`

Department of Information Technology
Faculty of Liberal Arts and Professional Studies

ITEC 4020 B – Fall 2023-2024
Internet Client-Server Systems
Professor: • Amran Bhuiyan

Tayyaba Riasat	219326826

Assignment
Due: Friday, November 17

Evaluation of ChatGPT Efficiency using MongoDB in an Internet Client Server Model

ChatGPT Efficiency Evaluation Report

Project Overview

Objective

The project aimed to evaluate the capabilities and efficiency of the ChatGPT model in answering queries from different domains (History, Social Science, and Computer Security) using a MongoDB-backed Node.js server-client system.

Tools & Technologies

- **Node.js (with Express.js for server-side operations)**

  The application is a web service built with Node.js and Express. A matrix calculation is performed to evaluate the overall accuracy and response time.

- **MongoDB**

  MongoDB is used as the database to store questions and their details.

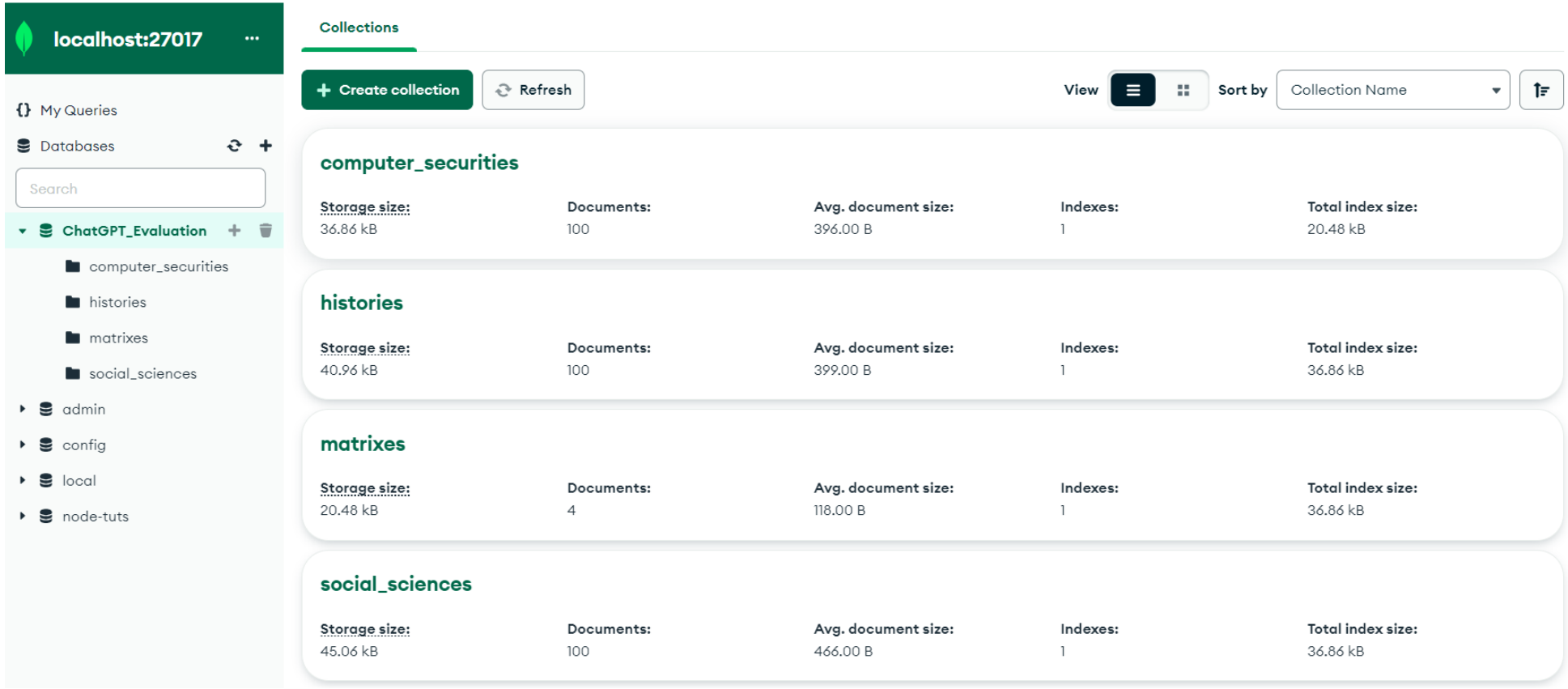
- **ChatGPT API**

  The OpenAI API is employed to generate responses to questions.

Part 1: Setting Up the Database

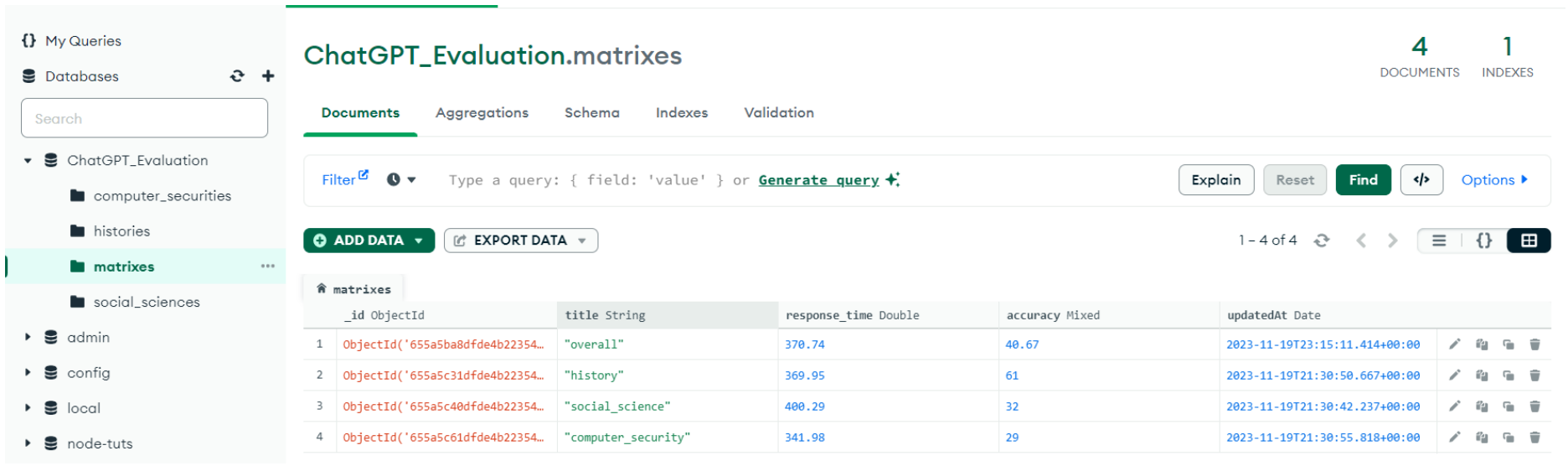
1.1 MongoDB Installation & Configuration

MongoDB has been successfully installed and configured. A database named 'ChatGPT_Evaluation' has been created with three collections. Since mongoose models were used to store and retrieve data from the database. Mongoose automatically looks for the plural, lowercase version of the model name. So by convention the name of the collections were changed to ‘histories’, ‘social_sciences’, ‘computer_sercurites’. Another collection ‘matrixes’ was used to store the average accuracies and response times per domain and overall.



1.2 Dataset Population

The ‘histories’, ‘social_sciences’ and ‘computer_sercurites collections were populated with 100 questions from the provided Google Drive repository. The ‘matrixes’ collection has four documents which were populated during the execution of the app and subject to change depending upon the data.



