Perfect Time - Plan Your Trip
Software Architecture Document

Version <1.1>

Revision History

Date	Version	Description	Author
01.12.2018	1.0	Created Document	Jan Rickel
04.12.2018	1.1	Updated Security Aspects	Jan Rickel

Table of Contents

Software Architecture Document

1. Introduction

This introduction provides an overview of the entire Software Architecture Document. It includes the purpose, scope, definitions, acronyms, abbreviations, references, and overview of the Software Architecture Document.

This document describes the architecture of the Perfect Time Application. This application was designed to allow smooth planning of vacations and holiday trips. To understand the purpose and the vision of the application, please refer to the official creators' blog.

1.1 Purpose

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

This document was designed to give the stakeholders and especially the customers of the Perfect Time Application an in-depth understanding on how the application was built. It will allow the customers to hand over the project to another team for further development and maintenance.
The document can be used to understand the code structure and to adapt or extend the code and the functionality of the software.

1.2 Scope

This document is closely related to the Software Requirement Specifications and the defined Use Cases. Both can be found in the GitHub repository of the Application. This document will influence the further coding and testing efforts and thereby the whole documentation of the Perfect Time Application.

1.3 Definitions, Acronyms, and Abbreviations

- MVC = Model View Controller
- SRS = Software Requirements Specification
- SAD = Software Architecture Document
- Flux = Flux architecture that is used by React
- Travel = one specific trip / vacation. For each travel the user will create an individual plan.
- Trip = Travel
- Location = one location visited during the travel. A location might be a city or a region. A travel contains multiple locations.
- Activity = one specific activity planned for a location. An activity might be a place or an event. Each activity must be assigned to a location.

1.4 References

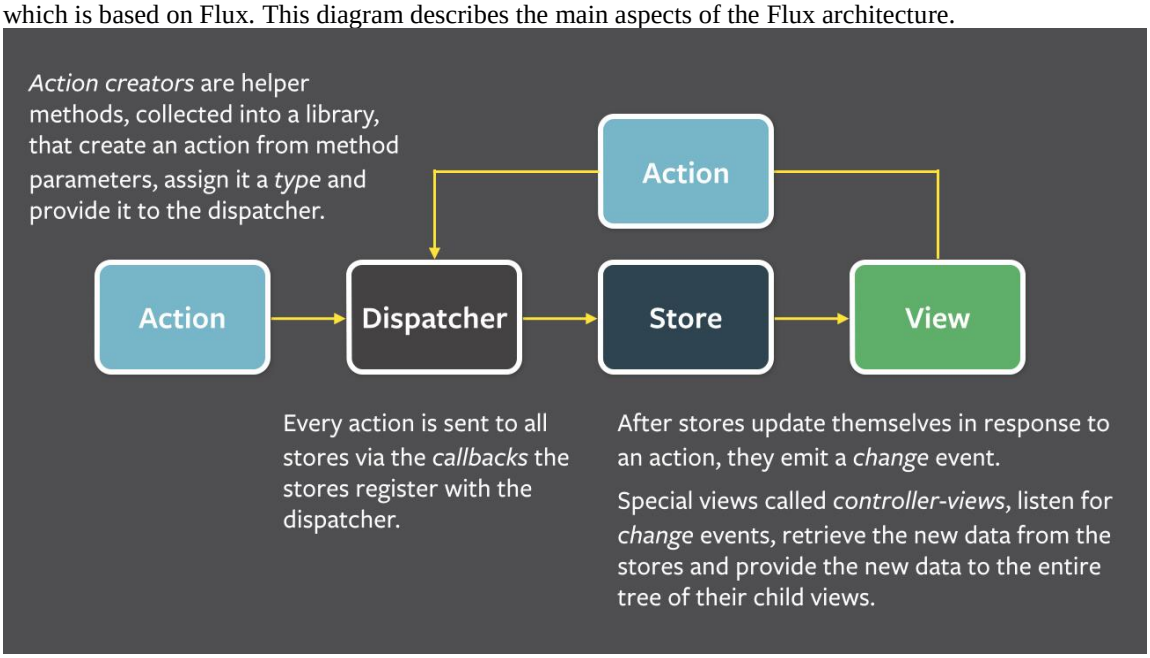
- Perfect Time Blog: https://perfecttime608150251.wordpress.com/
- Perfect Time YouTrack: https://perfecttime.myjetbrains.com/youtrack/agiles
- Perfect Time Git Repository: https://github.com/Tallround3r/PerfectTime
- Perfect Time Application Website: https://perfecttime-planyourtrip.firebaseapp.com/
- React JS Documentation: https://reactjs.org/

