Hoods

HOusing Open Data Survey

André Gaillard, Demira Dimitrova, Giacomo Lombardo, Iohan Sardinha, Leonardo Menti

May 2022

Contents

Introduction

While the internet has facilitated accessing information fast, it has also introduced certain challenges with the collection of useful data. For instance, the sheer amount of pages that has to be clicked through in order to find the fine-grained piece of information wanted can become a daunting task. Furthermore, fact-checking and filtering for reliable sources has become increasingly difficult. This problem has become unavoidable in deciding where to move, in particular which neighbourhood in a given city. While indices of quality of life exist, they are often too coarse to hint at anything more than the city or even only the country. The choice of a neighbourhood, hereinafter sometimes referred to as *barrio*, depends on many factors. A family might only care about commute time and green spaces while an Erasmus student may be more interested in nightlife and closeness to the beach. Depending who you ask or which reference you consider, opinions differ tremendously, making the process of making a clear decision virtually impossible.

In this work, we try to tackle these problems by providing a concise, intuitive overview of different neighbourhoods of Barcelona. This overview contains a rating depending on different priorities given by the user. To do so, we combine a multitude of data from a variety of different sources into an index that transparently recommends barrios to the user. This results in a compact tool that can facilitate the tedious decision-making-process of choosing the barrio most suitable for given quality of life demands.

In the first section, we will describe the data we use for the compact barrio-rating. Then, we are going to give an overview of the implementation. In particular, we will focus on the algorithmic choices, the frontend and the obstacles we encounter. The next section will contain detailed information of how we port the application to the cloud and what services we use. After describing our work methodology in the penultimate section, we conclude the report with contrasting the work with our initial ideas and goals and an outlook for future work.

Datasets

In this chapter, we detail the datasets we collect and the rationale behind using them. The technologies we use to do so and the limitations of them are deferred to the implementaitons section. We want to note here that Barcelona is structured into districts such as Eixample. Each district is then further subvidided into barrios which is the level we operate on mostly in our application.

2.1 Barcelona Open Data for Real Estate Prices

No matter which clientele a possible user of our website belongs to, the price will always play a role of paramount importance in the choice of where to move. Modern developments such as gentrification, greenification, lowering speed limits make the development of prices in different barrios agile, and thus hard to follow. These fluctuations can be very fine grained, with neighbouring barrios moving into different directions. In order to collect reliable data about renting prices in the barrios of Barcelona, we consult official data of the Ajuntament Barcelona which is publicly available. The dataset could be found at [1]. The dataset contains historic as well as more contemporary data where the rental prices of different barrios are layed out (with a few exceptions). Additionally, they provide information of the price per square meter which allows us to make a prediction about the average flat sizes in particular barrios. Unfortunately, the dataset delivered by the Ajuntament Barcelona is not completely clean and has a few draw-backs. Consequently, we preprocess the data in order to put it in a suitable format. This process will be described in the chapter about the implementation.

2.2 Google Maps Data

Nowadays, Google Maps plays an outstanding role in the collection of data about locations. With ubiquitous internet, it is hard to imagine exploring places without consulting Google Maps (or an alternative) for further information. Here, the elaborate Google Maps API and its many wrapper libraries come in handy. In particular, they allow to use the full functionality of Google Maps and with it, retrieve locations and corresponding data of arbitrary categories. This allows us to collect and process information about quality-of-life indicators such as the bar-density, the average rating of restaurants in the area or simply the number of playgrounds for

children. As such, the data retrieved from Google Maps plays a central role in the process of rating the barrio for the user, taking into account its priorities.

2.3 News Data

While the rental data and Google Maps already deliver a good overview of the barrios, they lack agility and flexibility. Consider for example an area that is heavily gentrified. Consequently, bars and restaurants will be omnipresent and, generally, well rated. However, at the same time such city planning developments might trigger price increases angering the long-term inhabitants. In a very extreme scenario, this could lead to riots or similar which would not be accounted for by the data we use so far. Therefore, we also need to consider contemporary information. To do so, we scrape the official Ajuntament Barcelona website [2] which has a subpage for every district in Barcelona where some news and events are listed. This allows us to extract updated news for each district and to display them to the user. We quickly note that on a news level, we do not need the smallest city unit (i.e barrio) and thus, districts are enough. The reason for that is firstly the lack of data for the smaller city units, making the reporting on a finer level non-descriptive. Secondly, we believe that news of districts mostly affect all barrios within the district, making it an expressive source of information even on barrio level.

2.4 Miscellaneous

To improve the user experience, we also scrape the Wikipedia entry of each barrio for the first sentence which is usually a concise and descriptive summary of the place. Furthermore, we also scrape its main image from Wikipedia. This will not influence the rating that we give the barrio but rather serves to make our frontend more appealing and give the user some further context.

Rating Implementation

In this chapter, we explain the core of our application, namely how we implement the rating of the barrios. Furthermore, we lay out the obstacles that we faced and how we overcome them. In the first part, we go over the implementation details of collecting the data. The second part illustrates how we created the rating from the collected data.

3.1 Data-Centric Implementation

This section zooms into the implementation of the data processing to give a detailed overview of how we process the data and which problems we have to overcome in order to have expressive data that can further be processed into our final barrio rating. Every of the following subsections will describe the components that influences our barrio rating index. First, we describe the process of cleaning the rental data. Then, we look at how we deal with Google Maps data. In the final section, we explain how we scrape the newspapers to obtain contemporary information about the district. Note that the newspaper information is not used in the calculation of the rating, but only provides additional information for the user.

