An-Najah National University
Faculty of Engineering
Department of Computer Engineering

Graduation Project 1

# Masarna

Students:
Dana Aqel
Lina Qurom

Supervisors:
Dr. Mona Demaidi
Dr. Ashraf Armoush

Presented in partial fulfillment of the requirements for bachelor's degree in Computer Engineering.
January 2024

## Acknowledgment

We're to preface by thanking Allah, all praise to Him, for the strength, patience, knowledge, and all that was necessary to work on and complete this project, and more. Secondly, we give thanks to our supervisors for their time and advice throughout the entire semester. Lastly, a big thank you goes out to our friends and family for their unwavering support and simply being there for us, always.

## Disclaimer

This report was written by Dana Aqel and Lina Qurom at the Computer Engineering Department, Faculty of Engineering, An-Najah National University. It has not been altered or corrected, other than editorial corrections, because of assessment and it may contain language as well as content errors. The views expressed in it together with any outcomes and recommendations are solely those of Dana Aqel and Lina Qurom. An-Najah National University accepts no responsibility or liability for the consequences of this report being used for a purpose other than the purpose for which it was commissioned.

## Table of Contents

## Abstract

A means to make travel planning easier, more efficient and much more fun, especially in group settings, is proposed by this project. The room for conflict between peers when deciding on a travel itinerary and the time spent planning for a trip are aimed to be minimized as much as possible, offering in turn a fun and smooth travel planning experience because that's what travel planning is supposed to be; fun and easy. The prior mentioned is all achieved via multiple features:

1. The user can create a new trip plan, which they can add friends to as members, add personal and group events to, view personal and group events' schedule in, upload media to and choose to download a generated reel, view event locations and route on the map, make use of external data suggestions for help during travel planning, plan collaboratively along plan members in the polls and comments section for every itinerary, and view trip events' budget.
2. The user can make use of the full notification system to stay up to date and have all information they need to know.
3. The user can connect with others via the full chatting system set up.
4. The user can benefit from the translator provided for any necessary translation needed due to situations of language barrier while traveling.
5. The user can interact with others by reporting them or maintaining friendship with them (requesting, declining, and canceling friendship)
6. The user can customize their profile page however they see fit.
7. The user can submit ratings and suggestions for the application, ensuring user feedback is always taken into account.
8. The user can utilize the system's smart chatbot for help with using the application or requesting tailored itinerary suggestions to the weather forecast.
9. The user can view the weather forecast up to 16 days forward, for multiple cities.

Additionally, the system is monitored and continuously kept up to date by the following features:

1. The admin can view reports on users, and reports and favorites on itinerary suggestions. The admin can also take necessary steps, in reference to the number of reports, such as deleting users from the system or suggestions that've gotten negative feedback.

2. The admin can view all updates to do with the application via the full notification system.
3. The admin can view the ratings and suggestions submitted by users, taking them into account, always.
4. The admin can interact with users for help and inquiries via the chatting system.

Moreover, the system is easily accessed and utilized via web and android, by both the user and admin.

Many of the features mentioned above can be found scattered across multiple applications, but our aim was to combine them along with others into one application, with much more attention to detail along with a seamless user interface, ensuring a smooth and fun travel planning and traveling experience, with all trip details and memories in one place.

To develop the system, we used multiple tools and frameworks, including Flutter, to create the cross-platform mobile application and website experience. More details will be discussed in the methodology section of this report.

# Chapter 1: Introduction

## 1.1 General background

Travel planning can be very tedious, especially due to the extensive amount of details to do with planning for a trip, and when in a group setting, the complexity and hardship is indefinitely amplified. Aside from that, the lack of a platform out there that combines all travel details in one place, doesn't make things any easier.