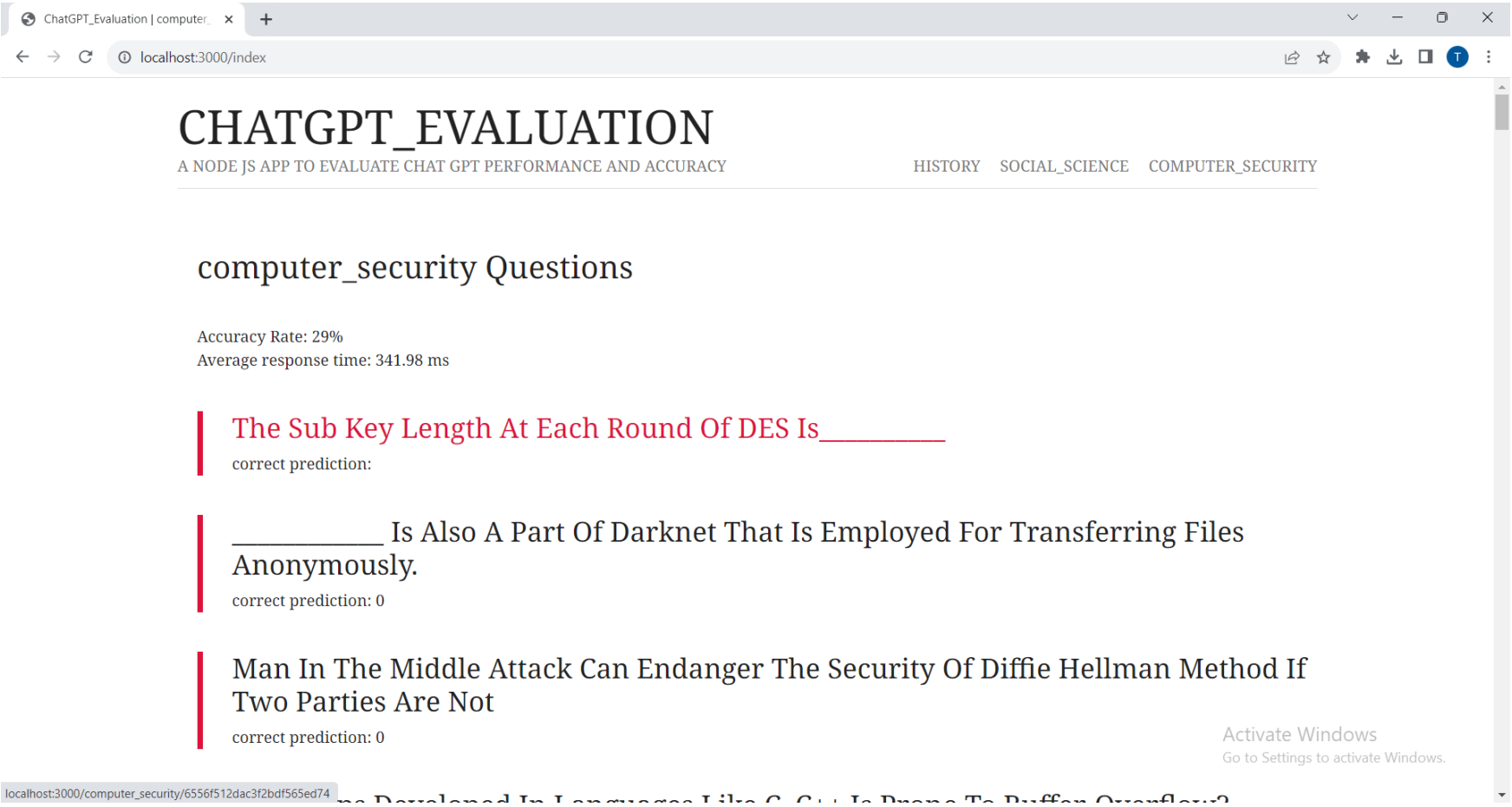
Part 2: Interacting with ChatGPT via Node.js

2.1 Client-Server Setup with Node.js

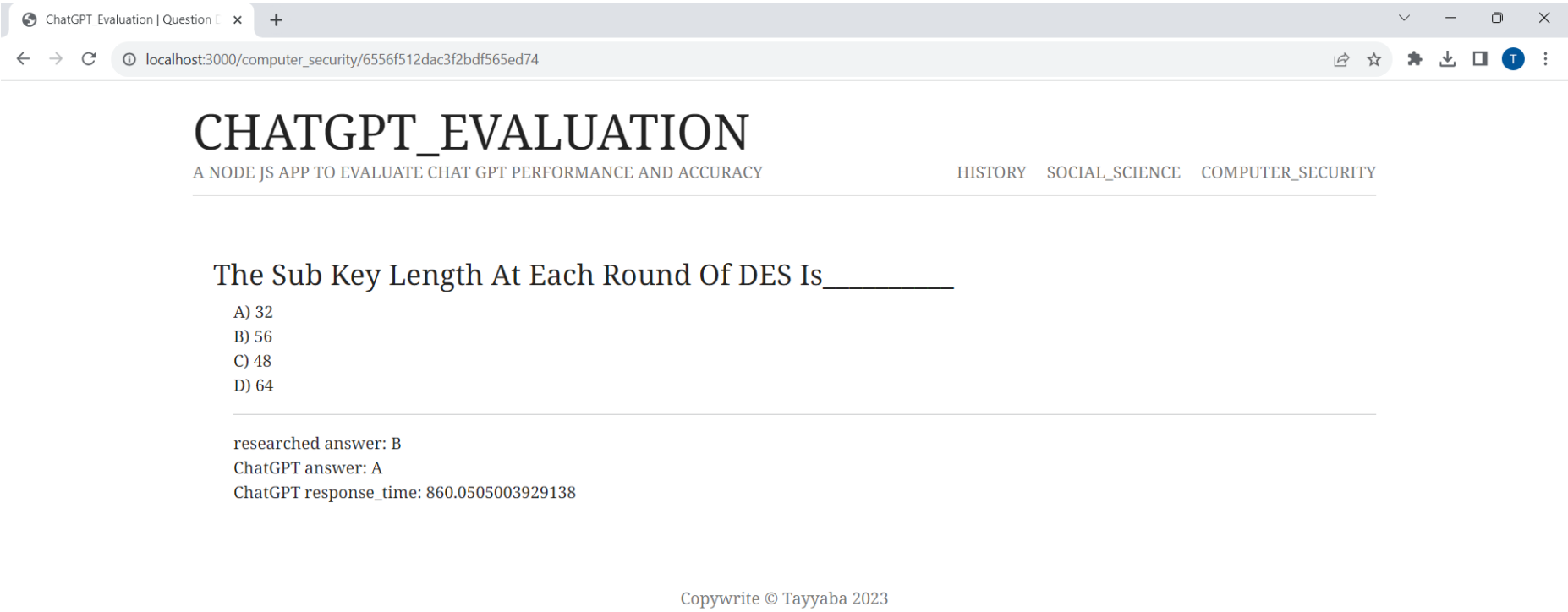
The Express framework with the MVC (Model-View-Controller) architectural pattern was used to develop the server-side application which interacts with the MongoDB database and the ChatGPT API. Routes, Models and Controllers were implemented to fetch questions from MongoDB and store ChatGPT's responses.

