Cairo University

Faculty of Computers and Artificial Intelligence

Department of Computer Sciences

PathFinder

Supervised by

Dr. Mohammed ElRamly

TA. Mohammed Atta

Implemented by

|  |  |
| --- | --- |
| 20200193 | Ziad Ibrahim Galal |
| 20200162 | Hamza Abdul-Hameed Ali |
| 20200348 | Omar Tarek |
| 20200636 | Yahia Ashraf |
| 20201222 | Yahia Mahmoud |

Graduation Project

Academic Year 2023-2024

Midyear Short Documentation

Table of Contents

[1 Abstract 3](#_Toc158562106)

[2 Background 4](#_Toc158562107)

[2.1 Project scope 4](#_Toc158562108)

[2.2 Motivation and Beneficiaries 4](#_Toc158562109)

[2.3 Main Techniques and Technologies 4](#_Toc158562110)

[3 Problem Definition 5](#_Toc158562111)

[4 Market Study 5](#_Toc158562112)

[5 Data Collection 6](#_Toc158562113)

[5.1 Web Scraping 6](#_Toc158562114)

[5.2 Crowd Sourcing 6](#_Toc158562115)

[5.2.1 PathFinder Tool 7](#_Toc158562116)

[6 Project Specifications 7](#_Toc158562117)

[6.1 Functional Requirements 7](#_Toc158562118)

[6.2 Non-Functional Requirements 8](#_Toc158562119)

[6.2.1 Performance Requirements 8](#_Toc158562120)

[6.2.2 Software Quality Attributes 9](#_Toc158562121)

[6.3 Stakeholders 9](#_Toc158562122)

[6.4 Use-Case Diagrams 10](#_Toc158562123)

[6.4.1 Student use case diagram 10](#_Toc158562124)

[6.4.2 University use case diagram 11](#_Toc158562125)

[6.4.3 Admin use case diagram 11](#_Toc158562126)

[6.5 System Design 12](#_Toc158562127)

[6.5.1 Component Diagram 12](#_Toc158562128)

[6.5.2 Sequence Diagrams 13](#_Toc158562129)

[6.5.3 ERD 20](#_Toc158562130)

[7 Machine Learning Models 21](#_Toc158562131)

[7.1 Chatbot Model 21](#_Toc158562132)

[7.1.1 NLU Training Data 21](#_Toc158562133)

[7.1.2 Rasa NLU Pipeline 21](#_Toc158562134)

[7.1.3 Rasa Core Policies 23](#_Toc158562135)

[7.2 Recommendation System 24](#_Toc158562136)

[8 Work Plan 26](#_Toc158562137)

[8.1 Task Table 26](#_Toc158562138)

[8.2 Gantt Chart 27](#_Toc158562139)

[9 References 27](#_Toc158562140)

List of Figures

[Figure 1 – Crowd Sourcing Tool. 7](#_Toc158534387)

[Figure 2 - Student use case diagram 10](#_Toc158534388)

[Figure 3 - University use case diagram 11](#_Toc158534389)

[Figure 4 - Admin use case diagram 11](#_Toc158534390)

[Figure 5 - Component Diagram 12](#_Toc158534391)

[Figure 6 - Student Use Chatbot Sequence Diagram 13](#_Toc158534392)

[Figure 7 - Student Sign-Up Sequence Diagram 14](#_Toc158534393)

[Figure 8 - Student Log-In Sequence Diagram 14](#_Toc158534394)

[Figure 9 - Student Search College Sequence Diagram 15](#_Toc158534395)

[Figure 10 - Student Compare Colleges Sequence Diagram 15](#_Toc158534396)

[Figure 11 - Admin Review Feedback Sequence Diagram 16](#_Toc158534397)

[Figure 12 - Admin Add University Sequence Diagram 16](#_Toc158534398)

[Figure 13 - University Sign-Up Sequence Diagram 17](#_Toc158534399)

[Figure 14 - University Announcement Sequence Diagram 17](#_Toc158534400)

[Figure 15 - University Add College Admin Sequence Diagram 18](#_Toc158534401)

[Figure 16 - Main ERD 19](#_Toc158534402)

[Figure 17 - PathFinder Tool ERD 19](#_Toc158534403)

[Figure 18 – Rasa pipeline and policies. 20](#_Toc158534404)

[Figure 19 – Recommendation system types. 23](#_Toc158534405)

[Figure 20 - Gantt Char 26](#_Toc158534406)

List of Tables

[Table 1 - Task Table 26](#_Toc158539056)

# Abstract

When students complete high school, especially Egyptian students, many often find themselves uncertain about which university or college they should enroll in to pursue a professional education. This lack of direction and data can lead to confusion and anxiety among students who are eager to embark on a fulfilling career path.

After high school, we are dedicated to assisting students in finding the perfect college that aligns seamlessly with their interests and aspirations. Our approach involves meticulous research and evaluation of numerous colleges and universities, ensuring we provide comprehensive and detailed information to empower students to make well-informed decisions about their higher education.

Our support extends to offering personalized recommendations and suggesting colleges that we believe are an excellent fit based on the student's preferences, academic ambitions, and future goals. We are committed to guiding students toward a path that meets their educational needs and inspires their success.

There are many tools needed to achieve our goals, such as Django REST framework, Rasa Open Source, MongoDB, React, and machine learning.