## 1.2 Objectives

The primary objectives of our travel planning system are centered around delivering a seamless and collaborative experience for users. Our foremost goal is to ensure that travel planning, especially in group settings, becomes a smooth and gratifying process where the opinions of all participants are taken into account. Furthermore, a key focus is placed on providing users with an easy and intuitive interface, guaranteeing a friendly and straightforward user experience. Additionally, the system aims to consolidate all trip details in one place to streamline organization and accessibility for users. By meeting these objectives, our travel planning platform seeks to revolutionize the way individuals and groups plan their journeys, emphasizing both convenience and inclusivity.

## 1.3 Significance

‘Masarna’ stands out as a significant innovation in the realm of trip organization. By introducing collaborative group trip planning, it addresses the common challenges of conflicting preferences and ensures inclusivity by incorporating every member's input. The platform's multifaceted features, including a weather forecast widget, route visualization on maps, budget tracking, and a smart chatbot offering tailored itinerary suggestions to the weather forecast, collectively elevate the planning experience. This not only simplifies the process but also infuses an element of fun into travel coordination. The consolidated nature of the platform ensures that all details are easily accessible in one place, mitigating the risk of overlooking crucial trip elements. In essence, the application reshapes travel planning by making it not only more efficient and enjoyable but also fostering seamless coordination among group members

## 1.4 Organization

This report is organized into chapters, starting with the introduction. This section is further divided into subsections covering the project's general background, objectives, significance and the report's structure. Then, the report continues with a chapter on constraints, standards and earlier coursework. The third chapter discusses literature review. The fourth chapter outlines the methodology employed in the project's development. Then, the fifth chapter goes into Masarbot in detail. After that, the report presents the results and analysis deducted from the project's implementation, along with the discussion of the entire development process. A conclusion and future work recommendations chapter follows after that and finally, the report ends with references.

# Chapter 2: Constraints, Standards, and Earlier Coursework

## 2.1 Constraints and limitations

1. Time: it was a challenge to be able to accomplish what we did and bring our idea and concept of our application to life during the limited time frame we had, causing us to work on the project under a lot of pressure almost all of the time.

2. Unfamiliarity: this was our first time working on a mobile application. It was also our first time working with Flutter, Firebase, Express.js, and Python, in an AI setting. Due to the prior mentioned, it took us a lot of time to learn and get good at the languages and frameworks; it wasn't a seamless or intuitive experience working with them at first.

3. Quality of training set: the chatbot would properly predict the intent of the user's input about 99% of the time. The 1% margin of error is due to the training data set not being exactly perfect, as that is very hard to attain, especially with the limited scope of possible conversation between the chatbot and user.

4. Short-term weather forecast: the weather external API we used offers the forecast for only up to 16 days forward, which limits the utilization of the chatbot's tailored activity suggestions to the weather, in the case of the trip not being during the 16-day frame. The chatbot wouldn't be able to provide weather suitable suggestions without valid/available weather forecast data.

## 2.2 Standards

### 2.2.1 MVC (Model View Controller)

We used the MVC model in our system, splitting it into three parts:

1. Model: represented by our main database, MongoDBAtlas, which stores all of the system's data, supplying the Express and Flask servers. Our secondary database is a Firebase one to handle the chat system's data.
2. View: represented by our GUI, in both web and android/mobile, used by users (travelers) to efficiently plan for their trips and the admin to effectively monitor and manage the system.
3. Controller: mainly represented by our Express server, supporting the communication between the Model and the View, and our Flask server, supporting the chatbot.

### 2.2.2 Agile Model

We used the Agile Model during our development process, segmenting it into the following stages:

1. Requirements: we held multiple discussions and brainstorming sessions together to decide on all of the features we wanted our system to offer.
2. Planning: we decided on what frameworks and languages we were going to use. Then, we assigned tasks and agreed on a rough agenda to follow.
3. Development: we worked on our assigned tasks, meeting almost daily to discuss progress and challenges faced, coming up with an appropriate resolution. After a task was completed, it was shared and discussed with our supervisors.
4. Testing: for backend related tasks, after finishing their development they were tested via Postman to ensure perfect functionality. For finished frontend tasks, they were tested via an android emulator and a physical android device as well. After that, integration between the frontend and the backend was finalized and tested via the android emulator and physical device, looking out for responses from the backend in the server's debug console and expected application behavior as well, ensuring a perfect and seamless implementation of every task.

