UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

TOYO TRAVEL COMPANION

propusă de

Sofron Radu

Sesiunea: *iunie / iulie, 2023*

Coordonator științific
Lect. Dr. **Simona Elena Vârlan**

UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

TOYO TRAVEL COMPANION

Sofron Radu

Sesiunea: *iunie / iulie, 2023*

Coordonator științific

Lect. Dr. Simona Elena Vârlan

Avizat,

Îndrumător Lucrare de Licență

Lect. Dr. Simona Elena Varlan

Data 22.06.2023 Semnătura ________________

DECLARAȚIE privind originalitatea conținutului lucrării de licență

Subsemntatul **SOFRON RADU**, cu domiciliul în **COMĂNEȘTI, JUDEȚUL BACĂU** născut la data de **22.06.2001**, identificat prin CNP **5010622046268**, absolvent al Universității „Alexandru Ioan Cuza" din Iași, Facultatea de **INFORMATICĂ** specializarea **INFORMATICĂ ÎN LIMBA ENGLEZĂ**, promoția **2020-2023**, declar pe propria răspundere, cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art.143 al. 4 si 5 referitoare la plagiat, că lucrarea de licență cu titlul: **TOYO TRAVEL COMPANION**, elaborată sub îndrumarea d-nei**. Lect. Dr. Simona Elenea Varlan**, pe care urmează să o susțină în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului său într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucruri științifice in vederea facilitării fasificării de către cumpărător a calității de autor al unei lucrări de licență, de diploma sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Data, Semnătură student

...........22.06.2023................. …………………………........

DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul „*TOYO TRAVEL COMPANION*”, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.
De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași, *iunie 2023*

Absolvent *Sofron Radu*

Contents

Chapter 1: Introduction

1.1. Technology today

In the last few decades, the world has been evolving at an unprecedented pace, faster than ever before. One of the main reasons why this has been possible was the constant and uninterrupted evolution of the technology. Thirty years ago, if someone wanted to know more about a particular subject, he would have to search for it in books. These days, everything someone wants to know is available at just a click away. The development of technology along with the emergence of the internet has truly improved people's lives. Whether it is the military, healthcare, education, or tourism domain, technology now plays a significant role in all of them.

In the tourism area, with the help of technology, people can now effortlessly search for accommodations, restaurants, flights, and other important aspects of a specific tourist destination. Not only that people can search for them, but they can make reservations, too, using web applications such as Booking, Trivago, or Kiwi. They no longer need physical maps (the traditional way in which people used to navigate the globe) as everything can be found very easily on their phones using services like Google Maps, Waze or, Moovit. Thanks to all these web applications, it is entirely safe to say that tourism has never been easier to practice.

Nowadays, web applications are an essential component of technology, their greatest advantage being that they are accessible to anyone, at any time and from anywhere around the globe. People can access them, regardless of the device they are using (a laptop, a smartphone, etc.). Due to these clear advantages, their number is huge. A web application usually consists of two components: frontend and backend. The frontend serves as the interface of the website, through which the users interact. The backend represents the core of the website, meaning that it ensures all the functionalities of the web application. However, web applications rely on other elements, too, without which they would not work as expected or be so powerful. Two of the most important ones are the Internet and the databases. The Internet provides the necessary infrastructure to make the communication possible between the users of the websites (also named clients) and the servers. On the other hand, databases are used to store data from web applications in an organized way. For instance, a website in the tourism domain can store user-related data (such as username, email, and password) or tourism-related data (such as accommodations, prices, and availability).

1.2. Motivation

As I have mentioned before, technology has truly become ubiquitous in every area, but this does not necessarily mean that there is no room for improvement. In the tourism domain, there are many useful web applications. However, as a tourist, I have encountered several difficulties that no website could help me thoroughly. Therefore, I have noticed a few limitations for the ones currently available, so my idea was to fill these gaps by creating a completely new web application.

The first issue I faced was that I had to access multiple websites to get the information needed about a specific tourist destination. For example, I had to access the Wikipedia website to know more things about the city, then web platforms like Tripadvisor or Lonelyplanet to discover some insights and travel itineraries regarding the city, and finally to access a website where I can check the weather forecast. Moving from one web application to another was annoying and time-consuming.

The second problem I encountered was that none of the websites I found was able to automatically generate a travel itinerary based on personalized options such as the number of days spent in that city and the tourist attractions that I want to see.

The last issue was the extensive selection of available options which sometimes was overwhelming: less important features were placed everywhere on the web page.

1.3. Proposed solution

Knowing the constraints of the current websites, my solution was to build a web application that contains all the necessary data about a tourist destination in one place only. Also, it can automatically create trip plans based on customized choices. The available tourist destinations on the website consist of every capital city of Europe. Therefore, people have the following options regarding a city, on the same page:

- to generate travel itineraries
- to read its description from Wikipedia
- to view relevant photos
- to examine some statistics (derived from users' evaluations)
- to see the reviews written by the others
- to find the latest news
- to check the weather conditions.

Additionally, people can add a bookmark to a city for easier access, or mark it as visited. Once a place is marked as visited, another option is unlocked: a page where the user can evaluate the tourist destination. The evaluation consists of two main components:

- to rate the city based on specific criteria
- to leave a review.

Because my website improves the quality of traveling, I decided to name it TOYO (an abbreviation of the syntax "Travel on your own").

1.4. Similar solutions

As I have previously stated, there are many web applications that improve the quality of a trip. Among all, Tripadvisor shares the most features similar to my website. People can see cities' descriptions, statistics, reviews, tourist attractions, and news and can easily interact with each other. Therefore, they can access customized travel itineraries created by other users. The interaction between them plays a significant role on this platform. Also, worth to say that this website has some extra options, like searching for hotels, restaurants, flights, things to do, and so on. Despite all these helpful features, Tripadvisor does not provide two important ones: the possibility to automatically create trip plans given a set of personalized options and the weather forecast.

Another similar website is Lonelyplanet. On this platform, the focus is on travel guides and accurate, detailed information, rather than on users' interactions and opinions. Thus, people cannot evaluate a city or leave a review. Moreover, weather forecasts and automatic creation of travel plans are missing, too.

In addition to these web applications, another honorable mention is Wikivoyage. This platform gives plenty of precise information about a specific city to its users.

Chapter 2: Technologies used

The second chapter covers an overview of all the technologies utilized in the development of my web application. The presentation will begin with the frontend, then move to the backend, and finally, we will talk about the database and the Application Programming Interfaces (abbreviated as APIs) I have used.

2.1. Frontend

For the frontend component, I chose a classic approach consisting of HTML (an abbreviation of Hypertext Markup Language), CSS (Cascading Style Sheets), and Vanilla Javascript (the term “Vanilla” refers to using plain Javascript, without any additional frameworks).

HTML is the most widely used method for creating websites. Its advantage is simplicity. It is utilized to build the structure of a web page but also provides other essential elements, such as meta tags. Generally, HTML is inseparable from CSS: if HTML creates a web page, then CSS styles it. Therefore, the use of CSS is fundamental. It is utilized to improve the layout and general design of a website. Also, it offers simple, efficient solutions to ensure the responsiveness of a web application. The versions of HTML and CSS that I used are the latest, explicitly HTML5 and CSS3.

HTML and CSS are not enough to create a more interactive and animated interface. This is where Javascript plays a significant role. It is the perfect tool to ensure a more dynamic interaction between the users and the website. Javascript is a high-level programming language used for web applications. Some of its key attributes are:

- runs in users’ browsers
- provides a lot of features
- is supported by a large number of browsers (almost all)
- can easily manipulate the HTML elements (also called Document Object Model or DOM)
- has many available frameworks.

Running in clients' browsers represents a big advantage because the users do not have to make a request to the server in order to complete various actions on the website, such as sorting or filtering. These can be achieved easily in Javascript by using its main feature - the possibility to interact with the DOM.

The decision to choose Vanilla Javascript instead of a Javascript framework was made based on two considerations:

1. over time, a framework may become outdated or lose popularity, especially on the frontend; this is less likely to happen with a programming language
2. the functionalities of my website could be achieved using either approach.

2.2. Backend

For the backend component, my choice was Python. Python is a high-level programming language that offers a large variety of libraries and frameworks for different fields, such as machine learning, artificial intelligence, or web development. Consequently, I have decided to use Flask as the web framework for my website.