# Background

## Project scope

A place for high school students to get personalized recommendations and suggestions for colleges that we believe are an excellent fit based on the student's preferences, academic ambitions, and future goals. We are committed to guiding students toward a path that meets their educational needs and inspires their success.

The main area of the application is our chatbot which interacts with the user answering their questions and gathering information about the student, and a recommendation system uses the gathered information to recommend a college for that student.

## Motivation and Beneficiaries

The primary motivation behind our project is to reduce the confusion and anxiety of high school students in university selection, especially in Egypt. By providing a comprehensive College inquiries and Recommendation System, we aim to help students with personalized recommendations and detailed insights into various colleges and universities.

The primary beneficiaries of our project are high school students looking to enter higher education. Through our project, we aim to help them make an informed decision about their future careers.

## Main Techniques and Technologies

Our system is composed of a web application that includes a chatbot. The main techniques and technologies used in our system are:

* **Rasa Open source**: Rasa is an open-source conversational AI framework designed for building contextually aware conversational AI.
* **Application**: Chatbot Development
* **Django REST framework**: Django REST framework is a Python framework used to build restful web API.
* **Application**: Web Application’s Backend
* **React.JS**: React.JS is a popular JavaScript library for building dynamic, and interactive user interfaces.
* **Application**: Web Application’s Frontend
* **MongoDB**: MongoDB is a flexible and scalable NoSQL database that uses a document-oriented data model.
* **Application**: Data Storage
* **MySQL**: MySQL is a popular open-source relational database management system (RDBMS) known for its reliability, performance, and ease of use.
* **Application**: Data Storage
* **Machine Learning libraries**: Machine learning libraries encompass a wide range of tools and frameworks used for building and training machine learning models and providing access to various other algorithms and techniques.
* **Application**: Building the Recommendation System

# Problem Definition

High school students, particularly in Egypt, face significant uncertainty and anxiety when it comes to selecting a suitable university or college for their higher education. The lack of accessible and comprehensive information about various institutions, coupled with the absence of personalized guidance, often leaves students feeling overwhelmed and directionless.

As a result, there is a need for a platform that offers tailored recommendations and detailed insights into colleges and universities, aligning with students' preferences, academic aspirations, and future career goals.

# Market Study

When facing the decision of choosing a college, people try several solutions:

1. **Asking other people:**

People tend to ask other people about their opinions regarding colleges they consider candidates. This is a decent solution. However, people around may just be a non-representative sample. They may mislead the student and guide him towards an unsuitable decision.

1. **Online research:**

A lot of websites provide data about colleges such as <https://www.dbse.co/universities>, <https://egecmena.com/ar>, <https://bigfuture.collegeboard.org/>, <https://www.appily.com/> and many more. However, those websites have a lot of problems such as outdated and insufficient data. It’s usually the case that you an extensive search is needed to know little information that help in taking a decision.

1. **Visiting multiple colleges:**

Visiting colleges is a very effective solution. However, it requires a lot of time and effort.

1. **Chatbots**:

Options like ChatGPT are available for students to use in order to help them decide. However, ChatGPT has a more general use and is not very good when addressing the domain at hand. Other specialized options are emerging such as <https://mainstay.com/>. However, they only support universities and colleges in foreign countries.

Our solution facilitates the process of choosing a suitable college through a single web app. This app is equipped with data about all colleges, a chatbot and a recommendation system. Hence, it saves a lot of time and effort in comparison with the various solutions stated above. It basically works as a hub that includes data of all colleges, ratings of other people, on-demand answers to inquiries and personalized recommendations to high school students.

# Data Collection

During our project, we needed to collect data such as university information, questions that the students might ask, and answers to these questions. While collecting these data we faced several challenges related to data, including:

* Lack of Centralization: The required data was scattered across multiple locations rather than being centralized in one place.
* Outdated Data: A big portion of the data we needed was not maintained and updated regularly.
* Accessibility Issues: Some data needed for our project was not readily accessible, requiring a big effort to acquire it ourselves.

To address these challenges, we used two primary approaches:

1. Web Scraping: by using web scraping techniques, we extracted relevant data from various online sources, gathering scattered info into a more centralized location for use in our project.
2. Crowdsourcing: due to the data unavailability, we needed to gather the information from diverse sources and contributors by ourselves.

In the following sections, we talk about the strategies used in more detail and the outcomes achieved through these two approaches.

## Web Scraping

The technique we used for web scaping is HTML parsing in which we send HTTP requests to websites, retrieving the HTML content of the page then parse the HTML to extract the desired data. This is done by using libraries such as BeautifulSoup for Python.

This approach helped us solve a big portion of the challenges associated with the lack of centralization and outdated data. Scrapping data from multiple websites and then cleaning and storing them helped us make the data more centralized. Also, during the process of scrapping these websites, we encountered multiple websites that contained the same data, which allowed us to identify and select the most recent and relevant information.

## Crowd Sourcing

To overcome the challenge of data accessibility, we spent a considerable effort on crowdsourcing. The process consisted of two steps.

Firstly, we sent forms to our colleagues and institutions, including other universities. These forms consisted of the questions which may be asked to our bot such as general information about colleges, etc.

Secondly, we developed a specialized tool to help us gather answers to these questions more easily. To be able to view and clean these answers more easily. We will talk about the tool in more detail in the following section.

