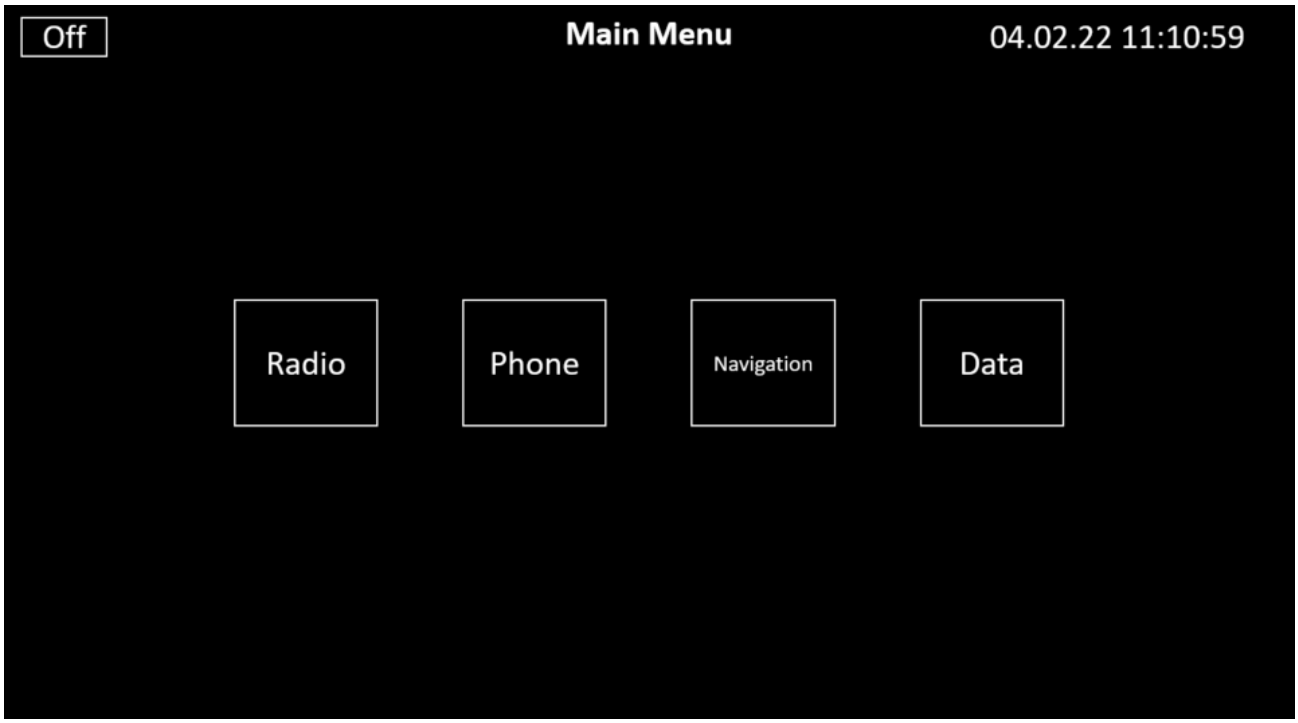

Abbildung 3: Mockup Hauptmenü

Über den Menüpunkt Data gelangt man in die Stammdatenverwaltung. Hier können Details zu gespeicherten Ladestationen und anderen gespeicherten Orten aufgenommen, gespeichert und gelöscht werden. Neu eingegebene Orte müssen mindestens einen Namen und ein Koordinatenpaar in einem validen Dezimalformat haben. Ist dies nicht der Fall, soll der Nutzer mittels Pop-Up darauf hingewiesen werden. Zusätzlich wurden für Zahleneingaben Textfelder geplant, in denen nur Dezimalzahlen, beziehungsweise ganze Zahlen eingetragen werden können. Die Ansichten für Stationen und sonstige Orte bauen auf der gleichen Vorlage auf. Mittels Klick in die Liste der gespeicherten Orte soll der Nutzer einen Ort aus der Datenbank laden können. Dadurch wird der aktuelle Ort auf der Karte angezeigt, sowie dessen Informationen in die entsprechenden Felder geladen. Orte können über die Schaltflächen gespeichert oder gelöscht werden. Um einen neuen Ort zu erstellen, kann der Nutzer in die Karte klicken und einen Namen vergeben.

Über die Schaltflächen in der oberen linken Ecke der Applikation soll in den Standby Modus gewechselt werden, sowie auf die zuletzt geladene Seite.

