**Documentation of Work**

**July 10**

**Project Setup and Initial Research**

1. **Installed Necessary Tools and Libraries:**
   * **Python and Flask:**
     + Installed Python and verified installation.
     + Created and activated a virtual environment.
     + Installed Flask and verified installation.
   * **Node.js, npm, React.js, and Tailwind CSS:**
     + Installed Node.js and npm and verified installation.
     + Created a new React app using Create React App.
     + Installed Tailwind CSS and configured it for use in the React app.
   * **Hugging Face's Transformers:**
     + Installed the transformers library and verified installation.
   * **Rasa:**
     + Installed Rasa and verified installation.
   * **Git and GitHub:**
     + Installed Git and verified installation.
     + Set up a GitHub repository and cloned it locally.
2. **Initial Research:**
   * **Hugging Face's Transformers:**
     + Explored the documentation and tried out basic examples to understand usage.
   * **Flask Basics:**
     + Read through the Flask Quickstart guide and created a simple "Hello World" Flask app.
   * **React.js and Tailwind CSS:**
     + Followed React Getting Started Guide and Tailwind CSS Guide.
     + Created basic React components and styled them using Tailwind CSS.

**Rasa Configuration**

1. **Initialized Rasa Project:**
   * Set up a new Rasa project using rasa init.
   * Configured config.yml for NLU pipeline and policies.
   * Configured domain.yml to define intents, entities, slots, responses, and actions.
   * Created nlu.yml with training examples for intents.
   * Defined conversation stories in stories.yml.
   * Created custom action in actions.py and updated endpoints.yml for action server.
2. **Running Rasa:**
   * Trained the Rasa model using rasa train.
   * Ran the Rasa server and action server.
   * Tested the assistant in the shell using rasa shell.

**Summary:**

* Successfully set up the development environment with necessary tools and libraries.
* Gained a basic understanding of core technologies through initial research.
* Configured and tested a basic Rasa assistant for handling simple interactions.

**July 15**

**Tasks Completed:**

1. **Environment Setup:**
   * Activated the virtual environment and verified all necessary tools and libraries were installed, including Flask and Hugging Face's transformers.
   * Installed PyTorch to ensure compatibility with Hugging Face's transformers.
2. **NLP Integration:**
   * Updated the NLP integration to use the text-generation task instead of conversational.
   * Implemented the generate\_response function in nlp.py using the GPT-2 model for text generation.
3. **Flask Backend Configuration:**
   * Defined the /process endpoint in routes.py to handle POST requests and generate responses using the NLP model.
   * Ensured the endpoint returns JSON responses correctly using jsonify.
4. **Testing:**
   * Manually tested the endpoint using curl to ensure it returns the expected JSON responses.
   * Created and ran a test script (test\_nlp.py) to automate testing of the endpoint.
5. **Debugging and Troubleshooting:**
   * Encountered and resolved issues with JSON responses being empty by ensuring proper response formatting and logging.
   * Added detailed logging to routes.py to track input and output during request handling.

**Week3**

**Tasks Completed:**

**1. Environment and Dependency Setup:**

* Ensured the virtual environment was activated and installed the necessary libraries including transformers, huggingface\_hub, and torch.
* Addressed missing CLI commands and ensured proper installation and setup of the Hugging Face CLI tools.

**2. Flask Application Configuration:**

* Verified and updated the \_\_init\_\_.py to ensure the Flask app and CORS were initialized correctly.
* Updated routes.py to define the /api/generate endpoint for text generation using the Hugging Face GPT-2 model.
* Created and verified run.py to ensure the Flask app runs on port 5001 to avoid port conflicts.

**3. Debugging and Testing:**

* Addressed and resolved the "Address already in use" error by changing the Flask server port.
* Successfully tested the Flask endpoint using curl to ensure it returns the expected JSON responses.

**4. Integration with Hugging Face:**

* Configured nlp.py to authenticate with Hugging Face and initialize the GPT-2 model for text generation.
* Verified that the Hugging Face model works correctly and returns appropriate responses.

**5. Frontend-Backend Integration:**

* Ensured the frontend correctly integrates with the Flask backend by updating the API service and testing the complete workflow from frontend input to backend processing and response.

**6. Final Testing:**

* Conducted thorough testing of the Flask backend and the /api/generate endpoint using curl to ensure functionality and correctness.