Flask, one of the most popular frameworks in web development, is a micro web framework designed for building web applications and APIs. Because it follows the principles of a micro framework, Flask focuses on simplicity and extensibility. Therefore, it provides only the fundamental components needed for a website, with the ability to easily add and integrate extra ones. Another important characteristic of this framework is flexibility. Flask does not impose any restrictions regarding the architecture of the web application, allowing developers the freedom to make their own architectural decisions.

The framework is built upon several elements, among which the most important ones are Werkzeug and Jinja.

Werkzeug is a Python library that furnishes key functionalities for a variety of web-related tasks. In Flask, it is utilized for handling HTTP requests and responses, managing cookies, URL routing, and more.

On the other hand, Jinja is a template engine for Python, used to generate dynamic content in web applications and several others. In Flask, its role is to create interactive HTML content based on templates. A template is an HTML file that, in addition to the standard HTML elements, can include special elements such as blocks, variables, control statements, or iteration constructs. Also, a worthwhile feature is that templates in Flask support inheritance.

Another aspect worth mentioning is that Flask incorporates default folders for improved file organization. A predefined directory named *templates* should store the HTML templates, and another one called *static* should contain all the other static files (photos, styles, scripts, etc.).

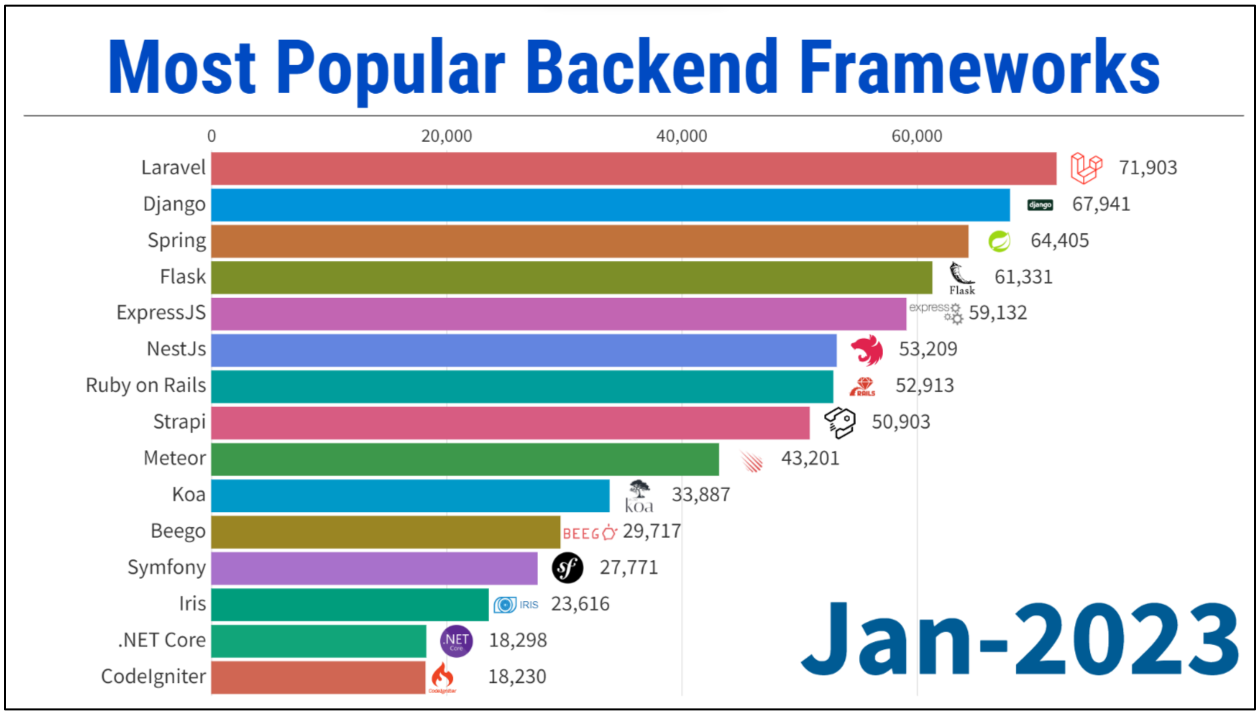


Figure 1. Most popular backend frameworks in January 2023 [1]

2.3. Database

When it comes to data storage, I have opted to utilize MySQL as the database of my web application. MySQL is a widely used relational database that is suitable for many types of applications. It utilizes SQL (an abbreviation of “Structured Query Language “) as its query language.

The main difference between a non-relational and a relational database is in how data is organized. In non-relational databases, data is usually stored in a very flexible manner, while in relational databases, the information is carefully arranged into tables with predefined structures, rules, and relations between them. Moreover, MySQL, like many other modern relational databases, presents a fundamental property known as ACID compliance, a feature that non-relational databases may not have. ACID stands for "Atomicity, Consistency, Isolation, Durability" and ensures a secure method for handling data, despite the possible presence of concurrent operations. Considering these distinctions, it is safe to say that relational databases, including MySQL, guarantee better data management and integrity. Another advantage of MySQL is that Python's integration with this database is straightforward and simple.

2.4. APIs

Application Programming Interfaces are another indispensable component of my web application. Without them, my website would not be able to provide several pieces of information, such as weather forecasts, tourist attractions in every city, or travel itineraries. According to the official definition, APIs are "*a source-code based software interface or intermediary that allows applications or software components to communicate with each other*" *[2]*.

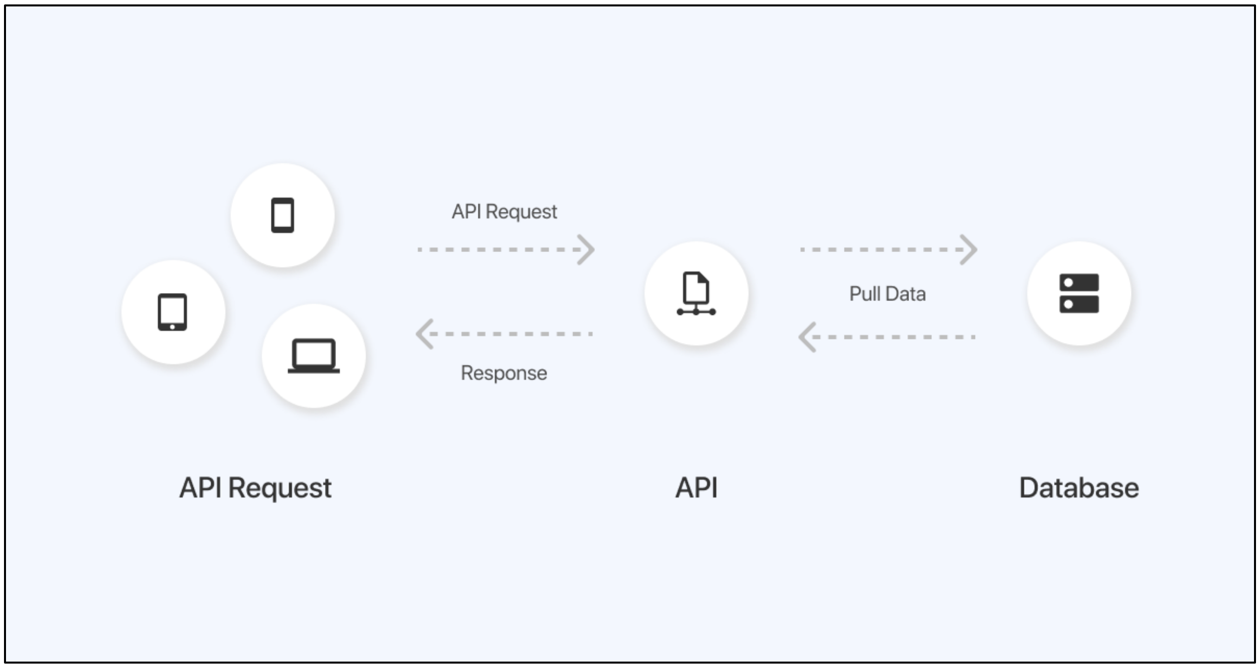


Figure 2. How API works [3]

My web application relies on multiple APIs. In the following subchapters, I will introduce and present each one of them.

2.4.1. Wikipedia

As the name suggests, Wikipedia API allows developers to access data from Wikipedia's vast database. This API, also known as Wikipedia API Sandbox, offers methods to extract different things, such as metadata, the entire webpage content, or just a summary of it. Furthermore, it provides advanced search capabilities - needed for retrieving specific information. Wikipedia API is owned by Wikimedia Foundation and its official documentation can be found at the following web address [4].