3.1.1 Rent Data Preprocessing

The Barcelona Open Data dataset `est mercat immobiliari lloguer mitja mensual` [1] provides a `.csv` file for every year from 2014 to 2021. Each file contains information about the average monthly rent and the per m^2 price in Euro for every barrio in the city of Barcelona. We expect the i-th row of the file to indicate that price information for a certain barrio of the city for every quarter of the year. However, it turns out that the prices per m^2 and the average rental price are scattered across the rows in the original `.csv` files. To illustrate that, we provide a few rows of one of the `.csv` files in table 3.1. It can be seen that the units of the row `lloguer_mitja` changes from monthly prices to monthly prices per square meter. The problem here is that this format complicates database queries for later retrieval and processing of the data. To deal with this problem, we opt for an offline processing step using simple Python scripts.

Any	Trim	Codi_Distr	Nom_Distr	Codi_Barri	Nom_Barri	Lloguer_mitja	Preu
2021	1	1	Ciutat Vella	1	el Raval	Euros/mes	759.2
2021	1	1	Ciutat Vella	1	el Raval	Euros/m^2 mes	12.8
2021	2	1	Ciutat Vella	1	el Raval	Euros/mes	774.2
2021	2	1	Ciutat Vella	1	el Raval	Euros/m^2 mes	12.9

Table 3.1: Example illustrating the need for aggregation within the provided tables

The final goal of the preprocessing step is to unify the price information in a single row, so that for barrio i both prices are present in row i. To stay consistent, we conduct the preprocessing step for every `.csv` file and aggregate them in a single big Pandas Dataframe.

Another problem we face is the inconsistent encoding if the characters in the dataset. Recall that the dataset is in Catalan. There seem to be some problems when decoding the special characters. To deal with that, we simply clean the data replacing the wrongly encoded characters so that they are correctly encoded on our machines.

This concludes the preprocessing step for the open-data. We provide a quick example of the final table (see table 3.2) which we later use for the analysis of different barrios and giving feedback to the user.

Year	Codi_Barri	Nom_Barri	Codi_Distr	Nom_Distr	Preu	PreuM2
2021	1	el Raval	1	Ciutat Vella	766.7	12.85

Table 3.2: Final layout of the preprocessed rental data

3.1.2 Priority Data Collection

In order to assess the quality of life of a barrio tailored to the individual interests, we need to define a few categories that the user can use for prioritizing the lifestyle that it wants to live:

- Restaurants & Cafes: This is the category that can be important for all kinds of clienteles, good cafes and restaurants often imply a lively barrio with lots of spaces to enjoy life.
- Nightlife & Bars: Here, we target a younger crowd that wants to live close to the nightly pulse of the city.
- Parks & Playgrounds: This category is mostly family centric. However, it also targets users that want to spend time outside in green spaces without being molested by cars or other city noises.
- Closeness to beach: As Barcelona lacks sweet water while being a Mediterranean city, closeness to beach can be a decisive factor for all groups of people alike. Note that this feature has not yet been implemented.

We choose this categories to be in accordance with the categories that Google Maps offers. This means that the collection of this data just amounts to simple API

calls to the Google Maps API. The broad overview of our collection algorithm works as follows:

1. Fetch the category data for each barrio center
2. Filter responses without rating
3. Filter responses that are out of barrio bounds
4. Store the ratings for further processing

The rationale behind filtering places without rating is that they do not provide information about the quality. Either they are very new and have not had enough visits, or they are very rarely frequented. We are aware of the fact that by only taking ratings of Google Maps can lead to skewed results, especially since the rating community of Google Maps might not be the most active. Taking other rating pages like Tripadvisor into account can make the scores more stable, more accurate and also more representative of the real quality. However, Tripadvisor, for example, does not provide a free API and is therefore out of scope for this project and left for future work.

Another problem we face is the inconsistency in retrieving of data. The Google Maps API might be elaborate, but it is also obscure. This leads to a multitude of problems, mostly because we send large amounts of requests at the same time. We provide a list of shortcomings that we encountered, and provide an idea of how one could overcome some of them:

- For some barrios and some categories, no responses are found at all. This is inconsistent with our experience when debugging and consulting the Google Maps website for the same data.
- Over different runs of scraping, different data is retrieved: We believe that this must come from the inner implementation of the Maps API and that there is nothing to be done about this. Since the differences over different runs are rather small, we consider this problem negligible.
- Even though we specify the types of objects we want to retrieve (i.e `park`), we retrieve too many objects for some categories in some specific barrios. For example, for the barrio *Raval*, we retrieve 21 parks which objectively cannot be true. This leads us to believe that the Google Maps types for search are rather slack. However, there is no way to figure out the exact definitions. An approach to solve this problem could be to call Google Maps again about information of the retrieved entity (by entity id) and to try to filter them by looking at the response and determining the real type.

Despite the above drawbacks, we consider the data retrieved usable and highly valuable in determining the quality of a barrio given certain priorities.

On a quick note, we make the simplifying assumption that the Google Maps data is static, meaning that we retrieve them only once. This also makes it feasible from a performance point of view since the calls to the Google Maps API are too slow to

deliver all the information in real time to the users. It is very easy to change the behaviour so that the data is automatically retrieved every now and then to update the values to reflect contemporary developments.

3.1.3 Commute Time

After having collected data to respond to user preferences in terms of quality of life, we still lack one big factor of the decision of where to move: commute time. Given an address that could be an office or a study place, we consider the commute time to that place from each barrio. For that, we collect the centroid of all barrios in terms of longitude and latitude. We briefly describe how we obtain these centroids:

There is no way to retrieve the polygon boundaries from the Google Maps API even though they can be seen in the vanilla Google Maps implementation. Therefore, we scrape the polygons from `wikimapia.org`, which has a user built database of polygons which are also editable by users. When clicking on a barrio in Wikimapia, a query to its database is triggered and a response containing the polygon points is returned. With some automation and browser console scripting we are able to retrieve the polygon information. As described in the frontend section, the polygons are not only used for the commute time but also for the display of the barrios in the frontend.