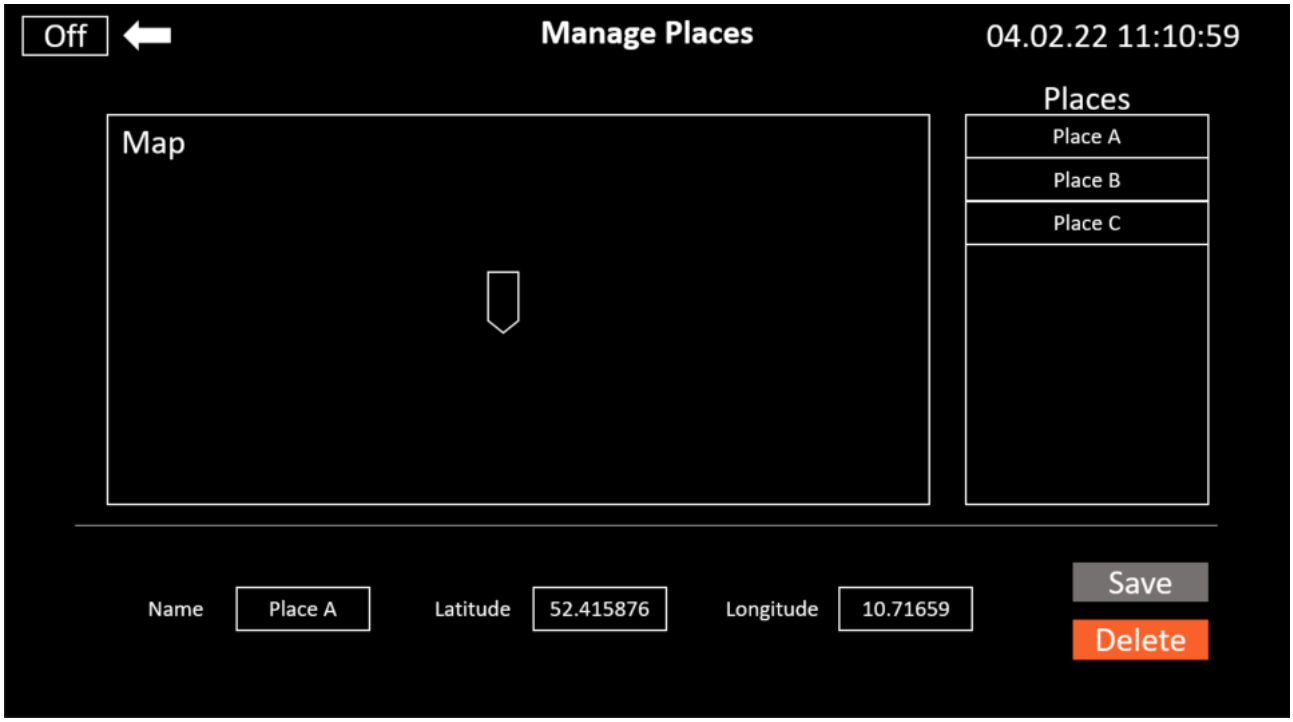

Abbildung 4: Mockup Verwaltung von Orten

Über das Hauptmenü gelangt der Nutzer ebenfalls in die Navigation. Diese besteht aus einer Karte, sowie zwei Listen für gespeicherte Orte und Routen. Diese können jeweils durch einen Klick auf der Karte angezeigt werden. Um neue Routen zu berechnen, können ein Ausgangspunkt und ein Endpunkt auf der Karte ausgewählt, und über die „Import from Map" Schaltflächen als Start-, beziehungsweise Endpunkt gesetzt werden. Dann kann der Nutzer eine Route berechnen, die dann automatisch auf der Karte angezeigt wird. Diese Route kann über Save Route mit einem Namen versehen und gespeichert werden. Zusätzlich ist es möglich, dass sich der Nutzer entlang der geladenen Route Ladestationen, die einen definierten Abstand nicht überschreiten, anzeigen lässt. Die gleiche Funktionalität gibt es auch für alle anderen auf der Karte angezeigten Orte innerhalb Deutschlands.

Die angezeigten Ladestationen können per Klick durch einen Absprung in die oben beschriebene Stammdatenverwaltung editiert werden.

Schaltflächen, die nicht verwendet werden können, z.B. „Stations around Marker", wenn kein Marker gesetzt ist, werden ausgegraut.

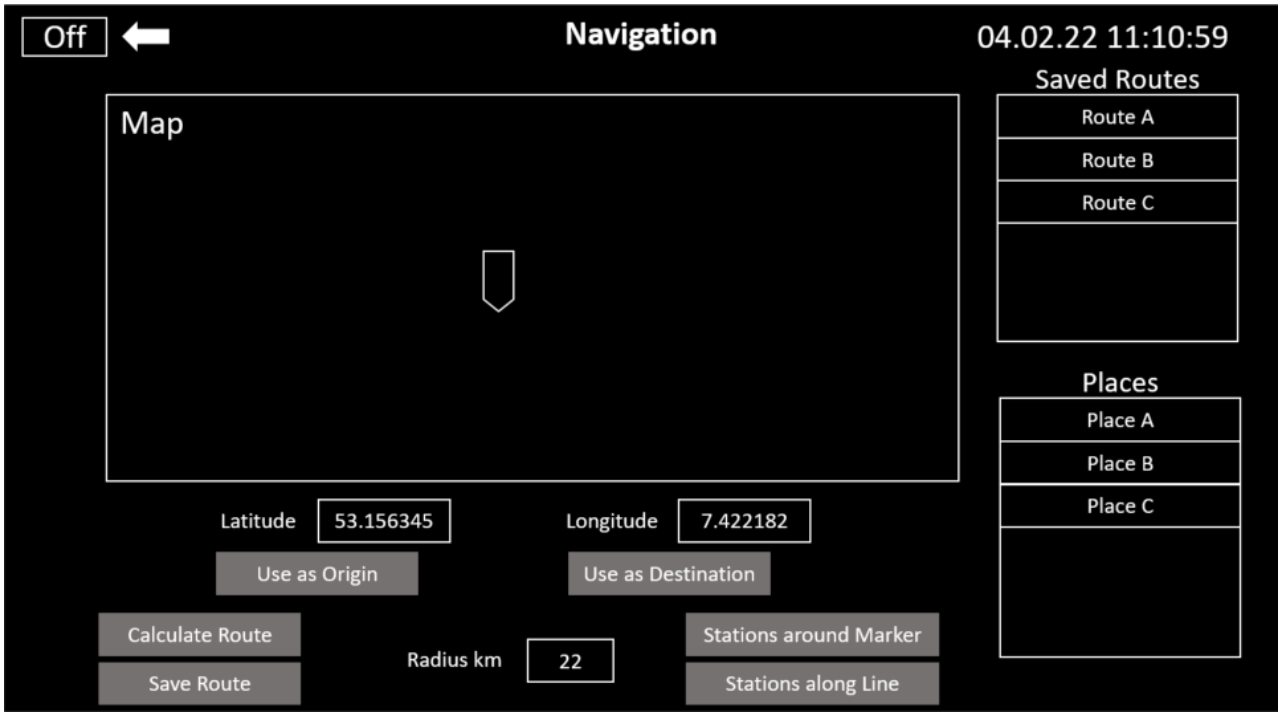

Abbildung 5: Mockup Navigation

Datenmodell

Um die beschriebenen Funktionaltäten zu erfüllen wurden drei Datenbankentitäten erstellt. Diese speichern einen beliebigen Ort, eine Route, oder Informationen zu einer Ladestation. Orte bestehen aus einer generierten ID, einem Namen und den Geo-Koordinaten für Längen- und Breitengrad. Routen bestehen aus denselben Attributen, und haben die Geokoordinaten für Start- und Endpunkt. Für die Ladestationen wurde eine vom Auftraggeber zur Verfügung gestellte CSV Datei (Kommaseparierte Wertetabelle) als Vorlage verwendet. Diese Datei wird automatisch in die Datenbank gelesen, wenn die Station-Tabelle bei Start der Applikation leer ist. Um die darin enthaltenen Daten abzuspeichern, muss ein Übersetzer erstellt werden, die die CSV Daten in das gewünschte Format überführen. Hierbei wurden die im Schaubild angezeigten Datentypen verwendet. Werte, die in der CSV Datei nicht diesem Datentyp entsprachen wurden mittels Parser in diesen überführt. Beispielsweise bedeutet „kostenlos“, dass „hasFee“ false ist, 22000 Watt wird zu 22 kW und “null” deutet auf einen fehlenden Wert hin. Wenn bei den „Amperage“ und „Voltage“-Werten mehrere Angaben gemacht wurden, wurde der höchste Wert verwendet. Zudem wurden sämtliche Tabellen mit Primärschlüsseln versehen. Diese wurde als Integer Wert festgelegt und automatisch generiert. In der Station Tabelle wurde ein Varchar verwendet, da in der mitgelieferten CSV Datei ebenfalls ein Varchar als ID verwendet wurde.