1.5 Overview

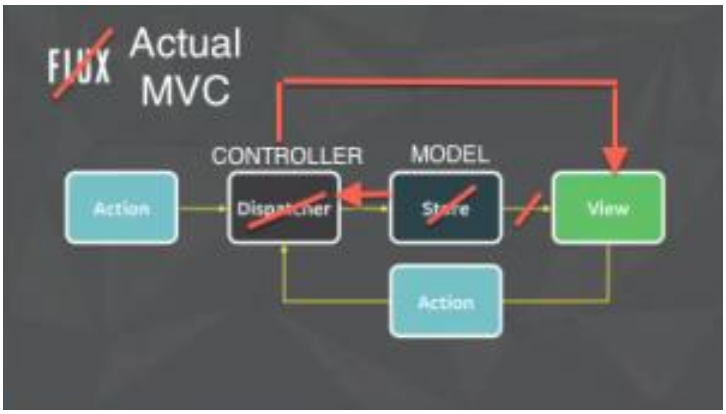
The document contains the following aspects. It explains the architectural representation of the Perfect Time Application. It states to goals and constraints of the chosen architecture. It details the Use Cases of the finished application. It describes the logical view of the Architecture, which includes the significant parts of the design model and the subsystems and packages. This document does not decompose the whole system into lightweight and heavyweight processes, as they are not applicable to this application. It also focuses on the deployment of the application. The implementation model of this document. It describes the selected database and storage of the application. The document will describe size and expected performance of the application. As the last aspect it describes the expected quality and how the chosen architecture should support the quality of the application.

2. Architectural Representation

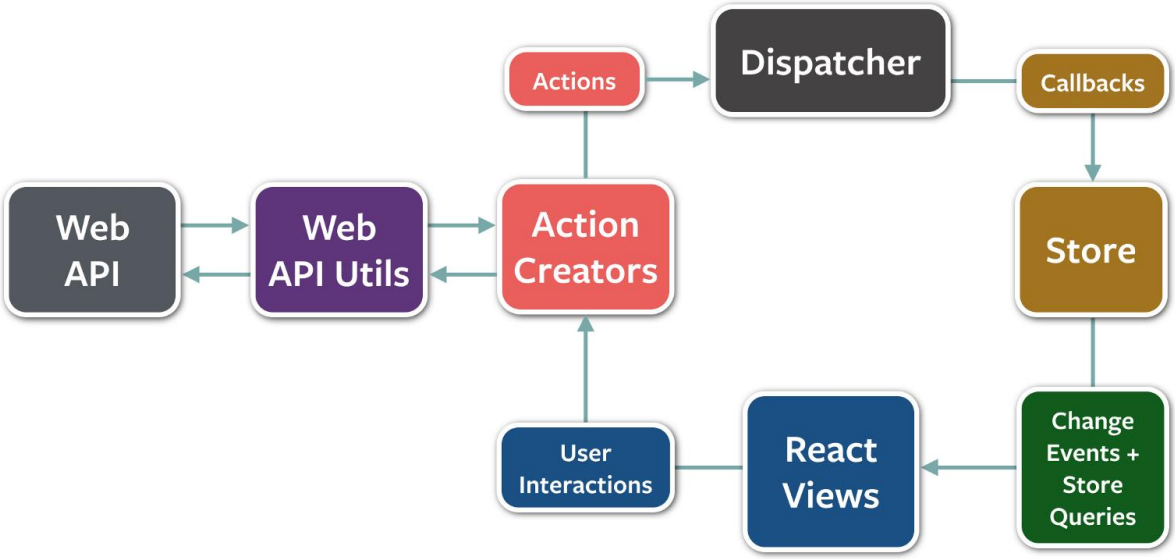
This section describes what software architecture is for the current system, and how it is represented. The Perfect Time Application is a React JS application. Therefor it implements the React architecture, which is based on Flux. This diagram describes the main aspects of the Flux architecture.



