Documentation and Product Report

This document contains an overview of the Locato project and product, Locato's implementation details, and a business analysis.

Project Overview

Locato is a tool for travellers and modern nomads. It uses open data to concisely highlight interesting parts of a given city. Locato was developed with big cities in mind - from Paris to Prague, Locato can make navigating a large city easier.

Have you ever wondered which part of a city is the safest? Or which part of the city has like-minded people? Locato can help you figure that out. It does so by having a set of data overlays for a list of cities. The user can choose a city of interest and the data to be shown, Locato then visualises the data as a heatmap. This way, the user can explore the city as a whole, rather than collecting bits and pieces of information about individual parts of the city.

The idea of Locato is to have a one-stop data aggregation for all quality of life metrics in large cities. We know that not everyone chooses their place to live or to stay during a trip based on the same preference. Therefore, we give the user the freedom to investigate the data they are interested in.

The project is realised as a web application that utilises both open data and deep learning with visual recognition. The client side is a React application with a sleek user interface, which is simple to use and navigate. It features a map component powered by Azure Maps used for displaying the selected data as a heatmap. The backend consists of a web API and a database. We expose our API endpoints so that any client implementation can make use of them. Additionally, the project contains means of collecting and processing open data from cities all around the globe. In the processing step simple algorithms, as well as convolutional neural networks, are applied.

Goals

The main goal of this project was to develop a proof of concept of an application that would visualise information about surroundings and points of interests on a map instantly. Let’s introduce some typical use-cases for Locato.

The first use-case is for a man who is travelling abroad with his family for a business meeting. Since he is travelling with his family but can not stay with them for the whole

duration of the trip he wants to make sure that the hotel is in a safe, family-friendly place with green parks.

1) A tourist wants to book a hotel in a foreign country. Because he is travelling with his family he wants it to be situated in a safe, family-friendly and green area.

Another use case might be for a company that is looking to expand its field of operations into another city where it does not yet have any property and is looking to buy some. Since the company produces a lot of noise it would prefer not to buy any property in an area that has a quiet surrounding since it would most likely deal with a lot of noise complaints.

2) A company wants to buy a property that is not in a quiet area

As a last use case we can think of a female student that is going to a large city for university. Because she is a student she can not afford to pay high rent. Therefore is looking for compromises everywhere she can but would not live in an unsafe area.

3) A student that wants to know where the rent could be low but the safety high.

Now with those three use cases, we can specify the goals we want to set for our application. It must be said that the main idea and the main purpose of this application suits the first use case the most. However, we find the other two use cases equally valid.

Before we lay the goals for the application itself we should pick a platform we will develop for. Since the specified use cases do not clearly say what platform the user is using we are free to speculate. We have pretty much three options - PC, Mobile and Web. The PC can be crossed out due to its hard accessibility when travelling. If we were to develop a mobile application we would lose the part where a business person travelling with a laptop would quickly access the app, choose a spot and carry on with their work. Therefore it seems that the web application is a sweet spot in the middle and that is also what we are going to develop Locato for.

From the use cases, we can clearly see that the user should be able to pick a city he is interested in.

1) The user should be able to pick a city where he wants to perform the visualisation.

Another clear requirement is that the user should be able to pick specific data that he wants to visualise.

2) The user should be able to pick a specific data he wants to visualise.