We compute the commute time by calls to the Google Maps Distance Matrix API. Using the commute location as destination and the barrios centroids as starting point. It is possible to retrieve up to 25 destinations per request, so three requests per user query suffice to fit all the barrios.

Each request has a mode of transport associated, which can be public transport, walking or car. We collect information for two different modes of transport: walking and public transport. The reason behind not using the car is the general assumption that commutes within the city by cars are mostly avoided.

The API then responds with the commute time in seconds for the queried starting point, destination pair which we later use to compute the overall barrio scores which we discuss in the following section.

3.1.4 District News

In order to get the news for every district we create a script to periodically scrape from the site Ajuntament Barcelona. The script starts fetching the data from a page that contains the links for every district of Barcelona. We move through every link to access the district web page and then we retrieve the information about every article we find on the page. We then store the news data for every district in an item that is illustrated in the following example. There, we can find the id of the district, the district name and a list where each entry is one news article, containing the date, the title and the URL for further reference.

```
{
 "id": 2,
 "district_name": "L'Eixample",
 "news": [
  {
```

```
    "date": "24/05/2022 - 11:01 h",
    "title": "Desactivado el aviso preventivo de contaminacion por PM10",
    "url": "https://ajuntament.barcelona.cat/eixample/es/noticia/desactivado-el-aviso-preventivo-de-contaminacion-por-pm10-5-2-2_1178224"
   },
   {
    "date": "23/05/2022 - 17:05 h",
    "title": "  Barcelona   Activa te conecta ",
    "url": "https://ajuntament.barcelona.cat/eixample/es/noticia/barcelona-activa-te-conecta_1178005"
   }
  ]
}
```


Recall that scraping is rather slow. Thus, we decide to do the scraping sporadically rather than do it on an incoming request. In particular, we decide to put the scraper script on an EC2 instance and trigger it once per day. We are dealing with key-value format here, therefore, whenever the script is triggered, we upload the data to a dedicated database on AWS DynamoDB so that the information can be retrieved on request. The script itself is run sporadically on an EC2 instance.

3.2 Rating Creation

In this section we discuss the scoring formula that is applied for each barrio depending on commute time and the user interests. The score can conceptionally be described as a weighted average of the different scores from rent prices, commute time and locations of interest taking the user preferences into account.

The weights are always 1 for the rent and distance and either 0 or 1 for locations of interest. It's 1 if the user has specified any priority, 0 otherwise. It is important to note that sometimes a barrio does not have any location of a type specified by the user as priority. In this case the weight is still 1 according to user interest. However, the score itself is 0. This way each individual score contributes equally to the final score.

We note that due to the above, a zero in a individual score lowers the final score but never lowers it to 0. This is important because scores below a certain threshold are reduced to zero. We discuss this in more detail below. Consider, for example, a barrio that is very far away and therefore has distance score of 0. However, if the barrio is very cheap in terms of rental prices, it can still score a considerable final rating. The logic behind this is the empirical experience that a user might be willing to trade off long commute times against a very attractive price.

The three individual scores have a particular way of being computed each of which will be discussed individually. In the end, the overall formula is presented.

3.2.1 Rent Score

The score of the rent is a measure of it's interpolation inside a range based on the deviation from the average rent price.

The average rent price per barrio is a simple average of the prices of each quarter in 2021. The deviation is experimentally tuned and is the average divided by 2.5. This way the score will be the interpolation of the price of a barrio in the range a minimum price to a maximum price, such that the minimum is the average minus the deviation and maximum average plus the deviation. This way, any price between the minimum and maximum gets a score between 0 and 1 which is also where we clip the values. To make the score more understandable for users, increasing the overall experience, we multiply the value by a factor of 5 to get a 0 to 5 rating.

3.2.2 Distance Score

The distance score is a the interpolation of the time needed for commuting discounted by a minimum time of 10 minutes in the range between zero and the maximum commute time which can be specified by the user, defaulting to 60 minutes. Similarly to the rent score, we clip the values to 0 and 1, to make sure that we are consistent in the way we rate. Finally this score is multiplied by 5 to convert the score to a user-friendly measure.

The possibility to change the minimum commute time (defaulting to 10 minutes) is implemented in the backend. However, we decide to not include the option in the frontend in order to not confuse the user.

3.2.3 Locations Score

Finally, we need to rate the points of interest according to the priorities of the user. These scores are precomputed and therefore ready-to-retrieve in the database. Recall that the scores of the points of interests are the ratings retrieved from Google Maps. To score a barrio, we just take the average of the ratings which is already in the desired range, making normalization obsolete. Some barrios may not have any location of a given type, in this case the type score is considered 0.

Recall the types of points of interest: restaurants & cafes, bars & nightlife and parks & playgrounds. Each of them can be specified as a priority by the user. If the user does not choose a particular type of priority, its weight is 0, otherwise it is weighted with 1. If all of them are 0, the point of interest score won't be considered in the final score of the barrio.

While the scoring system has proven to work well in our application, we want to note that it is possible to make the score more expressive. An idea that comes to mind is to smooth the weights of points of interest. For example, a user might not prioritize a lot of restaurants in a barrio, but it still contributes to the overall quality of life (i.e a nice-to-have). In order to implement such a scheme, more data and testing would be required. We therefore defer it to future work.

