

Developing a Website to Analyze and Validate Projects using LangChain and Streamlit

Mukesh Pillai
Dept. of Master of Computer Applications
Sardar Patel Institute of technology
Mumbai, India
mukesh.pillai@spit.ac.in

Pallavi Thakur
Dept. of Master of Computer Applications
Sardar Patel Institute of technology
Mumbai, India
pallavi.thakur@spit.ac.in

Abstract— This research study provides a new approach to project analysis and validation through the creation of a website using LangChain, a framework for building applications powered by Language Models, and Streamlit which is used to create LLM powered and Generative AI web applications quickly. In an era of rapid technological change, the necessity for effective project assessment tools has become increasingly crucial. The goal of this research is to overcome this obstacle by utilizing LangChain's natural language processing and understanding capabilities. The proposed website serves as a basic platform for project validation, making it easier to assess project feasibility, technical requirements, and work allocation. It serves as a starting point for developing a comprehensive system. The features of LangChain allow for the extraction of valuable insights from project data, technology, and team experience, resulting in a more automated and intelligent decision-making process. The developed website encompasses two main functionalities: individual and team project validation and task allocation. For individual projects, users can input project details and technologies, and their resume after which LangChain processes the information to determine feasibility and generate technical specifications based on the assessment of their resume. On the other hand, the team task allocation feature, utilizes LangChain to analyze frontend and backend specifications, and project requirements to intelligently allocate tasks based on team's experience level.

Keywords— *LangChain, Streamlit, Project Analysis, Validation, Task Allocation, Feasibility Analysis, Large Language Mode (LLM), Generative Artificial Intelligence (AI)*

I. INTRODUCTION

This research work introduces an innovative project, aiming to revolutionize project analysis and validation through the utilization of LangChain, an advanced language model technology, and Streamlit, a modern Python library for web applications. As technological advancements continue to reshape the project development paradigm, the need for sophisticated tools to assess project feasibility, technical specifications, and task allocation has become increasingly pronounced. For any software development, its lifecycle remains the same from feasibility analysis to implementation. [1]

This research sets out to address this demand by leveraging the capabilities of LangChain, an advanced language model developed by OpenAI. LangChain provides natural language processing and understanding, enabling the extraction of meaningful insights from project details, technological requirements, and team expertise. The proposed website serves as a pivotal platform for project validation, offering a streamlined process for users to input

project specifics and receive automated assessments regarding feasibility and technical specifications.

The research draws inspiration from recent breakthroughs in language models, particularly OpenAI's GPT-3.5 architecture, known for its advanced Natural Language processing capabilities. [2] The integration of Streamlit into the project facilitates the development of a user-friendly web application [3], ensuring accessibility and ease of use for individuals and teams involved in project development.

The overall goal is to provide a comprehensive system that not only accelerates the project validation process but also intelligently allocates tasks within a team, this research acts as a head start towards decision making of approaching problem statements that can be profitable for the organization or individual. This approach aligns with the broader trends in AI-driven decision-making processes and the growing emphasis on automation in project management.

The subsequent sections of this research work will delve into the technical aspects of the project, exploring how LangChain's capabilities are harnessed to achieve individual and team project validation functionalities.

II. LITERATURE REVIEW

The integration of LangChain, an open-source software library, has marked a significant advancement in developing custom AI applications through the utilization of Language Model (LLMs) - An LLM is a deep neural network model which has been trained on large amounts of data, such as books, code, articles, and websites, to learn the underlying patterns and relationships in the language that it was trained for. [4]

This development has garnered considerable attention within the AI community, positioning LangChain as a solution provider for various stages in the creation of tailored AI applications [5]. LangChain offers a versatile platform that harnesses the capabilities of language models, allowing for the implementation of a wide array of use cases. Notably, it has streamlined the process of developing AI applications by facilitating efficient interaction with LLMs. Despite this progress, the demands of custom AI applications extend beyond mere interaction through a web interface.

Web interfaces play a pivotal role in rendering complex technologies accessible to users. Streamlit, a widely recognized framework for crafting data applications, has gained acclaim for its simplicity and speed [3]. Existing

research underscores the significance of user-friendly interfaces in augmenting user engagement and optimizing interactions. The convergence of advanced language models with streamlined web interfaces has emerged as a prominent trend across various applications.

This research study addresses the challenge of efficiently implementing project features and functionalities within tight timelines. It proposes an innovative approach using LangChain and Streamlit to swiftly validate and prototype project ideas. The solution aims to streamline these processes for individual developers and teams working on projects with limited time.

The inspiration for this research project has been drawn from applications that have already been built using LangChain for a variety of use cases such as Automating Customer Service [6], summarization of PDF using Langchain [7] and Chat to Data Visualization application which generates visualizations on the given dataset using prompts. [8]

III. METHODOLOGY

Following is the proposed methodology where the frontend or client side is built using Streamlit and LangChain is used for creating LLM Application using Python.