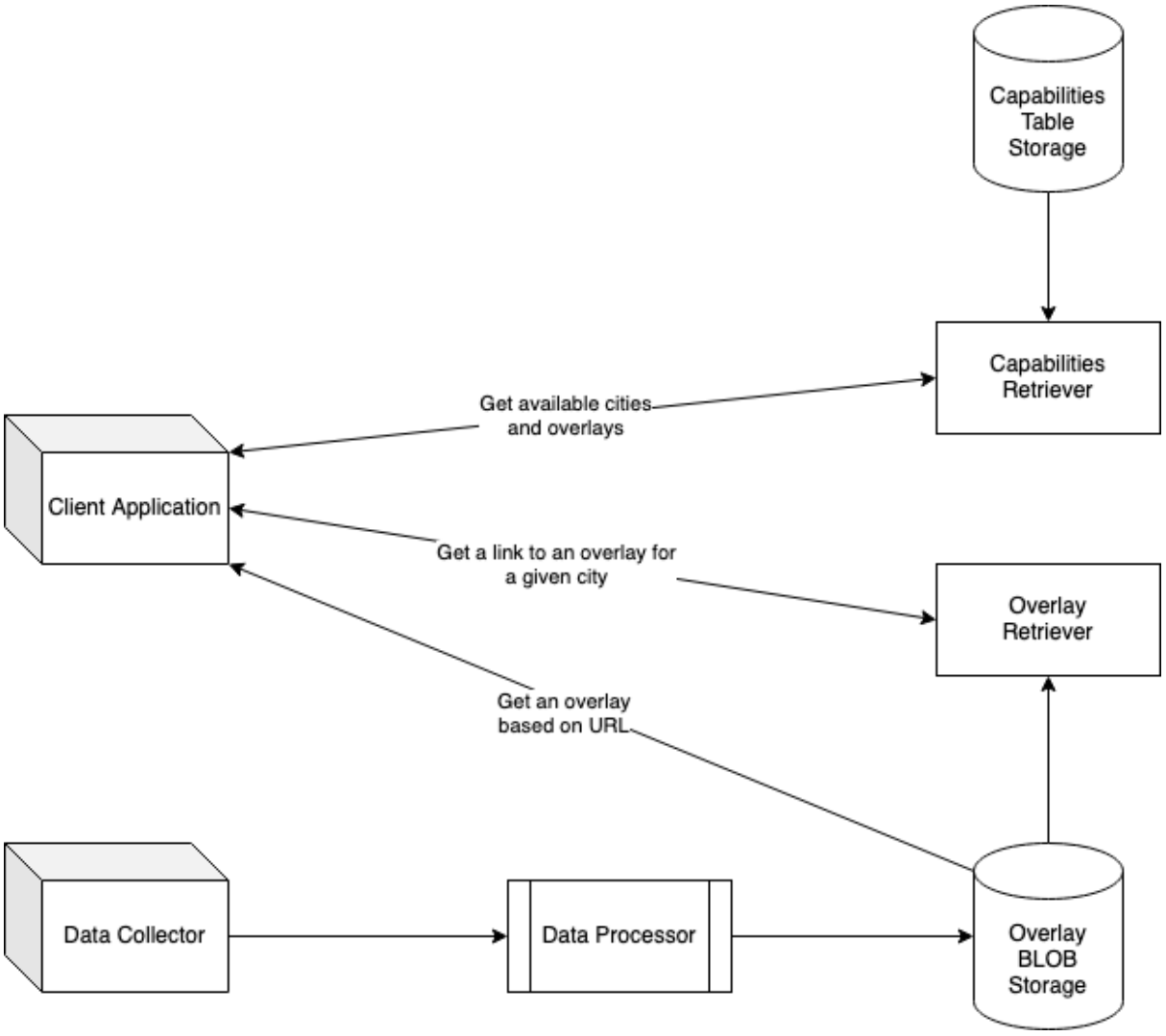
The data to visualise should be tightly connected to the interests of tourists, housing and quality of life.

3) The visualised data should be useful for tourists, the housing market and the quality of life.

As an example, the app should visualise the relative safety around the city of Prague.

Technical Documentation

The project consists of three logical components. The project's architecture can be seen on the following diagram.