Of course, crowd-sourcing has multiple drawbacks such as the quality of data due to different individuals contributing to the data. However, these problems can be mitigated by an effort to clean the data and double-check it before using it.

### PathFinder Tool

PathFinder tool is a simple website designed to facilitate the gathering of responses to the collected questions that our chatbot may encounter from diverse sources, including colleagues and institutions. it consists of multiple simple components:

Figure 1 – Crowd Sourcing Tool.

* Frontend component: a simple web interface that displays a randomized question from a predetermined pool of questions to the user. The user is then asked to either answer the question or skip it. If the user answers the question and submits then the answer is sent to the backend.
* Backend component: the backend receives the question's ID and the question's answer from the frontend via an HTTP POST request. It then validates that the answer is not empty. Finally, it stores the answer in the database using the question's id.
* Database: the database is a MySQL database hosted on pythonanywhere. It follows the schema previously defined by the ERD in figure 16.

PathFinder Tool serves as a complementary tool to web scraping and other data acquisition methods. By combining insights gathered through the tool with other sources of data. We can ensure that the data is well-rounded for our use.

# Project Specifications

## Functional Requirements

For Students

* Sign up for the system by providing personal details and email.
* Log in to the system using email and password.
* View comprehensive information about colleges and universities.
* Input preferences and queries in natural language to receive personalized recommendations.
* Receive detailed information about recommended colleges.
* Search for colleges based on name or its university and display it in an organized manner.
* Compare two colleges easily, in an organized manner.
* Access news updates related to universities' events, scholarship opportunities, and more.
* Filter displayed news content based on college/university.
* Select and compare two colleges side by side.
* Provide feedback on college choices and the recommendation process.
* Access chat history.

For Universities/Colleges

* Sign up by providing the necessary details.
* Sign in to the system using the provided credentials.
* Edit or delete uploaded information about colleges and courses.
* View all uploaded information and products.

For Admin

* View the University's details and upload information.
* Add/remove/edit universities’ details.

## Non-Functional Requirements

### Performance Requirements

* Response Time
  + **Requirement**: The system shall respond to user interactions within an average response time of 3 seconds, measured from the initiation of the request to the display of the corresponding response.
  + **Rationale**: Prompt responses enhance user satisfaction and engagement, improving overall user experience.
* Concurrent User Handling
  + **Requirement**: The system shall support a minimum of 100 concurrent users without degradation of response time beyond 2 seconds.
  + **Rationale**: The platform must handle a significant number of users simultaneously to accommodate peak usage times without compromising performance.
* Database Query Performance
  + **Requirement**: Database queries should execute within an average time of 200 milliseconds, considering a reasonable load and dataset size.
  + **Rationale**: Efficient database performance is crucial for data retrieval and application responsiveness.
* Chatbot Response Time
  + **Requirement**: The Chatbot shall generate responses to user inputs within 1 second, providing a quick and natural conversation flow.
  + **Rationale**: Swift responses from the Chatbot maintain engagement and encourage continued interaction.
* Error Handling
  + **Requirement**: The system shall display clear error messages within 3 seconds of an error occurrence, aiding users in understanding and resolving issues.
  + **Rationale**: Quick and informative error handling is crucial for seamless user experience and effective troubleshooting.

### Software Quality Attributes

* Usability

"The system shall provide intuitive user interfaces, clear navigation paths, and contextual help features to ensure that users can easily understand and utilize the system's functionalities.

* Reliability

The system shall achieve a system uptime of 99.9% availability, calculated monthly.

* Maintainability

The system shall be designed and implemented in a modular fashion, allowing for easy maintenance and future enhancements by facilitating the isolation and replacement of individual components without impacting the overall system functionality.

* Portability

The system shall be compatible with the latest versions of major web browsers (e.g., Chrome, Firefox, Safari) on Windows, macOS, and iOS, Android operating systems.

## Stakeholders

1. **End-user**
   * 1. **Student** high school graduates who need college and career advice based on their preferences, desires, and career goals.
     2. **University/Colleges** who want students to know more details about their programs, scholarships, events, and tuition fees.
     3. **Admins** who verify and suspend users, verify universities and handle feedback.
2. **Developers**.

Collect the data needed for the system, design, build, deploy, and test the system to be ready for users.

## Use-Case Diagrams

### Student use case diagram

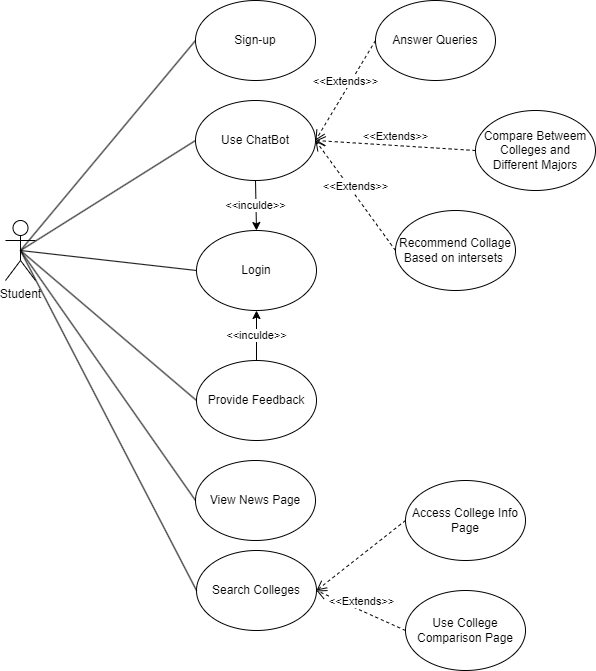


Figure 2 - Student use case diagram.

