# Guide Guru - Interactive Travel Guide Project Report

Tobias Kothbauer, Veronika Leitner

February 2, 2024

**Abstract**

In today's ever-changing world of travel, Guide Guru steps forward as a personal solution, ready to revolutionize how users embark on their journeys. Rooted in the fusion of cutting-edge technology and user-centric design philosophy, this project endeavors to craft a travel guide application that avoids limitations of traditional planning methods.

At its core, Guide Guru is driven by the vision of empowering travelers with a personalized experience, one that seamlessly aligns with their interests, preferences and hobbies. Leveraging the capabilities of advanced large language models, the application will function as an intuitive digital companion, which adapts to the desires of each user. By fostering a tight relationship between technology and user input, Guide Guru endeavors to simplify the travel planning process, offering a platform where every journey is curated to reflect the personality of the individual.

# Contents

# Chapter 1

# Aims and Context

The core objective of this project is to develop an intuitive and adaptable travel guide application that seamlessly integrates user input with language processing capabilities. Through the implementation of user surveys, Guide Guru should gather and evaluate individual interests, ensuring that every travel recommendation is finely tuned to match the user's specific desires and hobbies.

The envisioned application gives users a tool with a simple interface, allowing them to effortlessly select destinations and specify personal interests, thereby generating travel guides curated to their liking.

Moreover, Guide Guru will offer the practical functionality of exporting personalized guides in PDF format, enabling users to conveniently access their curated recommendations on various devices and platforms.

Upon completion, Guide Guru targets delivering a travel companion, that eases the planning process and allows a high levels of customization and efficiency. By placing the user at the center of the experience, Guide Guru aims to enhance the quality of travel adventures, empowering individuals to craft marvelous journeys that align, with their interests and preferences.

In order to accurately reflect user preferences in Guide Guru, a questionnaire was created. Thirty-three participants provided insights into their travel habits, hobbies, interests, and their perspectives on the ideal format for a travel guide. This data has allowed us to customize Guide Guru to cater to a diverse audience, requiring minimal user input while still meeting their needs effectively.

Chapter 2

# Project Details

Throughout our project, we carefully navigated through crucial steps to ensure success and produce great outcomes for our Guide Guru. From crafting detailed surveys to choosing the best technology tools, our path was defined by strategic choices aimed at crafting a tailored and efficient travel guide experience.

## 2.1 Questionaire

Initially, we developed a questionnaire based on our understanding of the essential information required to create a straightforward yet impactful prototype with highly personalized outcomes.

This survey included questions about user preferences, assessments of the importance of different travel factors, and open-ended questions to uncover insights into hobbies and preferred content for the travel guide.

Key findings from the survey revealed that the majority of participants preferred selecting their destination rather than receiving recommendations. Primary considerations for their travels included the type of travel (e.g., relaxation vacation), the overall environment (e.g., beach or city), and the preferred season for travel. Discovering new hobbies ranked as the least important aspect. All participants expressed a preference for receiving the travel guide in PDF format, with a strong emphasis on the importance of suitable images. Additionally, 85% favored selecting their hobbies from a predefined list, with sports and art & culture emerging as the most frequently mentioned categories. Regarding desired information in the travel guide, participants highlighted sights and hidden gems, culinary recommendations, entertainment options, cultural insights, and information on country-specific risks and hazards.

## 2.2 Technology Stack

Simultaneously, we conducted research on potential technology stacks. Two components were predetermined: utilizing React[1] for frontend development to enhance our web development skills with this technology, and selecting ChatGPT from OpenAI[2] as our

[1]https://react.dev

[2]https://openai.com

preferred large language model (LLM), thus our supervisor thankfully could provide us with an API key for this model.

Following several iterations, we opted to employ Python[3] for backend development after discovering its well-documented approach to integrating a LLM into a web application. As well Flask[4], a micro web framework, was chosen to handle requests and CORS policies.