First, let us understand the working of LangChain and then work upon the use case. There are essentially two ways to interact with LLMS:

1. Chatting, which involves writing prompts, sending it to the model and getting text responses in return.
2. Embedding which involves writing prompts, sending it to the model and getting a numeric array in return.

Both of these ways have the same problem which LangChain tries to solve as prompts are full of boilerplate texts. To get a proper response from the AI model we have to match the personality and instruction style of the model to get factual accuracy in response. LangChain solves this problem by providing a proper prompt template which has useful input with the proper boilerplate - proper instruction style. To handle the response properly LangChain provides an output parser.

Output parsers are classes that help structure language model responses. There are two main methods an output parser must implement:

"Get format instructions": A method which returns a string containing instructions for how the output of a language model should be formatted.

"Parse": A method which takes in a string (assumed to be the response from a language model) and parses it into some structure.

And then one optional one:

"Parse with prompt": A method which takes in a string (assumed to be the response from a language model) and a prompt (assumed to be the prompt that generated such a response) and parses it into some structure. The prompt is largely provided in the event the OutputParser wants to retry

or fix the output in some way, and needs information from the prompt to do so. [9]

Using LangChain, it is easily possible to swap between multiple LLMs for better performance and functionality. Then we have a chat message history which can be sent back to the model to remind it about the context of chat or use it for future purposes.

The main components of LangChain are:

LLMs (Language Models): LLM are deep learning models that are trained on a vast amount of data. These are the core reasoning engines in LangChain.

Prompt Templates: Prompt templates are pre-defined recipes for generating prompts for language models. A template may include instructions, few-shot examples, and specific context and questions appropriate for a given task. LangChain provides tooling to create and work with prompt templates, strives to create model agnostic templates to make it easy to reuse existing templates across different language models. [10]

They allow bundling up prompt formatting logic separately from the models themselves. There are PromptTemplates for LLMs and ChatPromptTemplates for ChatModels.

Output Parsers: These convert the raw text response from an LLM into a more usable format. For example, parsing text into JSON or just extracting the text content from a ChatMessage.

Chains: LangChain allows you to combine the above components into chains - LLM + PromptTemplate + Optional OutputParser. This bundles up pieces of logic for easier re-use.

Agents: Agents are autonomous programs powered by language models that can take actions and have conversations. They are composed of:

An Agent chain that decides what action to take next. It is a set of Tools that define capabilities the agent has access to.

An AgentExecutor runtime that executes the agent loop, calling the agent chain, taking actions, and collecting observations.

Agents build on top of the core LangChain components like prompts and chains, but add additional logic and structure to create more advanced autonomous behavior. The agent component determines the next step, tools provide capabilities, and the executor runs the full loop. Together these components allow developers to build conversational agents that can take multiple steps to accomplish tasks, while maintaining context and memory across turns. So agents are a way to leverage LangChain to create more sophisticated assistants and bots.

IV. IMPLEMENTATION

In the implementation of our use case, user inputs are gathered through a Streamlit web app with the following features:

1. Project Configuration:

Users choose between an individual or group project.

For individual projects, users provide their name, upload a resume, and input a project description encompassing both backend and frontend functionalities.

2. Tech Stack Input:

Users input the technological stack employed in their project.

3. Group Projects:

For group projects, users specify the number of team members (ranging from 1 to 5).

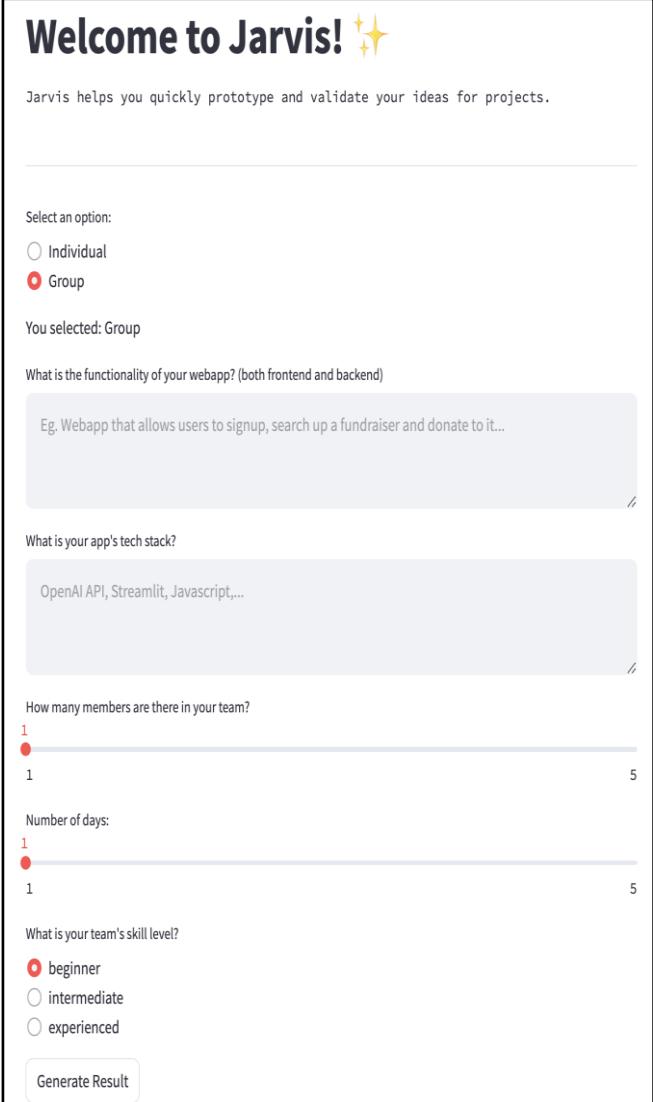
Users select the project duration in days (with a minimum of 1 and a maximum of 5).