2.2 Server Functionality

When a request is made to the server for a question, it retrieves the question from MongoDB. The server then communicates with the ChatGPT API to obtain the model's answer. After receiving the response, the answer is stored in the relevant MongoDB document and displayed on screen as well.



a) Request is made to retrieve a question from server

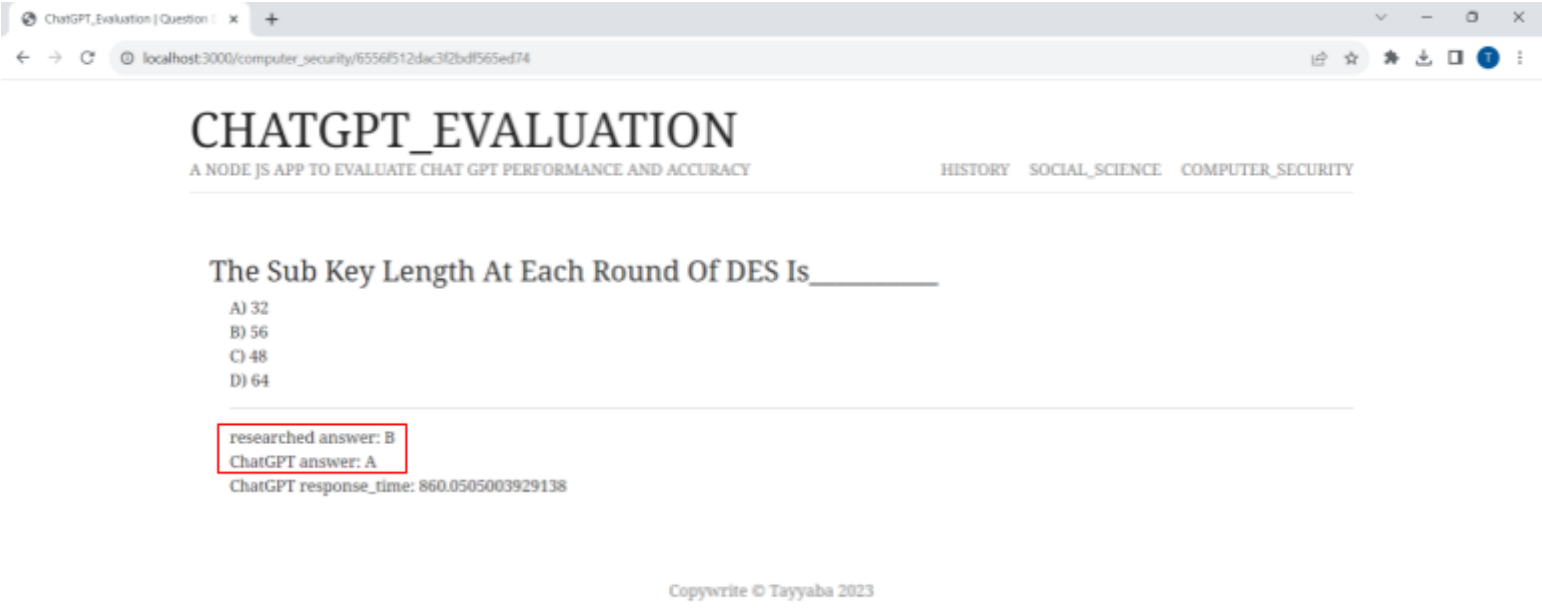


b) Server retrieves the question from MongoDB then communicates with the ChatGPT API to obtain the model's answer. After receiving the response, the answer is stored in the relevant MongoDB document and displayed on screen as well.

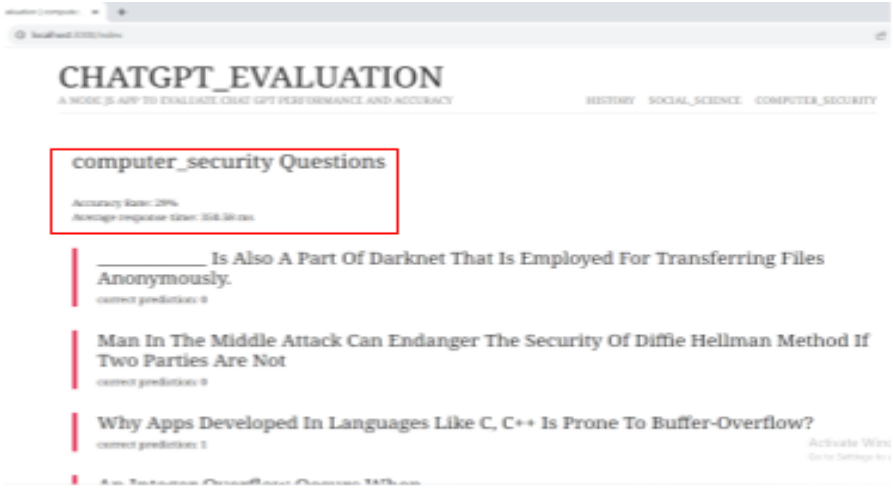
Part 3: Evaluation

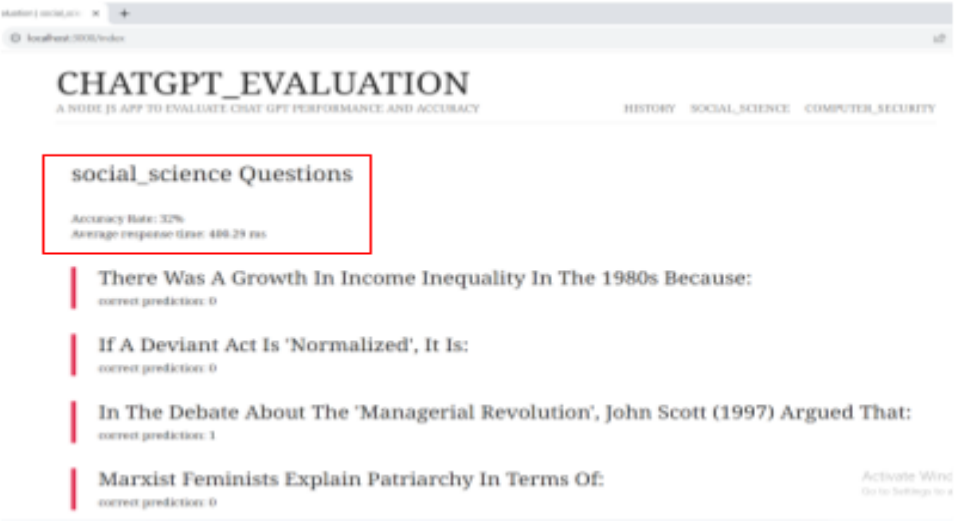
3.1 Efficiency Assessment

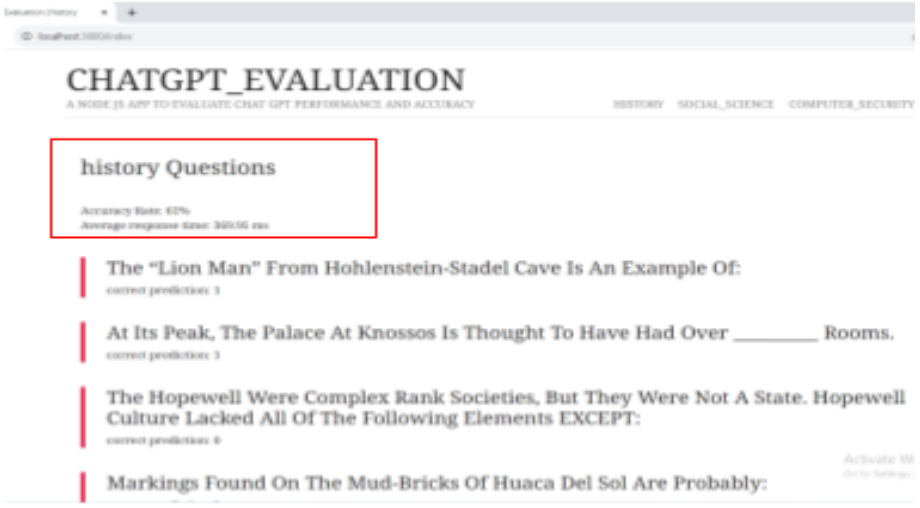
- The expected answer is contrasted with the ChatGPT answer.



- The accuracy rate per domain and overall accuracy rate have been computed.Additionally, The response time of each query has been recorded for performance evaluation.

