UNIVERSITETI I EVROPËS JUGLINDORE
УНИВЕРЗИТЕТ НА ЈУГОИСТОЧНА ЕВРОПА
SOUTH EAST EUROPEAN UNIVERSITY

Faculty of Contemporary Sciences and Technologies
Tetovo

Software Engineering

(Sem. 5, 2023/2024)

Final project:

Queryhelper

Students:
1. Ron Ismaili - 130050 - ri30050@seeu.edu.mk
2. Fehmi Havziu - 130063 - fh30063@seeu.edu.mk

Mentor:
Prof. Nuhi Besimi

January 2024, Tetovo

Table of contents

Introduction

The primary purpose of this project is to perform research on the capabilities of the ChatGPT 3.5 llm model in generating accurate SQL queries when given a prompt in natural language, alongside the Data Definition Language (DDL) of the database.
We have expressed our interest in experimenting with AI and after discussing our project idea with Prof. Nuhi, we got the green light to continue with our research project. We have made some minor adjustments in this documentation to better reflect our current project's state.

Student work

When it comes to the division of labor, we have divided the tasks as follows:

Ron Ismaili *(ri30050)*:

- Project outline
- User stories
- Class diagram
- Use case diagram
- Unit test documentation
- General documentation

Fehmi Havziu *(fh30063)*:

- Class diagram
- Use case diagram
- Software architecture
- Implementation of unit tests
- **BONUS:** Dockerizing the application
- **BONUS:** CI/CD pipeline

Implementation

To view the implementation of our project please feel free to visit our GitHub repository, all of our work and git activity can be found there. If you require access to the repository, please do not hesitate to contact us for assistance.

Project outline

Overview
Our idea is to utilize artificial intelligence to generate SQL queries for a specific dataset, in our case, for a library system. Queryhelper will take the user input, which will be given in natural language, and generate its SQL counterpart.

Customer
Our customer is a fictional library that wants to enhance their data analytics capabilities. They have contacted us to create a software that enables their staff to interact with their database in natural language, simplifying basic data analysis.

Product
The product we developed is an API that users can use to perform rudimentary data analytics using natural language to query the library systems database.

Users
Our primary end users come from a non-technical background, therefore we aimed to develop an API that not only works well but also offers usability and user-friendliness.

Management
We practiced agile methodologies for managing our software development process. Additionally we used Trello as our platform for tracking tasks and overall project progress/management.

Communication
Since this is a university project between two colleagues, most of the communication was done through face-to-face meetings.

Breakdown
Our system can be broken down into the following core components:
AI controller, AI service, SQL service and database.

AI controller:
- Endpoint in our AI controller that expects a request from the user.
- Utilizes the AI service in our AI controller to generate a response.
- Return the generated response back to the user.

AI service:
- The service takes the user's request and generates the appropriate prompt.
- The prompt gets sent to ChatGPT through langchain’s library.

- We receive the SQL query from ChatGPT and convert the JSON response into a string.
- We pass the SQL query to the SQL service so that it can be run against the database.
- We receive the records from the SQL service, create a response and return it.

SQL service:
- The service takes the query and runs it against the database.
- Once the execution finishes, the result is returned, using our RowMapper.

Database:
- The database schema is a basic library with 5 tables.
- We added some relationships and constraints on the tables.
- We filled the database with correct data.

Effort estimation

This project took us roughly 20-30 hours of collaborative effort to complete. This time included doing research, experimenting, developing, testing and consulting with professor Nuhi.

Technological stack

The back-end of Queryhelper is fully written in JAVA using the Spring Boot framework, the AI library langchain and Google's Gson library which we used for JSON serialization. To test our API we utilized Postman.