4. Team Experience:

In group projects, team members indicate their collective experience using a radio button.

This solution aims to expedite the validation and prototyping of project ideas or problem statements within tight timelines and limited resources, such as during a Hackathon event.

The design of the user interface is structured to efficiently collect essential information relevant to the use case.



The image shows a web form titled "Welcome to Jarvis!" with a subtitle "Jarvis helps you quickly prototype and validate your ideas for projects." The form is divided into several sections: 1. "Select an option:" with radio buttons for "Individual" and "Group" (selected). 2. "You selected: Group" 3. "What is the functionality of your webapp? (both frontend and backend)" with a text input area containing "Eg. Webapp that allows users to signup, search up a fundraiser and donate to it...". 4. "What is your app's tech stack?" with a text input area containing "OpenAI API, Streamlit, Javascript,...". 5. "How many members are there in your team?" with a range slider from 1 to 5, currently set at 1. 6. "Number of days:" with a range slider from 1 to 5, currently set at 1. 7. "What is your team's skill level?" with radio buttons for "beginner" (selected), "intermediate", and "experienced". 8. A "Generate Result" button at the bottom.

Fig. 1. User Interface for Group

There are two chains running in the code i.e. the Backend Chain and Frontend Chain which run asynchronously together and provide respective technical specifications based on the inputs and analysis done.

Here both, backend and frontend chain, connect with a ChatOpenAI component provided by LangChain which helps us to connect the language model of our choice, we have used *gpt-3.5-turbo* for generating the features, technical specifications and approval of the project.

Chains are used to sequence the execution of LLMs. A chain is a series of steps, each of which involves executing an LLM. In this code, chains are used to perform the following tasks:

Backend Evaluation & Frontend Evaluation: The backend evaluation and frontend evaluation chain consists of three steps: feature extraction, specification creation, and approval check. Each step in the chain executes an LLM using a prompt template.

Chains are a useful tool for automating complex tasks that require multiple steps. By using chains, we can break down

complex tasks into smaller, more manageable steps and then execute those steps in sequence.

In the backend and frontend evaluation chain, the sequence of prompt templates unfolds as follows:

Users input project details, project technologies, group size, and group experience via the web app.

The web app transmits this information to the backend feasibility evaluation chain.

In the first chain step, the Language Model (LLM) is executed for feature extraction using a prompt template. Extracted features are then forwarded to the subsequent step.

The second chain step involves the execution of the LLM for specification creation using another prompt template. Extracted features, along with additional information from user input, are utilized in this step. The resulting specification is then passed to the next stage.

The third chain step executes the same Language Model (LLM) for an approval check using a distinct prompt template. The generated specification, combined with supplementary details from the user input, is processed in this step. The resulting approval status and comments are subsequently relayed back to the web app.

Both chains operate asynchronously, and the web app presents the approval status and comments to the user.

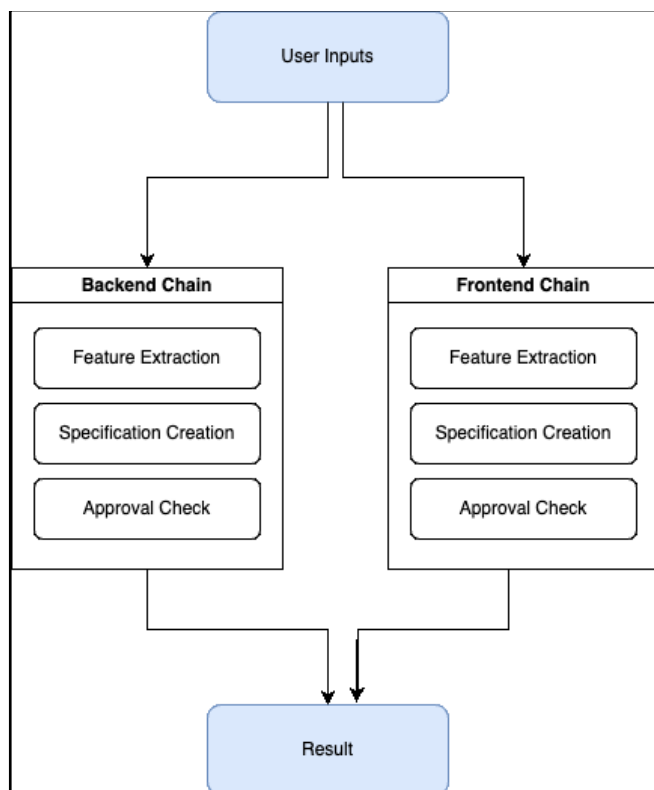


Fig. 2. Flow Diagram

This is how the prompt template looks like:

Here, `{project_details}` and `{project_technologies}` inside the prompt template are taken from the user input, similarly other inputs are referred to as well in the further templates.

```

"""Given the following project description and tech stack,
identify and elaborate on the key backend features that
would be necessary for development. The backend
features should not involve any frontend components or
styling. The backend features should be described in
terms of capabilities.
This project is to be completed within {duration} days.
The MVP should be the minimum 5 features necessary to
have a working prototype.
Project description: {project_details}
Technologies: {project_technologies}"""

```

Response: `{backend_features}`

Template 1

The Response from this template is passed on to create specifications.

```

"""Given the extracted backend features and the specified
skills/technologies, create a detailed technical
specification. This specification should include the
technologies to be used, the architecture, the different
routes/endpoints, their inputs and outputs. The MVP
should be the minimum 5 features necessary to have a
working prototype. You should ignore the technologies
for the frontend and focus on the backend. Also mention
how to approach and prioritize these features for the
given {duration} days and how it can be allocated among
the {group_size} members given that the group has
{group_experience}.

```

Backend Features: `{backend_features}`

Project Technologies: `{project_technologies}` """

Response: `{technical_specification}`

Template 2

After the specification creation, it is passed on for evaluation to the next template which determines whether the Backend is approved or not.

```

"""Given the developed technical specification, conduct a
thorough review of the MVP Features only for any
inconsistencies or issues. Also, evaluate whether the
MVP Features can be realistically completed within the
{duration} days for period {group_size} people,
considering the complexity and the technology stack
required. The MVP Features are specifically listed under
the heading 'MVP Features'. Please completely disregard
any features or sections listed under 'Additional Features'
or any similar headers.

```

This specification is only for the {aspect} aspect of the project, and should not be evaluated for other aspects.

Answer this question: Can the MVP Features be realistically completed within the {duration} days period for {group_size} people with this skill level: {group_experience}?

Output only a json with keys 'approval' and 'comments'. If yes, the value of 'approval' should be '1' and the value of 'comments' should be an empty string

If not, the value of 'approval' should be '0' and the value of 'comments' should be a string with the issues and inconsistencies listed as well what cannot be achieved within the {duration} days time frame based on the skill level of people: {group_experience}.

Technical Specification: {technical_specification}

Output only a json with keys 'approval' and 'comments'. ""

Response: {approval}

Template 3

Similarly, for the frontend chain, these types of Prompt Templates are created for validating and prototyping problem statements.

Let's consider that the user opts for a group-type project, and the designated problem statement is as follows:
Scenario 1:

The user enters this problem statement: The proposed smart city project aims to enhance urban living through a comprehensive digital solution. It includes interconnected components such as a Smart Traffic Management System utilizing AI for traffic analysis and predictive maintenance, a Public Safety System with facial recognition and predictive policing, a Waste Management System with smart bins and optimal pick-up scheduling, a Smart Energy Grid for real-time energy monitoring, Urban Farming integrating vertical farming and AI-driven pest prediction, a Public Transportation System with automated scheduling and AI optimization, and Environment Monitoring using IoT sensors for real-time air and water quality checks.

The user inputs the following tech stack in the provided input box:

"The project will leverage a diverse set of technologies, including but not limited to Python, TensorFlow, React.js, Node.js, PostgreSQL, Docker, Kubernetes, AWS, IoT protocols, and various data science and AI/ML libraries."

The prompt entered need not be perfect but it has to carry some sort of meaning and should be correct. The project is designated for one team member with a project duration of one day, and the user's skill level is classified as beginner.