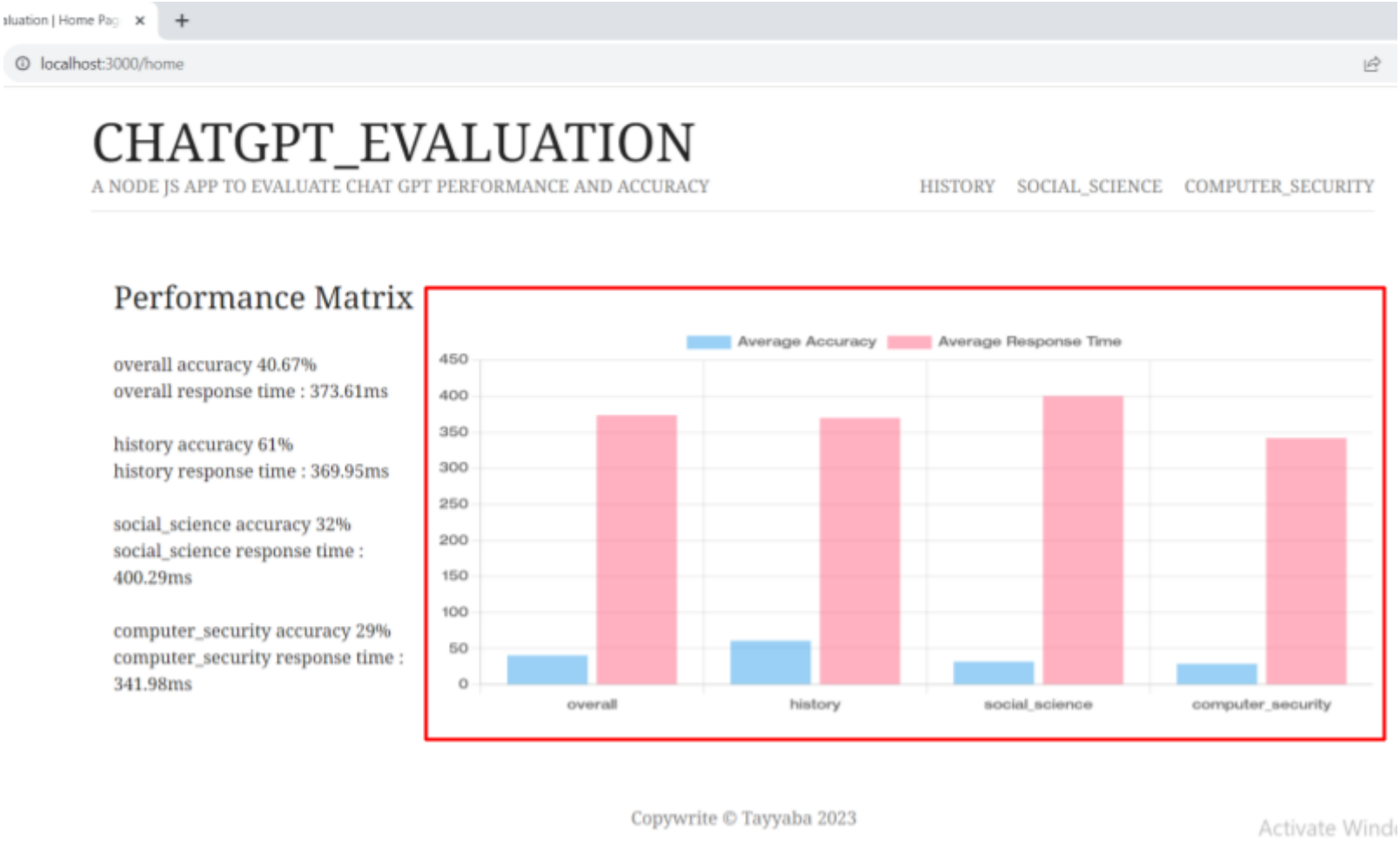






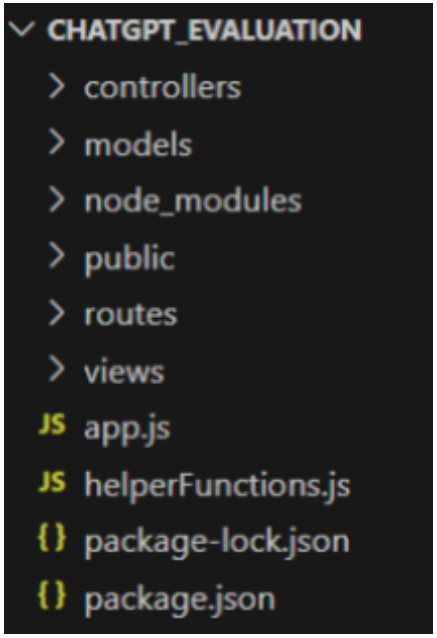
3.2 Data Analysis & Visualization

Chart.js is used to visualise the Chat GPT's accuracy rates across domains and the average response time for each domain.



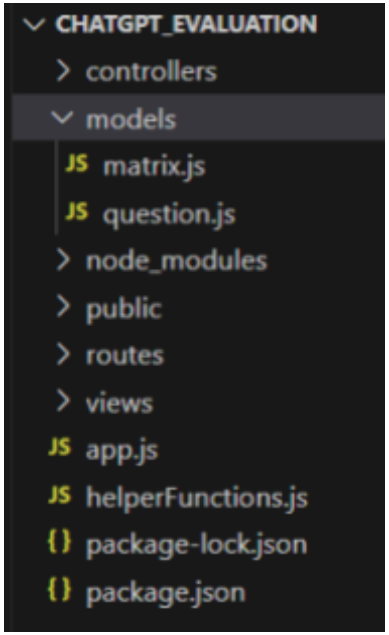
Part 4: Key Components:

Directory:



The app has the following folders.

1. **Models**
   - The model represents the data layer of the application. It is responsible for managing the data, business logic, and the application state.
   - The model consists of Mongoose schemas and models.
   - Following is the directory structure used


CHATGPT_EVALUATION
controllers
models
matrix.js
question.js
node_modules
public
routes
views
app.js
helperFunctions.js
package-lock.json
package.json

1. **Matrix.js**

   Defines the ‘matrixSchema’ and matrix model for ‘matrixes’ collection.

models > JS matrix.js > ...

```js
// 1) require mongoose
const { isNumber } = require('lodash');
const mongoose = require('mongoose');

// 2) use schema from mongoose
const Schema = mongoose.Schema;

// 3) define the Schema
const matrixSchema = new Schema({
    title:{type: String,required: true},
    accuracy:{type: Number,required: true},
    response_time:{type: Number,required: true}
},{timestamps: true});

// 4) create models
const matrix = mongoose.model('matrix',matrixSchema); //model for matrix
// export the matrix module
module.exports = matrix;
```

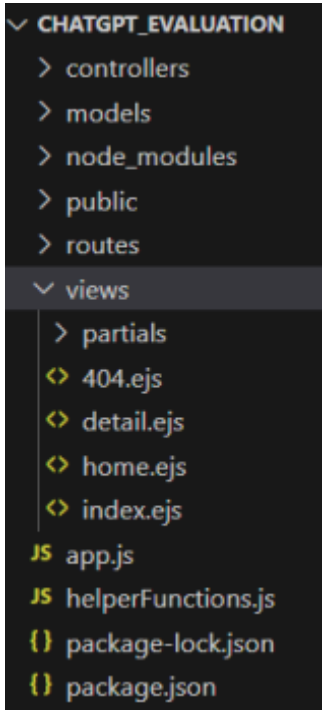
2. **Question.js**

Defines the ‘Schema’ and models for ‘histories’, ‘social_sciences’ and ‘computer_sercurites’ collections.

```
models > JS question.js > ...
// 1) require mongoose
const { isNumber } = require('lodash');
const mongoose = require('mongoose');

// 2) use schema from mongoose
const Schema = mongoose.Schema;

// 3) define the Schema
const questionSchema = new Schema({

    accuracy:{type: Number,required: true},
    question:{type: String,required: true},
    A:{type: String,required: true},
    B:{type: String,required: true},
    C:{type: String,required: true},
    D:{type: String,required: true},
    answer:{type: String,required: true},
    GPT_answer:{type: String,required: true},
    response_time:{type: Number,required: true}
},{timestamps: true});

// 4) create models
const History_question = mongoose.model('History',questionSchema); //model for history
const Social_Science_question = mongoose.model('Social_Science',questionSchema); //model for Social_Science
const Computer_Security_question = mongoose.model('Computer_Security',questionSchema); //model for Computer_Security
// export the question module
module.exports = {
    History_question,
    Social_Science_question,
    Computer_Security_question
};
```

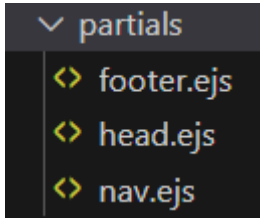
2. Views

- The view is responsible for rendering the user interface and presenting data to the users.
- The view is implemented using template engine EJS.
- Following is the directory structure used



1. Partials

Partials are used for shared components across multiple views. The folder contains headers, footers and navigation bars.



