Database, networks and web coursework report: -

Introduction: -

The goal is to develop a deployable blogging tool that will allow one user to run a blog on their server. The tool has distinct sites for authors and readers. While users can browse, like, and comment on published articles, the author can create, modify, and publish articles. The technological requirements call for the usage of SQLite for the data tier, Express.js for server-side functionality, and Embedded JavaScript Templates for server-side rendering. With an emphasis on dynamic data retrieval and storage in the SQLite database, the project satisfies the fundamental criteria, including author and reader home pages, settings page, article edit page, and reader home page. I've added front-end styling and a GUI extension to give the site a cleaner, more contemporary appearance.

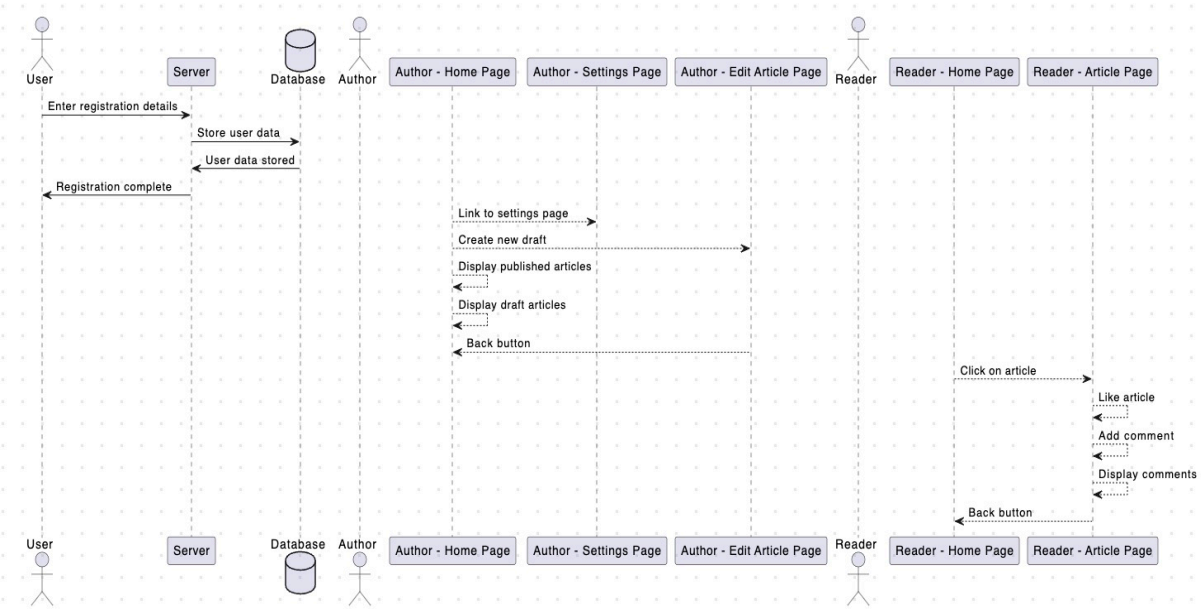


Diagram 1: - UML diagram showing the process of the website

Diagrams 2&3: Diagrams demonstrating all three tiers of your architecture, the endpoints that connect the client to the server.

- The web app can be accessed via http://localhost:3000
- The web app has 2 main pages
- Reader page: http://localhost:3000/
- Author page: http://localhost:3000/author
- Reader and author pages can be switched between using the buttons in the top right corner of the page.
- The reader page allows you to view the published articles in the database.
- The author page allows you to create and edit new articles.

