Zurich University
of Applied Sciences

School of
Engineering

INFPROG2 P05 – Individual Application Project – Variant: Public Transport Routing in Europe

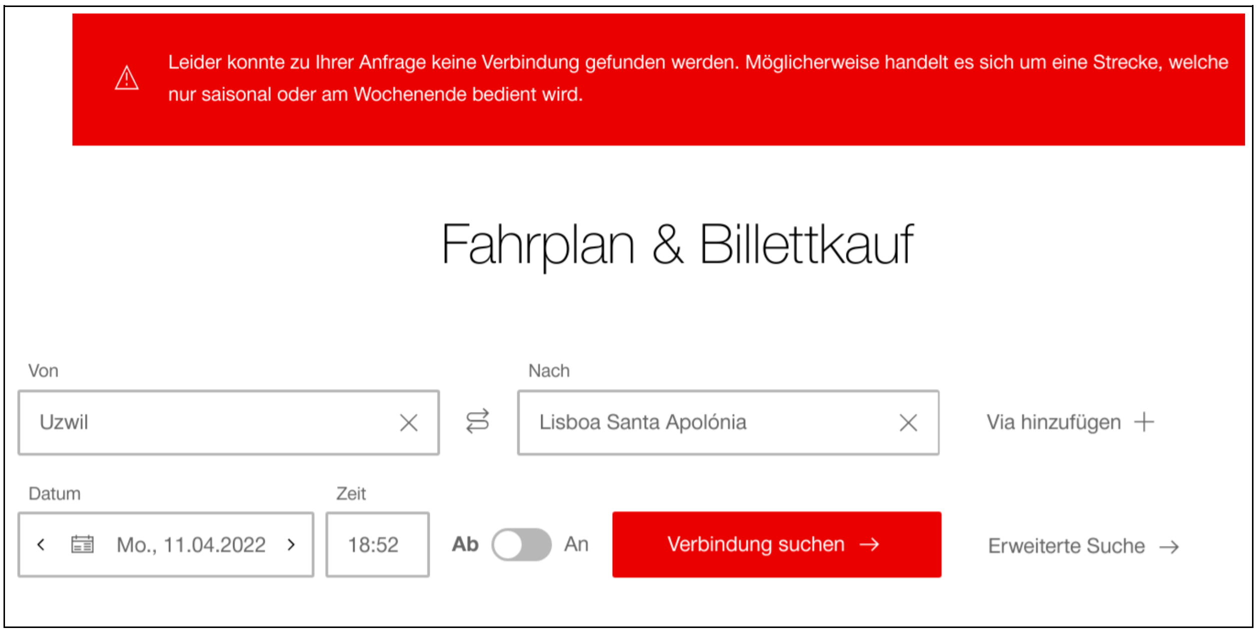
This project runs over 5 weeks (SW9-SW14 minus one) and thus yields 10 points. It is supposed to be solved in teams of 3 students. Only in exceptional cases and after consultation with the lecturer teams of 2 students are possible too.

This project work trains access to data on the Internet combined with data processing, including special types of data such as date, time and geolocation.

1.1 Problem statement

Our planet is subject to a climate crisis due to a high population and a high consumption by that population. Countering the climate crisis requires concrete proposals. One such proposal is to reduce the number of plane trips, finding substitutes such as train trips that emit less CO_2.

While for many years consumers have been able to consult and even book arbitrary flights online, just the basic lookup of international train or bus connections is still a problem. Often, the lookups only work until a certain city in a neighbouring country, but not beyond.