2.4.2. Unsplash

Unsplash is a web application where people can view and download high-quality images captured all over the world. The API furnished by Unsplash grants developers access to the entire collection of photos available on the website. Similar to the Wikipedia API Sandbox, it also offers sophisticated ways of searching, such as sorting images by date or relevance, and filtering based on orientation or keywords. As a result, obtaining stunning pictures of cities is seamless. For further details, check the referenced link [5].

2.4.3. OpenWeather

OpenWeather [6] stands as a web platform that contains a wide range of weather information APIs for a specific location, including Current Weather Data API (provides current weather data), Daily Forecast 16 days (offers a 16 days forecast), 5 Day / 3 Hour Forecast (furnishes a 5 days forecast, every 3 hours), and more. From all of them, the Application Programming Interface used for my website is 5 Day / 3 Hour Forecast. I have chosen this API because it is the only one that implies no money and, at the same time, gives more weather details. Also, advanced queries are supported. To access more in-depth information about it, visit the web address provided [7].

2.4.4. OpenCage

OpenCage [8] is a website that offers two types of APIs: for geocoding and reverse geocoding. Geocoding represents the process of converting place names or addresses into their corresponding geographic coordinates (longitude and latitude). On the other hand, just like its name indicates, reverse geocoding means the opposite being the action of transforming geographic coordinates into addresses or place names. For my web application, I have used Geocoding API to extract the longitude and latitude of the cities' central points. For more information about the API, access the designated link [9].

2.4.5. Geoapify

Geoapify [10] functions as a web platform providing a vast selection of APIs: for maps, addresses & locations, routes, places, and reachability. Among these categories, I have utilized one of the APIs for places.

Known by the name of Places API, this Application Programming Interface allows developers to retrieve an extensive and diverse range of data about specific areas, offering an advanced system for searching and filtering. Therefore, I have used it to extract 500 tourist attractions for each city, covering a radius of 7500 meters, as well as several details about them: their geographic coordinates, addresses, and opening hours.

To gain a deeper understanding of this API, visit the URL address provided [11].

2.4.6. GNews

GNews [12] is a website that furnishes news and events around the globe. Additionally, it offers an API that grants developers the ability to extract them, specifically for a searched keyword. Having this option, I have decided to use the API to retrieve the main news regarding a city. Its documentation can be found at the referenced web address [13].

Chapter 3: Architecture and implementation of the application

This chapter gives a thorough overview of my application. It comprises the presentation of the website, the web application and database architecture, and several essential implementation details.

3.1. Website functionalities

The first interaction between unauthenticated users and my web application occurs on the landing page. Here, a user can explore a short presentation of the website and navigate to the signup or login page.

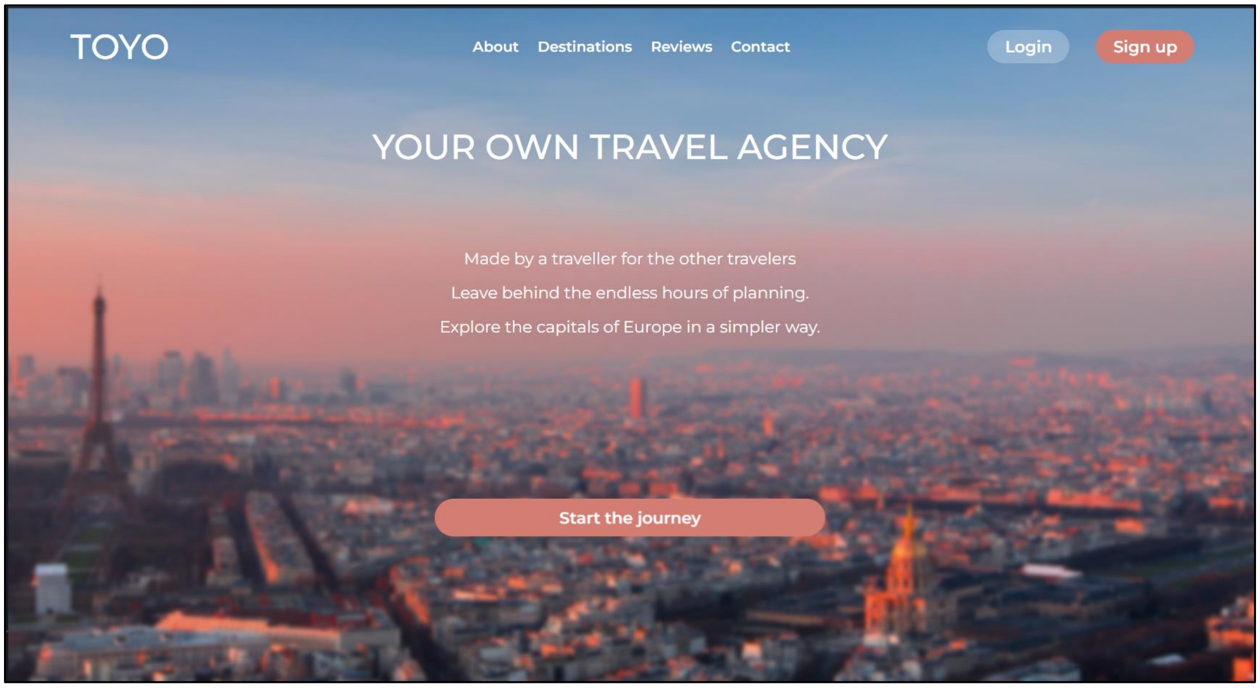
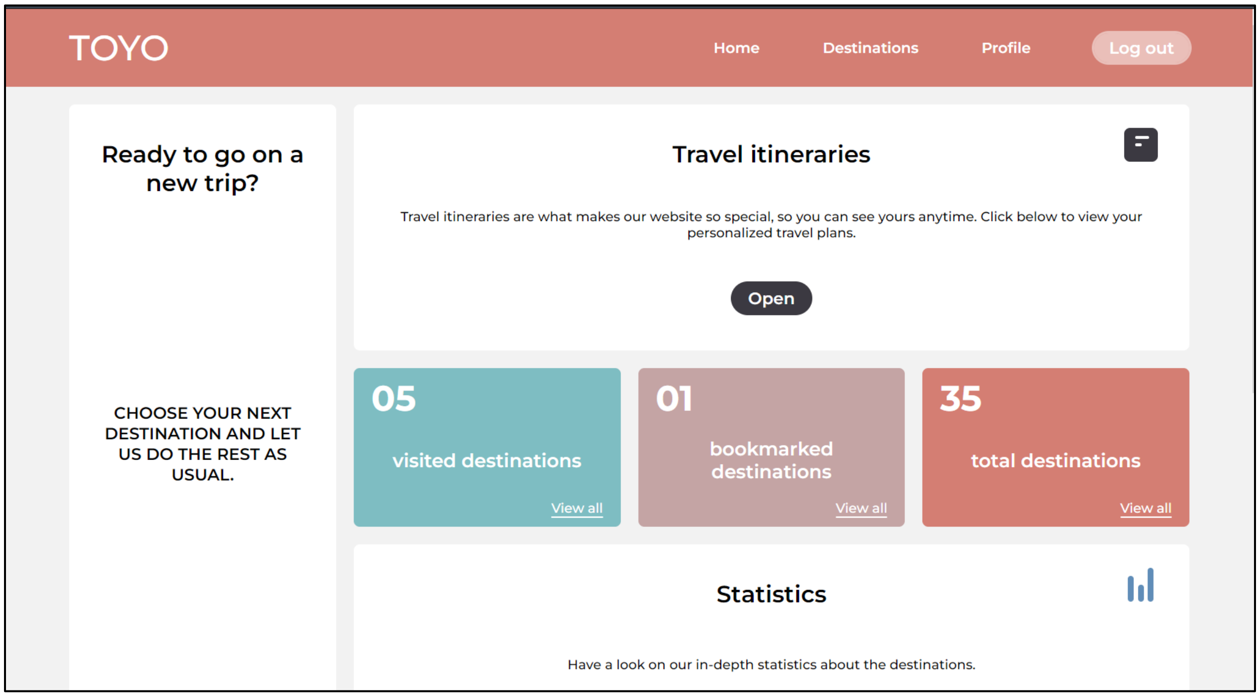

TOYO
About
Destinations
Reviews
Contact
Login
Sign up
YOUR OWN TRAVEL AGENCY
Made by a traveller for the other travelers
Leave behind the endless hours of planning.
Explore the capitals of Europe in a simpler way.
Start the journey

Figure 3. Landing page

After completing the registration or login process successfully, the user is redirected to the home page, which represents the core of the website. From here, he can navigate to different sections, including the profile, travel itineraries, and statistics pages. Also, he can access a page for the visited destinations, bookmarked destinations, or all destinations.