2. **404**

   The 404 error page which will be rendered when a request is made for a missing element. Mostly used for catching error purposes.

```
views > 404.ejs > ...
<html lang="en">
<%- include('./partials/head.ejs')%>
<body>
    <%- include('./partials/nav.ejs')%>

    <div class="not-found content">
        <h2>Error 404</h2>
        <div>OOPS, Page not found </div>
    </div>
    <%- include('./partials/footer.ejs')%>
</body>
</html>
```

3. **Detail**

   The page displays the details of questions like question, options, gpt answer and response time.

ChatGPT_Evaluation | Question

localhost:3000/computer_security/6556f512dac3f2bdf565ed74

CHATGPT_EVALUATION

A NODE JS APP TO EVALUATE CHAT GPT PERFORMANCE AND ACCURACY

HISTORY SOCIAL_SCIENCE COMPUTER_SECURITY

The Sub Key Length At Each Round Of DES Is_____

A) 32
B) 56
C) 48
D) 64

---

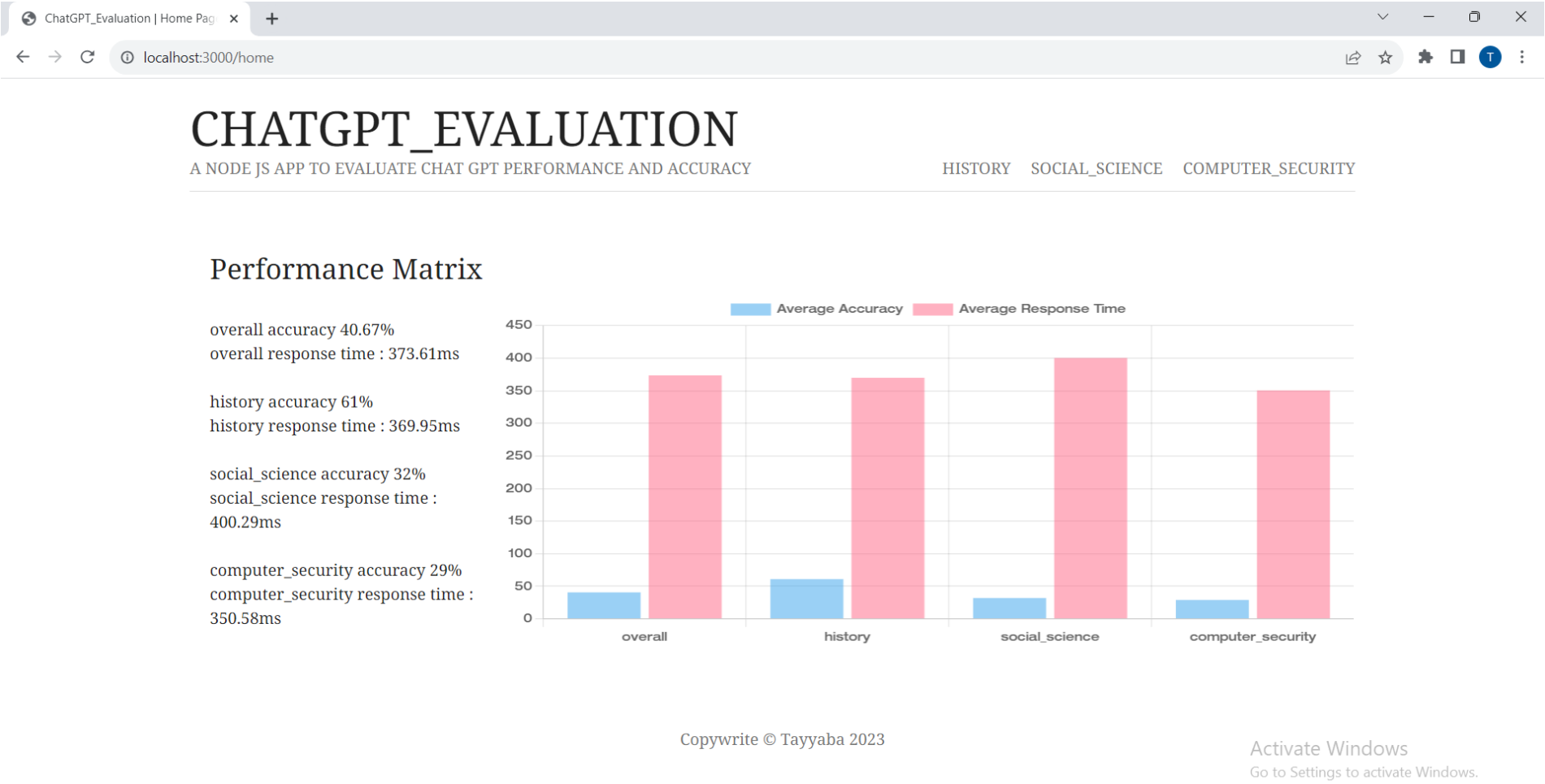
researched answer: B
ChatGPT answer: A
ChatGPT response_time: 860.0505003929138

views > detail.ejs > ...

```html
<html lang="en">
    <%- include('./partials/head.ejs')%>
<body>
    <%- include('./partials/nav.ejs')%>
    <div class="details content">
        <h3 class="title">
            <%= question.question %>
        </h3>
        <div class="content">
            <p >
                A) <%= question.A %>
            </p>
            <p >
                B) <%= question.B %>
            </p>
            <p >
                C) <%= question.C %>
            </p>
            <p >
                D) <%= question.D %>
            </p>
            <hr>
            <p >
                researched answer: <%= question.answer %>
            </p>
            <p >
                ChatGPT answer: <%= question.GPT_answer %>
            </p>
            <p >
                ChatGPT response_time: <%= question.response_time %>
            </p>
        </div>
    </div>
    <%- include('./partials/footer.ejs')%>
</body>
</html>
```

4. Home

This page displays the response time and accuracy overall and per domain both in text and graphical presentation.