Relationen sind im aufgeführten Modell nicht nötig. Abbildung 6 zeigt die entworfene Datenstruktur.

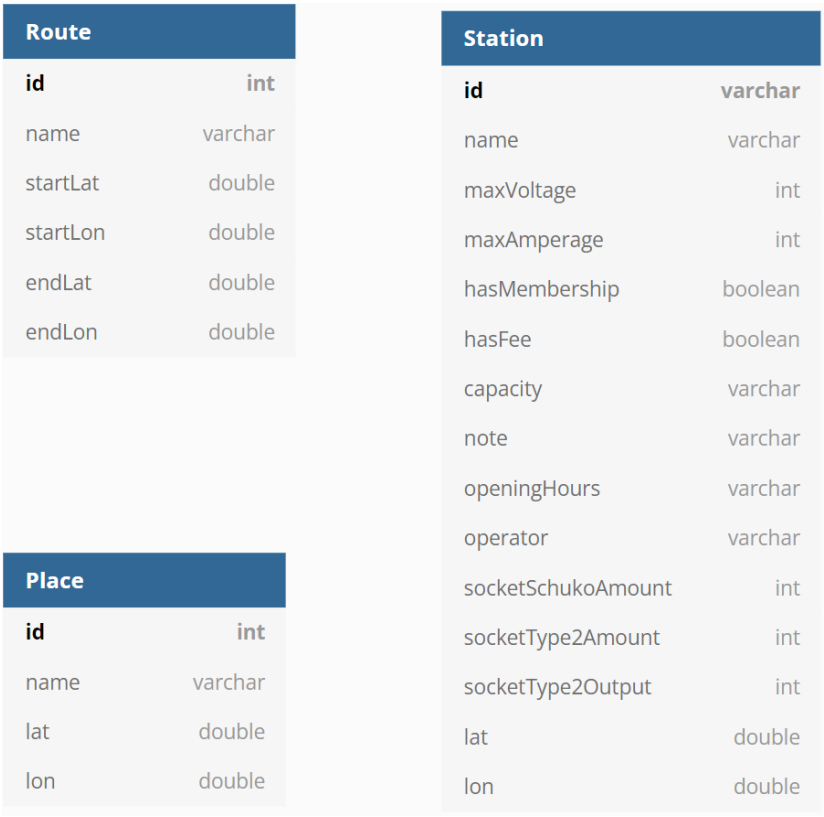

Abbildung 6: Datenbankmodell

Implementierung der Architektur

Die Applikation wurde von Anfang an monolithisch aufgebaut. Obwohl Spring Boot, welches hier für die Behandlung der Entitäten und ihrer Übersetzung via ORM in die Datenbank zuständig ist, üblicherweise in einem Client-Server-Modell eingesetzt wird, wurde es in diesem Projekt direkt mit dem JavaFX-Frontend innerhalb einer Applikation verbunden. Hierbei kam eine Bibliothek namens FxWeaver zum Einsatz, welche aus Spring Boot bekannte Architekturkonzepte wie Dependency Injection in und für JavaFX-Controller erlaubt.

Zunächst wurden die im Kapitel „Datenmodell" beschriebenen Entitäten definiert. Die Grundfunktionalitäten wurden mit Testfällen beschrieben und im Anschluss wurden Repositories und Services für die Übersetzung in die Datenbank zur Erfüllung dieser Testfälle implementiert. Für den vom Auftraggeber gelieferten Datensatz an Ladestationen wurde eine Übersetzungsebene implementiert. Als Datenbank kommt eine durch Spring Boot initialisierte, persistente H2-Datenbank zum Einsatz.

Anschließend wurden mit SceneBuilder Views erstellt, die von den davor umgesetzten Services Gebrauch machten. Die Struktur der JavaFX-Komponenten wurde hierbei im Model-View-Controller-Pattern (MVC) angelegt. Die Views wurden als FXML-Dateien eingebunden, die jeweilen Controller beinhalten die Geschäftslogik und verarbeiten die Modelle zur Ansicht und Behandlung im Frontend.

Implementierung der Funktionalitäten

Die Umsetzung der Grundarchitektur, mit der die konkreten Kundenanforderungen umgesetzt werden sollen, war nun abgeschlossen. Es wurde nun damit begonnen, in klar abgegrenzten Feature-Branches an den verschiedenen Anforderungen zu arbeiten.

Für die Einbindung einer Karte wurde eine Bibliothek namens mapjfx verwendet, die ein JavaFX-Element einer auf OpenStreetMap basierenden Karte zur Verfügung stellt. Nach einer ersten Orientierung anhand der Dokumentation und des Quellcodes der Bibliothek wurde eine View implementiert und die Funktionalität, Markierungen auf der Karte zu erstellen, umgesetzt.

Die bestehenden Views wurden um die benötigten Ein- und Ausgabekomponenten erweitert und optisch angeglichen. Hierbei wurde die Anordnung der Elemente auf Basis der Mockups nachgebildet. Die verschiedenen Views wurden miteinander vernetzt, indem der Nutzer über eine Menüführung durch die verschiedenen Views navigieren und wieder zurückkehren konnte. Hierbei bleibt das Hauptfenster mitsamt der Menüleiste am oberen Rand jederzeit bestehen im Inhaltsbereich in der Mitte wurden die verschiedenen Views angezeigt.

Zeitgleich wurden für die Kalkulation von Distanz zwischen Punkten, Umkreissuche und Distanz entlang eines Pfades benötigte geografische Berechnungsmethoden umgesetzt. Anhand dieser Methoden konnte nun eine Luftlinie zwischen zwei Punkten auf der Karte eingezeichnet und eine Berechnung der Fahrtdauer durchgeführt werden. Um die Nutzbarkeit der Applikation über diese Anforderung hinaus zu optimieren und die Navigationsfunktion noch realistischer umzusetzen, wurde zusätzlich zur Luftlinie eine reale PKW-Route eingezeichnet. Hierfür wurde eine quelloffene und frei verwendbare Schnittstelle im Internet angesprochen, die eine Route zwischen zwei Koordinaten berechnet und zurückliefert.