### University use case diagram

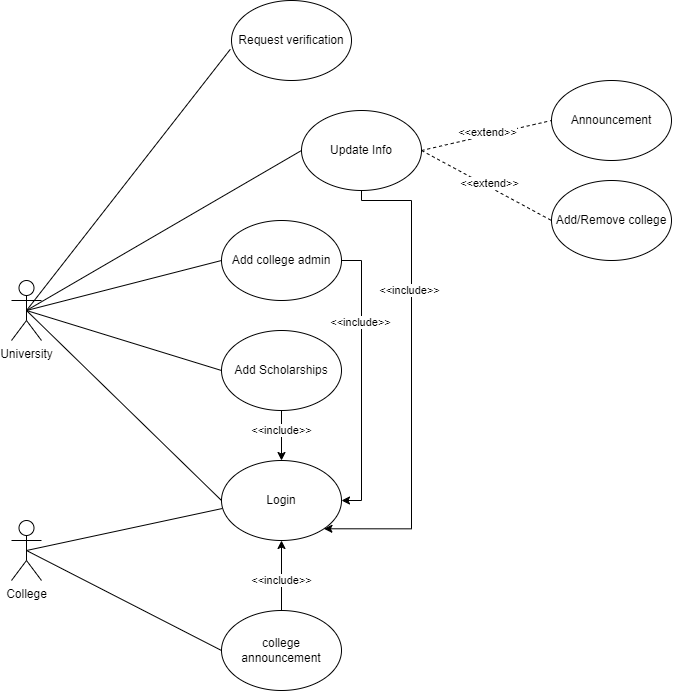


Figure 3 - University use case diagram.

### Admin use case diagram

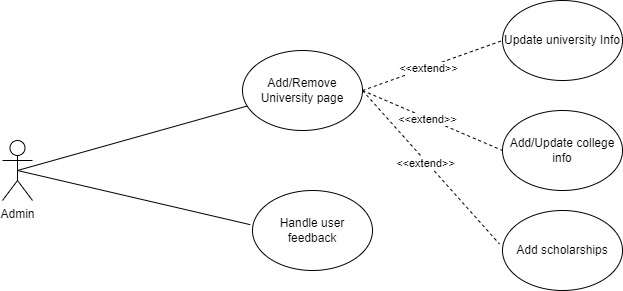


Figure 4 - Admin use case diagram.