Authors homepage requirements: -

Author.js and reader.js: -

```
const express = require('express')
const router = express.Router()
const db = require('../utils/sqlitePromises')
const getBlogSettings = require('../utils/utilFunctions')

/**
 * @api {get} /author Get all Articles
 */
router.get('/', async (req, res) => {
  const blog_settings = await getBlogSettings()
  const published_articles = await db.all(
    "SELECT * FROM articles WHERE article_status='Published' ORDER BY article_updated_at DESC"
  )
  const draft_articles = await db.all(
    "SELECT * FROM articles WHERE article_status='Draft' ORDER BY article_updated_at DESC"
  )
  res.render('author/index', {
    blog_settings,
    published_articles,
    draft_articles,
  })
})

/**
 * @api {get} /author/blog-settings Get Blog Settings
 */
router.get('/blog-settings', async (req, res) => {
  const blog_settings = await getBlogSettings()
  res.render('author/blog-settings', {
    blog_settings,
  })
})

/**
 * @api {post} /author/blog-settings Update Blog Settings
 */
router.post('/blog-settings', async (req, res) => {
  const { blog_title, blog_subtitle, blog_author } = req.body
  const blog_id = 1

  await db.run(
    'UPDATE blog_settings SET (blog_title, blog_subtitle, blog_author) = (?, ?, ?) WHERE blog_id = ?',
    [blog_title, blog_subtitle, blog_author, blog_id]
  )
```

```
45    res.json({ message: 'Blog settings updated' })
46    // res.redirect('/author/blog-settings')
47  })
48
49  /**
50   * @api {get} /author/create-new-article Create New Article
51   */
52  router.get('/create-new-article', async (req, res) => {
53    const blog_settings = await getBlogSettings()
54    res.render('author/create-new-article', {
55      blog_settings,
56    })
57  })
58
59  /**
60   * @api {post} /author/create-new-article Create New Article
61   */
62  router.post('/create-new-article', async (req, res) => {
63    const { article_title, article_subtitle, article_content } = req.body
64    const blog_settings = await getBlogSettings()
65    await db.run(
66      'INSERT INTO articles (article_title, article_subtitle, article_content, article_author) VALUES (?, ?, ?, ?)',
67      [
68        article_title,
69        article_subtitle,
70        article_content,
71        blog_settings.blog_author,
72      ]
73    )
74    res.json({ message: 'Article created' })
75    // res.redirect('/author')
76  })
77
78  /**
79   * @api {get} /author/edit-article/:article_id Edit Article
80   * @apiParam {Number} article_id Article ID
81   */
82  router.get('/edit-article/:article_id', async (req, res) => {
83    const article_id = req.params.article_id
84    const article = await db.get('SELECT * FROM articles WHERE article_id = ?', [
85      article_id,
86    ])
87    const blog_settings = await getBlogSettings()
88    res.render('author/edit-article', {
89      blog_settings,
90      article,
91    })
92  })
93
94  /**
95   * @api {put} /author/edit-article/:article_id Edit Article
96   * @apiParam {Number} article_id Article ID
97   */
98  router.put('/edit-article/:article_id', async (req, res) => {
99    const article_id = req.params.article_id
100   const { article_title, article_subtitle, article_content } = req.body
101   await db.run(
102     'UPDATE articles SET (article_title, article_subtitle, article_content, article_updated_at) = (?, ?, ?, CURRENT_TIMESTAMP)
103     [article_title, article_subtitle, article_content, article_id]
104   )
105   res.json({ message: 'Article updated' })
106   // res.redirect('/author')
107 })
108
109 /**
110  * @api {get} /author/article/:article_id/:action Publish or Draft Article
111  * @apiParam {Number} article_id Article ID
112  * @apiParam {String} action Action to perform
113  */
114 router.put('/article/:article_id/:action', async (req, res) => {
115   const article_id = req.params.article_id
116   const actionParam = req.params.action
117
118   if (actionParam === 'publish') {
119     await db.run(
120       'UPDATE articles SET (article_status, article_published_on) = (?, CURRENT_TIMESTAMP) WHERE article_id = ?',
121       ['Published', article_id]
122     )
123   } else if (actionParam === 'draft') {
124     await db.run(
125       'UPDATE articles SET (article_status, article_updated_at) = (?, CURRENT_TIMESTAMP) WHERE article_id = ?',
126       ['Draft', article_id]
127     )
128   }
129   res.json({ message: 'Article updated' })
130   // res.redirect('/author')
131 })
```

```
/**
 * @api {delete} /author/article/:article_id Delete Article
 * @apiParam {Number} article_id Article ID
 */
router.delete('/article/:article_id', async (req, res) => {
  const article_id = req.params.article_id
  await db.run('DELETE FROM article_comments WHERE article_id = ?', [
    article_id,
  ])
  await db.run('DELETE FROM articles WHERE article_id = ?', [article_id])
  res.json({ message: 'Article deleted' })
  // res.redirect('/author')
})

module.exports = router