The actors - the Client and the Data Collector - are interacting with the backend endpoints and databases. As the diagram suggests, there are two means of storing the available data. First, we store the capabilities of the backend. This comes down to a list of supported cities and for each city, a list of available overlays. Capabilities are stored in a database, which is operated by the backend. More specifically, by the Capabilities Retriever endpoints. Overlay is a specific JSON file: a so-called GeoJSON. The format encodes the data as points in GPS coordinates. More information on GeoJSON can be found at https://geojson.org/. This type of data can be stored in a BLOB storage and be accessed directly. The Overlay Retriever handles requests from the client containing a query for a city and a type of the overlay. As a result, the client receives a link, which can be used to access the wanted data directly (in a read-only mode, that is). This approach saves the backend's resources by making it the

client's responsibility to access the data. In the end, the data would have to be transferred regardless and we found this to be the most efficient way to do it.

Data Collection

Our project is based on several data sources. The main one is the geoportalpraha.cz portal which contains some statistics about the city of Prague. We collected and processed data about the noise

- https://www.geoportalpraha.cz/en/data/opendata/0AABB791-6C9B-4C9E-9262-2E3C48633EE5
- https://www.geoportalpraha.cz/en/data/opendata/F2207E6B-A08F-4BF3-97FF-9695D32D6384
- https://www.geoportalpraha.cz/en/data/opendata/BEE29CD7-98BA-4EB9-AB3D-079E6E338563
- https://www.geoportalpraha.cz/en/data/opendata/86489214-950D-4347-941E-C714F5F9E55B

About the safety

- https://www.geoportalpraha.cz/en/data/opendata/4A14E013-3B9D-4270-BEBE-64944C3DFA19
- https://www.geoportalpraha.cz/en/data/opendata/D7283D97-3909-4684-BA99-8FECCACBC2A6

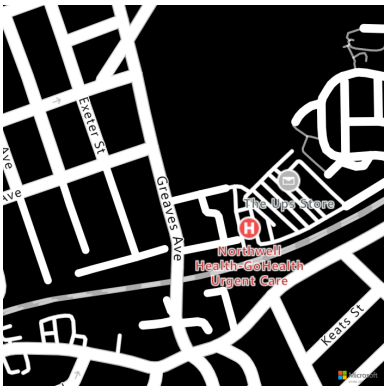
And about the bad climate or atmosphere

- https://www.geoportalpraha.cz/en/data/opendata/5BB4E2C5-9D4B-4B2B-BF0A-E0B98EE6013A

