Travlendar (Smart Commute)

# Objective

* Develop a scheduler web-app with a calendar interface that computes and accounts for travel time between appointments to make sure you’re never late for an appointment.

# Requirements

* **Authentication System:** Allow access per user to its own private environment through login authentication, and new users to register into the system.
* **Calendar interface:** Let the user interact with a calendar interface and manage appointments (events) within it, by performing the following actions: create, edit, delete and save.
* **Smart Commuting:** Automatically compute the travel time between appointments and warn for time overlapping.
* **Travel means by appointment and by day:** the web app must allow the user to specify the preferred travel means, and to be weekday sensitive.
* **User customizable:** let the user configure its workdays, predefined breaks and alert times.
* **Variability time between appointments:** allow the user to modify on the go the event durations (optional).

Introduction

# About the Developers

Hi. This is Pavel Pascacio and Antonio Del Negro, we are Automation & Control Engineering students at Politecnico di Milano. Smart Commute is the name of our web-based application which we’re developing since October 2018, and it is our first web software project. Our initial scope is to provide a functional application with its whole documentation for the mandatory course of Software Engineering, and afterwards to evaluate feasibility on launching the first version to the world wide web.

# Brief Summary

Smart Commute is a web-based application built in HTML, CSS, JavaScript and jQuery that aims to make the user’s appointments scheduling easier by considering the time it takes to go from one place to another. Smart Commute saves your appointments’ addresses and durations and shows you the best way to get from one to another, warns you if there’s no time, and helps you coordinate in an efficient fashion.

# User Personas

## Who would want to use Smart Commute?

Smart Commute is an app for the realtor, the entrepreneur, the busy law student. Basically, fits perfectly for anybody who must go from one place to the other every workday and attend meetings or errands.

This is not a mobile based application, nor it is intended to be (at least not yet). For the current purposes of the project, Smart Commute is centered on browser only usage, compatible with all major web browsers such as Google Chrome, Firefox, Safari, Edge and Opera.

## Why would they want to use it?

Because Smart Commute is the dedicated solution to a dynamic work environment that constantly demands relocation several times a day. There are dozens of solutions out there that provide the same functionality, however, it is not focused on a specific work environment and instead it normally is a general-purpose scheduler. The default calendar apps are effective in organizing events, however the lack of integration with a visual representation of such events allows for a dedicated solution.

## What are they looking for?

This persona wants to readily open the browser in their office/outside, get to smart commute and check the timing for the next appointment, how much time is left to go, and which transport means to take. Also, it wants to be able to dynamically plan for the day as it goes and change, delete or create new events at any time, so that Smart Commute assures a smooth workflow.

## How does Smart Commute provide the utilities that solve their problems?

It integrates a fully functional calendar with a mapping API and a routing API and uses a crafted algorithm to show in real time where and when to go according to the user’s plan of the day.

# Specifications (user)

## Login and access

Each user logs into its account by entering its email and password, the system validates the login and if successful the system connects the user into its account. If not successful, the system prevents entry but allows repeated attempts. After having validated the credentials, the user is redirected to the calendar interface automatically and if the user had previously saved a configuration and events, they will automatically load.

## Sign up

The user, by providing its name, last name, email and a password, will be able to register and create an account within the app. The system will then validate the login credentials and if successful it will connect the user into its account.

## Main Interface & Event Management

Once the user is in the calendar interface, it will have the possibility to take four main actions: to manage/view appointments, to configure settings, to view roadmap, and to logout. Managing appointments, configuring settings and viewing the roadmap are interfaces which by clicking their link, can be toggled among each other.

The functions of viewing appointments are within the same calendar interface. The events are automatically shown in the calendar interface, where it is possible to see the monthly view as default and the user can toggle views to daily, weekly, yearly, and schedule view. Differently from the date-based views, the schedule view lists all the appointments sorted in order of how recent.

Adding an appointment can be done via clicking the “+” button, and a configuration window will appear where the user can input the title, date, start time, end time, address and description of the event. The fields of title, start time, end time and date are mandatory, and description and address are optional. Once user has finished, “cancel” and “save & exit” options are available to save the event or to discard changes.