Given the importance of images as indicated by survey participants, and considering the multitude of cities and places of interest generated by user input, we needed a means to access online data for countless locations. To achieve this, we utilized a combination of the Google Places API[5] and Google Photos API[6] to download and display images from user recommendations, providing users with a simple yet effective insight into selected destinations.

To validate user input for potential travel destinations, we integrated the geonames API[7], which is user-curated and lists millions of places, allowing filtering for highly populated areas. This API facilitated the inclusion of destinations with populations exceeding 1000, ensuring a diverse range of travel destinations could be used to generate the guide.

## 2.3 Prototype Design

Equipped with insights gleaned from our survey, we embarked on the development of an initial mockup for Guide Guru. Leveraging Figma[8], a collaborative design tool, we crafted a clickable prototype encompassing the entire user journey and a blueprint for our PDF guide. This mockup (see figure 2.1) enabled us to pinpoint necessary input parameters, map out pages, identify reusable components, and discover the inherent structure of our web application in a fast and efficient manner.

## 2.4 Implementation First Prototype

## 2.5 User Testing

## 2.6 Final Prototype

## 2.7 Challenges

## 2.8 Lessions Learned

[3]https://www.python.org
[4]https://flask.palletsprojects.com/en/3.0.x/
[5]https://developers.google.com/maps/documentation/places/web-service/overview?hl=de
[6]https://developers.google.com/photos?hl=de
[7]https://www.geonames.org
[8]https://www.figma.com/de

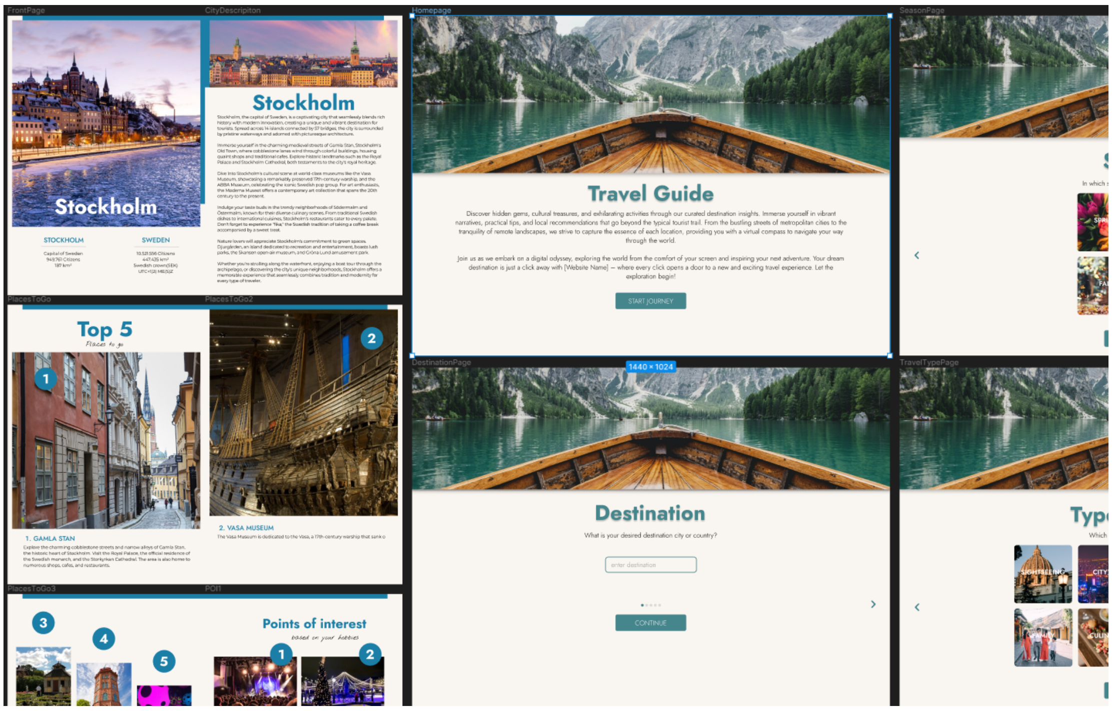


**Figure 2.1:** Mockup of Applicaton and Guide

# Chapter 3

# System Documentation

## 3.1 Backend

```python
import numpy as np

def incmatrix(genl1,genl2):
m = len(genl1)
n = len(genl2)
M = None #to become the incidence matrix
VT = np.zeros((n*m,1), int)  #dummy variable
```