Für ein interaktives Nutzererlebnis wurden die auf der Navigationsansicht durchgeführten Mauseingaben verwertet, sodass sich die Navigationsansicht vollständig mit der Maus bedienen lässt, ohne bspw. die Eingabe von Koordinaten über die Tastatur zu benötigen. Hierfür wurden Sprachmechanismen wie Properties und Bindings sowie das Observer-Architekturmuster zum Einsatz. Diese Funktionsweise wurde auch auf die Stations- und Ortsverwaltung übertragen. Für den detaillierten Datensatz der Stationen wurde das Builder-Architekturmuster eingesetzt.

Die Übersetzung der mitgelieferten Datensätze an Stationen wurde parallel zum wachsenden Funktionsumfang der Applikation mit weiteren Übersetzungsmodellen erweitert, bis sich die Stationsentität schlussendlich mit dem ursprünglich geplanten Datenmodell glich.

Nach der Implementierung weiterer Funktionsanforderungen, wie dem Standby-Menü und der Anzeige der Uhrzeit, wurde das in den Mockups geplante Farb- und Designschema mithilfe von Stylesheets angepasst.

Nachdem die Autoren den erzielten Funktionsumfang, dessen Korrektheit und die Optik und Nutzerfreundlichkeit der Applikation am Ende der Implementierungsphase evaluiert haben, wurde ein abschließendes gemeinsames Refactoring sowie die Vervollständigung der Quellcodedokumentation durchgeführt.

Die Applikation kann mit der erstellten .jar Datei ausgeführt werden.

Test / Qualitätssicherung

Die jeweilig umgesetzten Teilfunktionalitäten einer Anforderung wurden regelmäßig in das Versionierungssystem eingereicht. Nach jedem Abschluss einer Anforderung wurde ein gemeinsames Code-Review durchgeführt und ggf. erkannte Mängel beseitigt. Hiernach wurde die evaluierte Feature-Branch auf den Hauptzweig in Git migriert. So wurde sichergestellt, dass der Entwicklungsstand dem „master"-Branch klar und iterativ nachzuvollziehen ist und dass zu keiner Zeit unfertiger oder nicht funktionaler Quellcode auf dieser Ebene liegt. Diese Vorgehensweise ist im Anhang in der Abbildung 8: Auszug des Git Graphs deutlich zu erkennen.

Ebenso wurden Komponententests implementiert. Zwar wurde aufgrund der eingeschränkten Prüfbarkeit von JavaFX durch Unit-Tests nicht streng nach dem Konzept der testgetriebenen Entwicklung gearbeitet, allerdings wurden, wo immer möglich und sinnvoll, zuerst Testfälle definiert und diese danach durch Implementierung der Funktionen erfüllt. Wenn in einer Aufgabe die Anwendung des TDD nicht möglich war, wurde versucht, im Nachhinein Tests zu schreiben, die die implementierten Funktionen reproduzierbar und vollständig prüfen.

Um ein grundsätzliches Maß an stilistischer und syntaktischer Korrektheit sicherzustellen, wurde in der Entwicklungsumgebung eine statische Codeanalyse mit SonarLint durchgeführt. Die Entwicklungsumgebung wurde mit einer privaten Instanz von SonarQube verbunden, um einen Überblick über Metriken wie Testabdeckung neu geschriebener Funktionen zu behalten. Ein beispielhafter Screenshot ist im Anhang unter Abbildung 12: Codeanalyse mit SonarQube zu finden.

Das fertige Projekt wurde mithilfe von Maven zu einer ausführbaren Datei mitsamt ihren Abhängigkeiten verpackt. Diese wurde auf mehreren Systemen und mit unterschiedlichen Versionen der Java 8 JDK getestet und das Endergebnis wurde für reproduzierbar befunden.

Projektabschluss

Soll/Ist-Vergleich

Die Autoren bewerten das Projektziel als gänzlich erfüllt. Es wurde ein innovativer Routenplaner mit integrierter Unterstützung für Ladepunkte entwickelt. Im Folgenden sollen die Teilanforderungen im Detail evaluiert werden.

In der Nutzeroberfläche soll eine Karte angezeigt werden und über Definition von Start- und Endpunkten via Eingabe von Koordinaten eine Route geplant werden. Innerhalb einer von dem Nutzer ausgewählten Bereich soll nach Ladestationen, entlang einer Route oder um einen Punkt herum, gesucht werden können. Die Route soll hierbei durch eine gerade Verbindungslinie zwischen den beiden Punkten dargestellt werden.

Im Istzustand sind die oben genannten Funktionen gänzlich umgesetzt. Darüber hinaus kann der Anwender mittels Klicks auf der Karte und zwei entsprechenden Buttons Koordinaten direkt aus dem Kartenelement laden. Der Nutzer kann entweder mit einem Schieberegler oder einem Eingabefeld einen Radius einstellen, in dem nach Stationen gesucht wird. Neben der Verbindung als Luftlinie wird auch eine realistische, mit dem PKW befahrbare Strecke angezeigt. Der Nutzer hat nun die Möglichkeit, sowohl entlang der Luftlinie als auch entlang der Route nach Stationen zu Suchen.

In einer weiteren Anforderung wurde gefordert, dass der Nutzer mit Klick auf eine Ladestation eine Detailansicht öffnet, in der Informationen zu dieser Station zu finden sind.

In der tatsächlichen Applikation kann der Nutzer in dieser Detailansicht auch Informationen anpassen, Stationen löschen und andere Stationen anwählen.

Alle weiteren Teilanforderungen, wie die Implementierung eines Standby-Bildschirms, einer Uhrzeitanzeige und einer nutzerfreundlichen Menüführungen wurden exakt nach Vorgabe umgesetzt und bedürfen keiner weiteren Erläuterung.

In der Zeitplanung wurde punktuell von der geplanten Struktur abgewichen, die grundsätzliche Gewichtung der Projektphasen Analyse, Design, Implementierung und Dokumentation wurde jedoch eingehalten.