Ready to go on a new trip?

CHOOSE YOUR NEXT DESTINATION AND LET US DO THE REST AS USUAL.

Travel itineraries

Travel itineraries are what makes our website so special, so you can see yours anytime. Click below to view your personalized travel plans.

visited destinations

bookmarked destinations

total destinations

Statistics

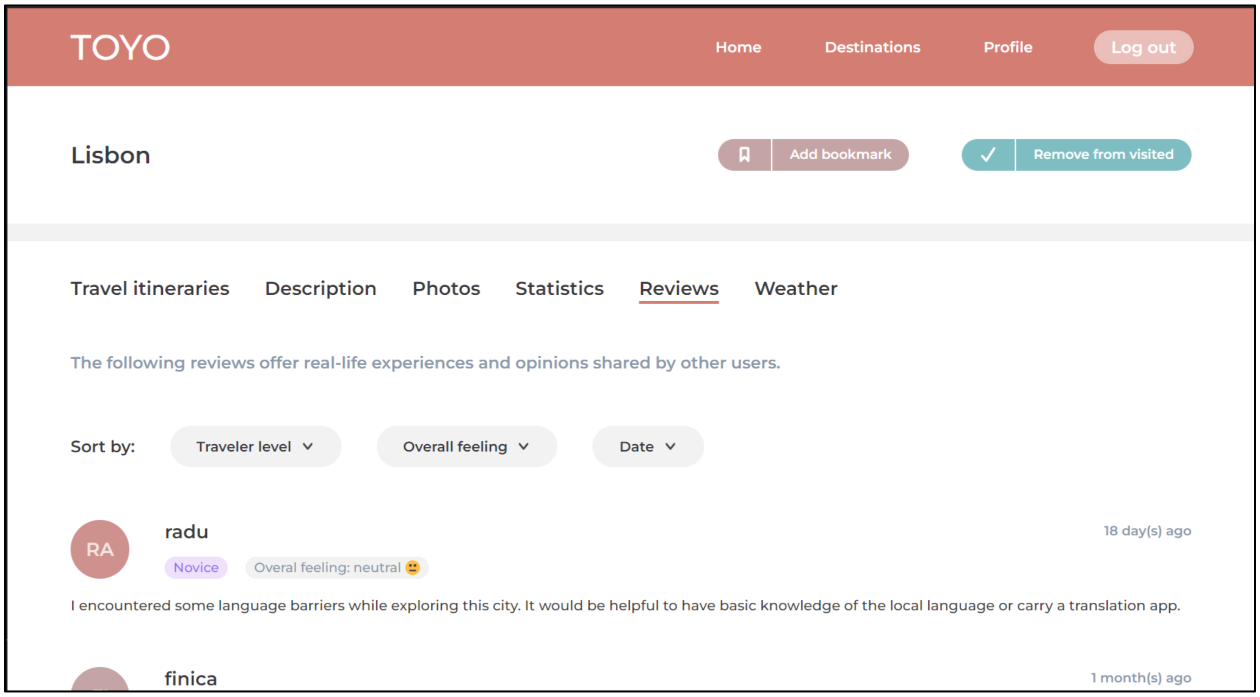
Have a look on our in-depth statistics about the destinations.

Figure 4. Home page

On the profile page, the user can view his profile details, upload a profile picture, look at some statistics about his activity on the platform, change his password, or delete his account.

On some of the other mentioned pages - statistics and travel itineraries - user interaction with the website is minimal. They can view more complex statistics about the cities, respectively their travel plans.

However, a greater level of interaction is available on the city page, which can be accessed from the destinations page, where the user can select a city that wants to explore. Once the city is chosen, its correspondent page is displayed. Here, the user can add a bookmark for the destination or mark it as visited. Also, he has access to a wide range of data about the city: descriptions, photos, statistics, reviews, weather forecasts, or news, and can create travel itineraries. An example of city page for Lisbon, containing all the available features, can be found in Figure 5.


TOYO

Home Destinations Profile Log out

Lisbon

Add bookmark ✓ Remove from visited

Travel itineraries Description Photos Statistics Reviews Weather

The following reviews offer real-life experiences and opinions shared by other users.

Sort by: Traveler level Overall feeling Date

RA radu 18 day(s) ago

Novice Overal feeling: neutral

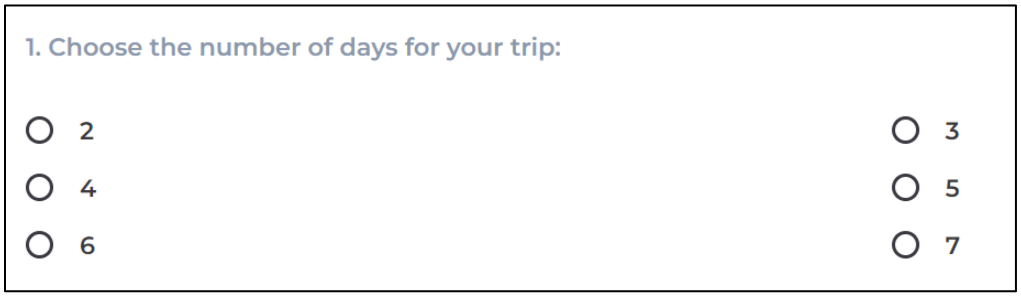
I encountered some language barriers while exploring this city. It would be helpful to have basic knowledge of the local language or carry a translation app.

finica 1 month(s) ago

Figure 5. City page for Lisbon

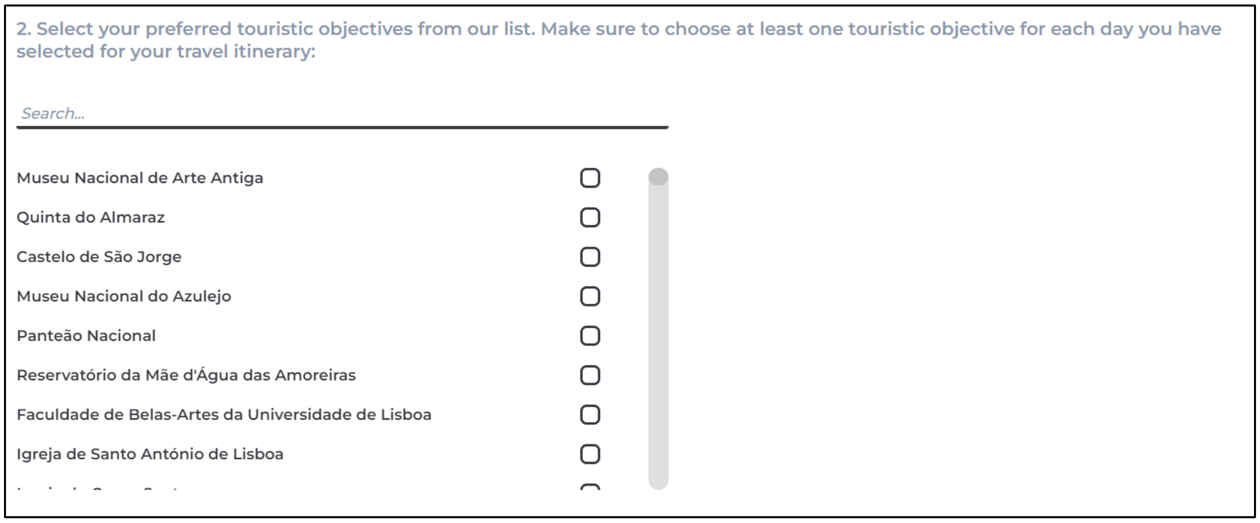
In order to create a trip plan, he has to perform a few steps:

- to choose the number of days spent (as can be seen in Figure 6)
- to select the desired tourist attractions from the list (Figure 7)
- to pick the algorithm that will be used (Figure 8)
- and optionally, to provide a name and a description for the itinerary (Figure 8)