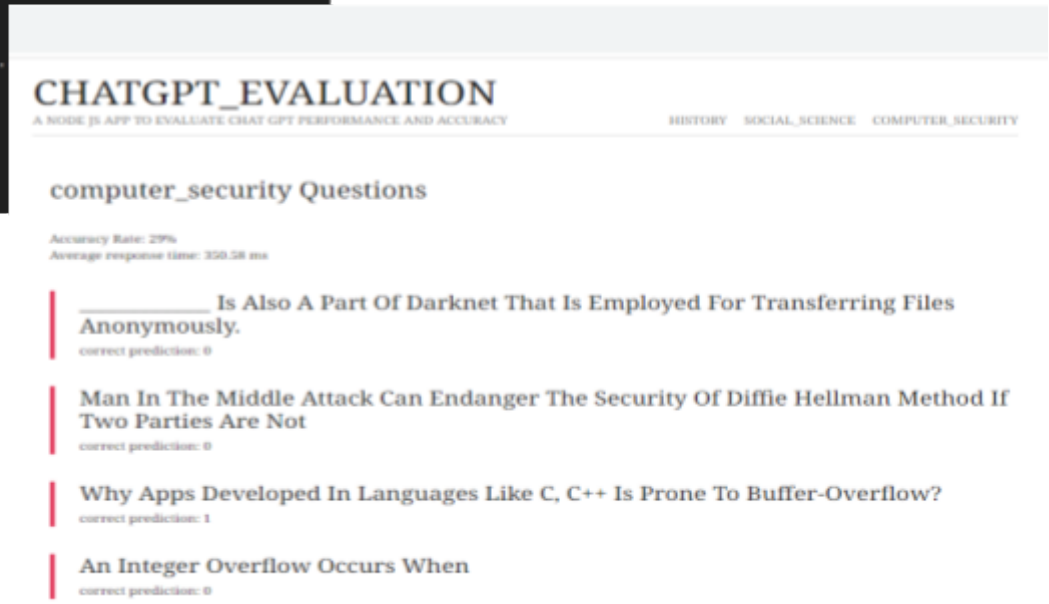
views > home.ejs > ...

```html
<html lang="en">
    <%- include('./partials/head.ejs')%>
<body>
    <%- include('./partials/nav.ejs')%>
    <div class="details content">
        <h3 class="title">
            Performance matrix
        </h3>
        <div class="row">
            <div class="col-3">
                <%
                    if(results.length >0){
                        let label = [];
                        let accuracyData = [];
                        let responseData = [];%>
                        <% results.forEach(result =>{
                            label.push(result.title);
                            accuracyData.push(result.accuracy);
                            responseData.push(result.response_time);
                                %>
                            <br>
                                <p >
                                    <script>
                                        var label = <%- JSON.stringify(label) %>;
                                        var accuracyData = <%- JSON.stringify(accuracyData) %>;
                                        var responseData = <%- JSON.stringify(responseData) %>;
                                    </script>

                                    <%= result.title %> accuracy <%= result.accuracy %>%
                                </p>
                                <p > <%= result.title %> response time : <%= result.response_time %>ms</p>
                        <% }); %>
                    <% } else{ %>
                        <p>No results to display.................</p>
                <% } %>
            </div>
            <div class="col-9">
                <canvas id="chart" height="125" width= auto></canvas>
            </div>
        </div>
        <script>
        const ctx = document.getElementById('chart');
        new Chart(ctx, {
          type: 'bar',
          data: {
            labels: label,
            datasets: [{label: 'Average Accuracy',data: accuracyData,},
            {label: 'Average Response Time', data: responseData,}
          ]
          },
          options: {
            scales: {
              y: {
                beginAtZero: true
              }
            }
          }
        });
    </script>
    <%- include('./partials/footer.ejs')%>
</body>
</html>
```

5. **index**

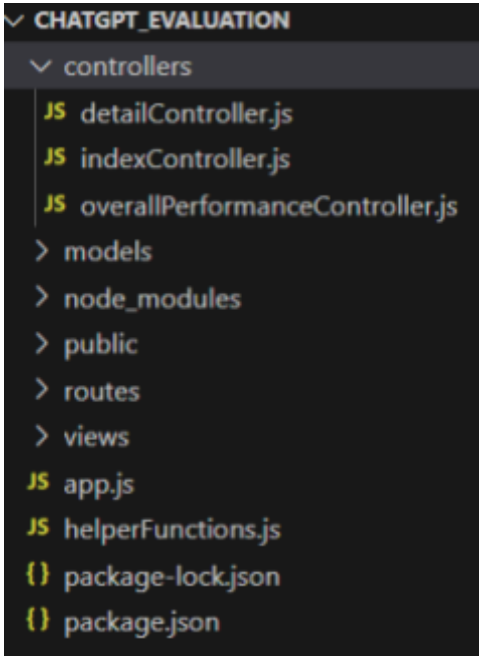
This page serves as the index of each domain which fetches all the questions from the database of the corresponding domain. This also shows the accuracy of each question and accuracy rates and average response time per domain.

```
views > index.ejs > ...
<html lang="en">
    <%- include('./partials/head.ejs')%>
<body>
    <%- include('./partials/nav.ejs')%>
    <div class="questions content">
        <h2><%= title %> Questions</h2>
        <p>Accuracy Rate: <%= accuracy%>%</p>
        <p>Average response time: <%= response_time%> ms</p>
        <% if(questions.length >0){ %>
            <% questions.forEach(question =>{ %>
                <a class="single" href="/<%= domain_title %>/<%= question._id %>">
                    <h3 class="question">
                        <%= question.question %>
                    </h3>
                    <p>correct prediction: <%= question.accuracy %></p>
                </a>
            <% }); %>

        <% } else{ %>
            <p>No questions to display.....

        <% } %>
    </div>
    <%- include('./partials/footer.ejs')%>
</body>
</html>
```



3. Controllers

- The controller acts as an intermediary between the model and the view. It handles user inputs, processes business logic, and updates the model and view accordingly.
- The controller folder is the second half of the controller pattern which contains the files that define the route handlers. These handlers are interacting with the model, and rendering views.
- Following is the directory structure used



1. detailController.js

- Controllers handle requests related to different question domains (history, social science, computer security).
- They interact with the OpenAI API to get responses and update the database with GPT answers, accuracy, and response time.

controllers > JS detailController.js > ...

```javascript
const indexRoutes = require('../routes/indexRoutes');
// import the question models from the models folder
const {History_question,Social_Science_question,Computer_Security_question} = require('../models/question');

// response time
const { performance } = require('perf_hooks');

// requiring openai
const OpenAI = require('openai');

// key
const openai = new OpenAI({
  apiKey: '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', // defaults to process.env["OPENAI_API_KEY"]
});

const history_detail = (req,res) =>{
    const id = req.params.id;
    History_question.findById(id)
        .then((result)=>{
            const question = `Question: ${result.question} A ${result.A} B ${result.B} C ${result.C} D ${result.D} .
            Do not use  dot in front or back, comma, any character,special characters, space and give the option letter only`;

            async function main() {
                const startTime = performance.now();
                const completion = await openai.completions.create({
                    model: "text-davinci-003",
                    prompt: question,
                    max_tokens: 4,
                    temperature: 0.2,
                });
                const endTime = performance.now();
                const responseTime = endTime - startTime;
              const gpt_answer = (completion.choices[0].text).replace(/[^A-D]/g, "").charAt(0);
              let acc = 0;
              if(result.answer==gpt_answer){
                acc = acc + 1;
              }
              History_question.findByIdAndUpdate(id,{GPT_answer:gpt_answer, response_time: responseTime,accuracy: acc}, {new: true})
                .then((result)=>{
                    History_question.findById(id)
                        .then((result)=>{
                            res.render('detail',{page_title: 'Question Details', question: result})
                        })
                        .catch((err)=>{
                            res.status(404).render('404', {page_title : 'page not found'});
                        });
                })
                .catch(err=>console.log(err));
            }
            main();
        })
        .catch((err)=>{
            res.status(404).render('404', {page_title : 'page not found'});
        });
}
```

```
const social_science_detail = (req,res) =>{
    const id = req.params.id;
    Social_Science_question.findById(id)
        .then((result)=>{
            const question = `Question: ${result.question} A ${result.A} B ${result.B} C ${result.C} D ${result.D} .
            Do not use  dot in front or back, comma, any character,special characters, space and give the option letter only`;
            async function main() {
                const startTime = performance.now();
                const completion = await openai.completions.create({
                    model: "text-davinci-003",
                    prompt: question,
                    max_tokens: 4,
                    temperature: 0.2,
                    top_p:0.1,
                });
                const endTime = performance.now();
                const responseTime = endTime - startTime;
              const gpt_answer = (completion.choices[0].text).replace(/[^A-D]/g, "").charAt(0);
              let acc = 0;
              if(result.answer==gpt_answer){
                acc = acc + 1;
              }
            Social_Science_question.findByIdAndUpdate(id,{GPT_answer:gpt_answer, response_time: responseTime,accuracy: acc}, {new: true})
                .then((result)=>{
                    Social_Science_question.findById(id)
                        .then((result)=>{
                            res.render('detail',{page_title: 'Question Details', question: result})
                        })
                        .catch((err)=>{
                            res.status(404).render('404', {page_title : 'page not found'});
                        });
                })
                .catch(err=>console.log(err));
            }
            main();
        })
        .catch((err)=>{
            res.status(404).render('404', {page_title : 'page not found'});
        });
}
```

```
const computer_security_detail = (req,res) =>{
    const id = req.params.id;
    Computer_Security_question.findById(id)
        .then((result)=>{
            const question = `Question: ${result.question} A ${result.A} B ${result.B} C ${result.C} D ${result.D} .
            Do not use  dot in front or back, comma, any character,special characters, space and give the option letter only`;
            async function main() {
                const startTime = performance.now();
                const completion = await openai.completions.create({
                    model: "text-davinci-003",prompt: question,max_tokens: 4,temperature: 0.2,
                });
                const endTime = performance.now();
                const responseTime = endTime - startTime;
              const gpt_answer = ((completion.choices[0].text).replace(/[^A-D]/g, "")).charAt(0);
              let acc = 0;
              if(result.answer==gpt_answer){
                acc = acc + 1;
              }
            Computer_Security_question.findByIdAndUpdate(id,{GPT_answer:gpt_answer, response_time: responseTime, accuracy: acc}, {new: true})
                .then((result)=>{
                    Computer_Security_question.findById(id)
                        .then((result)=>{
                            res.render('detail',{page_title: 'Question Details', question: result})
                        })
                        .catch((err)=>{
                            res.status(404).render('404', {page_title : 'page not found'});
                        });
                })
                .catch(err=>console.log(err));
            }
            main();
        })
        .catch((err)=>{
            res.status(404).render('404', {page_title : 'page not found'});
        });
}
module.exports = {
    history_detail,
    social_science_detail,
    computer_security_detail
};
```