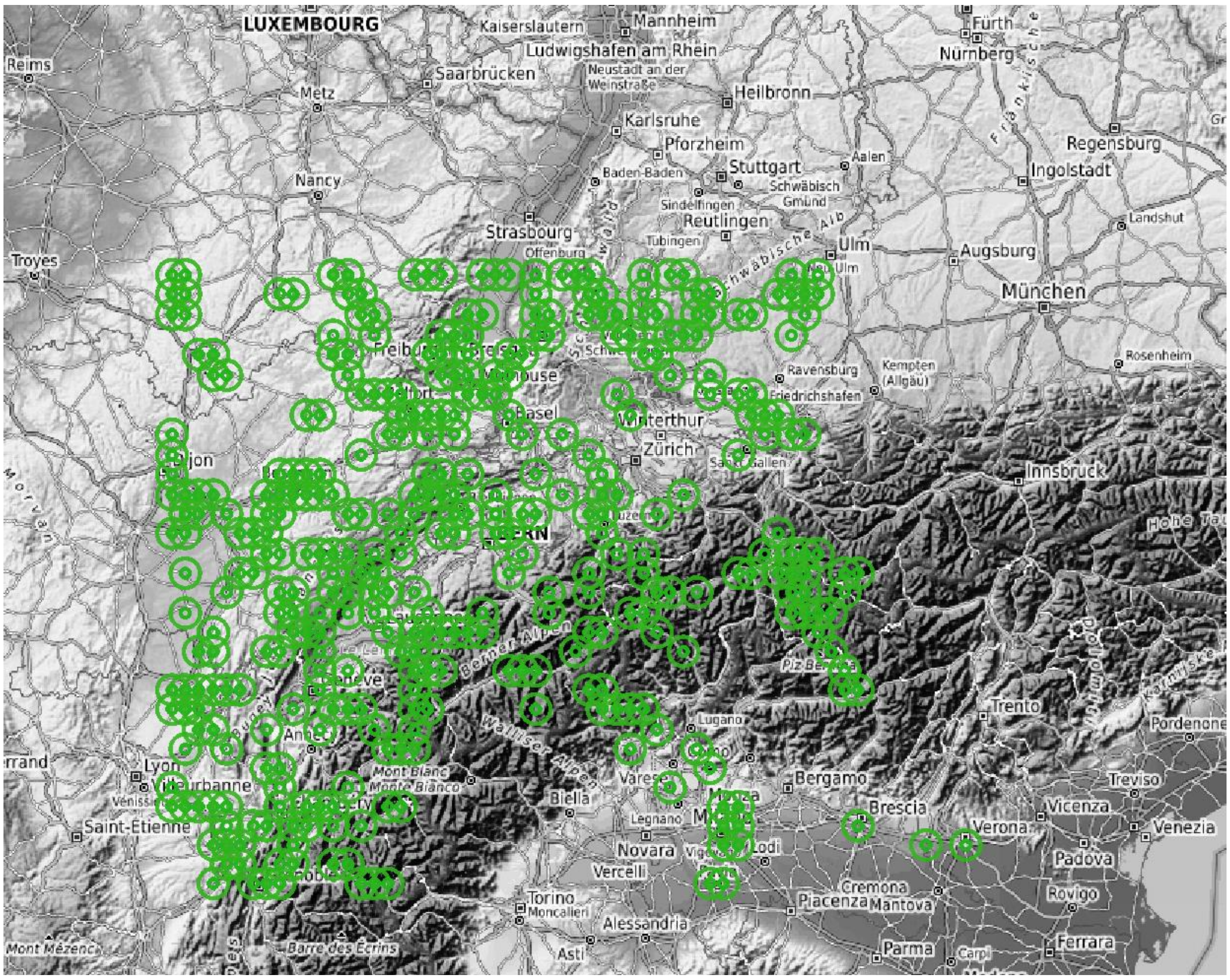

⚠ Leider konnte zu Ihrer Anfrage keine Verbindung gefunden werden. Möglicherweise handelt es sich um eine Strecke, welche nur saisonal oder am Wochenende bedient wird.

Fahrplan & Billettkauf

Von: Uzwil
Nach: Lisboa Santa Apolónia
Datum: Mo., 11.04.2022
Zeit: 18:52

The project idea starts here: Instead of receiving an answer such as the one shown above that a trip is not possible, could we give the customer a better experience by saying: Not the entire trip can be looked up, but around 70% of the distance can be covered, and the remaining 30% can be looked up at site XYZ?