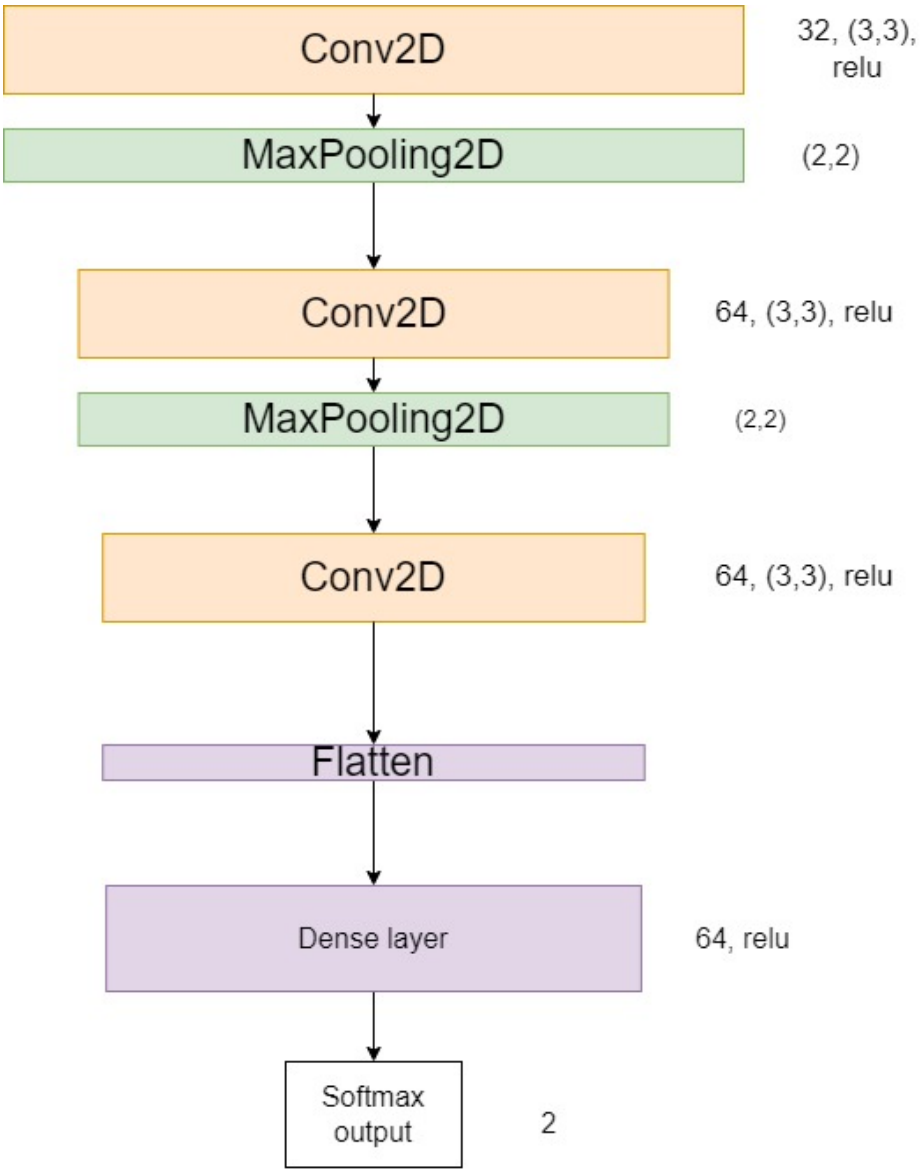
These data are collected using the collect_data.py script that takes some arguments to specify the behaviour we expect. After that for each dataset is generated a json file that the backend processes afterwards. Here is the example of the format

```
{ name: "Pollution in the area", color: "red", data: [ { lat:
14.496728031692513, long: 50.02016417623482 radius: 12.5, intensity: 0.401 },
] }
```

For the more general solution, we used the deep learning model for predicting possible unpleasant living based on the Azure open datasets of Boston, Chicago, San Francisco and Nyc Safety. We considered just a few categories such as drugs, graffiti, noise, ect.. that relates (for more information see the project readme and its files). These data are label gold data for map screenshots from the azure map api. So the sample feature for our CNN model looks like this, the 700x700 image of the map area . where the prediction says, if there is a potential dangerous or unpleasant condition.



The model is structured this way



Backend

The backend for our application Locato is completely written in C# leveraging the ASP.NET Core framework. For the minimal viable product, we have used Entity Framework as an abstraction for working with the database. Since the Entity Framework enables an easy switch to another type of database we can use SQLite for the development purposes and switch to a more scalable database later on.

The backend application is a simple Web API implemented using the REST principles. It is designed in a stateless manner in order to make the transition to serverless computing as easy as possible. We think the transition to a serverless model could save a non-trivial amount of resources in the long run.

The REST API contains basic CRUD operations on the database models. As said before the minimal viable product of Locato uses SQLite database which is by no means good enough for a production build.

The backend is hosted on Azure and is available at the following address https://azurecupbackend.azurewebsites.net/.

You can try to call a GET request that returns a list of currently available cities at https://azurecupbackend.azurewebsites.net/capabilities/GetAvailableCities.