Flux works slightly different from MVC. The main differences are the naming (dispatcher instead of controller and state instead of model) and the communication between the components. While in MVC the controller updates the View, in Flux the dispatcher updates the state and the state will change the view. MVC also allows changes in the model to be recognized by the controller (e.g. by an updated entry in the database). In Flux all changes are committed to the dispatcher, which will change the model and take the predefined action. The following diagram shows the differences between the architectural designs.



Another difference from MVC is the fact, that the different entities (controller, etc.) are not (necessarily) represented by own classes or files. In React they are represented by methods in each individual component. The React Flux architecture is slightly different from the normal Flux architecture.



React provides predefined functions to smoothen the communication between the Flux elements and to simplify the development process.

The Perfect Time Application uses Redux on top of React. Redux allows a global state, which can be used by all components of the web application. In plain React each component has its on state. Therefor communication and data consistency between components can be difficult to achieve. Using Redux solves this problem.

3. Architectural Goals and Constraints

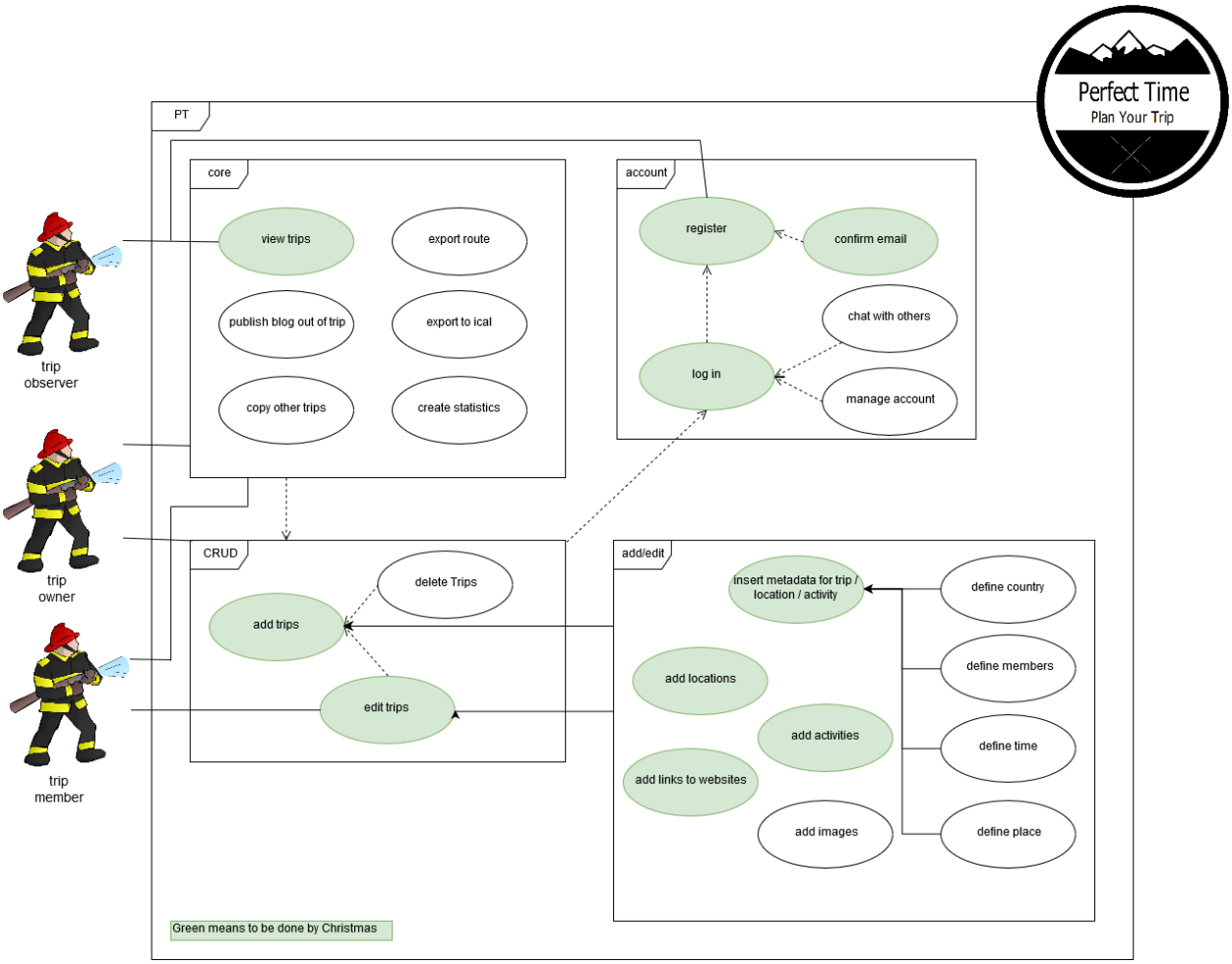
The Flux architecture allows the source code of Perfect Time to have a clear structure. The data is provided through the Firebase API. The received JSON-data gets mapped onto the defined models. With that standardized data the Redux state is filled. The updated state leads to an update of the shown elements.