Reporting

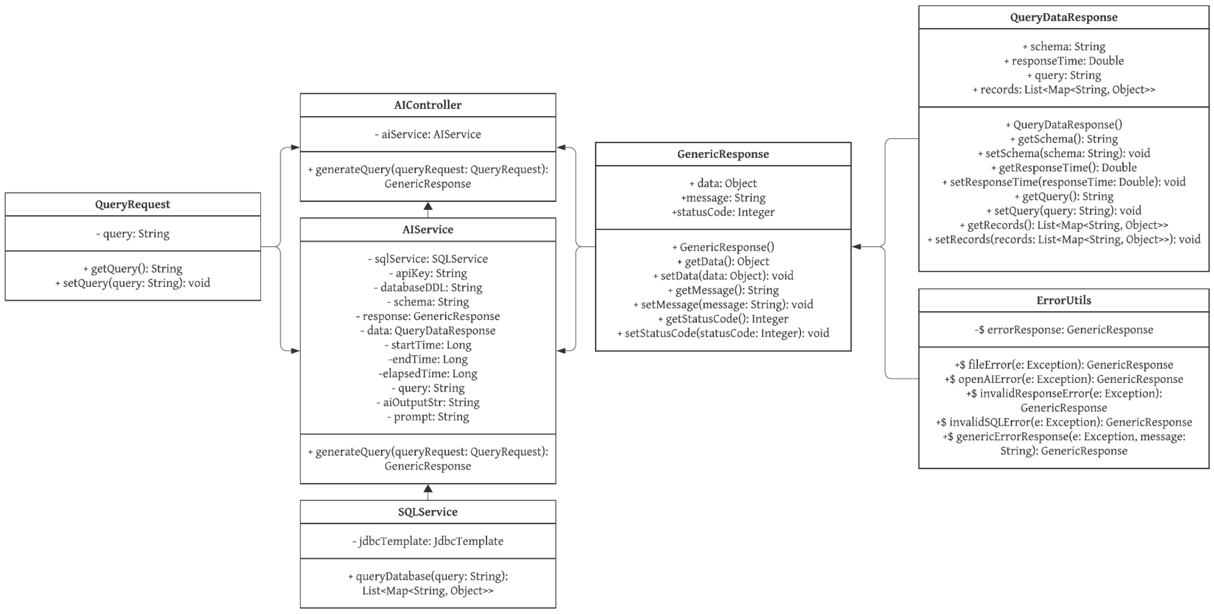
We have prepared a practical demonstration of our project for this course.

User stories

Queryhelper should make the day-to-day operations of our library clerks easier by having an API with which our clerks can communicate. Clerks should be able to send requests to the API where they specify, in plain english, what data they need from the database. The API should be able to retrieve this request, process it through the use of Artificial Intelligence, and send a response back to the clerk with all of the requested records. Queryhelper shall be sophisticated in its handling of errors. It must have the ability to detect missing API keys, if the AI service is down, if the generated query is valid, etc. If there is any issue, it should notify the user of what the issue is by responding with a message and code.