### 3.1.1 Large Language Model Integration

### 3.1.2 Integration of Google APIs

### 3.1.3 Geonames API

We utilized the geonames API to validate user input while keeping the range of possible destinations open. The geonames API contains data on approximately 4.8 million populated places, including details such as name, alternative names, country, population, geolocation, and categorization. To refine the user options, we set a threshold of including only places with over 1,000 inhabitants, ensuring that the travel guide's output remains relevant and accurate.

For the integration of this API into our Guide Guru, we used Express.js[1] to create the API Interface. After incorporating cors and axios middleware, we developed two interfaces for communicating with the geonames API.

To enhance user experience, we designed a mechanism to display only well-known and populous destinations upon the initial opening of the dropdown field. Upon rendering the destination page, a request is triggered, prompting an axios-request to the geonames API which can be seen in the below code. This request retrieves 25 cities with over 15,000 residents in its response, ensuring that users are presented with familiar and prominent locations.

```javascript
app.get('/api/bigCity', (req, res) => {
  const apiUrl = 'http://api.geonames.org/searchJSON?cities=cities15000&maxRows=25&
    username=fhtravelguidews23';
```

[1]https://expressjs.com/de/

```
  axios.get(apiUrl)
  .then(response => {
    res.json(response.data)
  })
  .catch(error => {
    console.error('Error fetching data:', error);
  });
});
```

As soon as the user begins to enter into the input field an request is sended for ten places that match the input an have more than 1,000 citicens to allow a high variety of possible travel destinations.

Once the user starts typing in the input field, a request is sent to retrieve ten places that match the input criteria and have more than 1,000 inhabitants, thus providing a diverse range of potential travel destinations.

The processing of the API response data is detailed in the description of our Destination Page at 3.2.2.

## 3.2 Frontend

As mentioned previously, we opted for React as our frontend technology. To maintain a component-based architecture, we divided our frontend into pages that leverage multiple components we designed and implemented. Additionally, we established a state management system to store both user input and data provided by the backend. This mixture of technologies enables us to create a straightforward yet efficient user journey.