## 2.3 Earlier coursework

Many courses that we've previously taken played a pivotal role in enabling us to work on this project, including Software Engineering, Advanced Software Engineering, Artificial Intelligence, Database, Web Programming, Data Structures and Algorithms, and Critical Thinking and Research Skills. Additionally, we independently learned Flutter [1] (Dart), Node.js, Firebase [2] and Python (Flask) [3] at the beginning of the semester, furthering our knowledge in software development.

# Chapter 3: Literature Review

The use and development of mobile applications for travel planning have gained a lot of popularity in recent years in efforts to make travel planning easier, more fun and more accessible. In comparison to other local applications, to our knowledge, ours distinguishes itself by integrating a wide range of features not commonly found collectively elsewhere. While competing apps may scatter similar functionalities, our platform integrates them seamlessly, providing users with a singular, comprehensive solution for travel planning. This distinctive approach underscores our commitment to enhancing the overall efficiency and sophistication of trip organization.

The prior mentioned competing apps include:

1. Wanderlog: includes map view of itineraries and trip budget.
2. TripIt: includes uploading images and simple collaborative planning.

# Chapter 4: Methodology

## 4.1 Backend development

For the backend's development, we broke it down to several parts:

1. Models: to represent the database structure. In order to design and communicate with our models, we used the Mongoose library, which offers an Object Data Modeling (ODM) layer for MongoDBAtlas. We were able to construct schemas with Mongoose, which governed the structure, validation guidelines, and behavior of our data.
2. Routes: to represent our APIs and their routes, their manipulation of data and communication with both the database and frontend (response), written in Node.js.
3. API: we opted for RESTful APIs to expose backend functionality, using different endpoints and HTTP requests such as (GET, POST, DELETE and PUT).
4. Middleware: we needed multiple middlewares for different reasons, including Puppeteer and Axios for external data, Multer for image file handling, Express and Bodyparser.
5. Uploads: a folder in the server where image files are added, as their path in the server is how they're stored in the database.

## 4.2 Frontend development

For the frontend's development, we used Flutter framework, in both web and the mobile application.

1. Design: we took inspiration from other travel planning and travel-related applications on Pinterest. We opted for vibrant and exciting colors to match with the user experience feel.
2. Framework: after careful consideration, we opted for Flutter for various reasons such as, it's a cross platform framework, making it easy to work in web and mobile development as the code is easily shared and is very similar, it's also open source with many documentations, resources and tutorials, making it tremendously easy for us to implement. It's also high-performance and satisfies all our needs.
3. Programming languages: we used Dart, powered by Google, to go hand-in-hand with our Flutter framework.

### 4.3 Database setup and configuration

We used MongoDBAtlas due to:

1. Flexibility: due to the constantly changing schema during development in our agile context, it was a game changer using MongoDB because it doesn't require a predetermined schema. Data handling was also flexible due to the noSQL document based architecture of the database.
2. Sharing: since the database is online, the documents are easily shared between us and any edits to the data are visible to the both of us. It was easy and efficient this way.
3. Scalability: the database is able to handle large amounts of data and be in communication with multiple servers (Express and Flask in our case) at once, seamlessly.
4. Performance: it's incredibly high-performance and easily accessed and dealt with via our two servers. Due to its driver easily enabling integration with Node.js and Python, it definitely made the development of the project much easier. It's also absolutely reliable and very user-friendly. A terrific choice for a database.

### 4.4 External data

We used Puppeteer to web scrape external data suggestions for various itineraries such as flights, shopping, nightlife and sightseeing activities, eateries and stays. We used Axios to pull the weather forecast data from the weather external API (weatherbit).

### 4.5 Calendar

We used the syncfusion Flutter calendar, for its vast range of capability, flexibility and simplicity of use, to enable users to view their events in monthly and daily view, to which they can add, edit and delete events, as well.