For humans, SBB operates its own search interface **sbb.ch** as well as the vendor-neutral interface **fahrplanauskunft-öv.ch**. For Python programs, there are several APIs – such as TRIAS, OJP and others. However, for this task the old and simple API [transport.opendata.ch]() can be used, especially its ‘Connections’ resource. It covers Switzerland, Liechtenstein and long-distance stations in France, as well as few key stations in neighbouring countries such as Milano or Karlsruhe. The associated ‘Locations’ resource is more limited, covering roughly a rectangle over Switzerland only as shown in the next figure. This API shall serve as starting point.



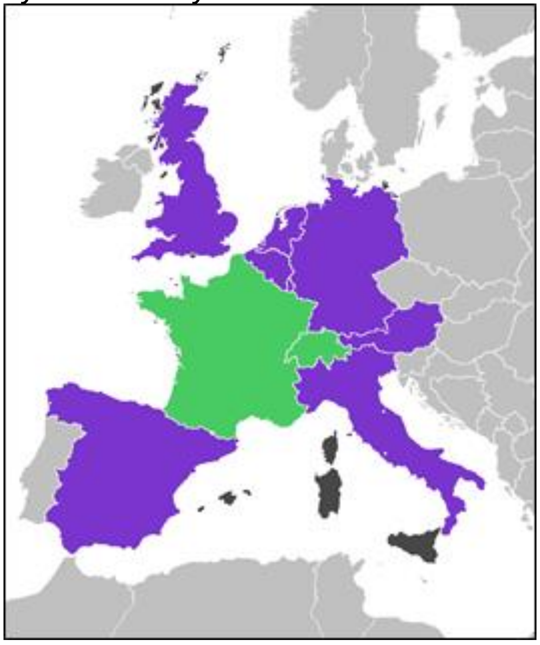
1.2 Functional requirements

Get to know the ‘Connections’ and ‘Locations’ resources of the API by reading up on their documentation.

1.2.1 Basic functional requirements (3 points)

The common requirements for this task to be fulfilled by all teams are as follows:

- Define a number of key cities and stations (ca. 50) around the borders of Switzerland and France across all directions. For each station, query automatically if a connection exists from your home location with the ‘Connections’ resource. Mark the station as reachable or unreachable in a tracking file with appropriate format, along with its geolocation (latitude, longitude). This information is all contained in successful API responses. Additional requirement: Count the stations in your tracking file. If less than 30 are reachable, you need to define more key stations until at least 30 are reachable.
- Build a database based on public transport search interfaces per country. For example, in Spain, trains can be looked up at **renfe.es**. These will be used to guide the user beyond the reachable cities and stations.
- Build a (textual) search interface. Ask the user’s home location in Switzerland and desired destination somewhere in the neighbouring countries Europe, symbolised by the violet colour in the map below.



1.2.2 Functional subtasks (2 points)

Only for teams of 2: solve 2 of 3 subtasks.

- Check if the destination can be reached directly. If not, calculate the subset of matching reachable connection stations. These should be within +/- 20° of the line of sight between home and destination; any station beyond that would lead to sub-optimal connections.
- For each connection station, determine the next possible connection including travel time, and calculate the percentage of the trip covered when reaching the connection station, again using line of sight. Moreover, give a travel recommendation to the user, including the next possible connection and where to search for follow-up connections.
- Implement a caching mechanism in a query wrapper. Whenever you query a connection and it does not exist, a blacklist should be populated with this negative information. In case the same connection is queried again, instead of wasting a network request, the blacklist should be consulted to inform that no such connection exists.

1.3 Code quality (3 points)

Create Python code that is …

- object-oriented with suitable class definitions and class tests
- clean and lean, passes the usual linter checks
- robust against both erroneous user input and network failures
- understandable what it does in each step, add comments where needed

1.4 Usability (2 points)

Provide a good user experience.

- Your program should be easy to use. It guides the user (e.g. the lecturer) through the interactions in a clear way.
- The user should always know where he is in the program flow and how to reach the next steps.