To edit/delete any shown appointment, the user can double click the shown event in any of the interface views (except schedule view) and the same fields as the create event window will appear, showing the previous event configuration, in addition to a “delete” button.

The user will have the possibility to create, edit, delete and save as many appointments as it wants. Each appointment will have the following fields: title, date, start time, end time, address and description.

## Settings configuration

Through the settings interface the user will be able to personalize its preferences, which will be the selection of the preferred transport, toggling between available transports (bike, walk, car, bus, tram, metro and train), specifying resting times (by default lunch and break times) and setting up the alerts.

To configure settings, the user clicks the settings tab from the calendar interface and once its redirected to the settings interface it has the possibility to modify the preferred transport, checking which transports to use (by toggling enable/disable buttons), specifying the resting times (which by default are called lunch and break times), and specifying the alerts.

Selecting the preferred transport will give priority to that type of transport, so whenever it is available once the Google maps API calculates the trajectory the roadmap interface will automatically show that given trajectory with the specified transport type. If the preferred transport is not available, it will prompt the user and ask for an alternative and show the fastest as recommended.

Enabling/disabling available transport types will force Google maps API to not show certain transports. For example, if you don’t desire to use tram and disable the option, under the scenario that your preferred transport isn’t available, even if for that travel tram is the recommended way, it will not be shown.

Specifying the resting times is a functionality that allows the user to have a daily fixed period (also configurable to any preferred days of the week) reserved to rests. Rests are considered a repetitive weekly event, and within the event period no events can be added, and travel time is not supposed to overlap that period either. By default, the resting times are named lunch time and break time. By clicking the edit rest times button the user will decide for each of the rest times their names, day periods, and which day of the week they should be active. Deselecting all days disables the resting time.

The alerts section decides how close to the maximum departure time (to the next appointment) should Smart Commute notify you. By default, it is set to fifteen minutes.

## Maps interaction

This interface shows the city map with the day roadmap of the user: all the current day’s events will appear as nodes, and upon clicking given nodes the appointment details will be shown. Additionally, the travel trajectory between appointments (nodes) will appear as calculated from the Google Maps API with the preferred transport, and by clicking on it, its details (duration, transport to be used and maximum departure time) will appear.

The roadmap is available through the button roadmap, present in the calendar and settings interfaces. The roadmap shows by default the Google Maps API centering your location and it loads any nodes (events) and edges (travels) previously computed, if any. This functionality allows the user to view the “roadmap” of its scheduled day according to the events it created. It shows the appointments as pins in the map that are clickable. Once clicked, they show the event details. Additionally, the “edges” which are the travel distances, are computed automatically thanks to the Google Maps API and shown accordingly, with a dialog box specifying which transport type is being used, and the duration of the commuting time.

## Logout from session

Logging out is a button available within the roadmap and calendar interfaces, and by clicking on it the user logs out, the app saves the generated data and the configuration data to the server, and it prompts back to the login site.

# Specifications (admin)

## Firebase as a resource for authentication, metrics and database (backend)

We have chosen Firebase as our main backend tool for Smart Commute. In this section we will detail the functionalities from Firebase used by the application.

### Firebase Auth and user management

Storage of user credentials and its verification is done through Firebase Auth API. In a sign-up process, the forms are filled to create a new user and store it onto Firebase Auth, along with an autogenerated user ID (unknown to the user). Similarly, if there is an email and password match during a sign-in process, a request is sent to Firebase Auth and for a given user its stored personal data will be loaded in the app.

The administrator has the possibility to manage all the users under the Firebase Auth section in the Firebase console.

### Firebase Database

Firebase DB is the main backend tool for Smart Commute. Just as Auth, we have initialized the DB tool provided by Google Firebase and linked a key to Smart Commute. Firebase Auth and DB are managed within the same console: that provides ease of management to the app administrator, for example: each unique autogenerated ID per user from Auth is linked directly to the Database every time a request is sent.