Based on these inputs, the generated output is as follows:

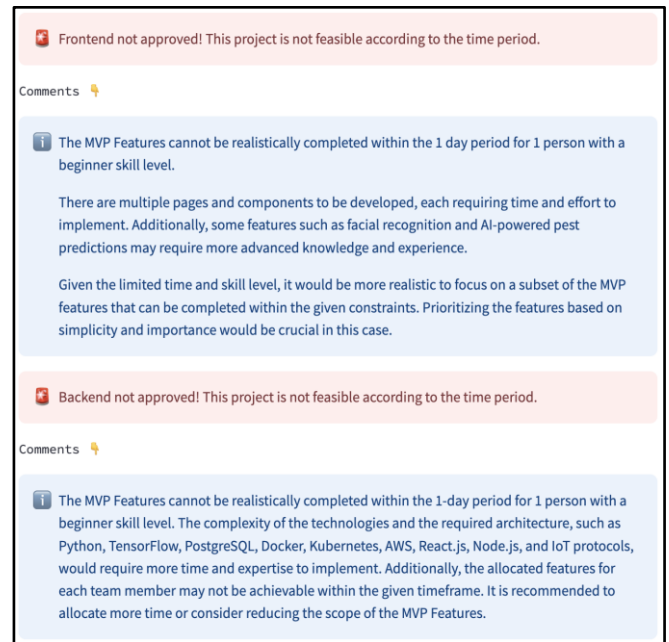


Fig. 3. Group Project Output (Scenario 1)

Given the minimal and unrealistic aspects of the problem statement, user skill level, and project duration mentioned earlier, the model rejected or disapproved the project due to its failure to meet the necessary criteria for successful completion.

Scenario 2:

In this scenario, the user opts for a group project type. The provided problem statement is:

"Developing a web application for task management within a software development team."

The user specifies the following tech stack: React, JavaScript, Python.

With a team size of 5 members, a project duration of 3 days, and an intermediate skill level, the model, upon analysis, generated the following outcomes:

✔ Frontend approved! This project is feasible according to the time period.

Frontend specifications 📌

Technical Specification:

1. User Registration and Login:

Technologies: React, JavaScript, Python, Firebase Authentication

Architecture: Single-page application (SPA)

Pages:

Registration page: Allows users to create an account by providing their email and password.

Login page: Allows users to log in using their registered credentials.

Forgot password page: Allows users to reset their password if they forget it.

Components:

Registration form: Collects user's email and password.

Login form: Collects user's email and password for authentication.

Forgot password form: Collects user's email to send a password reset link.

2. Dashboard:

Technologies: React, JavaScript

Architecture: Single-page application (SPA)

Pages:

Dashboard page: Displays all tasks assigned to the user and their current status.

Components:

Task list: Displays tasks with their details such as title, description, priority, due date, and status.

Filter and sort options: Allows users to filter and sort tasks based on criteria like priority, due date, or status.

Task status update: Allows users to mark tasks as complete or in progress.

3. Task Creation and Editing:

Technologies: React, JavaScript

Architecture: Single-page application (SPA)

Pages:

Create task page: Allows users to create a new task by providing a title, description, priority, and due date.

Edit task page: Allows users to edit the details of an existing task.

Fig. 4.1. Frontend Approval Output (Scenario 2)

Components:

Task form: Collects task details such as title, description, priority, and due date.

Team member dropdown: Allows users to assign a task to a specific team member.

4. Task Details:

Technologies: React, JavaScript

Architecture: Single-page application (SPA)

Pages:

Task details page: Displays the details of a task, including its title, description, priority, due date, and assigned team member.

Components:

Task details section: Displays the task details.

Comments section: Allows users to add comments or notes to a task.

File upload section: Allows users to attach files or documents to a task.

5. Notifications and Reminders:

Technologies: React, JavaScript

Architecture: Single-page application (SPA)

Pages:

Notification settings page: Allows users to customize their notification preferences.

Components:

Notification preferences form: Allows users to choose their preferred notification method (email, in-app) and set notification preferences for upcoming or overdue tasks.

Fig. 4.2. Frontend Approval Output (Scenario 2)

Approach and Prioritization:

Given the 3-day timeline and the intermediate skill level of the 5 members, the features can be prioritized and allocated as follows:

Day 1:

User Registration and Login: All members can work on this feature together as it is essential for user authentication and access control.

Day 2:

Dashboard: Divide the team into two groups:

Group 1: Works on the task list component and filter/sort options.

Group 2: Works on the task status update component.

Fig. 4.3. Frontend Approval Output (Scenario 2)

Day 3:

Task Creation and Editing: Divide the team into two groups:

Group 1: Works on the task form component.

Group 2: Works on the team member dropdown component.

Throughout the 3 days:

Task Details: Assign one member to work on the task details page and its components.

Notifications and Reminders: Assign one member to work on the notification settings page and its components.

By following this approach, the team can efficiently work on the frontend features and have a working prototype of the task management web application within the given timeline.

Fig. 4.4. Frontend Approval Output (Scenario 2)

In the generated output, we observe a detailed breakdown of task assignments for each page and component, outlining the necessary steps for developing the features essential for a minimum viable prototype. Additionally, the allocation of tasks is determined based on the specified duration. This description for a project that has tight deadlines and needs a working prototype faster, helps to reduce the time spent on brainstorming and start the development quickly.

Upon examination of the problem statement and the user-provided data, it is evident that both the frontend and backend have received approval following analysis by the model. The relevance and appropriateness of the problem statement and input data contribute to this positive outcome.