### 3.2.1 State Management

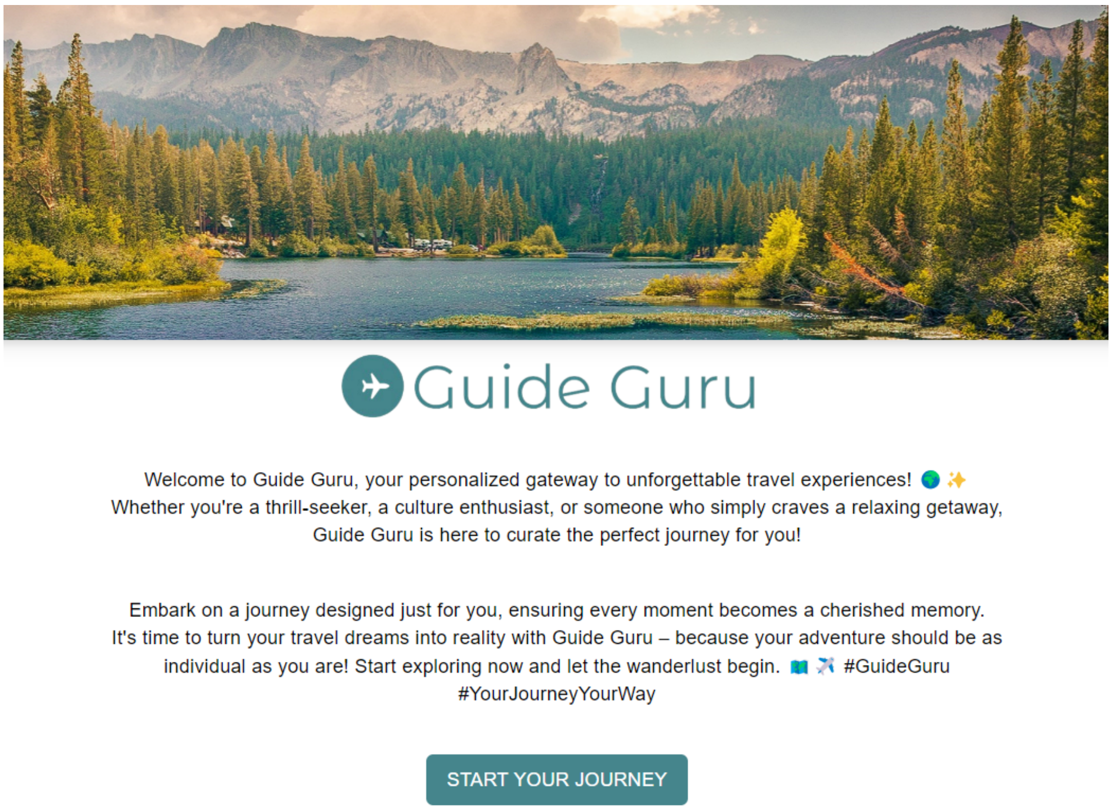
For state management, we utilized the Redux library to store both user input and data retrieved from the backend. The `inputSlice.js` file is responsible for storing the input of the user regarding the desired destination, seasons, type of travel, and selected hobbies they want to include in their journey. Additionally, we implemented a function within this slice to reset the user's input for new requests without requiring a page reload.

Within the `guideSlice.js` file, all information necessary to construct the guide is saved. This includes details such as destination, population, area, longitude, latitude, country information, images with appropriate dimensions, and recommended places.

In the file `store.ts`, these two reducers are configured and made accessible within the store variable. For instance, `store.getState().userInput.destination` provides access to the user's selected destination. To add data into the store, you can use functions like `dispatch(setDestionation("destination"))`.

### 3.2.2 Pages

In our mockup design, we implemented separate pages for each stage of the user journey. Each page features the header component, providing a consistent element throughout the experience for user familiarity. To ensure the validity of every guide request, we opted against traditional navigation methods and instead enabled users to navigate solely using buttons located at the bottom of each page.



Welcome to Guide Guru, your personalized gateway to unforgettable travel experiences! 🌍 ✨
Whether you're a thrill-seeker, a culture enthusiast, or someone who simply craves a relaxing getaway,
Guide Guru is here to curate the perfect journey for you!

Embark on a journey designed just for you, ensuring every moment becomes a cherished memory.
It's time to turn your travel dreams into reality with Guide Guru – because your adventure should be as
individual as you are! Start exploring now and let the wanderlust begin. 🗺️ ✈️ #GuideGuru
#YourJourneyYourWay

**Figure 3.1:** Homepage of Guide Guru

### Homepage

The homepage serves as the initial step in the user journey. To facilitate a seamless start to their personalized experience, we included our logo and a brief introduction explaining how to use the application. Additionally, to encourage continued engagement with the website, we incorporated a call to action featuring a button labeled *Start your journey*, directing users to the first input page.