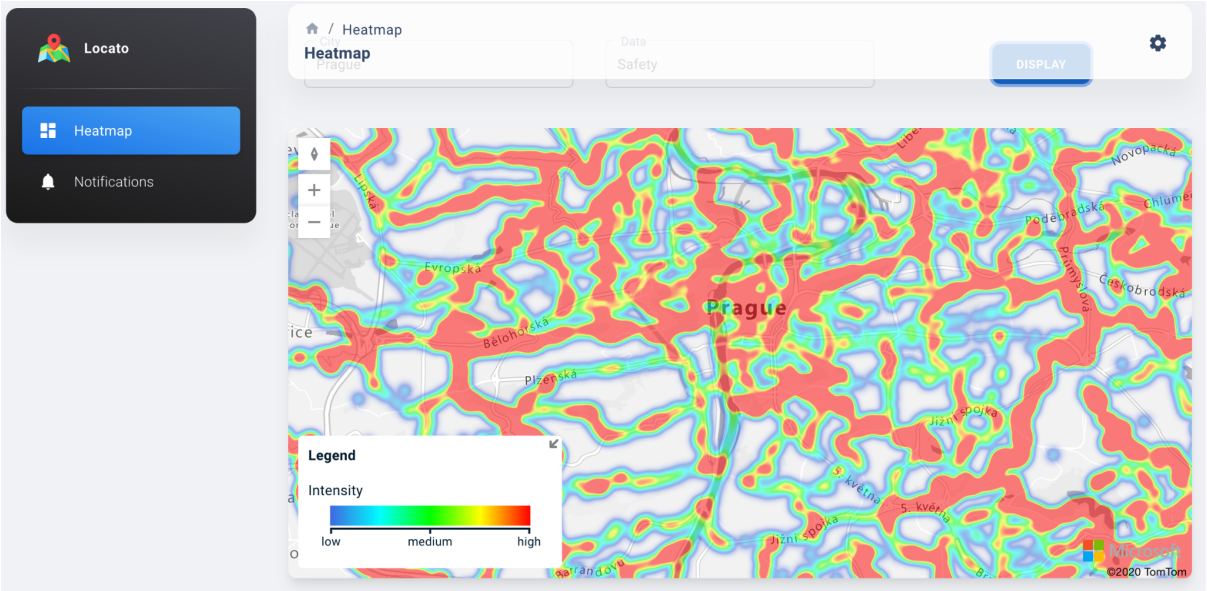
Frontend

For the minimal viable product, the client is implemented as a web application. A web application can be accessed from any device and thus is the best representation of what the project aims to accomplish. However, with the current architecture of the project, we have made it simple to implement a client as a desktop or a mobile application.

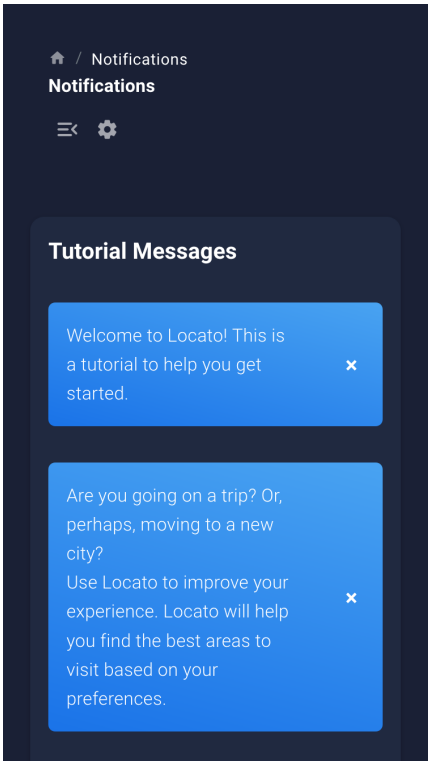
The frontend is written in ReactJS. It features a modifiable layout with light and dark themes, a navigation bar, and navigational colour choices. The navigation bar routes the user to the main page (Heatmap) and (currently) the tutorial (Notifications).

Since the project's business plan includes monetization based on a subscription model, we have laid out some ground for future components. The current user interface can easily host the future's profile page, payment page, and login page and registration pages.

The following pictures showcases the user interface - namely the main page. Here, the map component can be clearly seen taking up the majority of the viewport. The size of the map fits the device's height and leaves just enough leeway for the navigation bar. If one were to scroll up, they would see the option to choose a city and an overlay.