Die Planung des Datenbankschemas nahm weniger Zeit in Anspruch als ursprünglich angenommen, da verschiedenen Datentypen keine Verknüpfungen voraussetzten. Die hier eingesparte Zeit konnte in die Planung der Architektur reinvestiert werden.

Einige Arbeitsabschnitte stellten retrospektiv betrachtet eine größere Herausforderung dar und benötigten deswegen mehr Zeit. Hier sei die Handhabung der Bibliothek *mapjfx* zu erwähnen, deren Dokumentation teils unvollständig war. Dies führte in vielen Teilabschnitten der Implementierungsphase zu Verzögerungen. Besonders kritisch anzumerken sei hier das Auftreten eines Laufzeitfehlers in der Abschlussphase der Implementierung. Die Initialisierung der Karte funktionierte nicht immer, und das Problem war nicht durchgehend zu reproduzieren. Es stellte sich heraus, dass bei der Nutzung von vielen Funktionen dieser Bibliothek eine klare Reihenfolge bei der Initialisierung dieser eingehalten werden muss. Diese Fehlersuche und Behebung verlängerte die Abschlussphase deutlich.

Als weitere Herausforderung stellte sich die Berechnung von Distanzen und weiteren Metriken auf einem geographischen Koordinatensystem dar. Hierfür wurde viel Zeit in Recherche investiert, bis die Funktionen im Code ein mathematisch korrektes Ergebnis zurücklieferten.

Phase	**Soll**	**Ist**	**Differenz**
Vorbereitung	3	3	0
Analyse & Entwurf	8	8	0
Implementierung	59	61	+ 2
Dokumentation	15	21	+ 4
Puffer	10	4	- 6
Gesamt	112	112	0

Tabelle 2: Zeitplanung Soll vs. Ist

Fazit

Für ein abschließendes Fazit sei positiv anzumerken, dass die gestellten Anforderungen komplett implementiert und darüber hinaus noch weitere Funktionen implementiert wurden. Wie in Abbildung 7: Vergleich Navigation zwischen Mockup und Applikation zu erkennen, wurde die Applikation sehr nah an den ursprünglich entworfenen Mockups orientiert. Das gelungene „Look-and-Feel" der Applikation sowie der abgeschlossene Funktionsumfang lässt die Autoren ein sehr positives Gesamtfazit schließen.

Ausblick

Die erstellte Anwendung ist in Zukunft erweiterbar, insbesondere mit Blick auf die Verwendung innerhalb eines elektrischen Fahrzeugs. Beispielsweise können die Funktionen für das Radio und das Telefon implementiert werden, sowie eine Anzeige, die die Restlaufzeit der Batterie angibt und automatisch eine der angezeigten Ladestationen in Abhängigkeit zur Restlaufzeit in die Route einbindet. Die Datenbank kann dank der gewählten Repositories einfach um neue Felder erweitert werden. Außerdem ist die Anwendung derzeit nur für Orte in Deutschland freigeschaltet. Dies kann durch Einfügen entsprechender Ladestationen erweitert werden.