## System Design

### Component Diagram

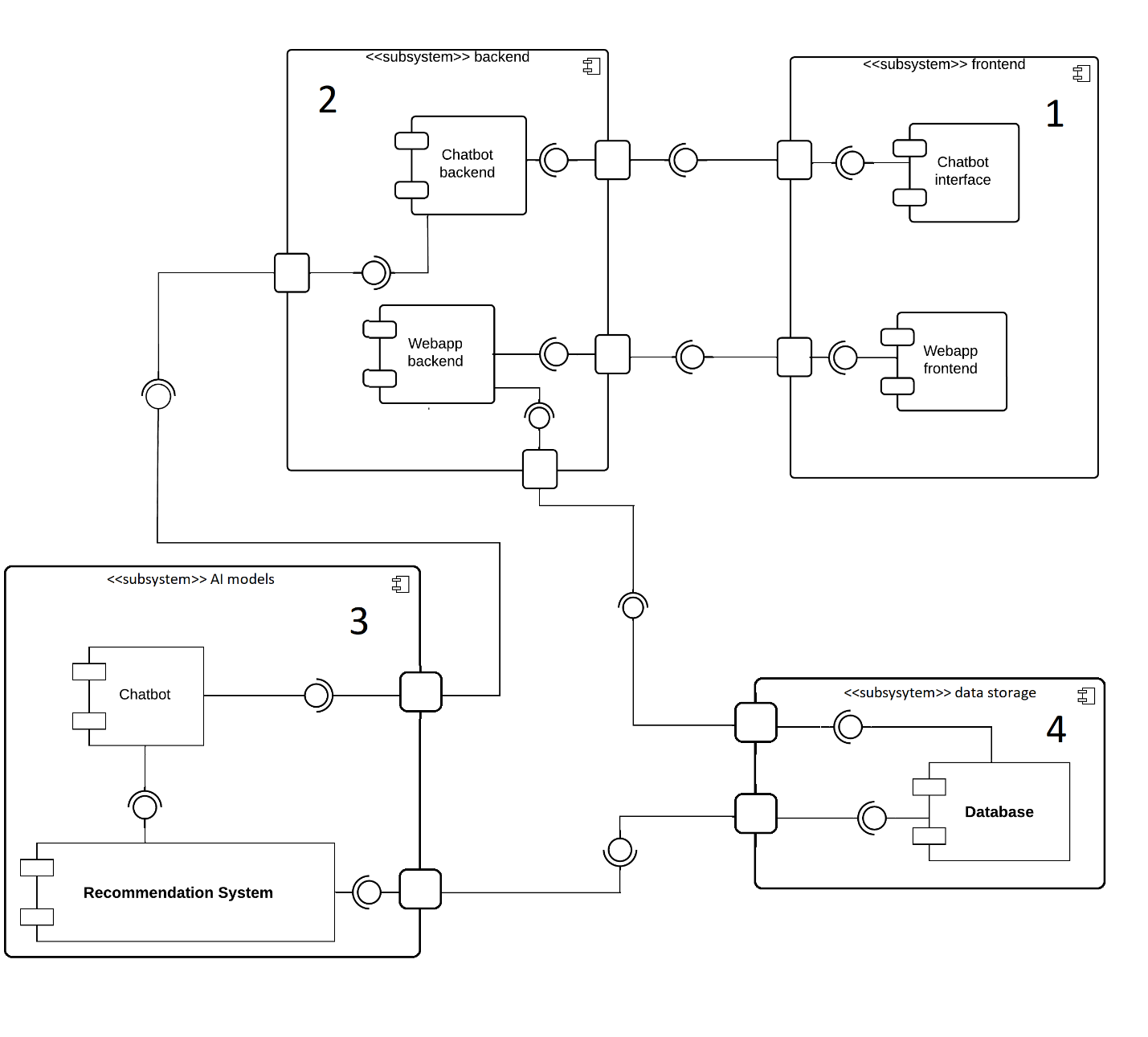


Figure 5 - Component Diagram.

There are four main subsystems in our component diagram:

1. Frontend subsystem

This subsystem represents an intermediate layer between the end user and the rest of the system. It includes two components:

* Chatbot interface

The main page of the chatbot that the user interacts with. This component sends the user message directly to the chatbot backend component.

* Webapp frontend

The user interface for the web application. It allows the user to interact with application features.

1. Backend subsystem

A layer that includes the server-side logic and handles requests from the frontend subsystem.

* Chatbot backend  
  It handles validation and preprocessing to the user message before passing it to the chatbot.
* Webapp backend: It handles business logic such as user authentication, data retrieval and receiving feedback.

1. AI models subsystem

It represents the AI components of the system such as the chatbot and recommendation system.

* Chatbot

A conversational agent that handles user inquiries. It is responsible for understanding the user messages and providing helpful responses.

* Recommendation system

It uses user requests and preferences to provide personalized recommendations.

1. Data storage subsystem  
   This subsystem represents data storage and management. It has only one component which is the database. The database stores the universities’ data, students’ data, and students’ feedback. The database follows the schema defined in the ERD of figure 16.

### Sequence Diagrams

#### Student Sequence Diagrams

1. Use chatbot.

A white sheet of paper with black lines

Description automatically generated

Figure 6 - Student Use Chatbot Sequence Diagram

1. Sign-up

A diagram of a program

Description automatically generated

Figure 7 - Student Sign-Up Sequence Diagram

1. Log-in

A diagram of a login process

Description automatically generated

Figure 8 - Student Log-In Sequence Diagram

1. Student search for college

A diagram of a diagram

Description automatically generated

Figure 9 - Student Search College Sequence Diagram

1. Student compare colleges

A diagram of a diagram

Description automatically generated

Figure 10 - Student Compare Colleges Sequence Diagram

#### Admin Sequence Diagrams

1. Admin review feedback

A diagram of a webpage

Description automatically generated

Figure 11 - Admin Review Feedback Sequence Diagram

1. Admin add university

A diagram of a webpage

Description automatically generated

Figure 12 - Admin Add University Sequence Diagram

#### University Sequence Diagrams

1. University Sign up

A diagram of a website

Description automatically generated

Figure 13 - University Sign-Up Sequence Diagram

1. University announcement

A diagram of a diagram

Description automatically generated

Figure 14 - University Announcement Sequence Diagram

1. University add college admin

A diagram of a company

Description automatically generated

Figure 15 - University Add College Admin Sequence Diagram

### ERD

#### Main ERD

A diagram of a diagram

Description automatically generated

Figure 16 - Main ERD

#### PathFinder Tool ERD

A black and white diamond with black text

Description automatically generated

Figure 17 - PathFinder Tool ERD

# Machine Learning Models

## Chatbot Model

Our Arabic chatbot is built using Rasa. Rasa is an open-source conversational AI framework designed for building contextually aware conversational AI. Rasa chatbots are split into two main parts, Rasa NLU and Rasa Core.

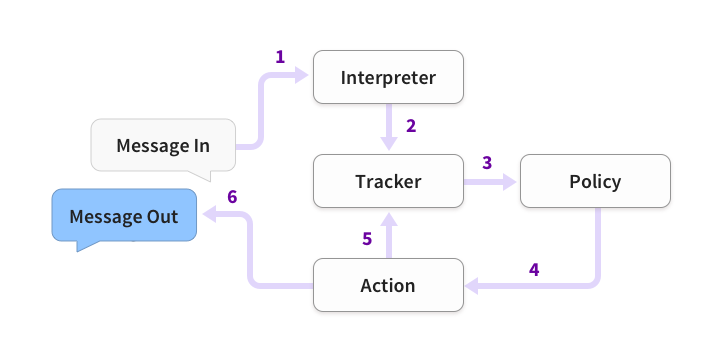
Rasa NLU interprets the user input and extracts entities and intent with the help of various pipelines. It then converts the input into a dictionary that includes the original text, intent, and entities identified, which is then sent to Rasa CORE. Rasa CORE is responsible for the chatbot’s action. It selects the appropriate action as per the user input and the specified policy then sends it back as a chatbot response.