This section was written from the client's point of view

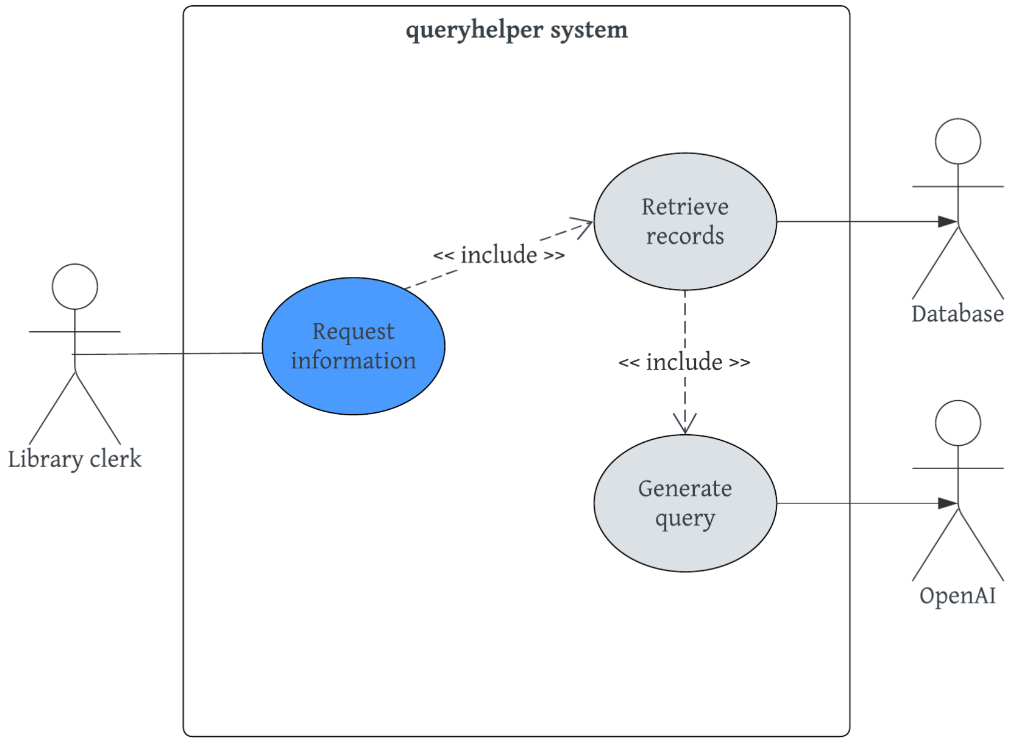
Class diagram


QueryDataResponse
+ schema: String
+ responseTime: Double
+ query: String
+ records: List<Map<String, Object>>
+ QueryDataResponse()
+ getSchema(): String
+ setSchema(schema: String): void
+ getResponseTime(): Double
+ setResponseTime(responseTime: Double): void
+ getQuery(): String
+ setQuery(query: String): void
+ getRecords(): List<Map<String, Object>>
+ setRecords(records: List<Map<String, Object>>): void
AIController
- aiService: AIService
+ generateQuery(queryRequest: QueryRequest): GenericResponse
GenericResponse
+ data: Object
+message: String
+statusCode: Integer
+ GenericResponse()
+ getData(): Object
+ setData(data: Object): void
+ getMessage(): String
+ setMessage(message: String): void
+ getStatusCode(): Integer
+ setStatusCode(statusCode: Integer): void
QueryRequest
- query: String
+ getQuery(): String
+ setQuery(query: String): void
AIService
- sqlService: SQLService
- apiKey: String
- databaseDDL: String
- schema: String
- response: GenericResponse
- data: QueryDataResponse
- startTime: Long
-endTime: Long
-elapsedTime: Long
- query: String
- aiOutputStr: String
- prompt: String
+ generateQuery(queryRequest: QueryRequest): GenericResponse
ErrorUtils
-$ errorResponse: GenericResponse
+$ fileError(e: Exception): GenericResponse
+$ openAIError(e: Exception): GenericResponse
+$ invalidResponseError(e: Exception): GenericResponse
+$ invalidSQLError(e: Exception): GenericResponse
+$ genericErrorResponse(e: Exception, message: String): GenericResponse
SQLService
- jdbcTemplate: JdbcTemplate
+ queryDatabase(query: String): List<Map<String, Object>>