#### Events Management

Every time an event is saved, deleted or updated, a push request is sent to Firebase DB through its API functionality. The event information is assured to be consistent thanks to the Fullcalendar.io logic, preventing any time inconsistencies while saving events.

The events data does not have a limit. Any time an event is created, it is appended into the user event data and stays active until the user decides to delete that data.

#### Settings Management

Default settings data is auto-generated once any user profile is created (after sign-up process) and logged into a .json file, and its size never changes. Every time the user modifies a setting, a request is sent to the database to update the settings specific values and it provides that information to the other app modules for due functionality.

#### Routing Data Management

Routing data consists on a duration value, a start and end point value, a transport type value and a routing array value which contains every route step to get to the end point. They are generated by the app engine and every time a consecutive event is created, this routing data is stored into the database and shown in the roadmap interface. Similarly, for any route objects previously generated in another session of the same user, these routes are preloaded once the user logs back into Smart Commute.

# App Engine

## Calendar and Events

The calendar interface will provide day, week, month and yearly views, and the addition of events can be accessed through any of these views. Also, a schedule view is available which lists all the events within a time period.

Specify about the algorithm behind the events creation and the whole front-end setup using Fullcalendar.io

## Routing

Routing data objects are autogenerated anytime there are consecutive events. Once a consecutive event is created, the app engine will automatically check time feasibility with possible transports and if it is possible to commute, creates this data object, shows the route in the roadmap interface and stores the object in the Database.

Ideally, there’s always route objects generated, being the number of consecutive, non-conflictive events, per day.

Specify about the leaflet tool (pending)

## Commuting Algorithm

The travel time (duration) will be computed and alerts will be generated whenever the maximum departure time is near. Additionally, suggests the preferred transport as main option and discards the transports disabled by the user.

Specify about how the app decides whether the preferred means of transport is feasible and whether it is possible to choose the second best.

# API’s

For the development of our project, we resorted to several application programming interfaces, either for the backend configuration, the calendar, the mapping & routing, operations and simulations, or the frontend configuration.

## Firebase

Firebase is an integrated solutions tool developed by Google which includes several tools, such as a cloud-based authentication system, a Database, Hosting service and others.

For Smart Commute, only Firebase Auth and Firebase Database were used. Firebase Auth manages, adds or deletes users into the system by providing a key ID to each signed up customer. Each customer upon logging into Smart Commute successfully, the specific key ID is used to pair its credentials with the user data saved in the Database, with which the user interacts in each logged session: it can save, delete or request data.

## Fullcalendar.io

Fullcalendar.io is a free API built in JavaScript + jQuery. It provides an easy implementation of a standalone calendar for web-based applications. Fullcalendar.io is the core engine of the Calendar module, and within it all the functionalities of event management are held. From viewing the calendar interfaces, the list of events and adding, updating and eliminating events, all operations are done within the Fullcalendar.io framework.

## Leaflet

Leaflet is a free API that provides mapping information, as well as addresses, location and traffic information. Leaflet is included and used as the core element of the Roadmap module, where the events and their commuting routes can be seen. More on this later.

## Leaflet Routing Machine

Leaflet routing machine is an add-on from Leaflet developed by Per Liedman. Works on top of Leaflet by using the traffic information and providing routing data from point to point. More on this later.

## Bootstrap

Bootstrap is an API used for frontend development, based on HTML + CSS + JS. It is used for embellishment purposes and for a sleek web design for Smart Commute.

## Gulp

Gulp is a tool that basically keeps constant connection between a localhost port and the source code of a website. It is used to appreciate in real time the simulation during the development of the code.

# Additional Information

# Specification (deprecated)

In the application, each user logs into its account by entering its email and password, the system validates the login and if successful the system connects the user into its account. If not successful, the system prevents entry but allows repeated attempts. After having validated the credentials, the user is redirected to the calendar interface automatically and if the user had previously saved a configuration and events, they will automatically load.

The user will have the possibility to create, edit, delete and save as many appointments as it wants. Each appointment will have the following fields: title, date, start time, end time, address and description.

The calendar interface will provide day, week, month and yearly views, and the addition of events can be accessed through any of these views. Also, a schedule view is available which lists all the events within a time period.