### 4.6 Map

We used Google's Flutter maps, for its precision and easy use, to display the user's events' locations and the connected route between them.

### 4.7 Translator

We used Flutter's Translator, as it's crucial on trips abroad, to avoid issues of language barrier, ensuring an easy and practical trip for users. It's very easy and intuitive to use, just what the user needs.

### 4.8 Rating

We used Flutter's rating-bar to enable users to submit ratings of the system, along with feedback and suggestions, ensuring a constant improvement of the system where user opinion is always taken into account.

### 4.9 Chatbot

We used Naive Bayes to classify activities as suitable for warm or cold weather via text analysis of each activity's name and description. We also used a feed-forward neural network to analyze user input and respond accordingly, including weather tailored suggestions, when the user asks for them. The chatbot was implemented to make the user experience customized, beneficial, easier and more fun.

## 4.10 Memories

We used Flutter's Chewie to have low-level access to video playback, as users are able to upload images and videos in the Memories section and choose to generate a reel from them, which they can download, as memories are important and precious to keep, especially trip memories.

# Chapter 5: Masarbot, Machine Learning and AI chatbot model

## 5.1 Language and tools

We used Python's Flask framework to create a server for communication between our Flutter frontend and our Python chatbot logic, via network requests. This enabled an easy and straightforward utilization of the chatbot in our frontend.

**Libraries and Frameworks:**

1. Natural Language Toolkit (nltk): Used for natural language processing and tokenization.
2. TensorFlow and Keras: TensorFlow is used as the deep learning framework. Keras is used as a high-level neural networks API. Components from TensorFlow and Keras are used for building, training, and loading machine learning models.
3. Scikit-learn: Used for various machine learning tasks, such as data preprocessing, model selection, and evaluation.
4. Pymongo: Python driver for MongoDB, used for interacting with our MongoDB database.
5. Spacy: Used for natural language processing and language model loading.
6. Flask: A web framework for building APIs, used for creating the backend of the chatbot service.

**Other Tools and Utilities:**

1. Regular Expressions (re): Used for pattern matching in handling date-related queries.
2. Datetime: Used for date formatting and handling.
3. Random module: Used for generating random suggestions from the list of activities.

## 5.2 Dataset

The dataset for our model includes two groups of data, the first being the activity suggestions, including the activity's name and description, used to train the Multinomial Naive Bayes classifier, and the second being the intents set, where specific patterns are

categorized under certain tags and have numerous appropriate response options, used to train the neural network.

### 5.3 Data processing

For the Multinomial Naive Bayes classifier, the activity is firstly pre processed before being used to train the model. That is done by using TF-IDF vectorization to convert activity names and descriptions into numerical vectors; the vocabulary is created based on all patterns in the intents, and each activity's description is transformed into a TF-IDF representation. The model is then trained to classify activities into ‘warm weather’ and ‘cold weather’ activities.

As for the neural network, user input is tokenized, lowercased, alphanumeric filtered, and stemmed. Then, the output tokens are used to create a bag of words to feed the neural network. The neural network is trained on the intents set, which is preprocessed exactly the same way as the user input is. The architecture of the neural network is that it has input nodes equal to the number of vocabulary (unique words extracted for the patterns in the intents set, after preprocessing) and output nodes equal to the number of unique intents. In this way, the neural network is able to process user input, match it to a corresponding intent and generate an appropriate response. If the user asks for weather appropriate activity suggestions, the neural network will respond with suggestions classified and determined by the Multinomial Naive Bayes model.

### 5.4 Development

The development process went in the following order:

1. Activity classification: The code, using pymongo, accessed our MongoDB database to retrieve the activity data, including activity description and name. Then, the data was preprocessed and labeled via warm and cold keywords. After that, the Multinomial Naive Bayes classifier was trained with the labeled activities (TF-IDF vectors) to predict whether an activity is warm or cold based on its TF-IDF representation.
2. Weather matching: After the activities were properly classified, with an accuracy rate of 0.9666666666666667, they were matched with their corresponding weather based on a threshold of 20 C. If the weather on a certain date was above 20 C, warm activities would be suggested, if not, then cold ones.

