A picture containing text, outdoor, building, white

Description automatically generated

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Student name: | Student 1: Karen Ferreira Magalhaes  Student 2: Thales Campos  Student 3: Vitor Freitas | | | | | |
| Student number: | Student 1: 3146094  Student 2: 3151261  Student 3: 3152612 | | | | | |
| Faculty: | Computing Science | | | | | |
| Course: | BSCH/BSCO/EXCH | | | Stage/year: | 2 | |
| Subject: | Software Development 2 | | | | | |
| Study Mode: | Full time | Icon  Description automatically generated |  | Part-time |  |  |
| Lecturer Name: | Haseeb Younis/ Muhammad Shoaib | | | | | |
| Assignment Title: | Prototype Documentation | | | | | |
| Date due: | 27/04/2025 | | |  | | |
| Date submitted: | 23/03/2025 | | |  | | |
| **Plagiarism disclaimer:**  *I understand that plagiarism is a serious offence and have read and understood the college policy on plagiarism. I also understand that I may receive a mark of zero if I have not identified and properly attributed sources which have been used, referred to, or have in any way influenced the preparation of this assignment, or if I have knowingly allowed others to plagiarise my work in this way.*  *I hereby certify that this assignment is my own work, based on my personal study and/or research, and that I have acknowledged all material and sources used in its preparation. I also certify that the assignment has not previously been submitted for assessment and that I have not copied in part or whole or otherwise plagiarised the work of anyone else, including other students.*  **Signed: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** | | | | | | |
| **Please note:** Students **MUST** retain a hard / soft copy of **ALL** assignments as well as a receipt issued and signed by a member of Faculty as proof of submission. | | | | | | |

**Documentation – Milestone 01**

**ChatGPTClient Class Documentation**

Overview: The ChatGPTClient class handles communication with the OpenAI ChatGPT API. It sends user messages, receives AI-generated responses, and processes the API response using the Gson library.

Package: main.java.com.tripper

Dependencies:

* java.io.\* (For input/output operations)
* java.net.\* (For handling HTTP connections)
* com.google.gson.\* (For JSON processing)

Constants:

* String API\_URL: The endpoint URL for OpenAI's ChatGPT API.
* String API\_KEY: The API key used for authentication (should be kept secret and not hardcoded in production).

Methods:

1. String getChatResponse(String conversationContext):
   * Sends a user message to the OpenAI ChatGPT API and retrieves the AI-generated response.
   * Parameters:
     + conversationContext: The user's input message to be processed by the AI.
   * Returns:
     + A string containing the AI-generated response.
   * Process:
     + Establishes an HTTP connection with the OpenAI API.
     + Constructs a JSON payload containing the request data.
     + Sends the request and reads the API response.
     + Parses the JSON response to extract the AI's reply.
     + Returns the processed response or an error message if the API call fails.

Exception Handling:

* Catches general exceptions during the API request and response handling.
* Prints stack traces for debugging and returns an error message when an issue occurs.

Security Considerations:

* The API key should not be stored in the source code; it should be loaded from environment variables or secure storage in production.
* Ensure proper exception handling to avoid exposing sensitive information in error messages.

**ConversationController Class Documentation**

Overview: The ConversationController class manages the chatbot's interaction flow, handling user input, processing responses, and guiding the conversation through predefined states.

Package: main.java.com.tripper

Dependencies:

* java.util.Scanner (For user input handling)

Attributes:

* ConversationState state: Stores conversation progress and user data.
* ConversationManager conversationManager: Manages chatbot responses.
* ChatGPTClient chatGPTClient: Communicates with OpenAI's API.
* Scanner scanner: Handles user input from the console.

Methods:

1. void run():
   * Manages the chatbot's state-based conversation flow.
   * Guides the user through different states: GREETING, COLLECT\_TRIP\_DETAILS, CONFIRM\_DETAILS, GENERATE\_RECOMMENDATIONS, OFFER\_PDF, END.
   * Integrates responses from ChatGPTClient.
   * Handles user inputs for trip details and PDF generation.
2. void typePrint(String message, int delayMs):
   * Simulates a typing effect when displaying chatbot messages.
   * Parameters:
     + message: The text to display.
     + delayMs: Delay per character in milliseconds.
3. String prompt(String message):
   * Displays a message and collects user input.
   * Parameters:
     + message: The prompt for the user.
   * Returns:
     + User input as a trimmed string.

Exception Handling:

* Ensures smooth conversation flow by handling user input errors.
* Uses try-catch to handle interruptions in the typing effect.

Functionality:

* Starts the chatbot with a greeting message.
* Collects and processes user trip details.
* Requests recommendations from ChatGPTClient.
* Offers an option to generate a PDF checklist.
* Ends the conversation with a farewell message.

**ConversationManager Class Documentation**

Overview: The ConversationManager class handles the chatbot's predefined responses to user input. It generates friendly and engaging messages to guide users through the interaction process.