This structure can be described with an example. A component (e.g. LocationsPage.js) defines the method *compose()* which calls *firestoreConnect()* to fetch the required data from the database. The received data gets transferred into an array called *locations*, which is saved in the state. The method *render()* uses the data as input for HTML code. This code gets displayed with the imported CSS attributes.

Through this design the application's maintainability is increased. Still, mixing the Flux elements in a single method is possible and sometimes necessary. Separating the elements as strict as in the Vaadin Java Framework for example is not possible in React JS.

4. Use-Case View

The application provides the use cases shown in the following diagram.



4.1 Use-Case Realizations

N/A

5. Logical View

This section describes the architecturally significant parts of the design model, such as its decomposition into subsystems and packages. First an overview over the classes is given. Then the elements of the Flux-architecture are shown on an example. Since the whole class diagram (which is a converted file diagram, because React is based on Java Script and does not use classes) is compromised of many files, which all include some state, dispatcher and view functions, it would create a mess, if a diagram with all Flux elements would be drawn. The simple example shows an understandable case, which can be transferred to

all the other classes / files.

5.1 Overview

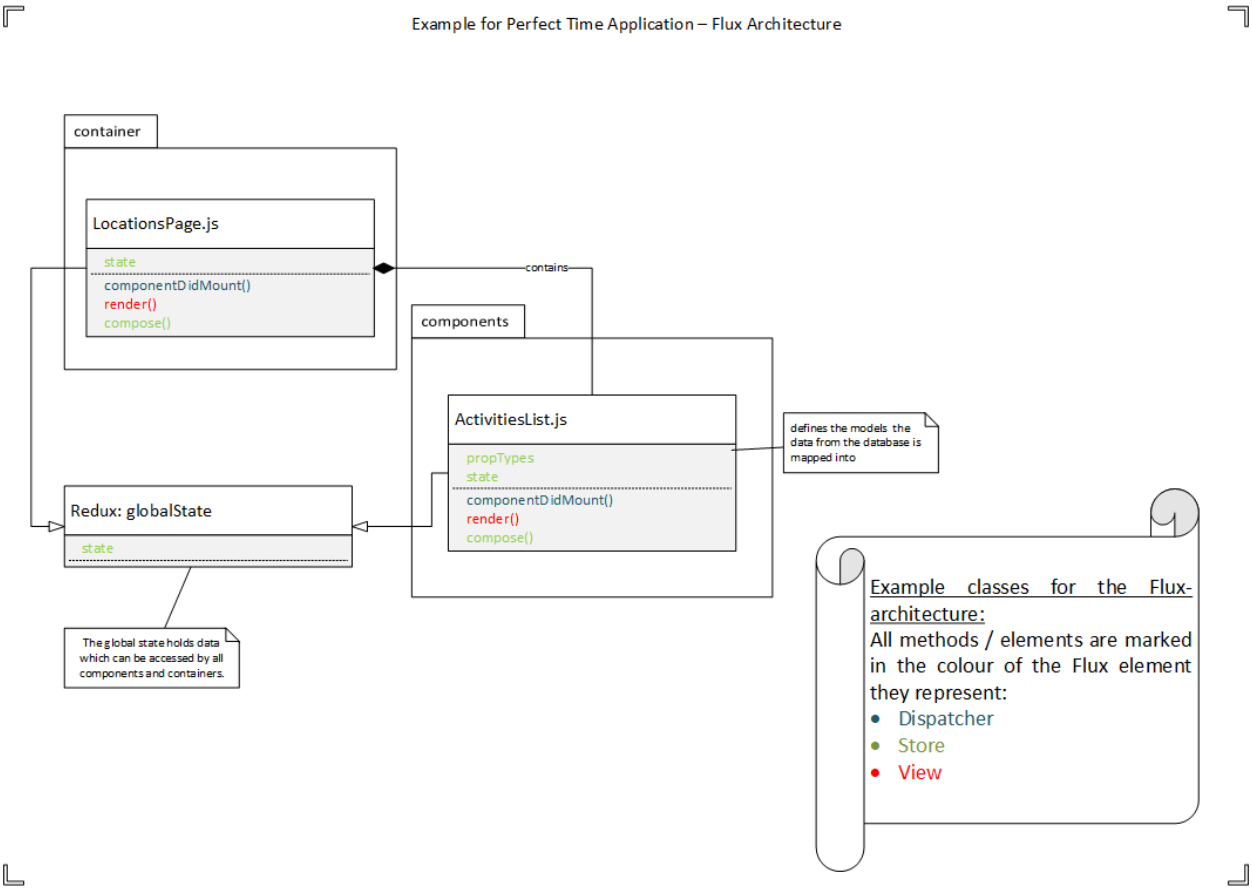
The following diagrams show the defined containers and the components of the application and their methods. The components are used in different containers. Each container represents a page / single view a user can access. A container can include several components. A component is an element shown on a page (e.g. a list or a button).