Let's have a look at the Backend Approval Output as well:

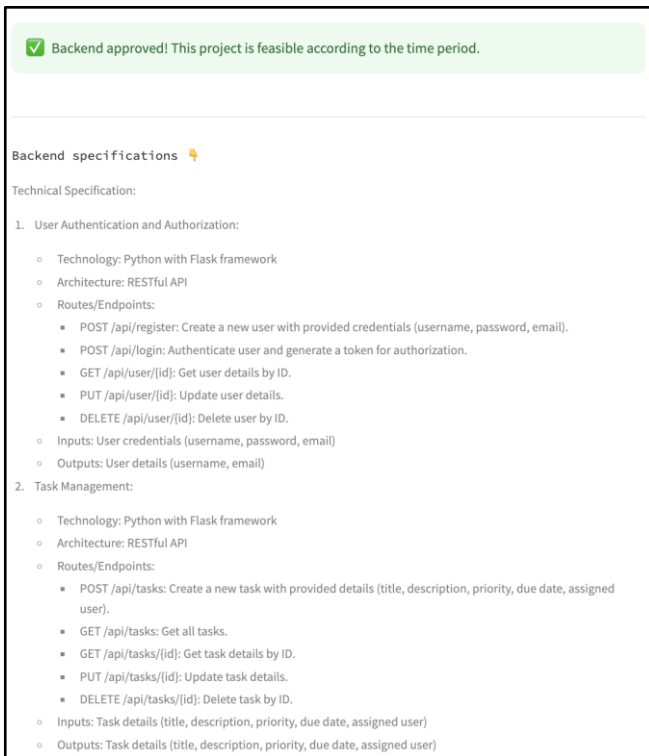


Fig. 5.1. Backend Approval Output (Scenario 2)

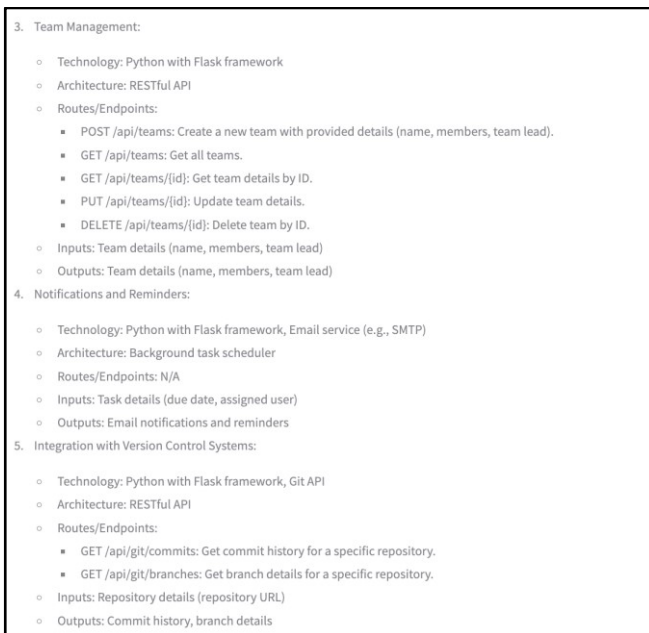


Fig. 5.2. Backend Approval Output (Scenario 2)

On approval from the backend chain, we get the minimum functionalities that need to be implemented for a working prototype.

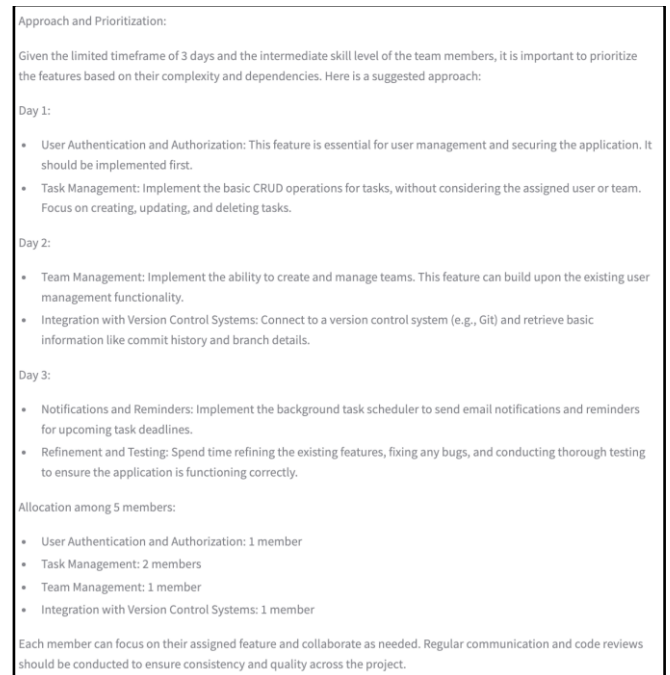


Fig. 5.2. Backend Approval Output (Scenario 2)

The provided output outlines the method and prioritization of task assignment. While task allocation is determined by both duration and team size, the distribution based on team members takes into account the group skill input, where the skill level is assumed to be consistent for each individual within the team.