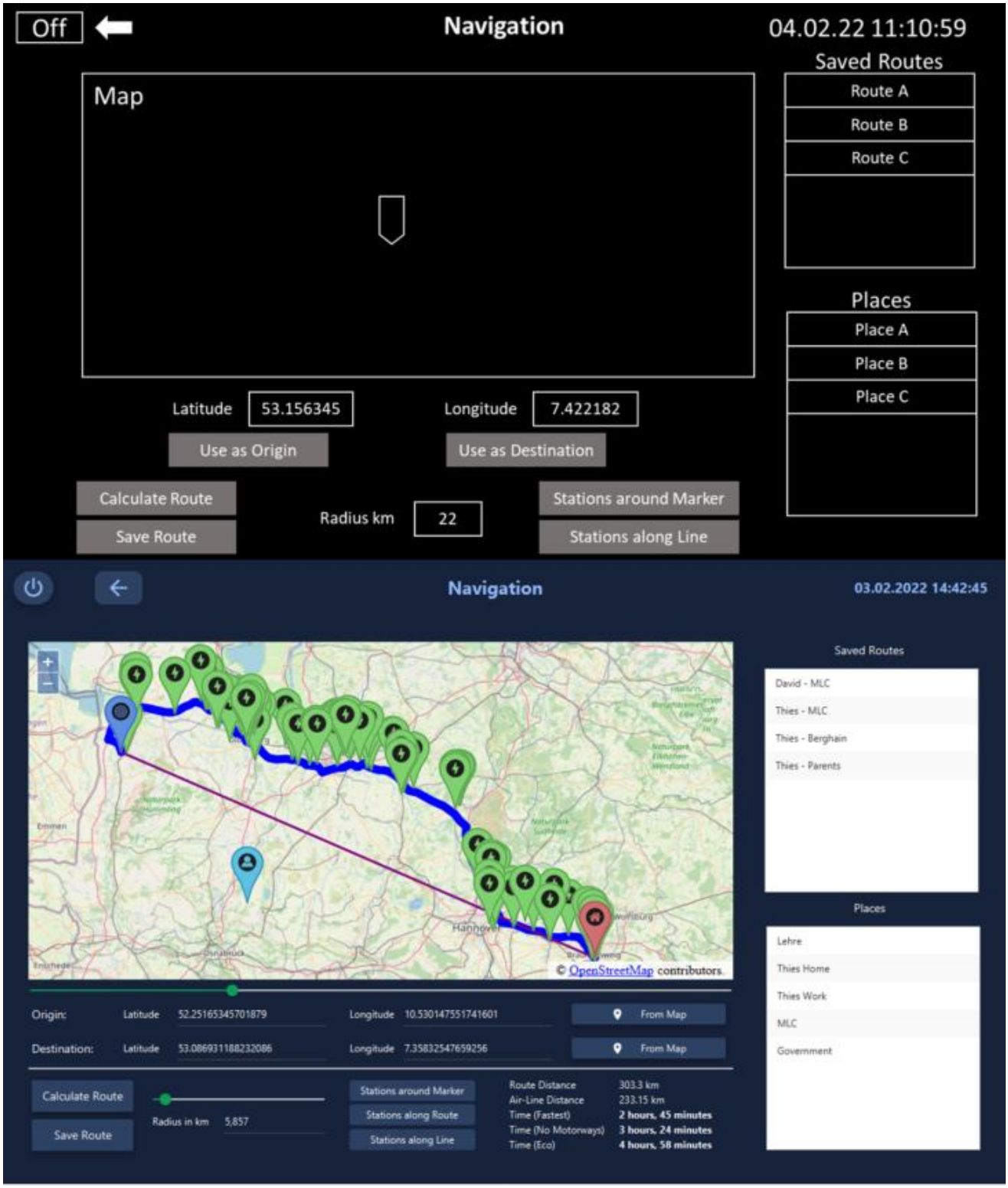

Abbildung 7: Vergleich Navigation zwischen Mockup und Applikation

Glossar

BitBucket 5 ..... Online-Service für Versionskontrollsysteme
Builder-Pattern 10. Entwurfsmuster der objektorientierten Programmierung zur dynamischen Erzeugung von Objekten
Dependency Injection 9 ..... Entwurfsmuster um während der Laufzeit Abhängigkeiten zwischen Objekten herzustellen
FxWeaver 9 ..... Bibliothek zur Verbindung von Spring Boot und JavaFX innerhalb einer Applikation
Git 10 ..... Versionskontrollsystem für iterative Softwareentwicklung
H2 9 ..... Relationales Datenbankmanagementsytem
JavaFX 4, 9, 10 ..... Java-Framework zur Erstellung von Applikationen mit grafischer Oberfläche
Maven 11 ..... Werkzeug zur Verwaltung von Softwareprojekten und ihrer Abhängigkeiten
MVC 9 ..... Model-View-Controller, gängiges Architekturmuster zur Aufteilung von Applikationen mit grafischer Oberfläche. Beschreibt das Datenmodell, seine Präsentation und die Programmsteuerung
Observer-Pattern 10 ..... Entwurfsmuster zur Weitergabe von Änderungen an einem Objekt
ORM 9 ..... Objektrelationale Abbildung, um Objekte in einer Datenbank auf geeignete Art und Weise abzuspeichern
Pair-Programming 5 Technik, bei der zwei Entwickler gemeinsam programmieren, wobei eine Person den Code schreibt während die andere Person konstruktive Anweisungen gibt und den Code kontrolliert
Properties und Bindings 10 ..... Sprachmechanismen in JavaFX um Beziehungen zwischen Variablen herzustellen. Eng verwandt mit dem Observer-Pattern
Refactoring 10 ..... Strukturelle Überprüfung und Anpassung des Quelltextes mit Fokus auf Lesbarkeit, Wartbarkeit und Verständlichkeit
Repository 9 ..... Im Spring Boot-Kontext: Schnittstellen der ORM zur Verwendung der Datenbank
SceneBuilder 9 ..... Werkzeug zur Entwicklung von JavaFX-Oberflächen
SonarLint 10 ..... Lokales Werkzeug zur statischen Codeanalyse
SonarQube 11 ..... Online-Werkzeug zur statischen Codeanalyse
Spring Boot 9 Werkzeug zur einfachen Entwicklung von Spring -Applikationen. Spring ist ein Framework zum Aufbau der Infrastruktur von Java-Projekten
Stylesheet 10 ..... Formatvorlage für das Design von Nutzeroberflächen
TDD 10 ..... Test Driven Development, Ansatz der testgetriebenen Entwicklung
Varchar 8 ..... Zeichenfolgen in Datenbanksystemen