1. Choose the number of days for your trip:

○ 2	○ 3
○ 4	○ 5
○ 6	○ 7

Figure 6. Number of days choice

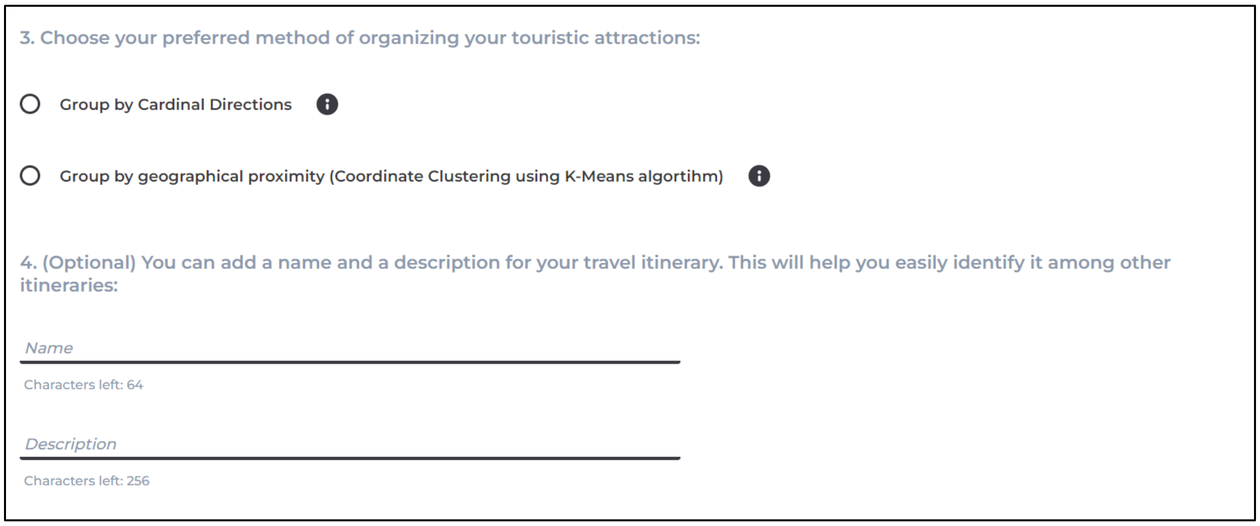


2. Select your preferred touristic objectives from our list. Make sure to choose at least one touristic objective for each day you have selected for your travel itinerary:

Search...

- Museu Nacional de Arte Antiga ☐
- Quinta do Almaraz ☐
- Castelo de São Jorge ☐
- Museu Nacional do Azulejo ☐
- Panteão Nacional ☐
- Reservatório da Mãe d'Água das Amoreiras ☐
- Faculdade de Belas-Artes da Universidade de Lisboa ☐
- Igreja de Santo António de Lisboa ☐

Figure 7. Tourist attractions selection



3. Choose your preferred method of organizing your touristic attractions:

○ Group by Cardinal Directions

○ Group by geographical proximity (Coordinate Clustering using K-Means algortihm)

4. (Optional) You can add a name and a description for your travel itinerary. This will help you easily identify it among other itineraries:

Name

Characters left: 64

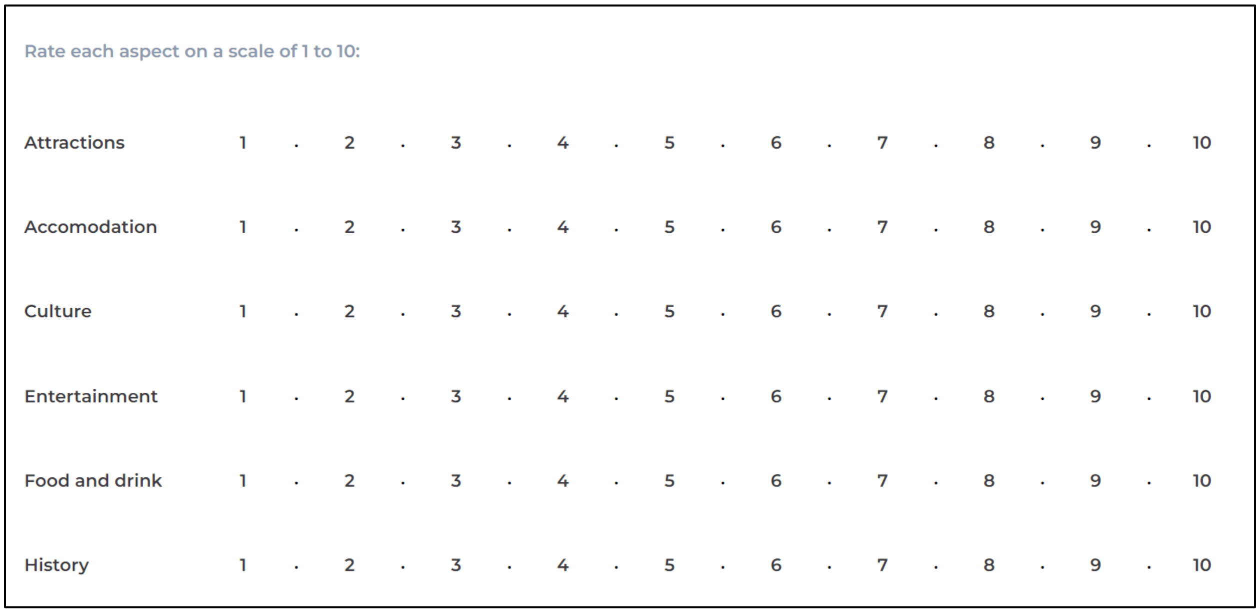
Description

Characters left: 256

Figure 8. Algorithm choice and details insertion

When a city has been marked as visited, a new feature becomes available, allowing users to evaluate the destination in two distinctive ways:

- to give grades for certain aspects (Figure 9)
- to leave a review (Figure 10)


Rate each aspect on a scale of 1 to 10:

Attractions	1	2	3	4	5	6	7	8	9	10
Accomodation	1	2	3	4	5	6	7	8	9	10
Culture	1	2	3	4	5	6	7	8	9	10
Entertainment	1	2	3	4	5	6	7	8	9	10
Food and drink	1	2	3	4	5	6	7	8	9	10
History	1	2	3	4	5	6	7	8	9	10

Figure 9. Evaluation based on grades

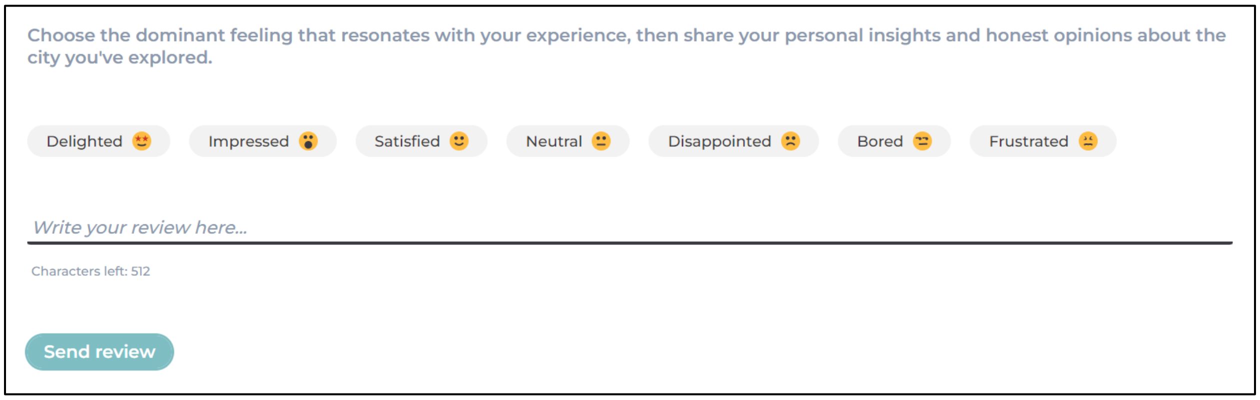

Choose the dominant feeling that resonates with your experience, then share your personal insights and honest opinions about the city you've explored.

Figure 10. Evaluation based on review

The last available page serves for a specific trip plan that has been already created. Here, the users can review the details of the travel itinerary.

For a deeper understanding, a diagram that illustrates all the possible interactions can be found below.