Moreover, a roadmap view will be available. This interface shows the city map with the day roadmap of the user: all the current day’s events will appear as nodes, and upon clicking given nodes the appointment details will be shown. Additionally, the travel trajectory between appointments (nodes) will appear as calculated from the Google Maps API with the preferred transport, and by clicking on it, its details (duration, transport to be used and maximum departure time) will appear.

The travel time (duration) before mentioned will be computed and alerts will be generated whenever the maximum departure time is near. Additionally, suggests the preferred transport as main option and discards the transports disabled by the user.

Through the settings interface the user will be able to personalize its preferences, which will be the selection of the preferred transport, toggling between available transports (bike, walk, car, bus, tram, metro and train), specifying resting times (by default lunch and break times) and setting up the alerts.

# Description (deprecated)

In the application, each user logs into its account by entering its email and password, the system validates the login and if successful the system connects the user into its account. If not successful, the system prevents entry but allows repeated attempts. After having validated the credentials, the user is redirected to the calendar interface automatically and if the user had previously saved a configuration and events, they will automatically load.

Once the user is in the calendar interface, it will have the possibility to take four main actions: to manage/view appointments, to configure settings, to view roadmap, and to logout. Managing appointments, configuring settings and viewing the roadmap are interfaces which by clicking their link, can be toggled among each other.

The functions of viewing appointments are within the same calendar interface. The events are automatically shown in the calendar interface, where it is possible to see the monthly view as default and the user can toggle views to daily, weekly, yearly, and schedule view. Differently from the date-based views, the schedule view lists all the appointments sorted in order of how recent.

Adding an appointment can be done via clicking the “+” button, and a configuration window will appear where the user can input the title, date, start time, end time, address and description of the event. The fields of title, start time, end time and date are mandatory, and description and address are optional. Once user has finished, “cancel” and “save & exit” options are available to save the event or to discard changes.

To edit/delete any shown appointment, the user can double click the shown event in any of the interface views (except schedule view) and the same fields as the create event window will appear, showing the previous event configuration, in addition to a “delete” button.

To configure settings, the user clicks the settings tab from the calendar interface and once its redirected to the settings interface it has the possibility to modify the preferred transport, checking which transports to use (by toggling enable/disable buttons), specifying the resting times (which by default are called lunch and break times), and specifying the alerts.

Selecting the preferred transport will give priority to that type of transport, so whenever it is available once the Google maps API calculates the trajectory the roadmap interface will automatically show that given trajectory with the specified transport type. If the preferred transport is not available, it will prompt the user and ask for an alternative and show the fastest as recommended.

Enabling/disabling available transport types will force Google maps API to not show certain transports. For example, if you don’t desire to use tram and disable the option, under the scenario that your preferred transport isn’t available, even if for that travel tram is the recommended way, it will not be shown.

Specifying the resting times is a functionality that allows the user to have a daily fixed period (also configurable to any preferred days of the week) reserved to rests. Rests are considered a repetitive weekly event, and within the event period no events can be added, and travel time is not supposed to overlap that period either. By default, the resting times are named lunch time and break time. By clicking the edit rest times button the user will decide for each of the rest times their names, day periods, and which day of the week they should be active. Deselecting all days disables the resting time.

The alerts section decides how close to the maximum departure time (to the next appointment) should Smart Commute notify you. By default it is set to fifteen minutes.

The roadmap is available through the button roadmap, present in the calendar and settings interfaces. The roadmap shows by default the Google Maps API centering your location and it loads any nodes (events) and edges (travels) previously computed, if any. This functionality allows the user to view the “roadmap” of its scheduled day according to the events it created. It shows the appointments as pins in the map that are clickable. Once clicked, they show the event details. Additionally, the “edges” which are the travel distances, are computed automatically thanks to the Google Maps API and shown accordingly, with a dialog box specifying which transport type is being used, and the duration of the commuting time.

Logging out is a button available within the roadmap and calendar interfaces, and by clicking on it the user logs out, the app saves the generated data and the configuration data to the server, and it prompts back to the login site.