3.2.4 Score Formula

$$Score = \frac{RentScore + DistanceScore + LocationScore * HasLocations}{2 + HasLocations}$$

$$RentScore = min(max(RentScore', 0), 5)$$

$$RentScore' = 5 * \frac{1 - (Rent - MinimumRent)}{MaximumRent - MinimumRent}$$

$$Rent = \text{Rent of individual barrio}$$
$$MaximumRent = AverageRent + \frac{AverageRent}{2.5}$$
$$MinimumRent = AverageRent - \frac{AverageRent}{2.5}$$

$$DistanceScore = max(DistanceScore_{Walking}, DistanceScore_{PublicTransport})$$

$$DistanceScore_i = min(max(DistanceScore'_i)); i \in m$$

$$DistanceScore'_i = \begin{cases} 5 * \frac{1-(Time-10)}{MaxTime} & \text{if time¡} MaxTime \\ 0 & otherwise \end{cases} i \in m;$$

$$m \in \{Walking, PublicTransport\}$$

$$Time = \text{Time from commute location to individual Barrio}$$
$$MaxTime = \text{Maximum time the user is willing to take in the commute}$$

$$LocationScore = \frac{\sum_{i=t} LocationScore'_i * HasLocation'_i}{\sum_{i=t} HasLocation'_i}$$

$$LocationScore'_i = \text{Average Score of i kind of location in Google Maps}; i \in t$$

$$Haslocations'_i = \begin{cases} 1 & \text{if i kind of location is important for the user} \\ 0 & otherwise \end{cases} i \in t$$

$$t = \text{Types of locations} = bars, cafes, discos, parks, playgrounds, restaurants$$

$$HasLocations = \begin{cases} 1 & \text{if at least one kind of location is important for the user} \\ 0 & otherwise \end{cases}$$

Front-end Implementation

In this section we are going to detail the implementation of the front-end of our application. More specifically, we will discuss the design choices and the frameworks and libraries used in order to develop the final product.

The front-end side of the application is built using the React framework, in addition with some libraries that will be presented later. This design choice is driven by the need of a single web-page application with frequent and dynamic interactions both with the user and with the back-end functionalities. By using React, we can do set up a single page application in a very quick fashion and subdivide the implementation of the various functionalities into the typical component-centered paradigm used in React apps. Additionally, to provide a clean and responsive web-app we implement some simple CSS provided by the Bootstrap library for React. Although nowadays, there is a multitude of websites with very generic and similar designs due to the popularity of Bootstrap, we decide to go for it given our experience with its features and the very intuitive implementation of it. Additionally, designing a unique looking website can be a decisive factor for attracting new customers to the application.

4.1 Front-end Functionalities

The web application provides some very simple functionalities to visualize the data and interact with it which we describe in the following.

4.1.1 Barrio scores visualization

The main functionality of the application is to rate the various barrios of Barcelona given parameters specified by the user resulting suggested areas for moving into the city.

This functionality is implemented with a simple, interactive form that lets the user provide the parameters needed for the score computation. The frontend then display the results in an intuitive way through a map representation.

More specifically, the input form consists of:

- **Address input**: in this field the user inserts his/her workplace address (or any address the user may find interesting), helped by the Google Maps API Autocomplete feature. This is the only required field in the form and an additional check is performed to verify whether the inserted address is in the city of Barcelona or not.

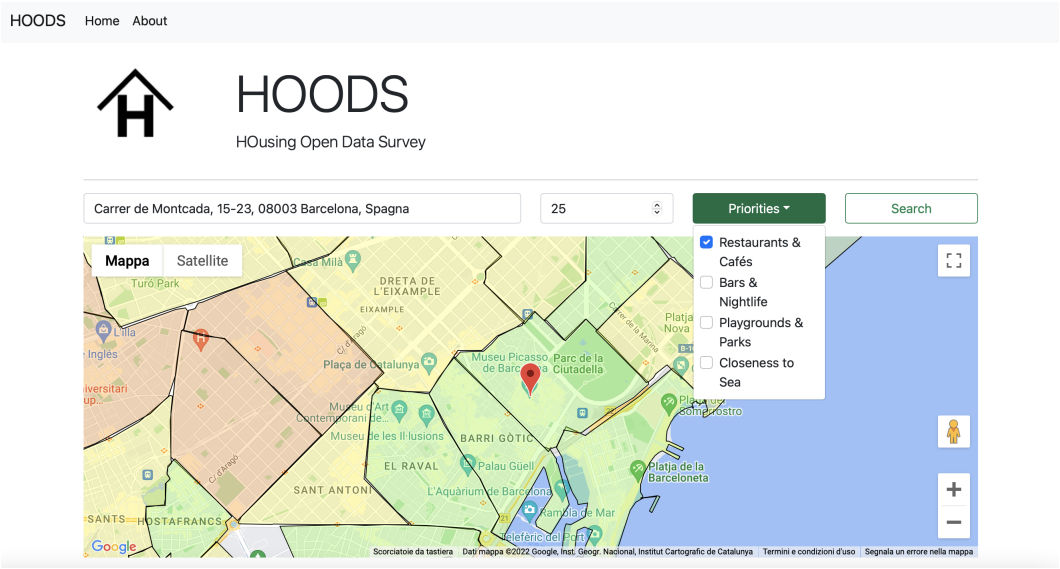

Figure 4.1: Input form and visualization of the Barrio scores

- **Maximum commute time**: the user can additionally specify the maximum amount of minutes that he/she intends to spend doing home-to-work commuting.
- **Priorities**: these check-boxes allow the user to specify his/hers additional priorities, already described in the previous sections of the document.

Once the form is submitted, the back-end API is invoked and when the results are ready the map is updated by coloring the various barrios accordingly to their score. The various colors, ranging from red to dark green, provide an intuitive overview of the recommended areas.