Anhang

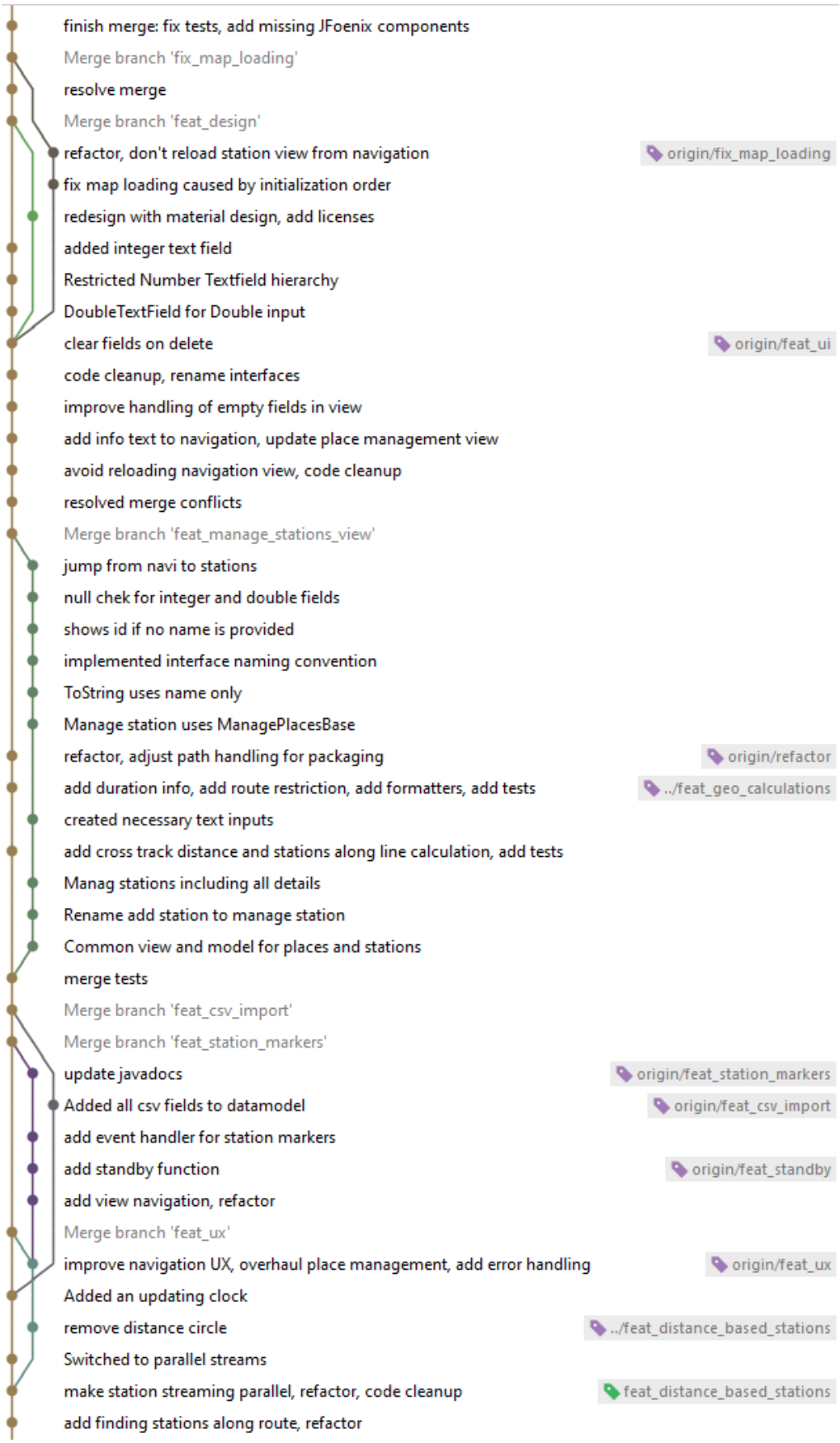

Abbildung 8: Auszug des Git Graphs

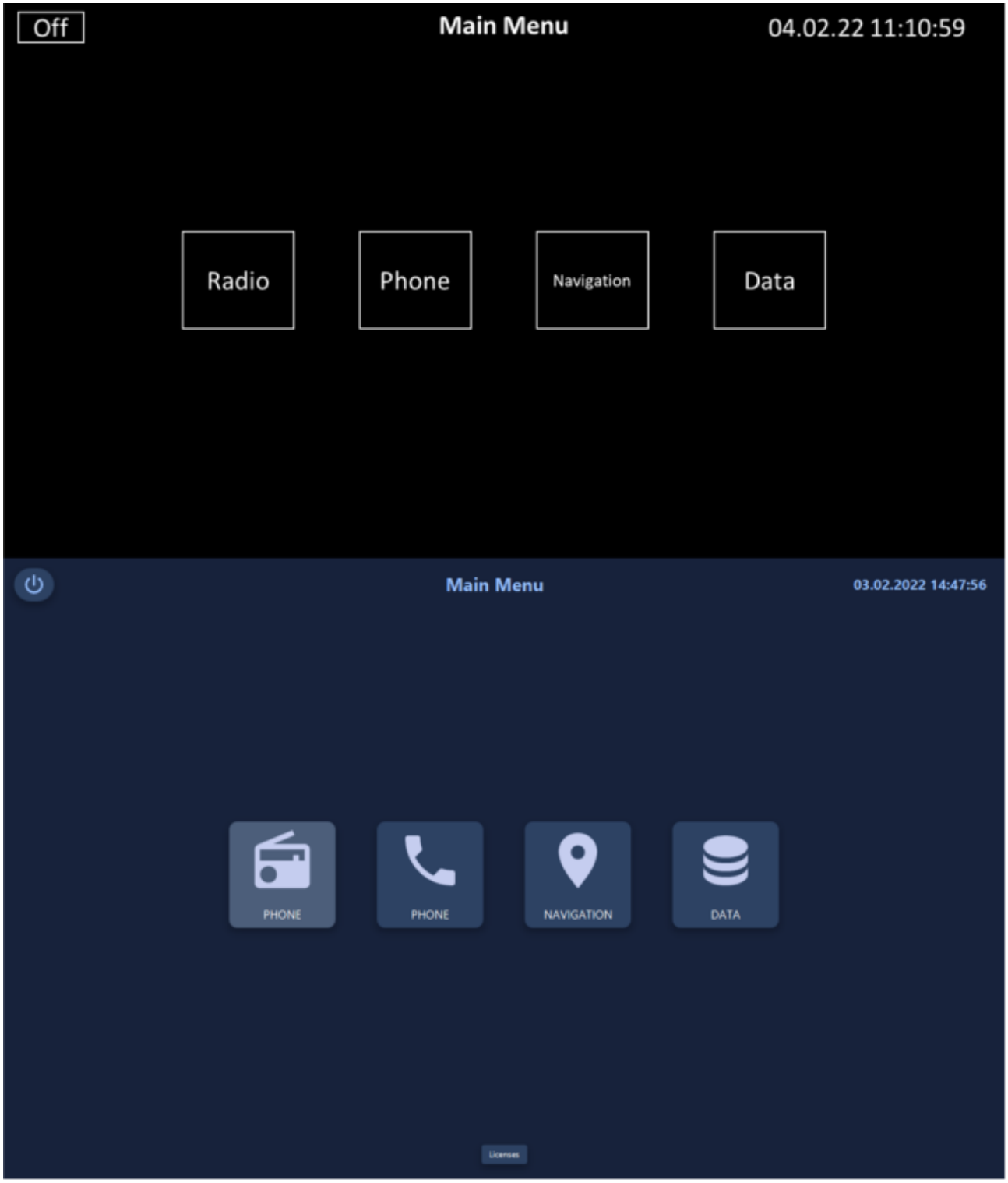

Abbildung 9: Mockup vs. Applikation für das Hauptmenü

Abbildung 10: Mockup vs. Applikation für die Uhr

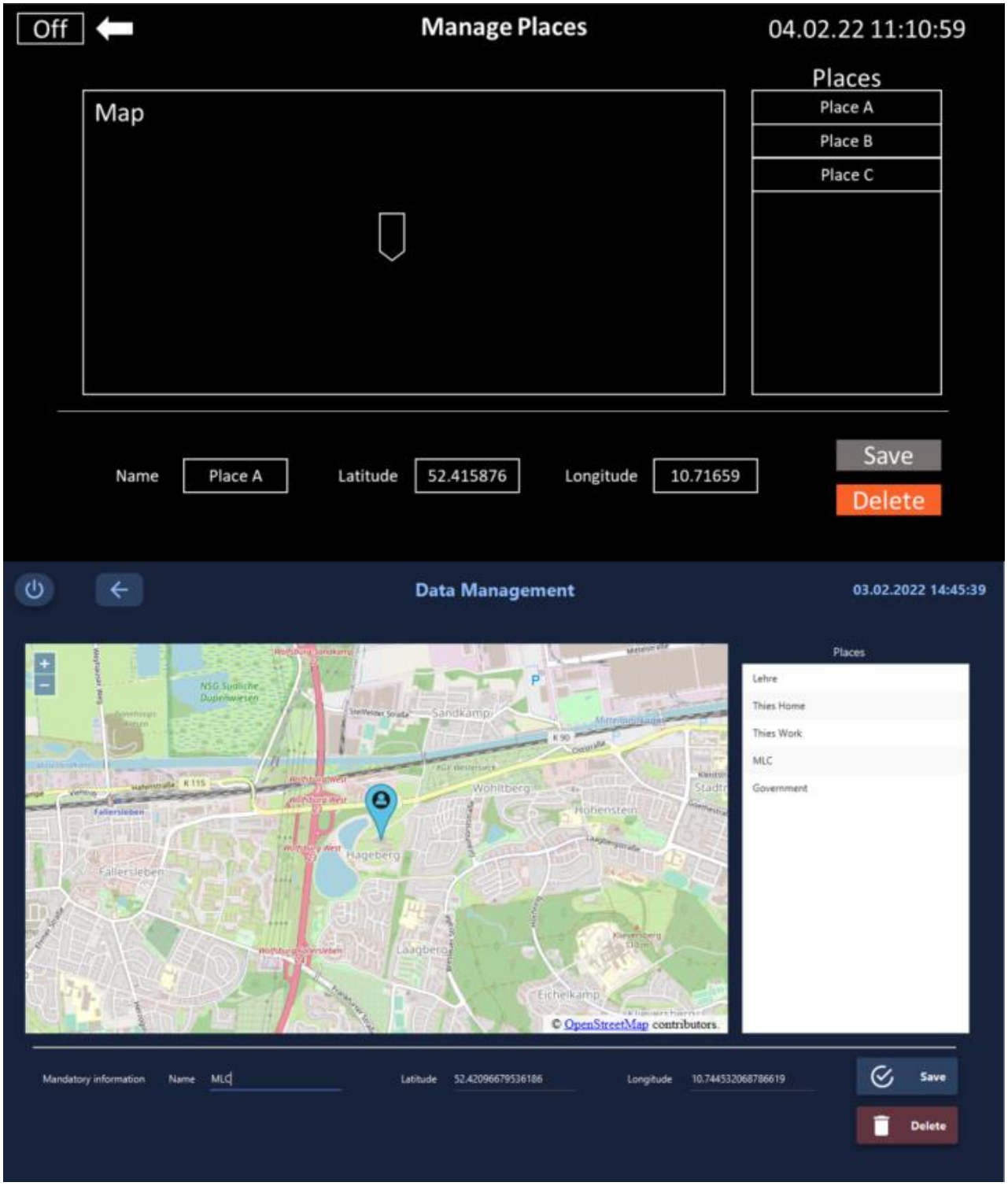