The alternate scenario arises when the user opts for the "Individual" project type. In this case, the user initiates the process by uploading their name and resume. Subsequently, an analysis of the resume is conducted, following which the previously mentioned workflow unfolds for the user.

The User Interface for this option is depicted as follows:

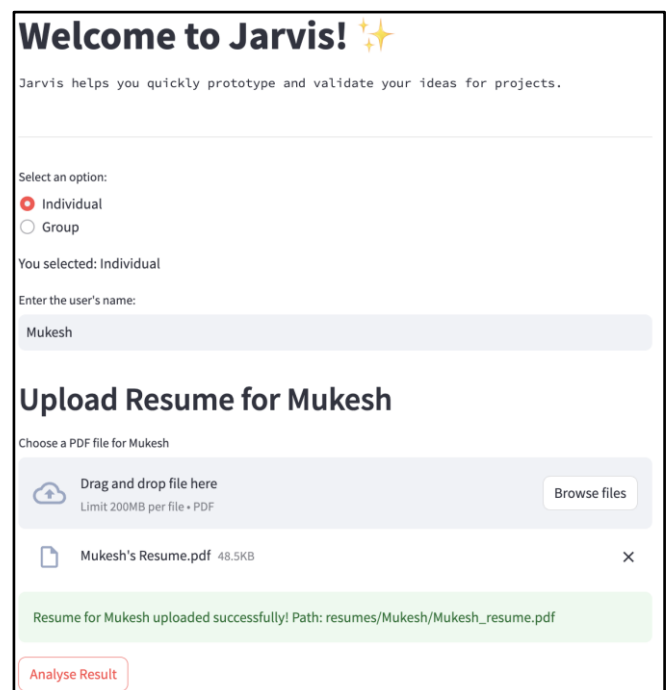


Fig. 6. User Interface for Individual

Embeddings, vectors, and PDF chunks are used to efficiently analyze and process resumes for candidate evaluation.

Embeddings: Embeddings are a powerful tool for representing text as vectors of numbers. This allows the code to compare and contrast resumes by comparing their vector representations. Embeddings are also used to generate summaries of the resumes and to analyze the technical skills and work experience of each resume.

Vectors: Vectors are a way of representing text as a sequence of numbers. This makes it easier for the code to process the text and to extract information from it. Vectors are also used to represent the vector representations of the resumes.

PDF chunks: PDF chunks are used to break the text of the resumes into smaller pieces. This makes it easier for the code to process the text and to extract information from it. PDF chunks are also used to prevent the GPT-3.5-turbo language model from being overwhelmed by the large size of the resumes.

After analysis of the resume summary of the user, which contains the technical skills and work experience is passed to backend and frontend chain for processing based on the prompt templates specific to this scenario.

The analysis is stored in the backend in JSON format so that it can be referenced further by the chains:

```
{  
  "name": name,  
  "summary": technical skills & work experience  
}
```

The User Interface displays a message after successful analysis:

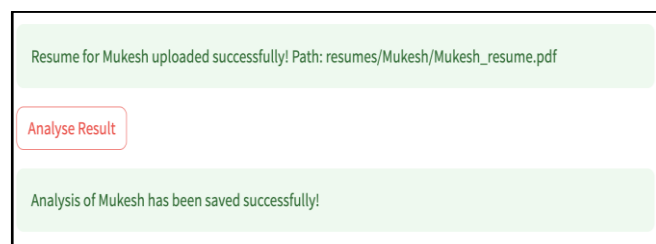


Fig. 7. Resume Analysis Completion

The inputs other than those mentioned previously remain unchanged. However, a slight adjustment has been made in the prompt template to consider the resume summary, replacing the group experience, given the individual nature of the effort or project. The resulting output also remains the same but is now derived from the analysis of the user's resume. Additional details in the output, compared to the previous version, are illustrated in Fig. 8.

Based on the user's technical skills and experience, they have a strong proficiency in Python, JavaScript, and React JS, which are the main technologies used in this project. They are also familiar with Django and Flask, which are the web frameworks utilized. Their experience in developing web applications using Django Web Framework aligns well with the chosen architecture. Additionally, their familiarity with speech recognition and SMTP libraries can be beneficial for implementing collaboration and communication features.

To prioritize and allocate these features within the given 5 days, it is recommended to follow an iterative development approach. Start by implementing the user authentication and authorization system, as it is a fundamental requirement for the application. Then, focus on the task management and project management features, as they form the core functionalities of the application. Next, implement the collaboration and communication features to facilitate team coordination. After that, integrate with version control systems to enhance the development workflow. Finally, develop the reporting and analytics functionality to provide insights into task and project progress. Throughout the development process, ensure scalability, performance, data storage, and security measures are implemented.