### Destination

In the destination page, users are prompted to choose their preferred travel destination from options provided by our geonames API interface. Upon the initial rendering of the page, a request is made for 25 cities with more than 15,000 inhabitants. The responses from this interface populate the options, where the city name serves as the value and part of the label, along with the country name to prevent confusion between places with similar names. Additionally, information such as city population, country, latitude, and longitude is included in the options to be stored in the state when the user selects a destination. We utilized the `react-select` library to build this selection component, and the required HTML structure is as follows:

```
<form onSubmit={formSubmit}>
  <div className="my-4 flex justify-center">
    <Select
      isSearchable={true}
      options={cityOptions}
      onInputChange={loadOptions}
      required={true}
      onChange={handleChange}
      placeholder="start typing or select a city ..."
    />
  </div>
  <Button text='Continue' type='submit'/>
</form>
```

To send requests to the interface and build the adapted options as described above when the user starts typing into the input field, we utilize the following TypeScript function. This function is called from the `onInputChange` event of the `<Select>` component:

```
const [cityData, setCityData] = useState([]);
const [cityOptions, setCityOptions] = useState([]);
const dispatch = useDispatch();

const loadOptions = (inputValue: string) => {
  axios.get('http://127.0.0.1:4000/api/city?city=' + inputValue, {}).then((response)
     => {
    setCityData(response.data.geonames);
  }).catch(error => console.log(error));
};

useEffect(() => {
  createOptions();
}, [cityData]);

function createOptions() {
  const newOptions = []
  cityData.forEach((city) => {
    newOptions.push({
      ...city.name,
      value: city.name,
      label: city.name + ' (' + city.countryName + ')',
      population : city.population,
      country: city.countryName,
      lat: city.lat,
      lng: city.lng,
    })
  })
  setCityOptions(newOptions);
}

const handleChange = (selectedOption) => {
  dispatch(setDestination(selectedOption.value));
  let cityCiticens = selectedOption.population.toString().replace(/\B(?=(\d{3})+(?!\
    d))/g, ",");
  dispatch(setCitizens(cityCiticens));
  dispatch(setCountry(selectedOption.country));
  dispatch(setLatitude(selectedOption.lat));
  dispatch(setLongitude(selectedOption.lng));
};
```

Within the `loadOptions` function, a request is made to the interface, and the hook for `cityData` is adapted accordingly. This change triggers the `useEffect`, which constructs the new options for the selection, refreshing the `cityOptions` hook.

Once the user selects a destination, the `handleChange` function is accessed. This function adds the necessary and formatted information into the stores. Therefore, the required data is already saved, and the `onSubmit` function simply redirects to the next page.

### Seasons

On the season page, users are required to select at least one of the four seasons. The selection methodology is described within the `CardBlock` component at 3.2.3. To ensure that the user has selected at least one season, the form submission process checks the season store for elements. If the season store is empty, the message indicating the selection requirements switches from the regular color to bold red as can be seen in the HTML and TypeScript code below.

```
<form onSubmit={handleSubmit(onSubmit)}>
  <Heading headingLevel='h3' text="In which season would you like to travel?"/>
  <Paragraph text="Select one or multiple seasons for your journey."
    className={`${textColor} ${isBold ? 'font-bold' : ''}`}/>
  <CardBlock category="seasons" name="seasons"/>
</form>
```

```
const navigate = useNavigate();
const [textColor, setTextColor] = useState('text-black');
const [isBold, setIsBold] = useState(false);

function formSubmit(data: UserInput) {
  if (store.getState().userInput.seasons.length === 0) {
    setTextColor('text-error');
    setIsBold(true);
  } else {
    navigate('/type');
  }
}

const onSubmit: SubmitHandler<UserInput> = data => formSubmit(data);
```

Once the requirements are met, the user is redirected to the travel type page.

### Travel Type

The functionality of the travel type page mirrors that of the seasons page. Users must select at least one element in the `CardBlock` component to ensure a correct output from the large language model. Additionally, this requirement aims to motivate users to enter a category, which will be used to tailor the guide to their individual needs.

### Hobbies

The hobby page shares similarities with the previous pages, but includes three different `CardBlock` components, one of which contains the `HobbySelection` component described at 3.2.3.