3. Handling and responding to user input: After weather tailored suggestions were properly set up, we set up the neural network. We used a feed-forward neural network with dense layers. The input layer accommodates a bag-of-words representation of the user input, and the input size is determined by the length of the vocabulary created from all the patterns in the training data. As for the output layer, it has units equal to the number of unique intents in the training data, it also uses the softmax activation function to produce probabilities for each intent. The number of hidden layers and the number of neurons in each hidden layer can be adjusted based on the complexity of the problem. The activation function used in the hidden layers is ReLU (Rectified Linear Unit), a commonly used activation function in neural networks for introducing non-linearity. For the loss function and optimizer, the model is compiled using the categorical cross-entropy loss function, which is suitable for multi-class classification problems, and the Adam optimizer is used for training the model.
4. Putting everything together: after the neural network was properly set up, trained, and processed and responded to user input correctly, it was time to include the weather tailored suggestions. After successfully predicting the user's intent, if it was asking for tailored suggestions, it would call for the weather tailored suggestions functions. If it were any other intent, it would randomly pick a response from the responses array of that intent.
5. Integration: in order to integrate with our frontend, we set up a flask server for network requests, specifically POST, enabling the bot to receive and respond to API requests sent from the frontend. The user's input via the GUI is sent as a POST request to the server, the neural network handles it accordingly, and responds with a response sent back to the frontend via another POST request.

# Chapter 6: Results, Analysis, and Discussion

The main goal for this project was to develop a system offering all features needed by a user when trip planning and an admin when managing it. The platform is easy, intuitive and beneficial to use. It offers a wide range of features that take care of all details to do with travel planning, not offered by any other competing application. The system is also easily utilized and accessed via web and mobile. With the development of such an extensive and expansive system, came many challenges including:

1. Lagging: due to the size of the project, it was oftentimes very hard and tedious to run and test, especially via the android emulator, as it required perfect Wi-Fi connection and enough disk space to run smoothly, which wasn't always attainable.
2. Variety: due to the variety of middlewares, languages and frameworks that we used, lots of incompatibility and indirect integration issues arised. Errors and exceptions seemed to be all over the place, everywhere, at some points.
3. Chatbot: the NB classifier had a small margin of error in categorizing activity data and the neural network also had a small margin of error in recognizing intents. With all of that combined, the bot happened to sometimes incorrectly respond to user input. The small margin of error would be fixed with better training data, but realistically, absolutely perfect data and performance isn't attainable.

# Chapter 7: Conclusion and Future Work

In summary, this project effectively offers a seamless travel planning experience, especially in a group setting. Its primary aim was to make travel planning easier for users, by taking care of all details in one place, such as group planning and viewing events in the calendar, map, budget, etc., more fun for users, by offering a generated reel for uploaded media, and more beneficial for users, by offering a chatbot for help and recommendations, a weather forecast widget, and a translator.

Future work could focus on expanding the range of possible conversation with the chatbot, offering a wider range of itinerary suggestions, and IOS support of the system.

While this conclusion doesn't introduce new information not previously mentioned, it underscores the project's significance, accomplishments and potential avenues for future development. The journey from research and development to testing and error resolution has led to an efficient platform that holds promise in transforming the landscape of travel planning.

# References

[1] Wael Abo Hamza (2023). *Learn flutter from zero to hero.* Retrieved September 20, 2023 from https://youtu.be/6bSP4vazmyw?si=tLDpuRs9UJhv5OFo

[2] Wael Abo Hamza (2023). *Intro firebase (2023).* Retrieved December 27, 2023 from https://youtu.be/gZSqLbxXjjs?si=df_c_ImrVdmoo13K

[3] Patrick Loeber (2020). *Chatbot with PyTorch.* Retrieved Jan 20, 2024 from https://youtu.be/RpWeNzfSUHw?si=p6simDEBx2USxarA.