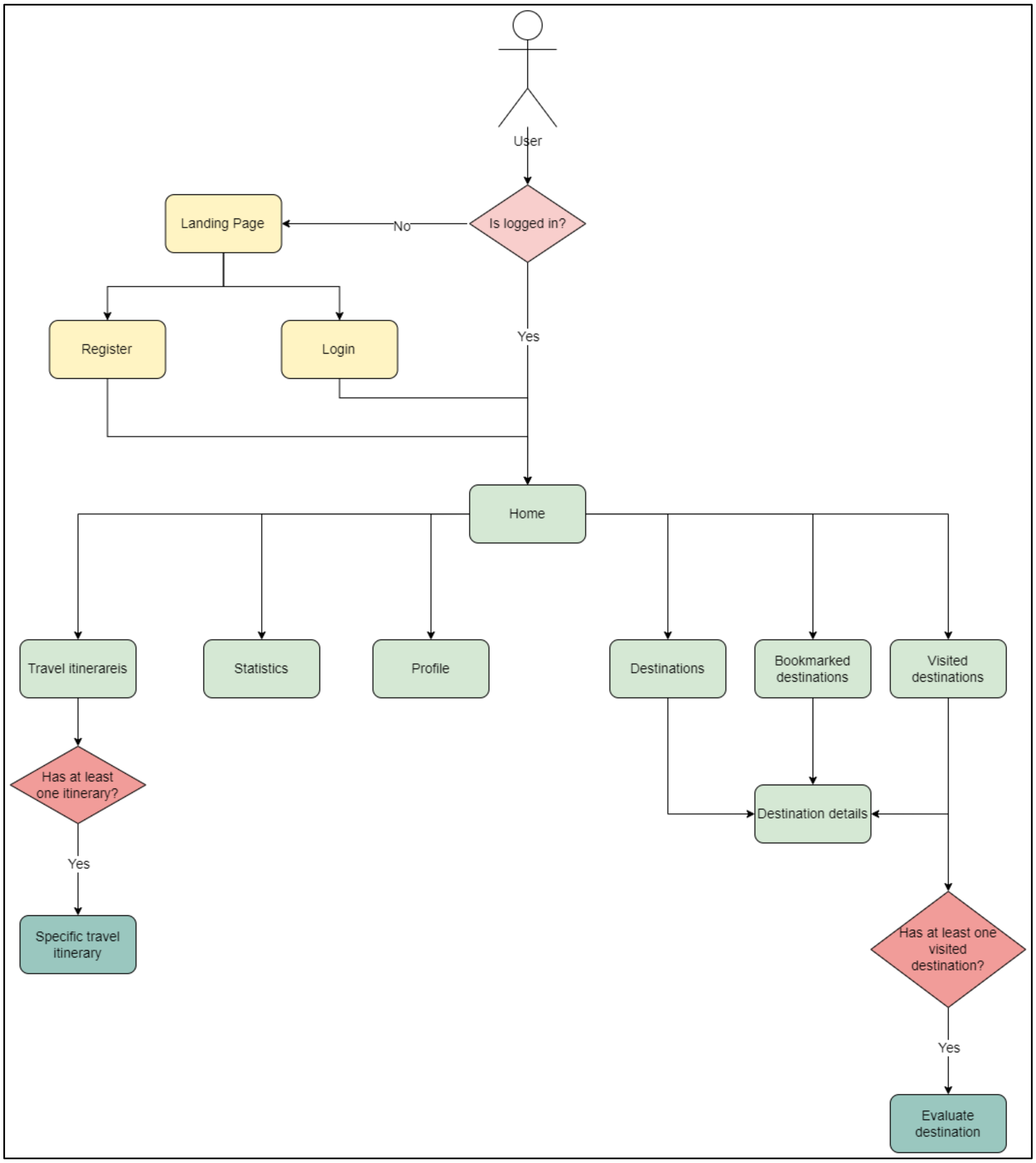


Figure 11. Website functionalities

3.2. Architecture of the web application

The architecture of my web application is depicted in the next diagram:

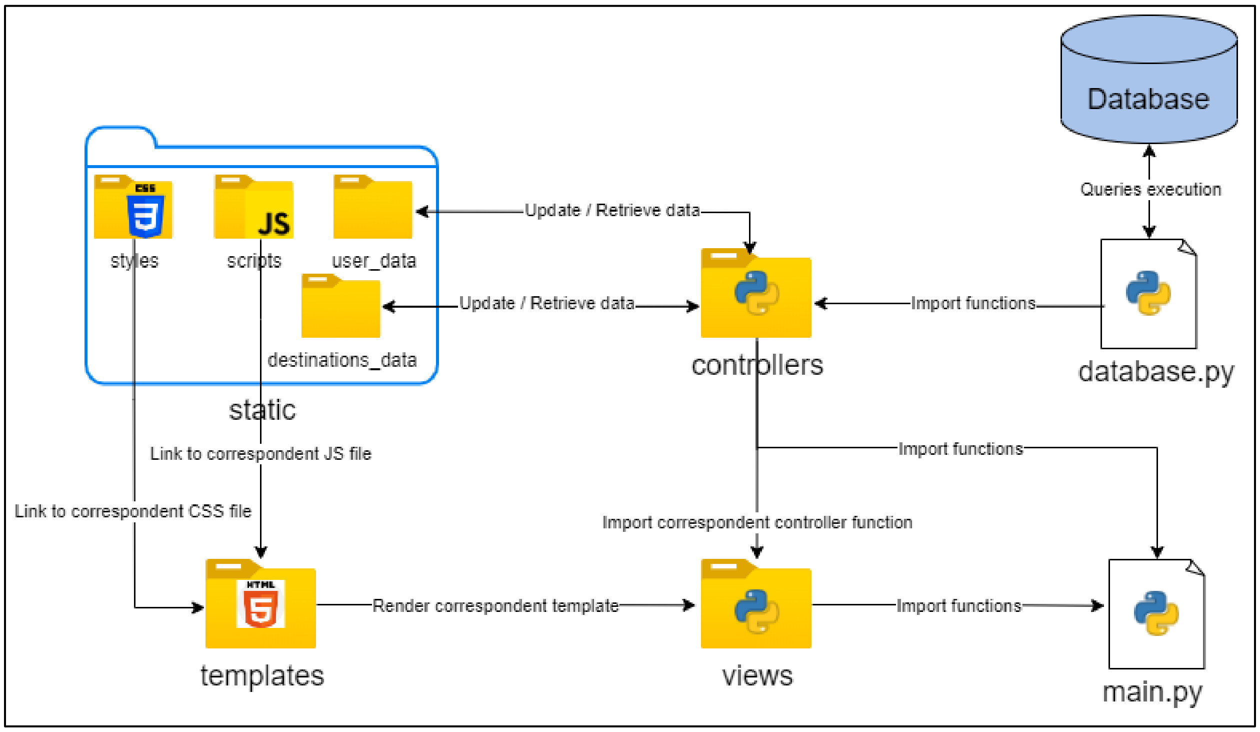


Figure 12. Diagram of the architecture of the application

In the following, I will explain the architecture considered in detail.

For my web application, I have chosen to implement a lightweight variation of the traditional MVC (Model-Controller-View). An MVC design pattern involves a separation of concerns where the model is utilized to manage the data, the controller is responsible for handling requests and user inputs, and it also ensures the communication between the model and the view, and the view has the obligation of rendering the user interface together with its specific data.

In my case, the controller performs the tasks of the model. As a result, the entire logic of my web application is found inside the controllers. Data transmission between a controller and a view is made using a JSON-type serialization. Every controller and view have a route defined in it. In Flask, a route can be specified by using the *@app.route* decorator. However, when working with multiple files, it is necessary to create a blueprint for each route. A blueprint can be easily done, as it is shown below:

```
# Create blueprint
home_controller_blueprint = Blueprint("home_controller_blueprint",__name__)

# Define route for blueprint
@home_controller_blueprint.route("/api/home")
```

In order to launch the app, I have built a separate file called *main.py*. This file creates and configures a Flask object, which represents the core of the website. Every blueprint made has to be imported and registered here, too. Once the object is set up, the application starts.

When dealing with relational databases, one major concern is SQL injection. SQL injection represents a security vulnerability that allows attackers to inject or manipulate malicious SQL statements. Consequently, intruders can gain unauthorized access, and insert, modify, or delete data. An efficient solution to prevent SQL injection is to use an external file, in my case *database.py*, that contains all the necessary SQL queries encapsulated in parametrized functions and creates a database object. Then, controllers import and utilize the database object and the required functions from this file. This method is well known in the industry and is called *database driver*.

As I have previously mentioned, Flask provides two default folders, called *static* and *templates*. The *static* directory is used to keep all the static files of the application, such as photos, icons, and so on. Additionally, it stores the CSS and Javascript files created for their respective HTML files, which are preserved within the *templates* folder. Because HTML files support inheritance, I have created an HTML template called *pre_login_base.html* that is inherited by two templates, *index.html*, and *post_login_base.html*. Also, *post_login_base.html* is inherited by all the other HTML files available, providing the website header and footer (the shared sections between the pages).