4.1.2 Barrio info visualization

Additionally, the user can interact with the map by clicking on a certain Barrio in order to visualize its overview. This is performed through another API call that retrieves the desired information, which is then displayed in an overlapping window as it can be seen in 4.2.

4.2 API Interaction

To implement the dynamic functionalities of our web-app we need to provide a way to interact with the various APIs designed. Another strength of React is the easy implementation of such back-end to front-end interaction through the various libraries for HTTP requests. In our application, we use the very popular Axios library[3] that allows us to quickly design interactions with the API and provides asynchronous functionalities to dynamically update the web-page in a responsive and user-intuitive fashion. For instance, one crucial function is the retrieval of the scores given the address and the additional parameters provided by the user: the

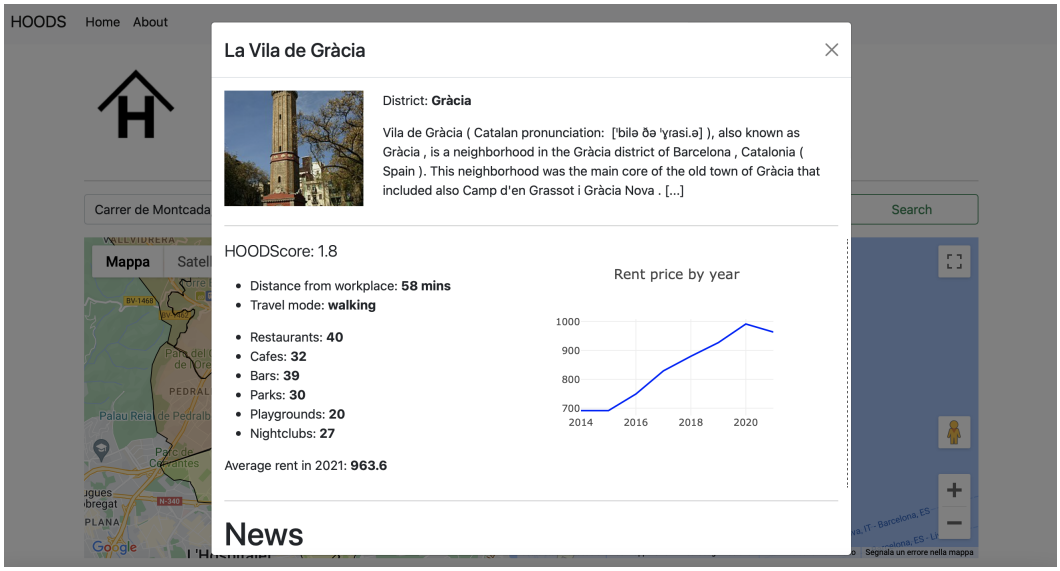

Figure 4.2: Visualization of the additional information of a Barrio

API call in question is quite computation-heavy and can require a couple of seconds in order to provide the results. Through using Axios, we can dynamically update the page to inform the user that the results are loading while waiting for the API response and then display the received information once the results are ready.

4.3 Google Maps API

Given the fact that our web-app revolves around geographical data, a crucial functionality of our front-end is the representation of it in a clear and appealing fashion. In order to do so, we use the Google Maps JavaScript API library for React [4] to implement the front-end functionalities that involve processing of geographical data. Thanks to the API, we are able to display firstly a map to visualize Barcelona and to draw on top of it the various Barrios, which are then colored according to their score. In order to do this, we use the Polygon component offered by the Maps library that allows us to display on the map the borders of each barrio, which are stored in JSON format in a static file.

Additionally, we can design more interesting interactions with the map: for instance, when clicking on a barrio the user is served with its overview as well as with its score and the various data previously described in this report. Consequently, the rating is transparent and easy to understand for the user.

Backend Implementation and Deployment

This chapter is describes the process and decisions taken to host the full application on the Amazon Web Services (AWS) Cloud provider. Since the system consists of several parts, each of these are mentioned in a separate section.

5.1 Secrets Manager

AWS Secrets Manager helps to protect secrets needed to access any resources in combination with fine-grained permissions. The service makes it possible to easily rotate, manage, and retrieve database credentials, API keys, and other secrets throughout their lifecycle. [5] Several of the components deployed on the Cloud require the use of secret API key in order to Access the Google Maps API. The secret is stored in the manager and accessed through code on the places where it is required.

An example of a secret retrieved in Python code:

```
client = boto3.client('secretsmanager')
response = client.get_secret_value(
            SecretId='MAPS_API_KEY'
            )

MAPS_API_KEY = response['SecretString']
```

5.2 AWS Lambda

AWS Lambda is a serverless, event-driven compute service that enables to run code for virtually any type of application or backend service without provisioning or managing servers. [6] We decide to deploy the components that perform computations as a Lambda function due to its intrinsic scalability and its pricing system. Lambdas are payed for based on duration of running time and number of requests to the function, while data transport is free within the same region with DynamoDB and S3.