The picture below contains a part of the tutorial - currently done in the Notifications sections. The individual steps of the tutorial can be dismissed using the close button, however since the application in its current state does not feature user profiles, the tutorial resets after each reload of the page.



The code for the application can be easily extended by adding developer-defined components. The design follows the rules of MaterialUI, which guarantees unified user experience across a big range of devices and systems. Any MaterialUI control element can be seamlessly plugged in without looking out of place.

The core of the application is in the MapWrapper component. This component renders both the data selection dialog and the map component. The map is interactive and powered by Azure Maps. Once the overlay is chosen, it is rendered as a layer using the map's API.

The map component is highly customizable and allows great flexibility in the future. For example, the intensity and opacity of the heatmap can be altered. The style in which the data is represented can change as well, using weights and clustering.

User Documentation

It is expected that the user only interacts with the client application. The application, in its final form, will be distributed with the necessary API keys and secrets resolved. Currently, these are kept in a private manner and not made public.

Upon launching the client application, the user is greeted with the Heatmap page. Here, the user can specify the city by addressing the corresponding input field. A dropdown menu appears, revealing a list of available cities. Upon selecting a city, the user can then choose an overlay in a similar manner from the data input field. Once the overlay is selected, clicking the Display button renders a colourful heatmap in the map component below.

These steps are documented on the Notifications page, which is accessible by the navigation panel on the left.

Business Analysis

To perform our business analysis let's first summarise what we need to keep the application up to date and running.

For the data, Locato heavily relies on open datasets available online often maintained by national institutions. On the other hand, we can consider those datasets to be accurate and representative. Such datasets are mostly free of charge and available to use right away. Another approach is to buy some datasets online - there are plenty, or to create our own datasets by scraping maps and business reviews from google and creating our own by aggregating already existing datasets.

The resources needed to keep the application alive must be accounted for as well and will make the majority of our expenses if we exclude the pay of the developers that would either maintain or improve the application throughout time. The cost of the servers needed for the backend and frontend part of Locato should not be marginal and well scalable which is important for us. The majority of the cost is expected to be on the database storage and traffic. Since the main purpose of this application is to help tourists we can not effectively distribute our computing units across the globe without first having some real user data we could rely on. Also, the use of advanced techniques such as CDN would be very costly at the very beginning. We will therefore not consider any of those somewhat advanced cloud architectures and will focus on simple and straightforward solutions. The expected cost of running the application in the cloud should be in the tens of dollars tops.

Locato is primarily meant to be free of charge. To cover the cost and make some revenue we have several options. The first is to include advertisements. As a team, we have agreed that such types of applications are somewhat annoying to use and advertisements decrease the user experience by a large factor. Another approach is to make the application paid. That would make it more luxurious but also lower the influx of new users significantly.

The final option is to make the final application as a service. Meaning that the application would offer some cities and some specific data visualisation free of charge. The rest of the cities and data visualisations would be paid in a subscription manner.

From the surveys of our peers, we have concluded that the subscription is the best option. The very easy way this application is scaled makes it very convenient to adjust the price of the subscription on the run. The survey also revealed that most of the surveyed people

would be hesitant to use such an application. The most common reason was that they had not used any similar application before. We think it is possible to go from making ends meet to creating revenue in the order of months. This is however mostly a guess and is not based on any factual data.

Conclusions

The Locato project promises to make the lives of people who travel easier. It might also reveal interesting facts about a city a person has been living in. From the perspective of its developers, Locato seems to be a quick and easy to use way to gather large amounts of information about a city as a whole.

The minimal working product demonstrates the capabilities of the application perfectly. The business plan is fair for both the users and the investors. Used technologies cover nearly all of the Azure Cup 2022 competition's tracks, spanning from AI to open data and big data.

We have made sure to build on the Azure foundation, having successfully deployed and hosted the backend of the application. Azure Maps and Azure Datasets proved to be a great means of achieving our goal.