**Objective:**  
Integrate a kitchen helper functionality into the Drift AI system, enabling users to search for recipes, find ingredient substitutions, and generate responses using natural language processing.

**Steps and Work Done**

**1. Integration of RecipeNLG Dataset**

**1.1 Downloading and Loading the Dataset:**

* The RecipeNLG dataset was downloaded in CSV format and stored in the backend/data/ directory.
* A new script, recipe\_nlg.py, was created to load and query the dataset using the Pandas library.

**Code:**

python

Copy code

import pandas as pd

def load\_recipenlg\_dataset(file\_path):

dataset = pd.read\_csv(file\_path)

return dataset

def search\_recipe\_by\_ingredient(dataset, ingredient):

results = dataset[dataset['ingredients'].str.contains(ingredient, case=False, na=False)]

return results

def search\_recipe\_by\_dish(dataset, dish):

results = dataset[dataset['title'].str.contains(dish, case=False, na=False)]

return results

**2. Backend Updates**

**2.1 Updating recipes.py:**

* recipes.py was updated to include functions for searching recipes by ingredient and dish name using the RecipeNLG dataset.
* The dataset is loaded once when the application starts to optimize performance.

**Code:**

python

Copy code

from .recipe\_nlg import load\_recipenlg\_dataset, search\_recipe\_by\_ingredient, search\_recipe\_by\_dish

dataset = load\_recipenlg\_dataset('backend/data/recipenlg.csv')

def get\_recipe\_by\_ingredient(ingredient):

recipes = search\_recipe\_by\_ingredient(dataset, ingredient)

return recipes.to\_dict(orient='records')

def get\_recipe\_by\_dish(dish):

recipes = search\_recipe\_by\_dish(dataset, dish)

return recipes.to\_dict(orient='records')

**2.2 Updating routes.py:**

* New endpoints /get\_recipe and /get\_substitution were added to handle recipe and substitution queries.
* These endpoints return JSON responses based on the queried data.

**Code:**

python

Copy code

@main.route('/get\_recipe', methods=['GET'])

def get\_recipe():

ingredient = request.args.get('ingredient')

dish = request.args.get('dish')

if ingredient:

recipes = get\_recipe\_by\_ingredient(ingredient)

elif dish:

recipes = get\_recipe\_by\_dish(dish)

else:

return jsonify({'error': 'Please provide an ingredient or dish'}), 400

if recipes:

return jsonify({'recipes': recipes}), 200

else:

return jsonify({'error': 'No recipes found'}), 404

**3. Testing and Optimization**

**3.1 Functional Testing:**

* Tested the /get\_recipe endpoint with various ingredients and dish names.
* Verified that the /get\_substitution endpoint returned correct ingredient substitutions.

**3.2 Performance Optimization:**

* Implemented Flask-Caching to cache frequent queries, reducing response times.

**Code:**

python

Copy code

from flask\_caching import Cache

cache = Cache(config={'CACHE\_TYPE': 'simple'})

cache.init\_app(app)

@cache.memoize(60)

def get\_recipe\_by\_ingredient(ingredient):

recipes = search\_recipe\_by\_ingredient(dataset, ingredient)

return recipes.to\_dict(orient='records')

**4. Deployment Preparation**

**4.1 Environment Setup:**

* Ensured all environment variables, such as API keys and dataset paths, were correctly configured.

**4.2 Docker Deployment:**

* Created a Dockerfile to containerize the application, simplifying deployment to various environments.

**Dockerfile:**

Dockerfile

Copy code

FROM python:3.9-slim

WORKDIR /app

COPY . /app

RUN pip install --no-cache-dir -r requirements.txt

EXPOSE 5001

CMD ["flask", "run", "--host=0.0.0.0", "--port=5001"]

**4.3 Final Testing:**

* Conducted final tests in the production environment to ensure all functionalities work as expected.

**5. Documentation and Final Steps**

**5.1 API Documentation:**

* Documented all API endpoints, including /generate, /get\_recipe, and /get\_substitution, detailing their usage, inputs, and outputs.

**5.2 Developer Documentation:**

* Provided instructions on setting up the development environment, loading the dataset, and deploying the application.

**5.3 Final Review:**

* Reviewed the entire system for any issues, ensuring that all components work together seamlessly.