Shifting the attention back to the storage aspect, I have decided to save users' profile pictures, weather forecasts and news in the *static* directory. The reasons behind the decisions will be discussed later on, in the subchapter that focuses on implementation details.

3.3. Architecture of the database

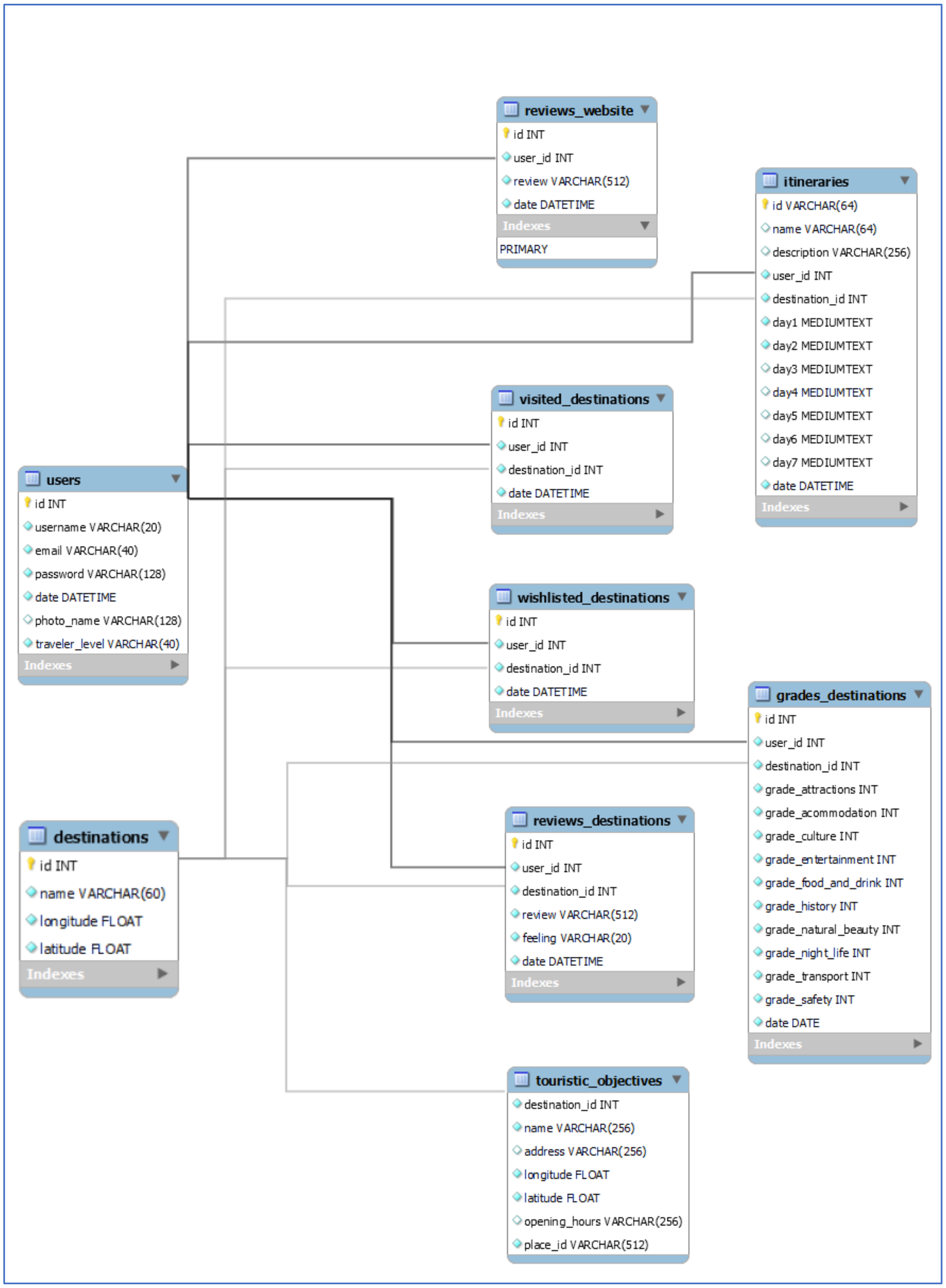


Figure 13. Diagram of the database

As stated before, I have chosen to utilize MySQL. The database contains nine tables, explained below:

1. *users* - consists of a user id and his account details; for the case in which the user has just registered, the traveler level inserted is the lowest;
2. *destinations* - covers all the available destinations on the website, as well as their geographic coordinates; additionally associates a unique id to each city
3. *visited_destinations* - incorporates all the visited destinations by any user
4. *wishlisted_destinations* - holds all the bookmarked destinations by any user
5. *reviews_destinations* - covers each review of the destinations
6. *grades_destinations* - includes every evaluation based on grades of the destinations
7. *touristic_objectives* - consists of 500 tourist attractions for every city and several details about each one of them
8. *itineraries* - stores all the travel itineraries created by any user
9. *reviews_website* – holds all the reviews about the website

Any table except for *touristic_objectives* contain a primary key. These primary keys are automatically generated. For the *itineraries* table, a unique id is created by using the *uuid* module from Python. For the others, a key is created by incrementing the maximum value of an existent id. Because a city offers 500 tourist attractions which are displayed simultaneously, and then the generated itinerary is stored with its tourist objectives in the *touristic_objectives* table, there is no need for a primary key.

The data included in *touristic objectives* and *destinations* tables was stored using some of the mentioned APIs (as will be seen in the following subchapter).

3.4. Implementation details

The following subchapters cover some of the most significant technical aspects concerning essential functionalities of my web application.

3.4.1. Preparatory components

In the *utils* directory, a directory that has nothing to do with the actual website, several Python scripts can be found. They have been used to retrieve various data for my application. All of the scripts have utilized the *requests* module from Python.

A fundamental step in the process of creating my web application was to extract cities' geographic coordinates and tourist attractions. I retrieved cities' coordinates using the API provided by Opencage. For the tourist objectives, I utilized the Places API offered by Geoapify. Both APIs have a limited number of requests per day in their free versions. Therefore, the only solution was to store the information extracted into the database. The URLs employed to retrieve the data can be seen below.

```
# Create URL
url=f"https://api.opencagedata.com/geocode/v1/json?q={city}&key={key}&limit
=1&pretty=1"
```

OpenCage API

```
# Create URL
url=f"https://api.geoapify.com/v2/places?categories={category}&filter=circl
e:{coordinates[0]},{coordinates[1]},{meters}&limit={limit}&lang=en&apiKey={
key}"
```

Places API (by Geoapify)

Places API extracts the tourist attractions in the language of the respective country. Consequently, it would be hard for the users to identify their desired tourist objectives from the list, so I have opted to extract the names from the database, translate them into English, then update the database accordingly. The translations have been performed using the *translate_text* function from the *translators* module in Python:

```
# Translate every name
for word in words
      new_words.append(ts.translate_text(word[0], translator="bing",
      to_language="en"))
```

The *translators* module utilizes translation services from many well-known providers, such as Google, Bing, and Yandex.

Also, I have decided to download the images available for each city. This choice was made because the initial dimension of a picture provided by Unsplash API was quite big (more than 5MB, even 15 MB). As a result, the upload of multiple photos at the same time was making the website quite slow and the waiting time too extensive. The waiting time was too big. Also, I did not want to display only a few then the users to have to click to view more because it would not be a great experience. Thus, the idea was to download the images and resize them. Resizing them (respecting the proportions) to the maximum number of pixels that a photo can occupy, significantly decreased their dimensions. The resized pictures were saved in the *static* folder. The URL utilized can be found below.

```
# Create URL
api_url=f"https://api.unsplash.com/search/photos?page=1&per_page=20&order_b
y=relevant&orientation=landscape&query={city}&client_id={key}"
```

Unsplash API

The images have been resized using the *Image* object from the *PIL* module. The following example illustrates the resizing of cover pictures:

```
for city in cities:
       # Open city photo
       image = Image.open("webapp/static/assets/cities_photos/covers" +
      city + ".jpg")

        # Set the new size
        new_size = (400, 267)
        # Resize
        resized_image = image.resize(new_size)

        # Save the resized photo
        resized_image.save("webapp/static/assets/cities_photos/covers" +
      city + ".jpg")
```

3.4.2. Registration and authentication

For the registration, a few checks are being made:

- the username and email cannot be already used
- the password must have a minimum of 8 characters
- the password to match the reintroduced password

If one of the conditions is not satisfied, the application redirects the user to the same page, displaying a proper message; *flash* and *redirect* functions are provided by Flask.

```
# Try to return input username from database
username_rows = extract_username(dba, username)
# Try to return input email from database
email_rows = extract_email(dba, email)

# Checks
if len(username_rows) == 1:
      flash("username_error")
      return redirect("/register", code=302)
if len(email_rows) == 1:
      flash("email_error")
      return redirect("/register", code=302)
if len(password) < 8:
      flash("password_error")
      return redirect("/register", code=302)
if password != password_confirmation:
      flash("passwords_error")
      return redirect("/register", code=302)
```

Otherwise, the user is inserted into the database and is redirected to the home page. Before the insertion, the current date is extracted and the password is hashed using the *generate_password_hash()* function from *werkzeug.security* module.