Zoom in to get a better look at the details

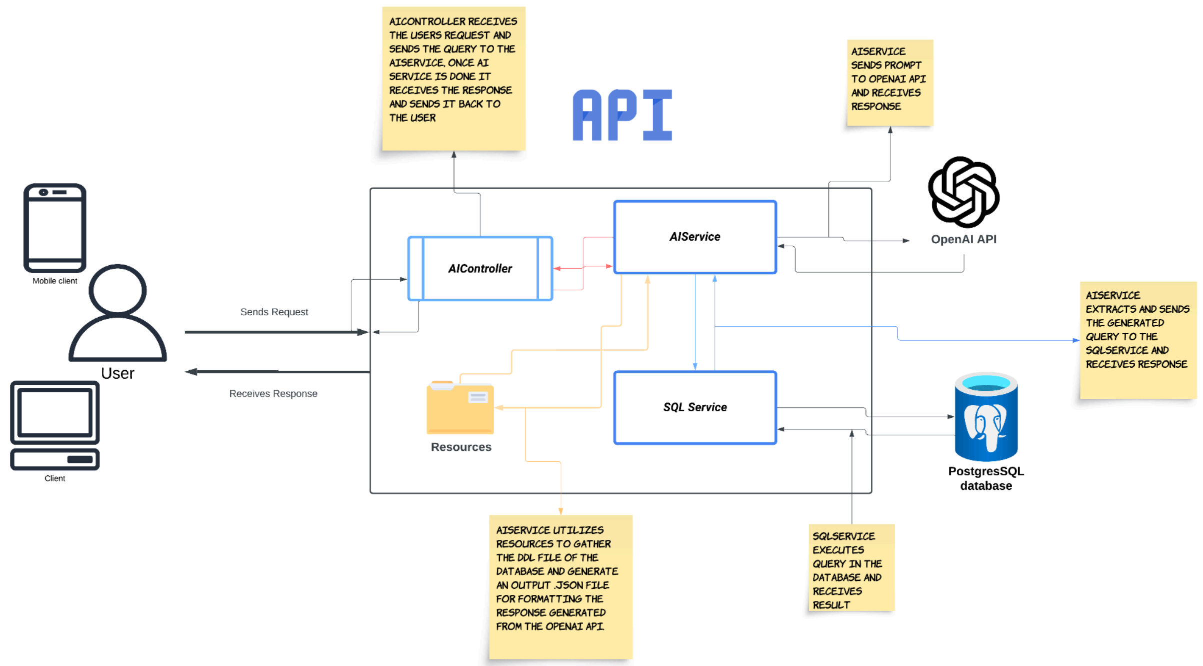
]

Use case diagram



Zoom in to get a better look at the details

Software architecture


AICONTROLLER RECEIVES THE USERS REQUEST AND SENDS THE QUERY TO THE AISERVICE. ONCE AI SERVICE IS DONE IT RECEIVES THE RESPONSE AND SENDS IT BACK TO THE USER
API
AISERVICE SENDS PROMPT TO OPENAI API AND RECEIVES RESPONSE
Mobile client
User
Client
Sends Request
Receives Response
AIController
AIService
OpenAI API
AISERVICE EXTRACTS AND SENDS THE GENERATED QUERY TO THE SQLSERVICE AND RECEIVES RESPONSE
Resources
SQL Service
PostgresSQL database
AISERVICE UTILIZES RESOURCES TO GATHER THE DDL FILE OF THE DATABASE AND GENERATE AN OUTPUT .JSON FILE FOR FORMATTING THE RESPONSE GENERATED FROM THE OPENAI API
SQLSERVICE EXECUTES QUERY IN THE DATABASE AND RECEIVES RESULT

Zoom in to get a better look at the details

Unit testing

After discussing what functionalities we should test we concluded that the following unit tests are the most important:

AIService

- testGenerateQueryFileError - We test if the error handling for file reading works.
- testGenerateQueryOpenAIError - We test if the error handling for the communication with OpenAI works.
- testGenerateQueryInvalidResponseError - We test if the error handling for response conversion works.
- testGenerateQueryInvalidSQLError - We test if the error handling for invalid SQL queries works.

SQLService

- testQueryDatabase_Success - We test if a valid SQL query succeeds.
- testQueryDatabase_Failure - We test if an invalid SQL query fails.

We did not test generating a query because we are dealing with AI and its response can be quite unpredictable at times, therefore unreasonable to test (this is our opinion after consulting with Prof. Nuhi).