2. **indexController.js**
    - Manages domain-specific routes and retrieves questions for the specified domain.
    - Uses the matrix helper function to calculate accuracy and response time for the domain.

controllers > JS indexController.js > ...

```javascript
// import the question models from the models folder
const { History_question, Social_Science_question, Computer_Security_question } = require('../models/question');
const matrix_model= require('../models/matrix');
const matrix = require ('../helperFunctions'); // matrix helper function
let domain = "";
const history_index = (req, res) => {
    domain = 'history';
    res.redirect('/index');
}
const social_science_index = (req, res) => {
    domain = 'social_science';
    res.redirect('/index');
}
const computer_security_index = (req, res) => {
    domain = 'computer_security';
    res.redirect('/index');
}
const domain_index = async (req, res) => {
    try {
        let result;

        if (domain === 'history') {
            result = await History_question.find().sort({ updatedAt: 1 });
        } else if (domain === 'social_science') {
            result = await Social_Science_question.find().sort({ updatedAt: 1 });
        } else if (domain === 'computer_security') {
            result = await Computer_Security_question.find().sort({ updatedAt: 1 });
        } else {
            throw new Error('Invalid domain');
        }
        const { accuracy, response_time } = await matrix(result);
         // save in db
         matrix_model.findOneAndUpdate({title:domain}, { accuracy: accuracy, response_time: response_time }, {new:true})
         .then((matrixResult)=>{
            res.render('index', { title: domain, domain_title: domain, page_title: domain, questions: result, accuracy:
                matrixResult.accuracy, response_time: matrixResult.response_time });
         })
         .catch((err)=>{
            res.status(404).render('404', {page_title : 'page not found'});
         });
    } catch (err) {
        console.error('Error in domain_index:', err);
        res.status(404).render('404', { page_title: '404' });
    }
};
module.exports = {
    history_index,
    social_science_index,
    computer_security_index,
    domain_index
};
```

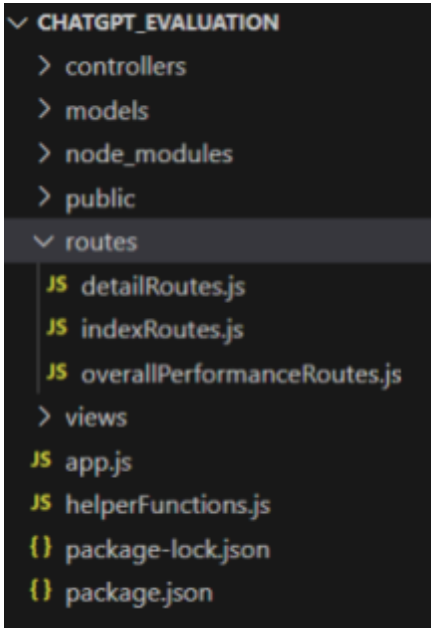
3. **overallPerformanceController.js**
    - Handles routes related to overall performance.
    - Retrieves data from all domains, calculates overall accuracy and response time, and updates the matrix model.

controllers > JS overallPerformanceController.js > ...

```javascript
const indexRoutes = require('../routes/indexRoutes');
// import the question models from the models folder
const {History_question,Social_Science_question,Computer_Security_question} = require('../models/question');
const matrix_model= require('../models/matrix');
const matrix = require ('../helperFunctions');
// response time
const { performance } = require('perf_hooks');
// requiring openai
const OpenAI = require('openai');
// key
const openai = new OpenAI({
  apiKey: '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',
  // defaults to process.env["OPENAI_API_KEY"]
});
const backslash_route = (req,res)=>{
    res.redirect('/home');
}
const getOverallMatrix = async (req,res) => {
    try{
        const datas = await History_question.aggregate(
            [
                {$unionWith:"social_sciences"},
                { $unionWith: "computer_securities" },
                { $project: { accuracy: "$accuracy", response_time:"$response_time"} }
            ]
        );
        const { accuracy, response_time} = await matrix(datas);
        // save in db
        matrix_model.findOneAndUpdate({title:'overall'}, { accuracy: accuracy, response_time: response_time }, {new:true})
        .then((result)=>{
            matrix_model.find()
                .then((matrixResult)=>{
                    res.render('home',{title:'Home Page', page_title:'Home Page', results:matrixResult})
                })
                .catch((err)=>{
                    res.status(404).render('404', {page_title : 'page not found'});
                })
        })
        .catch((err)=>{
            res.status(404).render('404', {page_title : 'page not found'});
        });
    }catch (error){
        console.log(error);
    }
}
module.exports = {
    getOverallMatrix,
    backslash_route
};
```

4. Routes

- The routes folder is the first half of the controller pattern which contains the files that define the routes. These are responsible for processing HTTP requests to the desired controller by using an express router.
- Following is the directory structure used



1. detailRoutes.js

Redirecting the question request to the detail controllers by using the id of that particular question and domain name correspondingly.

routes > JS detailRoutes.js > ...

```
const express = require('express');
const detailController = require('../controllers/detailController');

const router = express.Router();

router.get('/history/:id',detailController.history_detail);
router.get('/social_science/:id', detailController.social_science_detail);
router.get('/computer_security/:id',detailController.computer_security_detail);

module.exports = router;
```

2. **indexRoutes.js**

Redirecting the request of the index of each domain to the specified index controller.

```
routes > JS indexRoutes.js > ...
const express = require('express');
const indexController = require('../controllers/indexController');
const router = express.Router();
// redirecting the question routes by category
router.get('/history',indexController.history_index);
router.get('/social_science', indexController.social_science_index);
router.get('/computer_security', indexController.computer_security_index);
// domain routes
router.get('/index',indexController.domain_index)
module.exports = router;
```