The application uses two separate Lambda functions. Both of them check the received request and implement only GET functions with the correct input parameters. One of the Lambdas is used to calculate the score of a barrio following the method described in the section 'Score formula'. This function is accessing the database with rent data and making calls to Google Maps API to retrieve the commute time from the user input address to each of the barrios. Based on this data

is following the Scoring formula to calculate a score per barrio specific to the input parameters. The function is returning a JSON object for all of the 73 barrios, with the following structure:


```
{
      "nom_barri": "el Raval",
      "mode": "public transport",
      "duration_value": 7.416666666666667,
      "duration": "7 mins",
      "score": 3.935110746317873
 },
 ...
```


The second Lambda function is provides more detailed information per barrio such as the number of cafes, bars, picture, rent price per year, description and news. The collected information is passed to the front-end to display in the overlapping window on click. The only input parameter needed is the unique barrio id. The function accesses and queries our rental data database based on this id. Furthermore, it pulls the scraped news data from another table in the database and similarly pulls the specific data per district id. The function returns response a JSON object in the following structure:


```
{
 "cafes_avg": 4.168421052631578947368421053,
 "discos_num": 2,
 "img": "https://upload.wikimedia.org/wikipedia/commons/thumb/6/61/
Poble_nou-_barcelona_-_panoramio.jpg/300px-Poble_nou-_barcelona_-
_panoramio.jpg",
 "playgrounds_num": 12,
 "Codi_Districte": 10,
 "parks_num": 16,
 "Preu": {
      "y2020": 1013.6,
      "y2021": 1076.6,
      "y2014": 649.21,
      "y2015": 649.21,
      "y2016": 685.35,
      "y2017": 987.18,
      "y2018": 999.62,
      "y2019": 999.63
 },
 "bars_avg": 3.96857142857142857142857143,
 "discos_avg": 4.3,
 "Nom_Barri": "Proven als del Poblenou",
 "center": {
      "lng": 2.2045898466666665,
      "lat": 41.4113721
 },
 "restaurants_avg": 4.00526315789473684210526315 8,
 "parks_avg": 3.94375,
 "Nom_Districte": "Sant Mart ",
 "cafes_num": 19,
 "bars_num": 35,
 "playgrounds_avg": 2.7,
 "PreuM2": {
      "y2020": 1013.6,
```

```
        "y2021": 1076.6,
        "y2014": 649.21,
        "y2015": 649.21,
        "y2016": 685.35,
        "y2017": 987.18,
        "y2018": 999.62,
        "y2019": 999.63
    },
    "restaurants_num": 38,
    "description": "Proven als del Poblenou  is a neighborhood in the Sant Mart  district of  Barcelona ,  Catalonia  ( Spain ).",
    "id": 70,
    "news": [
        {
            "title": "Desactivado el aviso preventivo de contaminaci n por PM10",
            "date": "24/05/2022 - 11:01 h",
            "url": "https://ajuntament.barcelona.cat/santmarti/es/noticia/desactivado-el-aviso-preventivo-de-contaminacion-por-pm10-5-2-2_1178224"
        },
        ....
    ]
}
```

Generally, it is a preferred pattern to keep the Lambda functions as simple as possible and refrain from monolithic deployments. Those two Lambda functions together perform all of the computations needed for the functionalities of the system.

5.3 API Gateway

AWS provides an API Gateway which can handle all the tasks involved in accepting and processing up to hundreds of thousands of concurrent API calls, including traffic management, CORS support, authorization and access control, throttling, monitoring. [7] The Lambda integration, uses the API gateway as a proxy. It allows for a defined event JSON to be expected and returned by the Lambda function instead of building our own and managing the routing. Furthermore, by using this service we can route any HTTP method to the Lambda.

In order to trigger the Lambda execution, we have created two REST API gateways. Sending a request to one of them, would trigger the corresponding function. The gateway is granted permissions to execute the triggered lambda. The service is also responsible for receiving the response from the function and responding to the request it has received on its own.

It would be possible to unify the API's and provide only one gateway to the front-end to query. In this example scenario, the two lambdas would be triggered depending on the path of the request to the API. However, assigning an API proxy per Lambda allowed us to work easily with logs and monitor requests.

5.4 DynamoDB

DynamoDB is a NoSQL proprietary database service provided by AWS. It is structured as document based key-value structure. In this project we have deploy two tables on the DynamoDB in AWS. One is serving for keeping the rent data and another one is being populated with the scraped news every 24 hours.

The service comes with auto-scaling provisioning, which is useful for our case during the initial upload of the rent data from the open source. The auto-scaling activities are pre-configured and tracked separately for read and write. On the time of data population the capacity units for writing reach 7 and are reduced to 1 unit after some time.

There is no need to use or store database credentials to access because all of the components connect to each other through the permission policies of each service.

5.5 EC2 instance

Amazon Elastic Compute Cloud (Amazon EC2) provides scalable computing capacity in the Amazon Web Services (AWS) Cloud. [8] EC2 serves us as a virtual server instance to which we could access through command line and run code on. The cloud provider is responsible for scaling the chosen infrastructure while the developers can choose the specific type of server to run. We are using the smallest kind `t2.micro` of Linux machine which is available within the free tier for 750 hours.

For the aims of this project, we made use of an EC2 instance to host and run the code for scraping the district news from the selected websites. The script is being executed through a cronjob on the server that is setup to run every 24 hours. Each time the script is executed it connects to the table in DynamoDB and populates the new data. The EC2 machine is granted permissions to write to the database.

5.6 S3

Amazon Simple Storage Service (Amazon S3) is an object storage service offering scalability, data availability, security, and performance. Customers of all sizes and industries can store and protect any amount of data for virtually any use case, such as data lakes, cloud-native applications, and mobile apps. [9] Moreover, static applications could be stored on the service, which is the case for the frontend of this project.

In order to provide a functioning website, we store in an S3 bucket the front-end application, after it has been built in a local environment. By enabling public access and website hosting in the S3 settings, we are then provided with an accessible URL in order to make our website publicly available.

5.7 CloudWatch logs

Availability of centralised logs are a crucial part of the deployment of any system. AWS provides us with the CloudWatch service to store and access logs from most of the AWS services that are used.

During deployment of the aforementioned services, we use these logs to debug issues between the different components. For example, when configuring the API Gateway proxy to communicate with the Lambda function one of the issues we encountered is that Lambda computes the correct results but did not return them in the expected format to the proxy and thus, the proxy returned an error. The way to follow such a request track is to refer to the logs for the corresponding service. There we filter the request ID and find an eventual exception or error code.