Package: main.java.com.tripper

Methods:

1. String getGreeting(String userName):
   * Generates a personalized greeting for the user.
   * Parameters:
     + userName: The name of the user.
   * Returns:
     + A friendly welcome message including the user's name.
2. String askForTripDetails():
   * Prompts the user to provide details about their trip.
   * Returns:
     + A request message asking the user for trip information.
3. String friendlyResponse(String dynamicResponse):
   * Formats and returns a friendly response incorporating dynamically generated recommendations.
   * Parameters:
     + dynamicResponse: The AI-generated travel recommendations.
   * Returns:
     + A structured response including the recommendations.

Functionality:

* Ensures a smooth and engaging chatbot experience.
* Provides user-friendly prompts and structured responses to enhance interaction.
* Acts as an intermediary between user input and AI-generated recommendations.

**ConversationState Class Documentation**

Overview: The ConversationState class maintains the state of a chatbot conversation. It stores user-specific data such as name, trip details, and confirmation status to ensure a smooth interaction flow.

Package: main.java.com.tripper

Dependencies:

* java.util.ArrayList (For handling dynamic lists)
* java.util.List (For managing location storage)

Attributes:

* String userName: Stores the user's name.
* List locations: Holds the list of travel locations provided by the user.
* String tripDetails: Contains user-supplied details about the trip.
* boolean detailsConfirmed: Indicates whether the trip details have been confirmed by the user.

Methods:

1. ConversationState():
   * Constructor that initializes the locations list and sets detailsConfirmed to false.
2. String getUserName():
   * Retrieves the user's name.
   * Returns:
     + The name of the user.
3. void setUserName(String userName):
   * Sets the user's name.
   * Parameters:
     + userName: The name of the user.
4. List getLocations():
   * Retrieves the list of locations.
   * Returns:
     + A list of location names.
5. void addLocation(String location):
   * Adds a new location to the list.
   * Parameters:
     + location: The name of the location to add.
6. String getTripDetails():
   * Retrieves the trip details provided by the user.
   * Returns:
     + The trip details as a string.
7. void setTripDetails(String tripDetails):
   * Sets the trip details.
   * Parameters:
     + tripDetails: A string containing trip information.
8. boolean isDetailsConfirmed():
   * Checks if the trip details have been confirmed by the user.
   * Returns:
     + True if confirmed, false otherwise.
9. void setDetailsConfirmed(boolean detailsConfirmed):
   * Updates the confirmation status of the trip details.
   * Parameters:
     + detailsConfirmed: Boolean value indicating confirmation status.

Functionality:

* Tracks user input and progress throughout the chatbot conversation.
* Stores key trip-related information for personalized recommendations.
* Ensures state persistence for an improved chatbot experience.

**InputParser Class Documentation**

Overview:  
The InputParser class processes and extracts travel-related information from user input. It uses regular expressions and string manipulation to identify the travel month and potential locations provided by the user.

Package:  
main.java.com.tripper

Dependencies:

* java.util.\* (For handling collections and arrays)
* java.util.regex.\* (For regular expression matching)

Constants:

* Pattern monthPattern:  
  A regular expression pattern to match any month name (case-insensitive).

Methods:

1. TripDetails parseTripDetails(String input):
   * Parameters:
     + input: The user-provided string containing trip details.
   * Returns:
     + A TripDetails object containing the extracted travel month and locations.
   * Process:
     + Uses regex to identify a month in the input and sets the travel month in the TripDetails object.
     + Splits the input into tokens and identifies capitalized words (excluding month names) as potential locations.
     + Removes punctuation from the tokens and adds valid locations to the list.

Functionality:

* Identifies and extracts the travel month and location information from a free-form user input string.
* Uses regular expressions for month matching and simple string checks for locations.
* Ensures proper location parsing even if months and locations are mixed in the input.

**NLPInputParser Class Documentation**

Overview:

The NLPInputParser class is responsible for processing input text using natural language processing (NLP) techniques. It uses OpenNLP tools to perform sentence detection, tokenization, part-of-speech (POS) tagging, and optional lemmatization. The primary goal is to parse trip details, such as locations and travel dates, from user input.

Dependencies:

* OpenNLP Library (tools for sentence detection, tokenization, POS tagging, and lemmatization).

Attributes:

* SentenceDetectorME sentenceDetector: Detects sentence boundaries in the input text.
* TokenizerME tokenizer: Tokenizes the text into individual words.
* POSTaggerME posTagger: Tags each token with its part-of-speech.
* DictionaryLemmatizer lemmatizer (optional): Lemmatizes tokens to their base forms.

Constructor:

* Initializes models for sentence detection, tokenization, POS tagging, and lemmatization (if available).
* Loads model files from the local file system for OpenNLP tools.

Methods:

1. parseTripDetails(String input):
   * Parameters: String input – The input text containing trip details.
   * Returns: TripDetails – A TripDetails object containing locations and the travel month parsed from the input.
   * Description:
     + Detects sentences in the input.
     + Tokenizes sentences into words.
     + Performs POS tagging on the tokens.
     + Identifies proper nouns (NNP/NNPS tags) as potential location names.
     + Identifies date-like tokens (e.g., "12/12/2025" or month names).
     + Collects and returns the locations and travel month as part of the TripDetails object.

Security Considerations:

* Ensure the model files are securely stored and not exposed to unauthorized access.
* Handle any exceptions that might occur during model loading or processing.