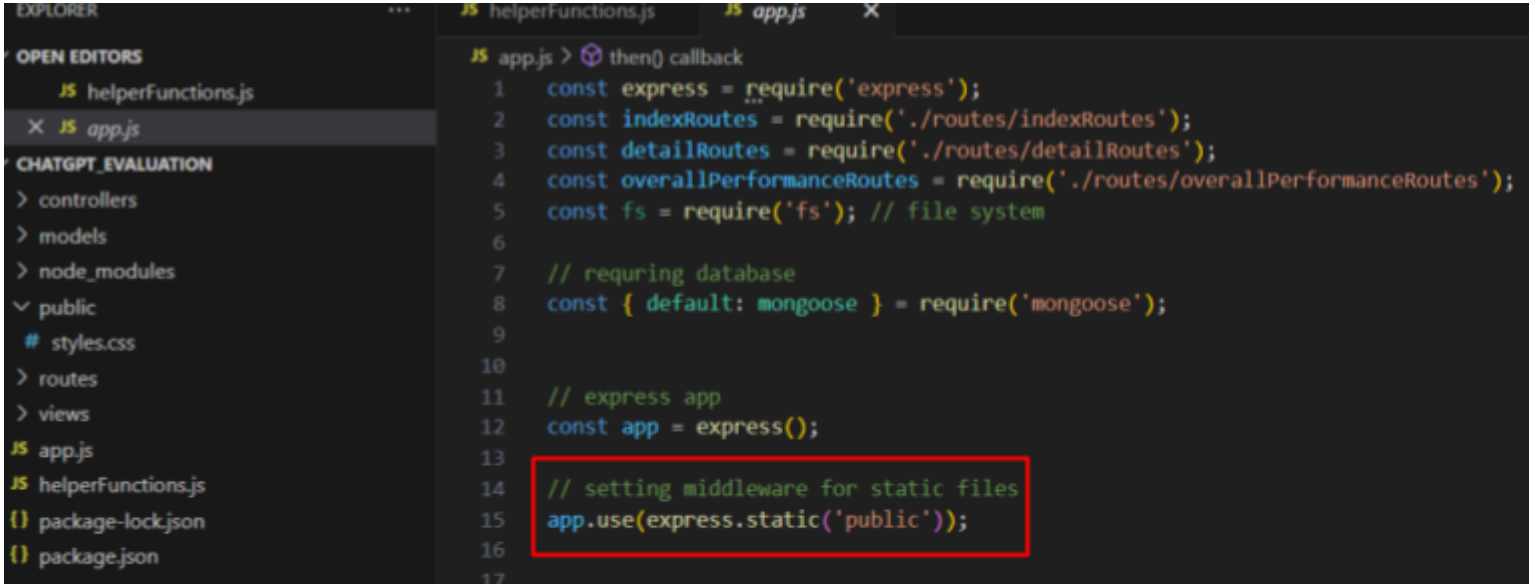
3. **overallPerformanceRoutes.js**

Redirecting the / and home page request to overall performance controller.

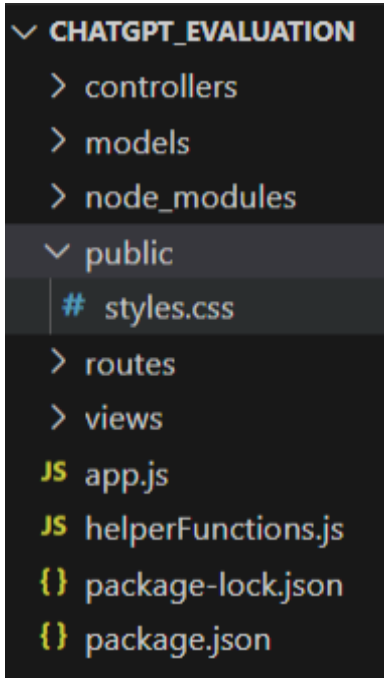
```
routes > JS overallPerformanceRoutes.js > ...
const express = require('express');
const overallPerformanceController = require('../controllers/overallPerformanceController');
const router = express.Router();
router.get('/',overallPerformanceController.backslash_route);
router.get('/home',overallPerformanceController.getOverallMatrix);
module.exports = router;
```

5. Public

- The public contains static pages like stylesheets and accessed from app.js by using static express middleware.



- Following is the directory structure



The app has the following files in the main directory.

1. **App.js (Express App)**
    - The app uses the Express framework for routing and handling HTTP requests.
    - Static files are served from the 'public' directory.
    - MongoDB connection is established using Mongoose.
    - EJS is set as the view engine.

app.js > ...

```js
const express = require('express');
const indexRoutes = require('./routes/indexRoutes');
const detailRoutes = require('./routes/detailRoutes');
const overallPerformanceRoutes = require('./routes/overallPerformanceRoutes');
// requring database
const { default: mongoose } = require('mongoose');
// express app
const app = express();
// setting middleware for static files
app.use(express.static('public'));
// connect with database
dbURI = "mongodb://localhost:27017/ChatGPT_Evaluation";
mongoose.connect(dbURI)
    .then((result)=>{
        // listen for requests
        app.listen(3000);
    })
    .catch(err=>console.log(err));
//  register view engine
app.set('view engine', 'ejs');
// redirecting home page
app.use(overallPerformanceRoutes);
// domain routes
app.use(indexRoutes);
// details routes
app.use(detailRoutes);
// 404 page
app.use((req,res)=>{
    res.status(404).render('404', {page_title : 'page not found'});
})
```

2. helperFunctions.js

- The matrix is a helper function that processes an array of data.
- It calculates accuracy rates and average response time based on the provided data.

```js
helperFunctions.js > ...
const matrix = async (datas)=> {
    try {
        let accuracy_rate = 0.0;
        let responseTime = 0.0;

        datas.forEach(data => {

            accuracy_rate = Math.round((accuracy_rate + data.accuracy)*100)/100;
            responseTime = Math.round((responseTime + data.response_time)*100)/100;
        });

        if (datas.length > 0) {
            accuracy_rate = Math.round(((accuracy_rate * 100) / datas.length) * 100) / 100;
            responseTime = Math.round((responseTime / datas.length) * 100) / 100;
        }


        return { accuracy: accuracy_rate, response_time: responseTime };
    } catch (error) {
        console.error('Error in matrix of helperFunction:', error);
        throw error;
    }
}

module.exports = matrix;
```

3. Challenges Encountered:

3.1. Asynchronous Operations:

The application heavily relies on asynchronous operations, especially when interacting with databases and external APIs. Proper error handling and debugging is done by using promises.

3.2. Data Consistency:

For ensuring data consistency across different controllers and models is crucial the output from chat gpt is formatted in a specific way where gpt is asked to give answers only from the provided option with only letters also the temperature and max token is set accordingly. For further formatting only the letter from A - D is accepted without any special characters and spaces. Any discrepancies in data could lead to inaccurate overall performance metrics.

3.3. Error Handling:

While error handling is present in some places, there are still some places where comprehensive error handling is essential to handle the unexpected scenarios. Thus, this application is lacking proper error handling for all the unexpected situations.

3.4. Security Concerns:

The application is not expecting any inputs from users thus it will face less security vulnerabilities, such as SQL injection or cross-site scripting.

4. Insights:

4.1. Code Organization:

The application is using MVC and modularization approach which is very important to maintain and reusability of the code. By using these approaches I learned how to modularize the code and use it in different contexts and scenarios.

4.2. Logging and Debugging:

By implementing extensive logging to facilitate debugging and monitoring during development helped me to understand the different concepts like sync and async functions, routes handlers,openai API, schemas and models etc.

4.3. Testing:

By implementing unit tests and integration tests to ensure the reliability of the application, especially for critical functionalities like GPT response generation and matrix calculation I learned how to generate a specific query for desired answer.

Performance matrix.

overall accuracy 40.67%

overall response time : 373.61ms

history accuracy 61%

history response time : 369.95ms

social_science accuracy 32%

social_science response time : 400.29ms

computer_security accuracy 29%

computer_security response time : 350.58ms

Conclusion

The project successfully evaluates the efficiency of ChatGPT in answering questions from different domains.The combination of MongoDB, Node.js, and ChatGPT API demonstrates a robust system for handling and analysing data. The visualisations provide clear insights into ChatGPT's performance. Overall, the project achieves its objective of assessing ChatGPT's capabilities in a server-client system.