Figure 18 – Rasa pipeline and policies.

### NLU Training Data

NLU training data consists of example user utterances categorized by intent. Training examples can also include entities. Entities are structured pieces of information that can be extracted from a user's message. You can also add extra information such as regular expressions and lookup tables to your training data to help the model identify intents and entities correctly. Training data is split into two parts, Training examples and Conversation data.

#### Training Examples

* Intent Training Examples: in which each intent has examples that help identify the correct intent.
* Entities: Entities are structured pieces of information that can be extracted from a user's message. They are annotated in training examples with the entity's name.

#### Conversation Data

Stories and rules are both representations of conversations between a user and a chatbot. They are used to train the dialogue management model. Stories are used to train a machine learning model to identify patterns in conversations and generalize to unseen conversation paths. Rules describe small pieces of conversations that should always follow the same path and are used to train the RulePolicy.

### Rasa NLU Pipeline

In Rasa, a pipeline is a set of data processing units arranged in series, where the output of one element serves as the input for the subsequent element. Incoming messages undergo processing through a sequence of components defined in the config.yml file. These components are executed sequentially, comprising the natural language understanding (NLU) pipeline. Choosing an NLU pipeline allows customization and fine-tuning of the model on the dataset. The pipeline consists of five types of components, each serving a specific purpose:

* **Tokenizer**
* **Featurizer**
* **Intent Classifier**
* **Entity Extractor**
* **Response Selector**

#### Tokenizer

Tokenization refers to the process of converting a sequence of text into smaller parts, known as tokens. These tokens can be as small as characters or as long as words. The primary reason this process matters is that it helps machines understand human language by breaking it down into bite-sized pieces, which are easier to analyze.

The tokenizer we use in our Arabic chatbot is the “WhitespaceTokenizer”, which efficiently breaks down Arabic text into individual tokens based on whitespace characters such as spaces, tabs, and line breaks. This tokenizer is particularly useful in Arabic text processing pipelines as it allows for straightforward segmentation of text into meaningful units, which can then be further processed for other tasks.

#### Featurizer

Featurizers in Rasa are components responsible for converting tokens from the tokenizer into numerical features that machine learning models can understand and learn from. These features capture various aspects of the input text.

In our Arabic chatbot, we use multiple featurizers to capture different aspects of the input and increase our chatbot robustness since if one featurizer performs poorly the others may compensate for its performance. Finally, the output features are fused to create a combined feature representation of our tokens. The featurizers are as follows:

* RegexFeaturizer: During training the RegexFeaturizer creates a list of regular expressions defined in the training data format. For each regex, a feature will be set marking whether this expression was found in the user message or not.
* LexicalSyntacticFeaturizer: The LexicalSyntacticFeaturizer is a component meant to create features that are useful when detecting entities. It moves with a sliding window over every token in the user message and creates feature configurations such as, whether the current token is a title, or is the token before/after is a title, and if it is at the end or beginning of a sentence, in general, these features are useful to know to be able to detect entities in the user message such as names, cities, etc
* CountVectorsFeaturizer: The CountVectorsFeaturizer builds a vocabulary based on tokens observed in the training data and each token becomes a feature. The CountVectorsFeaturizer then generates a feature vector based on the presence or absence of each token. In our pipeline, we use two of this featurizer
  + Word level CountVectorsFeaturizer in which word token counts are used as a feature.
  + Char level CountVectorsFeaturizer in which n-chars of token counts are used as a feature.

#### Intent Classifier

Intent classifiers are components responsible for determining the intent behind user messages, based on feature vectors, according to the intents defined in the training data, and then return rankings of intents. In our Arabic Chatbot, we use two intent classifiers DIETClassifier and FallbackClassifier

* DIETClassifier: DIET (Dual Intent and Entity Transformer) is a multi-task architecture for intent classification and entity recognition. It utilizes transformer-based architectures which is a type of neural network commonly used in NLP tasks.
* FallbackClassifier: The FallbackClassifier classifies a user message with the intent nlu\_fallback in case the previous intent classifier wasn't able to classify an intent with confidence greater than or equal to the threshold of the FallbackClassifier (usually 0.5).

#### Entity Extractor

Entity extractors extract entities, such as person names or locations, or any other information from the user message. We use EntitySynonymMapper along with the DIETClassifier from the previous component to extract the entities.

* DIETClassifier: which is also used in entity extraction.
* EntitySynonymMapper: this component maps extracted entities into specified values. For example, it maps both NYC and New York City to NY

#### Response Selector

Response selectors are components responsible for handling retrieval actions, which involve selecting a response from a predefined set of responses based on the predicted intent, its subintents, and the context of the conversation. Retrieval actions are used for tasks like frequently asked questions (FAQs, or any other cases where intent can be split into sub-intends

The response selector used in Rasa is called ResponseSelector. It works by ranking all subintents within the specified intent, which is defined using the retrieval\_intent parameter in the pipeline configuration. The retrieval\_intent parameter allows you to designate certain intents as retrieval intents, indicating that they are associated with a set of predefined responses.

### Rasa Core Policies

In Rasa, policies are responsible for making decisions about what actions the chatbot should take based on the current conversation state and user input. Policies determine the bot's behavior by mapping predicted intents and entities, and taking into consideration the chat history, to appropriate dialogue actions, such as responding with a message, asking for clarification, or executing an external function.