Abbildung 11: Mockup vs. Applikation für die Datenverwaltung

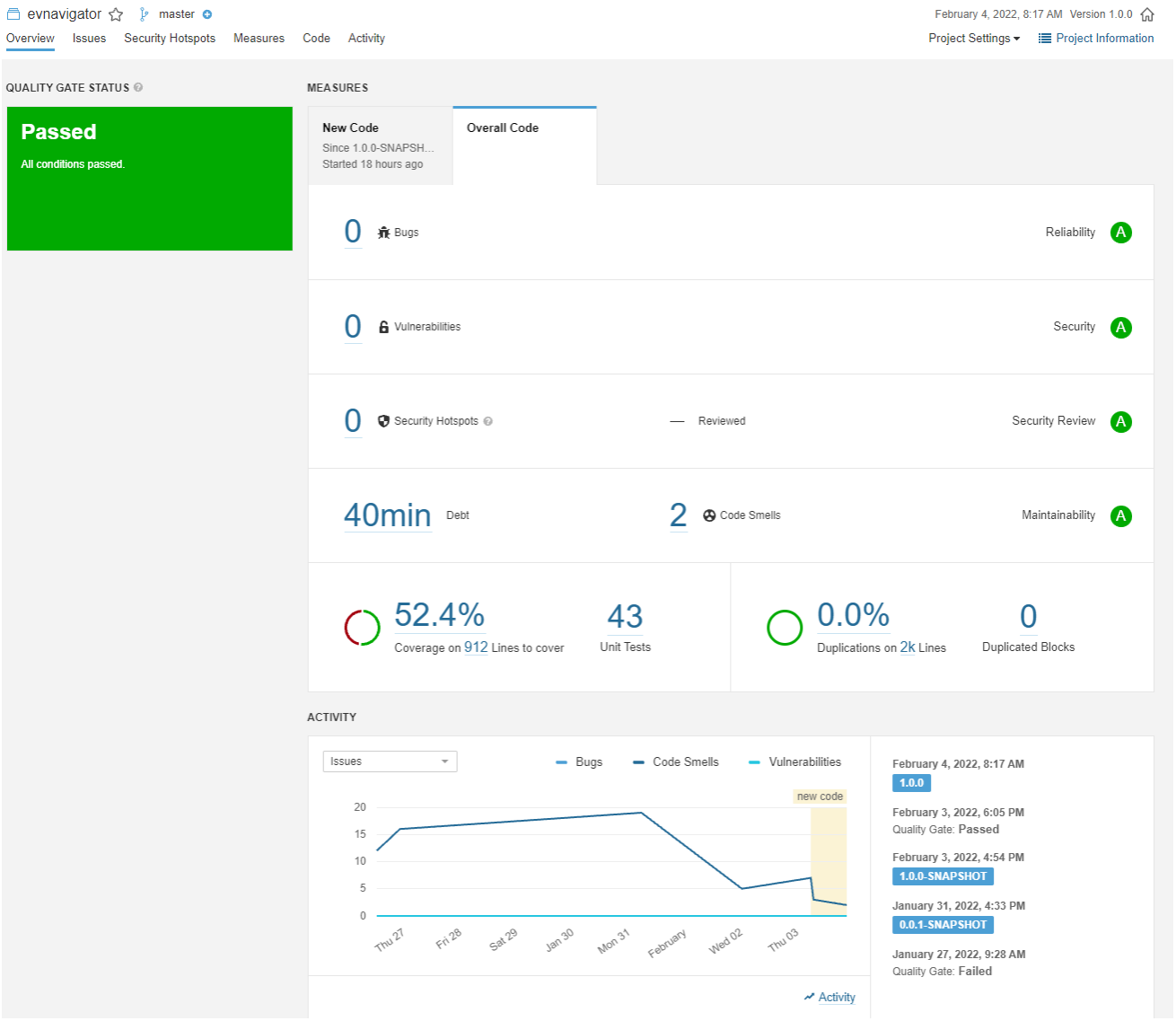

Abbildung 12: Codeanalyse mit SonarQube

Impressum

Justo, David
SE-A/34
david.justo@volkswagen.de

Bücker, Thies
SE-A/34
thies.buecker@volkswagen.de