Container:

Components:

5.2 Architecturally Significant Design Packages

The following example illustrates how the Flux architecture of the React framework is designed in the Perfect Time application. It can be translated to any other component / container.



6. Process View

(N/A)

7. Deployment View

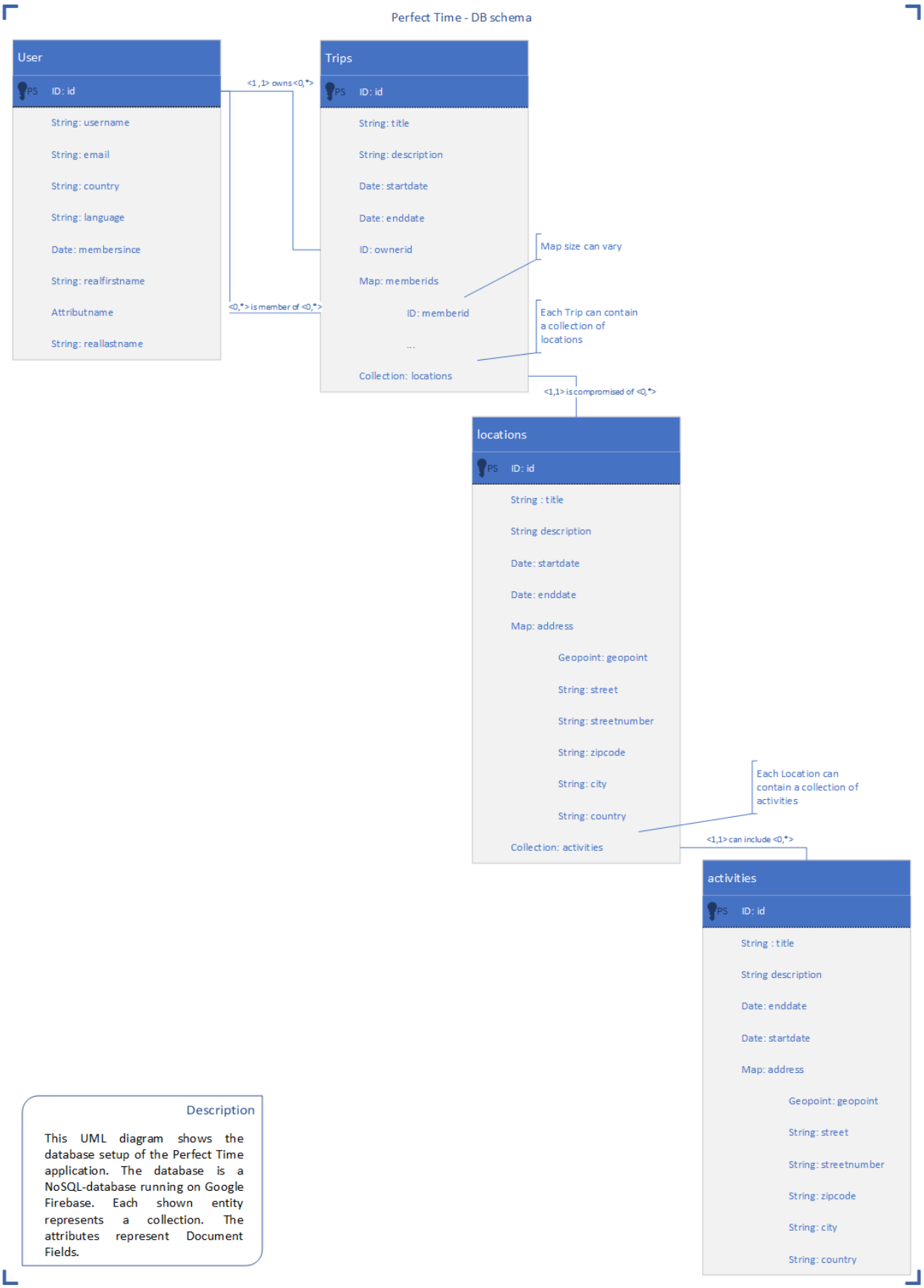
The Perfect Time Application is running on Google Firebase. Firebase provides the database, the media storage, the user functions (registered users, passwords, etc.) and the runtime environment. The application is deployed through a npm-install / command. Concludingly, the server architecture is provided by Firebase and Firebase is integrated into the Perfect Time application. The application can be accessed by any modern web browser that support CSS and Java Script.