Fig 8. Additional Output for Individual Analysis

In case of rejection, the output differs as well, the changes are depicted in the below figure:

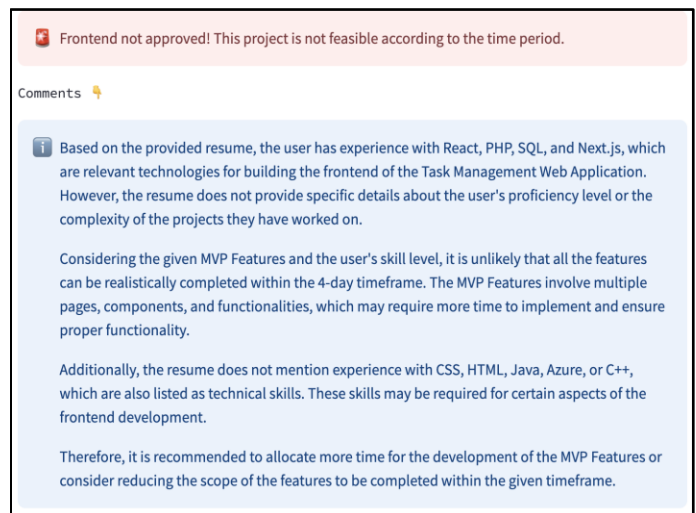


Fig. 9. Output for Rejection Scenario

V. CONCLUSION

As a result, this study obtained a minimum of five essential features from both the frontend and backend perspectives, crucial for rapidly prototyping or initializing the first iteration of a project. This paper documents our experimentation and exploration of LangChain's capabilities in addressing the challenges of project validation and prototyping within stringent time constraints. Prompt patterns are analogous to software patterns and aim to provide reusable solutions to problems that users face when interacting with LLMs to perform a wide range of tasks. Prompt patterns significantly enrich the capabilities that can be created in a conversational LLM. The presented website serves as an initial step toward automating the brainstorming and prototyping process to some degree. Although human knowledge and intelligence remain indispensable, utilizing this tool expedites the process and incorporates additional insights. This platform, or tool, empowers students or developers to swiftly prototype,

validate, and allocate tasks, offering a valuable resource, particularly in time-sensitive scenarios like hackathons.

VI. FUTURE WORK

Future efforts will focus on refining the prompt templates for enhanced accuracy and establishing appropriate evaluation criteria. Currently, our reliance on human knowledge is pivotal for determining the adequacy of results, despite the extensive training of models on large datasets. Recognizing the existence of potential edge cases that may be overlooked, addressing these scenarios is a priority for future iterations.

The frontend and backend chains operate independently, devoid of awareness regarding each other's existence. A forthcoming challenge involves devising a solution to synchronize their operations. Despite being in its initial stage concerning chains, templates, and suitable inputs, there is a necessity to elevate the complexity of the tool. Numerous available APIs could be seamlessly integrated to automate additional processes, thereby aiding developers in saving valuable time.

VII. REFERENCES

- [1] S. Conger, "Software development life cycles and methodologies: Fixing the old and adopting the new", 2010. doi: 10.4018/jitsa.2011010.
- [2] OpenAI. (2023). "gpt-3-5" [Online]. Available: <https://platform.openai.com/docs/models/gpt-3-5> (November 1, 2023).
- [3] Khorasani, M.; Abdou, M.; Hernández Fernández, J. Streamlit at Work. In Web Application Development with Streamlit: Develop and Deploy Secure and Scalable Web Applications to the Cloud Using a Pure Python Framework; Apress, 2022.
- [4] Application of Large Language Models to Software Engineering Tasks: Opportunities, Risks, and Implications; pp. 4-5. doi: 10.1109/MS.2023.3248401.
- [5] Topsakal, O.; Akinci, T.C. Creating Large Language Model Applications Utilizing LangChain: A Primer on Developing LLM Apps Fast. In Proceedings of the International Conference on Applied Engineering and Natural Sciences, Konya, Turkey, 10–12 July 2023; Volume 1, pp. 1050–1056. doi: 10.59287/icaens.1127.
- [6] K. Pandya and M. Holia, "Automating customer service using LangChain: Building custom open-source gpt chatbot for organizations," 2023. doi: 10.48550/arXiv.2310.05421.
- [7] Comparative Study and Framework for Automated Summariser Evaluation: LangChain and Hybrid Algorithms. doi: 10.48550/arXiv.2310.02759.
- [8] Chat2VIS: Generating Data Visualizations via Natural Language Using ChatGPT, Codex and GPT-3 Large Language Models. doi: 10.1109/ACCESS.2023.3274199.
- [9] LangChain. "Output Parsers - Python Documentation."-Langchain. Available: https://python.langchain.com/docs/modules/model_io/output_parsers. (November 2, 2023).
- [10] LangChain. "Prompt Templates - Python Documentation."-Langchain. Available: https://python.langchain.com/docs/modules/model_io/prompts/prompt_templates. (November 2, 2023).