All of our services have CloudWatch monitoring enabled that could be useful in the future to respond to more operational issues.

5.8 Identity and access management

A core service of AWS is the Identity and access management(IAM), which serves to define roles, policies, users and user groups within the AWS project.

One of the requirements we have considered since the beginning is that each of the developers accessing AWS resources must have a separate account so that permissions could be managed granularly. At first we create an organisation to hold the member accounts for the project. AWS Organizations is integrated with other AWS services so you can define central configurations, security mechanisms, audit requirements, and resource sharing across accounts in your organization. [10] However, considering the size of the project and time limitations, we come to the conclusion that such an approach is too cumbersome. Managing an organisation on AWS allows to create up to four different accounts which are to be used as Organisational Units. Moreover, Policies and User groups should be applied to those units. In this project in contrast, all of the members are expected to access and modify resources on the cloud provider. Therefore, the creation of individual users from the root account is the chosen solution. Under other circumstances where longevity and further scaling of the project is expected, the team is aware that it must use an AWS Organizations.

Since all of the members are expected to access the full system and each of its components, the policy group assigned to the generated Users is covering administrator access. The user accounts are provisioned with an auto generated password and on the first login into the account they must create a new password. Once a user is logged in, the account shows the Root Account ID and then the username for the IAM user like in the following example: `Account ID: 9589-5784-4018, IAM user: Demira`. Moreover, it is possible to audit access to services per individual user.

An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. An IAM role is similar to an IAM user, in that it is an AWS identity with permission policies that determine what the identity can and cannot do in AWS. [11]. In this project we have worked with several roles as each of the system components has attached its own role covering the access to other services.

Each of the Lambdas, API gateways, EC2 instances and S3 buckets has a corresponding individual role with policies to grant access only to the services for the specific functionalities. For example, only one of the Lambdas needs access to the Secret manager in order to retrieve the API key to the Google maps API, while the other Lambda function has no need for those permissions and thus, it does not have them.

Methodology

In this chapter, we first briefly describe our methodology, group organisation and the division of responsibilities. Then, we have a closer look on how we implemented the 12-factor methodology.

6.1 Working Methodology

While implementing an elaborate SCRUM methodology would be out of scope for this project, we use a Kanban-like workflow using a Trello board. The progress is discussed in about three meetings per week where also the responsibilities are newly distributed if need be. The meetings usually consists of a round where everybody details the progress that has been made and the obstacles that have been encountered. In a quick discussion round, the problems are addressed with fresh inputs of the other group members. In general, we distribute one task per group member. However, if there are seemingly insurmountable obstacles within certain tasks, we add a second team member to the task, mostly depending on the previous experience with the topic containing the issue.

When finalising the application, as the tasks get more and more coupled, we let go of the one-task per person policy and work all together to combine the different components and finalize the full application.

Due to a limitation in time, we refrain from creating different working environments. Generally, we develop and test locally and deploy as soon as a functionality is ready. From a versioning point of view, we keep things simple. In particular, we use git where we mostly work on one branch. This is possible since all group members usually work on different parts.

We steadily adapt the documentation on the git-repo for the other users to be seen. It has to be noted, however, that the technologies used (i.e python and nodejs) are not particularly prone to build and install problems, since they are shipped with elaborate dependency managing systems.

6.2 12-Factor Methodology

We want to give an overview of the 12-factor and how we use or why we may not use each of the practices:

- Codebase: Due to the simplicity of this application, we do not follow this point completely. In particular, during working on the application, we host all the scripts and the frontend in the same repository. However, our lambda functions (backend) are hosted in AWS directly and the deployed frontend (after running `npm build` is pushed to a separate repository.
- Dependencies: Each python script is maintained using pip's `freeze` functionality. The frontend is hosted statically and the lambdas stay within the AWS editor. Therefore, this principle is completely followed by our application.
- Config: All of our configs are stored in the AWS secret manager, leading to this principle being fully satisfied.
- Backing Services: We follow this principle by using AWS Dynamo DB and accessing it from within the lambda functions.
- Build-Release-Run: This principle is out of scope for our project. As soon as we have tested the application, we add it to the source control and deploy it.
- Port Binding: We hold this principle by routing through the API Gateway API
- Concurrency: Since we do not expect many users, we break this principle by not developing the app to be concurrent. However, AWS Services like the API Gateway and Lambda introduce some scalability for us.
- Disposability: Dealing with startup and shutdown is out of scope for this project due to time and user limitations.
- Dev/Prod Parity: Since we do not distinguish between development and production deployment, we do not follow this principle in our application.
- Logs: We adhere to this principle by collecting the logs of our services into the AWS cloud watch.
- Admin-Processes: Since we are a small group with a rather small project, we abstain from separate administrator processes and thus do not stick to the principle. However, we do use some role management to implement the most-restrictive permissions policy for the services.

Outlook

While we present a fully functioning app, it still has a lot of room for improvement. In this section. We contrast the final application to the initial one that can be found in the draft document to. Additionally, we discuss shortcomings, how we can improve them as well as unimplemented features. We conclude the chapter with a general outlook.

7.1 Deviation from the Draft

This section is organized in two parts. The first part explains the functionalities that have been discarded from the original set of ideas and why this was the case. In the second part, we contrast the initial time planning to how the time has actually been spent and why this was the case.