8. Implementation View

(N/A)

9. Data View

The Perfect Time application uses Google Firebase to store data. Thereby it uses a NoSQL-database. The application is design to allow the user to define custom fields for his/her trip/location/activity (e.g. ticket number for a concert). Therefor the following diagram shows only the predefined data fields.


Perfect Time - DB schema
User
PS ID: id
String: username
String: email
String: country
String: language
Date: membersince
String: realfirstname
Attributname
String: reallastname
<1 ,1> owns <0,*>
<0,*> is member of <0,*>
Trips
PS ID: id
String: title
String: description
Date: startdate
Date: enddate
ID: ownerid
Map: memberids
ID: memberid
...
Collection: locations
Map size can vary
Each Trip can contain a collection of locations
<1,1> is compromised of <0,*>
locations
PS ID: id
String : title
String description
Date: startdate
Date: enddate
Map: address
Geopoint: geopoint
String: street
String: streetnumber
String: zipcode
String: city
String: country
Collection: activities
Each Location can contain a collection of activities
<1,1> can include <0,*>
activities
PS ID: id
String : title
String description
Date: enddate
Date: startdate
Map: address
Geopoint: geopoint
String: street
String: streetnumber
String: zipcode
String: city
String: country
Description
This UML diagram shows the database setup of the Perfect Time application. The database is a NoSQL-database running on Google Firebase. Each shown entity represents a collection. The attributes represent Document Fields.

The media data (pictures, videos, etc.) are stored in a special storage in Firebase. It will be identified through the primary keys of each data tuple.

10. Size and Performance

The size of the application depends mainly on the number of trips which users have created. Other important figures are the number of registered users and the number of views of the applications. React applications are easily scalable. However, the “bottleneck” of this application is Firebase. The application is running on the free plan. Thereby the storage size and the number of views per period are limited. This can be changed easily be upgrade to a paid plan for the application.

11. Quality

The application is easily extensible. Adding further React components to the application is simple. As described in chapter 10, the application can be enlarged by switching to a paid Firebase subscription. As a Google service, the creators believe Firebase to be consistently available and maintained. Thereby, the application should run reliably. The integration limits the portability of the application to other firebase accounts. In cannot simply be transferred to another backend.
Regarding safety and privacy concerns, the creators of the application to provide proper security through the user login. However, exploits in the Firebase and the React source code cannot be ruled out completely. Privacy might also be a concern. Google is not allowed to access the user data which Perfect Time is storing in Firebase. Still the Perfect Time team cannot promise that the NSA or other US agencies won’t collect data of Firebase users.
The user login is saved in a protected database, which is provided by Firebase. Therefor, they are not freely accessible. The Perfect Time application will only allow the upload of certain file types (e.g. JPEG). Apart from that, no integration of a malware detection toll is planned for.