For login, validation of the password is done using the *check_password_hash()* function from *werkzeug.security* module, too. If the login is successfully done, the website redirects the user to the home page. Else, the user is redirected to the same page and a message is displayed.

Once a user is redirected to the home page, whether he came from the register or login page, the web application stores in the *session* his id and the fact that he is logged in. The *session* represents a special dictionary offered by Flask. Also, the framework permits developers to set the time until the session expires.

```
# Store in session user id and log in flag
session["logged_in"] = True
session["user_id"] = extract_user_id(dba, email)
```

Session storage

```
app.config["PERMANENT_SESSION_LIFETIME"] = timedelta(days=7)
```

Session available period

Both register and login utilize POST requests.

3.4.3. Profile options

As I have stated before, on the profile page, the user can upload/update his profile picture, change his password, or delete his account. POST requests are used, too.

To add a profile picture, he has to select one from his device. The upload button is by default disabled. File validation is done immediately using Javascript and consists of a check regarding its extension and size. If this step is passed successfully, the upload button becomes available. Once clicked, the image is saved in the *user_data* directory. Flask allows developers to easily configure a default folder for uploads, as can be seen in the following box:

```
app.config["UPLOAD_FOLDER"] = "webapp/static/assets/user_photos/"
```

To change his password, he must provide the current one. The match between the password inserted and the one from the database is checked using *check_pasword_hash()*, too. In case of a success, the password is updated.

3.4.4. City data

On this page, a lot of options are available. I will provide details for some of them, which have a complex approach.

The weather forecast is retrieved using the mentioned API from OpenWeather.

```
# Create URL
url=f"https://api.openweathermap.org/data/2.5/forecast?q={city}&appid={key}
&units=metric"
```

Unfortunately, this API has a limited number of requests per day in its free version, too. However, there are enough requests available to update the weather hourly. Due to the constant updates, I have decided to save the data in JSON files, having one for every city. The same problem occurred for the news. Therefore, I have chosen to create a folder inside the *static* directory named *destinations_data*. This folder contains other directories, one for every city, suggestively named. Inside these directories, two JSON files are found: one named *weather.json* and the other *news.json*. First rows of a JSON file for weather forecast:

```
{"cod": "200", "message": 0, "cnt": 40, "list": [{"dt": 1687273200, "main":
{"temp": 23.59, "feels_like": 23.55, "temp_min": 23.59, "temp_max": 23.69,
"pressure” ...
```

The amount of data received is large. From all this information, I have chosen to display the essential ones like the temperature and the sky status.

In order to update the weather hourly, when a user clicks on a city, the application compares the file modification date with the current date. If an hour has passed, it renews the weather forecast. In the case of news, since the number of possible requests is even more limited, they are updated once every 12 hours.

The following code is used for the weather forecast (the one for the news being very similar to this code):

```
last_modification_timestamp = os.path.getmtime(json_path)
# Convert from timestamp to datetime
last_modification_time=datetime.fromtimestamp(last_modification_timestamp)

# Calculate time elapsed since the last modification
current_date = datetime.now()
# Return time elapsed in seconds
time_elapsed = (current_date - last_modification_time).total_seconds()
# Convert to hours
hours_elapsed = time_elapsed / 3600

if hours_elapsed < 1:
      # No update ...
else:
      # Update ...
```

Another aspect that is worth highlighting is the sorting of reviews: after traveler level, after overall feeling, or after the date. The sorting is implemented in Javascript, for a better user experience, as it is faster.

3.4.5. Travel itineraries generation

To create a travel plan, the user must provide all the necessary data explained in the *Website functionalities* subchapter: the number of days spent, the tourist attractions that he wants to visit, and the algorithm to be utilized. By default, the button for trip itinerary generation is disabled. The website automatically enables it when all the requested information is furnished.

The creation of a travel itinerary is made using the *k-Means++* clustering algorithm, where the number of clusters *k* is equal to the number of days spent. I have used *k-Means++* class from the *cluster* module within the *sklearn* library and *pandas* module, the *matplotlib* library for the visual representation, and the *uuid* module to generate unique, random IDs for every travel itinerary. If the user has not provided a name for the travel plan, then its correspondent ID becomes the name, too.

In the *k-Means++* algorithm, I have used the geographic coordinates of the tourist attractions, converted into *pandas* data.

Chapter 4: Conclusion and future improvements

Reviewing all the functionalities offered by my web application, it is safe to say that the goal has been achieved. I have created a visually appealing website whose main focus is providing multiple details about a destination in one place only and generating travel itineraries. The web platform has a user-friendly interface and is fully responsive. Navigating the website on a phone or tablet will be perfectly fine. Also, the security component is not missing. It is safe enough for this type of application.

In the future, I would like to add more features such as:

- the possibility to create a travel journal for each destination
- a section where the users can view all the trip journals that are made public
- the chance to evaluate tourist attractions, too
- a map generation for every travel itinerary
- and most important - a stronger and more efficient algorithm for trip plans.

To create better travel itineraries, a viable solution would be to implement the *Equal-size spectral clustering* algorithm, its results being remarkable. As the author stated, this algorithm represents a modification of the spectral clustering algorithm *"that builds clusters balanced in the number of points" [19]*. The main difference between the *k-Means++* algorithm and *Spectral clustering* consists of the assumption made by *k-Means++*. As a result, the clusters tend to be spherical. On the other hand, the *Spectral clustering* algorithm can return different shapes for clusters, making the travel itineraries more natural. However, to generate accurate trip plans, it is mandatory to take into consideration that the clusters' cardinals have to be approximately equal.

In conclusion, the web application provides all the desired features by me, even if there is room for several more. True to its name, it serves as a reliable travel companion.

Bibliography

Miguel Grinberg, *Flask Web Development*, 2nd Edition (2018)

Daniel Gaspar, Jack Stouffer, *Mastering Flask Web Development* (2018)

Fatimah Rahmat, Nurul Shakirah Binti Mohd Zawai, Mohamad Iqbal hakim Che Omar, *Python and MySQL for Beginner*, 1st Edition (2023)

Taeho Jo, *Machine Learning Foundations* (2021)

Liviu Ciortuz, Alina Munteanu, Elena Bădărău*, Exerciţii de învăţare automata* (2022)

Ulrike von Luxburg, *A tutorial on Spectral Clustering*, in *Statistics and Computing*, vol. 17(4), 2007, pp. 1-32.

Carmen Adriana Martinez Barbosa, *Equal-size spectral clustering* in *Towards Data Science*, 2023

Webography

1. https://statisticsanddata.org/data/most-popular-backend-frameworks-2012-2023/
2. https://www.lexisnexis.co.uk/legal/glossary/api
3. https://acho.io/blogs/how-to-pull-data-from-an-api
4. https://en.m.wikipedia.org/wiki/Special:ApiSandbox
5. https://unsplash.com/developers
6. https://openweathermap.org
7. https://openweathermap.org/forecast5
8. https://opencagedata.com
9. https://opencagedata.com/api
10. https://www.geoapify.com
11. https://www.geoapify.com/places-api
12. https://gnews.io
13. https://gnews.io/docs/v4#introduction

14. https://flask.palletsprojects.com/en/2.3.x/
15. https://pypi.org/project/translators/
16. https://pypi.org/project/Pillow/
17. https://github.com/spantiru/companion-lab/blob/master/Lab11.ipynb
18. https://scikit-learn.org/stable/modules/clustering.html#clustering
19. https://towardsdatascience.com/equal-size-spectral-clustering-cce65c6f9ba3