In our chatbot, we use multiple different policies to choose the action and the one with the highest confidence level is chosen, and in case there is a tie it is decided using policy priorities where a higher number means higher priority. The used policies are:

* RulePolicy (priority: 6): The RulePolicy is a policy that handles conversation parts that follow a fixed behavior (e.g. business logic). It makes predictions based on any rules you have in your training data. It also handles the fallback intent in which the user asks for something outside the scope of the chatbot.
* MemoizationPolicy (priority: 3): The MemoizationPolicy remembers the stories from your training data. It checks if the current conversation matches the stories in your stories.yml file. If so, it will predict the next action from the matching stories of your training data with a confidence of 1.0. If no matching conversation is found, the policy predicts None with confidence 0.0.
* UnexpecTEDIntentPolicy (priority: 2): UnexpecTEDIntentPolicy allows your bot to react to unlikely user turns. It has the same model architecture as TEDPolicy. The difference is at a task level. Instead of learning the best action to be triggered next, UnexpecTEDIntentPolicy learns the set of intents that are most likely to be expressed by the user given the conversation context from training stories. It predicts the action action\_unlikely\_intent only if the intent is unlikely to happen in the conversation flow.
* TEDPolicy (priority: 1): TEDPolicy is a machine learning policy that uses several transformer encoders to learn the best action that should be done given the dialog’s history.

## Recommendation System

Figure 19 – Recommendation system types.

A recommendation system is a data filtering tool that recommends the most relevant items to a particular user or a customer using machine learning algorithms. In our case, items are colleges and users are high school students.

There are four main types of recommendation systems or engines: Content-Based filtering, Collaborative filtering, Knowledge-Based filtering and Hybrid recommendation systems. We will care the most about Knowledge-Based recommendation systems.

**Knowledge-Based Recommendation Systems (KB RS):**

A knowledge-based recommendation system is a system that make recommendations based on specific queries made by the user, not on a user’s rating history. It prompts the user to give some rules or guidelines on what the results should look like. The user may also give an example of an item. The system searches through its item database and tries to find a similar match. There are two main types of knowledge-based recommendation systems:

**1. Constraint-Based:**

It uses existing knowledge bases that contain explicit rules about how to relate user requirements with item features. It is very similar to an expert system in knowledge base systems. It is considered a constraint satisfaction problem.

**2. Case-Based:**

It depends on letting the user choose a target or an anchor item the algorithm finds a similar item to recommend. Results are usually treated as new target cases with some interactive modifications. This is similar to Content-Based systems where items similar to ones the user previously liked are suggested. The main difference is that most knowledge-based systems depend on the description of the items in the form of relational attributes in knowledge bases rather than as text keywords like in Content-Based systems.

**2.1. Similarity Metrics:**

Similarity metrics are needed to retrieve examples similar to the specified item. For continuous variables, similarity can be the difference between two numbers. For example, two different colleges have similar fees if the difference in fees is close to zero. It can also be more complex and require the use of statistical measures such as the standard deviation. When it comes to categorial variables, similarity is much more challenging. Domain hierarchies are often used in this case. Domain hierarchies are tree graphs where each category can have a parent category and subcategories. The similarity of two items can then be measured by the length of the path between the items.

**2.2. Critiquing:**

Once some result is found using similarity metrics and recommended to the user, the user is able to provide feedback and customizes the results to match what he is looking for. The user specifies a change request on some attributes of the item they like. The change request can be a directional critique or a replacement critique. A directional critique provides feedback on the direction in which the recommendation should be adjusted. For example, if the system recommended a college that is 100 km away, the user may request a shorter distance. On the other hand, a replacement critique suggests alternative items to replace the initially recommended item. For example, if the system recommended a computer engineering college, the user may suggest recommending a computer science college instead.

**KB RS Strengths:**

* Complex Item Domain: KB RS is very proficient when the items have many complex aspects to consider. Those aspects often need expert knowledge in the domain. KB RS effectively captures those complex aspects by utilizing knowledge bases crafted by domain experts. This strength is especially significant when the item is of great importance and requires a lot of thought, such as a university or a college.
* Avoids Cold Start Problem: User data is not needed because what the user wants is explicitly defined. The recommendation process is accurate and can start and work well without requiring existing rating data.

**KB RS Weaknesses:**

* Knowledge Acquisition Bottleneck: A big problem lies in the creation of the knowledge base. It requires the conversion of the knowledge possessed by domain experts into consumable representations.