To ensure tailored and appropriate travel guide outputs, we've increased the minimum selected options to three hobbies, either inside the `CardBlock` or the `HobbySelection` component. This adjustment aims to prevent undefined or incorrect outputs for points of interest when selected hobbies are not possible at the chosen destination (e.g., skiing in Hawaii). By requiring at least three hobbies to be selected, we anticipate that the guide will provide some relevant points of interest.

The functionality of this page remains largely the same, but includes a small computation for the `CardBlock` and the additional selected hobbies in the `HobbySelection` component, as demonstrated below.

```
const [textColor, setTextColor] = useState('text-black');
const [isBold, setIsBold] = useState(false);

function formSubmit(data: UserInput) {
  const totalHobbies = store.getState().userInput.hobbies.length +
    store.getState().userInput.addHobbies.length;
  if (totalHobbies < 3) {
    setTextColor('text-error');
    setIsBold(true);
  } else {
    navigate('/result');
  }
}
```

### Result

At the result page, users are presented with their personal input. We've provided users with the option to validate their input before generating the guide. This is important because the guide generation process can take one to two minutes of computation time, and users should have the possibility to establish it when they are pleased with their input. On this page, users can navigate to each of the previous input pages and change their selection or start a new travel guide with new parameters.

When the user clicks on the generate button, a request with the stored input data is sent to the backend to be computed by the large language model. The resulting guide is then stored in the `guideStore` to be accessed in the generated PDF document (see 3.2.4), to which the user is subsequently redirected.

In the code below the request to the backend to generate the guide and the handling of the output is shown:

```
const fetchChatResponse = () => {
  setIsLoading(true); // shows the loading animation, interactive map and message

  // load the stored inputs
  const destination = store.getState().userInput.destination;
  const seasons = store.getState().userInput.seasons;
  const travelType = store.getState().userInput.travelType;
  const hobbies = store.getState().userInput.hobbies;
```

```
    const addHobbies = store.getState().userInput.addHobbies;
    const country = store.getState().guide.country.name;

    // request to the LLM
    axios.post('http://127.0.0.1:5000/', {
      destination: destination,
      seasons: seasons,
      travelType: travelType,
      hobbies: hobbies,
      addHobbies: addHobbies,
      country: country
    }).then((response) => {
      // save data into guideStore
      dispatch(setDescription(response?.data.destination_text));
      dispatch(setDestination(destination));
      dispatch(setArea(shortenAnswerForNumbers(response?.data.city_area)));
      dispatch(setCountryCitizens(shortenAnswerForNumbers(response?.
        data.country_citicens)));
      dispatch(setCountryArea(shortenAnswerForNumbers(response?.
        data.country_area)));
      dispatch(setCurrency(response?.data.currency));
      let places = response?.data.places_text;
      let placesArr = places.split(/: |\n\n/);
      dispatch(setPlacesToGo(placesArr))
      places = response?.data.poi_text;
      placesArr = places.split(/: |\n\n/);
      dispatch(setPOI(placesArr));
      setIsLoading(false);
      navigate('/exportGuide')
    }).catch(error => console.log(error));
  };
```

During the guide generation process, the user is notified that it may take some time. Additionally, an interactive map component (see 3.2.3) is displayed, showing the selected destination marked with data retrieved from the geonames API (described at 3.1.3). This allows the user to explore their chosen travel destination while waiting for the guide to be generated.

### Export Guide

This page is only needed to include the PDF Viewer and the React PDF that is described in the subsection 3.2.4.

### 3.2.3 Components

#### Button

#### CardBlock

#### CardInput

#### Header

#### Heading

#### HobbySelection

#### InteractiveMap

#### Paragraph

### 3.2.4 React PDF

# Chapter 4

# Summary

Give a concise (and honest) summary of what has been accomplished and what not. Point out issues that may warrant further investigation.

# Appendix A

# Supplementary Materials

The appendix is a good place to attach a user guide, screenshots, installation instructions, etc. Add a separate chapter for each major item.

## References