7.1.1 Discarded Functionalities

The following is a comprehensive list of discarded functionalities, frameworks and infrastructure parts that were originally planned to be in the app:

- Elastic Beanstalk: The idea behind using AWS Elastic Beanstalk (PaaS) was to simplify the deployment of our backend that was supposed to serve the frontend statically. The choice of PaaS suggests itself for this kind of project since we do not need a lot of control over the configuration of the platform. However, during development, we decide to replace our full fledged Django-REST-backend with a serverless backend using Lambda. We justify this choice with the lack of state in our backend due to the queries to retrieve the data from Google Maps being to slow for realtime data collection. Instead we store the precomputed data in the database and retrieve it on request while it is sporadically updated. This makes a stateful backend obsolete. Therefore, we replace it with Lambda functions which is also more cost-saving since we pay per request.
- Codepipelines: Even though CI/CD pipelines would have been nice to have, they had to yield to other priorities due to a limitation in time. In the end, our application does not consist of that many microservices nor is it particularly difficult to build. This is why the removal of CI/CD technology is easily accepted.

- Django Backend: The reason why we abandoned the Django Backend is already stated above in the paragraph about Elastic Beanstalk. We therefore refrain from detailing it more.
- Elastic Search/Kibana/Tableau: In the end, we decide against using a graph visualizer. While we do present a graph, it only consists of historical data of four points in time. Therefore, using a javascript library to plot it is more than enough.
- Docker (Containerization): While container really facilitates deployment on different machines while decreasing start up time of applications, they are not needed for our services due to their structure. Using only a static frontend and a serverless backend makes the use of docker unnecessary.

7.1.2 Time Planning Differences

We present the difference in time planning. The results can be found in table 7.1. In general, we are very satisfied with our planning. The reason why the hours spent on data collection exceeds the time planned is that it is not easy to get used to the new APIs, as well as the inconsistencies within the data. The key take-away here is that a buffer really is needed since tasks tend to take longer than one expects rather than shorter.

Task	**Time Planned**	**Time Spent**	**Responsible Students**
Initial Meeting	10	10	All
Draft Proposal	20	20	All
Final Deliverables	30	35	All
Data Collect and Pre-Process	10	30	Andre, Leo, Iohan, Giacomo
Backend development	55	60	Demira, Iohan
Frontend development	25	50	Andre, Giacomo
AWS Setup	15	15	Demira, Leo
Testing and CI/CD	20	-	-
Meetings with group	-	15	All
Buffer	15	-	-
Total	**200**	**205**	-

Table 7.1: The time planned on tasks compared to the time actually spent and the responsible students

7.2 Outlook

While we have achieved the main goals according to the initial planning, the application still displays some rough edges whose improvements are left for future work.

In particular, the rating creation requires more thorough fine-tuning and testing, taking into account other sources of data, like Tripadvisor for instance. Furthermore, it is not clear if a simple averaging scheme is accurate enough, or if more advanced

statistics can improve the overall accuracy of the ratings. We additionally believe that using smoothing techniques instead of a 0,1 weight, could give the user a more fine-grained feedback of different barrios.

We believe that we collected a good amount of criterias to let the user prioritize. However, such a list is never complete and one can think of a large amount of other categories, e.g crime statistics or noise. Extending the amount of priorities gives more control to the user, making our application an even more valuable tool for the exploration of Barcelona's housing market and quality of life parameters.

On the side of the deployment, there is still work to be done in professionalizing the workflow, which will especially be critical as the application grows. As such, CI/CD pipelines and automatization, more careful role management and strict separation of microservices will rise to paramount importance for the maintenance of the app.

Finally, we want to point out that at some point, a stateful backend might be needed to integrate more real-time functionalities like the integration of predictions through Machine Learning services in terms of user preferences or the integration of collection of user statistics.

Conclusion

In this work, we implemented a web application that allows users to retrieve valuable data about different neighborhoods in Barcelona. Not only did we implement the collection of the data, we also combined the information in a compact and transparent way. More specifically, we created a rating for each barrio based on the data and the user's preferences so that the user can easily comprehend the way that the rating was created. This helps the user decide which barrio suits best for a possible move.

The application consists of a well designed front-end with a small but expressive set of features that lets the user interact with our backend. The backend retrieves data and combines the information from it to give feedback to the user within a short time span. The data is a combination of data collected in preprocessing applications and real-time retrieval. It is gathered from a collection of webpages and APIs, such as Open Data Barcelona, Google Maps or the official webpage of the Ajuntment Barcelona. The preprocessed data is sporadically updated to maintain contemporariness. All the services are deployed using a microservice architecture on AWS with the keypart, i.e the backend, being fully serverless.

While the application still leaves room for extension and improvement, we were able to create an application that helps the users decide in which barrio's in Barcelona to move, alleviating them from the tedious task of manually going through pages of unprocessed data on the internet.

Bibliography

[1] *Average monthly rent and average rent per surface of the city of Barcelona*. URL: https://opendata-ajuntament.barcelona.cat/data/en/dataset/est-mercat-immobiliari-lloguer-mitja-mensual.

[2] *Ajuntament de Barcelona official website*. URL: https://ajuntament.barcelona.cat/.

[3] *Axios*. URL: https://www.npmjs.com/package/axios.

[4] *Google Maps Library for React*. URL: https://www.npmjs.com/package/@react-google-maps/api.

[5] *AWS Secrets Manager*. URL: https://aws.amazon.com/secrets-manager/.

[6] *AWS Lambda*. URL: https://aws.amazon.com/lambda/.

[7] *AWS API Gatewar*. URL: https://aws.amazon.com/api-gateway/.

[8] *Amazon Elastic Compute Cloud*. URL: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html.

[9] *Amazon Simple Storage Service*. URL: https://aws.amazon.com/s3/.

[10] *AWS Organizations*. URL: https://aws.amazon.com/organizations/.

[11] *AWS IAM roles*. URL: https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html.