# Work Plan

## Task Table

|  |  |  |  |
| --- | --- | --- | --- |
| Task | Task Title | Description | Task Statues |
| 1 | Determine project idea | Decide what the main idea for the project is and see if there are any problems with it. | Completed |
| 2 | Searching and studying related work | Search for similar projects and see how they were completed. | Completed |
| 3 | Determine project features | Decide what features our project will have. | Completed |
| 4 | Study needed technologies | Learn needed technologies to be able to implement our project. | In Progress |
| 5 | Web scraping | Collect data using web scraping. | In Progress |
| 6 | Pathfinder tool development and deployment | Develop and deploy the Pathfinder Tool to help with data collection. | Completed |
| 7 | Crowdsource data | Send the Pathfinder tool to other people to crowdsource data. | In Progress |
| 8 | Clean collected data | Clean the collected data to ensure its quality. | In Progress |
| 9 | Develop Prototype | Develop a chatbot prototype along with a simple interface | Completed |
| 10 | Write Documentation | Design UML Diagrams, and finish mid-year documentation to show progress | Completed |
| 11 | Develop Backend | Develop web application’s backend | Planned |
| 12 | Develop Frontend | Develop web application’s frontend. | Planned |
| 13 | Chatbot development | Develop the Arabic chatbot and ensure its accuracy in predicting user intents. | Planned |
| 14 | Build recommendation system | Build the recommendation machine learning model. | Planned |
| 15 | Testing and Evaluation | Test the chatbot, recommendation system, and the web application as a whole | Planned |

Table 1 - Task Table

## Gantt Chart

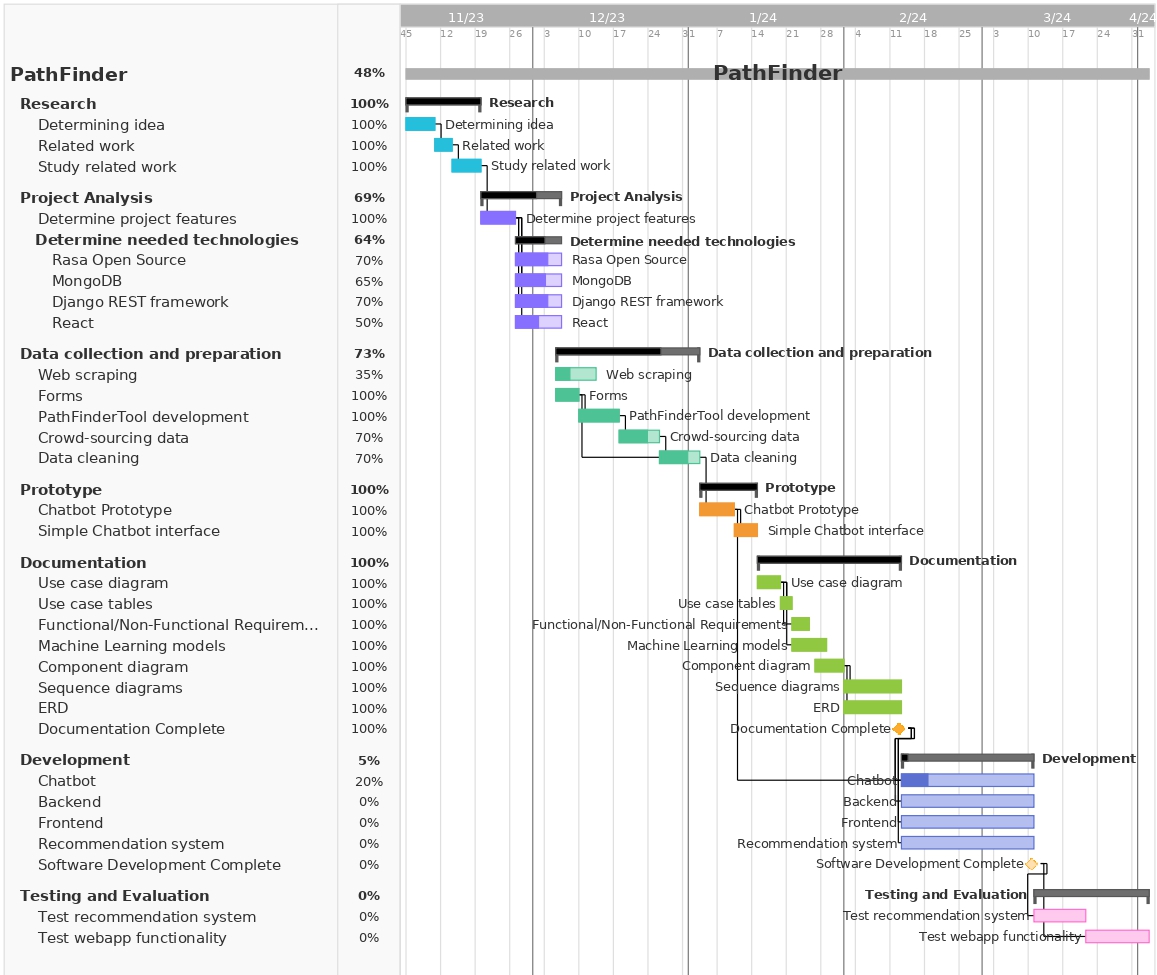


Figure 20 - Gantt Chart

# References

Paul0. (n.d.). Knowledge-Based RS. Medium. <https://medium.com/@paul0/knowledge-based-rs-b4547b84108a>

jwu2. (n.d.). Knowledge-Based Recommender Systems: An Overview. Medium. <https://medium.com/@jwu2/knowledge-based-recommender-systems-an-overview-536b63721dba>

Rasa. (2024). Rasa Open Source Documentation. <https://rasa.com/docs/rasa/>

Rasa. (n.d.). Conversational AI with Rasa, Pipelines and Policies. <https://learning.rasa.com/conversational-ai-with-rasa/pipeline/>

Knoldus. (n.d.). Understanding the Rasa NLU Pipeline. Knoldus Blog. <https://blog.knoldus.com/understanding-the-rasa-nlu-pipeline/>