```

This code is to be a backend implementation of an API using the Express framework in Node.js. It defines various routes to interact with blog settings and articles. Let's break down the important parts of this code and explain them:

Importing Dependencies:
The code imports the necessary modules: express for creating the router, sqlitePromises for interacting with SQLite database (presumably with async/await support), and utilFunctions for a function called getBlogSettings.

router.get('/') - Getting all Articles:
This route handles a GET request to the root path (/). It retrieves blog settings, published articles, and draft articles from the database using await db.all() calls. The results are then rendered using a view template (res.render) with the blog_settings, published_articles, and draft_articles variables.

router.get('/blog-settings') - Getting Blog Settings:
This route handles a GET request to /blog-settings. It fetches the blog settings from the database using await getBlogSettings() and renders the settings using a view template.

router.post('/blog-settings') - Updating Blog Settings:
This route handles a POST request to /blog-settings. It receives the updated blog settings (blog_title, blog_subtitle, blog_author) from the request body and updates the database using await db.run() with an SQL UPDATE statement.

router.get('/create-new-article') - Rendering the New Article Form:
This route handles a GET request to /create-new-article. It fetches blog settings and renders a form template to create a new article.

router.post('/create-new-article') - Creating a New Article:
This route handles a POST request to /create-new-article. It receives the new article details (article_title, article_subtitle, article_content) from the request body, and after

fetching the blog settings, it inserts the new article into the database using an SQL INSERT statement.

router.get('/edit-article/:article_id') - Rendering the Edit Article Form:
This route handles a GET request to /edit-article/:article_id. It fetches the article with the given article_id from the database and renders a form template to edit the article.

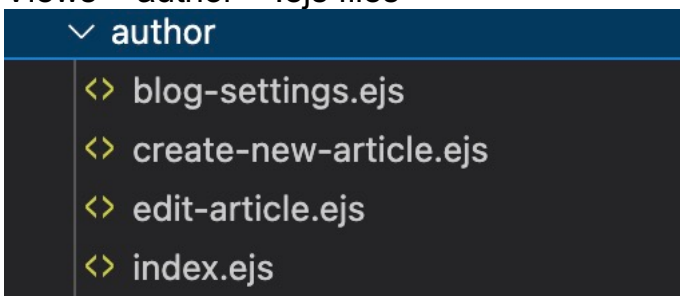
router.put('/edit-article/:article_id') - Updating an Article:
This route handles a PUT request to /edit-article/:article_id. It receives the updated article details (article_title, article_subtitle, article_content) from the request body and updates the corresponding article in the database using an SQL UPDATE statement.

router.put('/article/:article_id/:action') - Publishing or Drafting an Article:
This route handles a PUT request to /article/:article_id/:action. It allows an article to be published or drafted based on the action parameter ('publish' or 'draft'). The article's status and other details are updated in the database using SQL UPDATE statements.

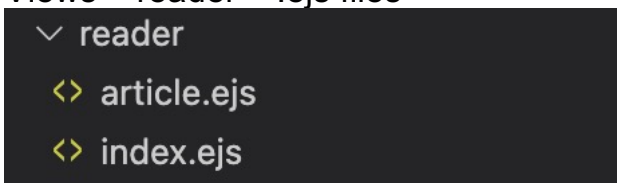
router.delete('/article/:article_id') - Deleting an Article:
This route handles a DELETE request to /article/:article_id. It deletes the specified article (and its associated comments) from the database using SQL DELETE statements.

module.exports = router:
This line exports the defined router instance so that it can be used in other parts of the application.

Views > author > .ejs files

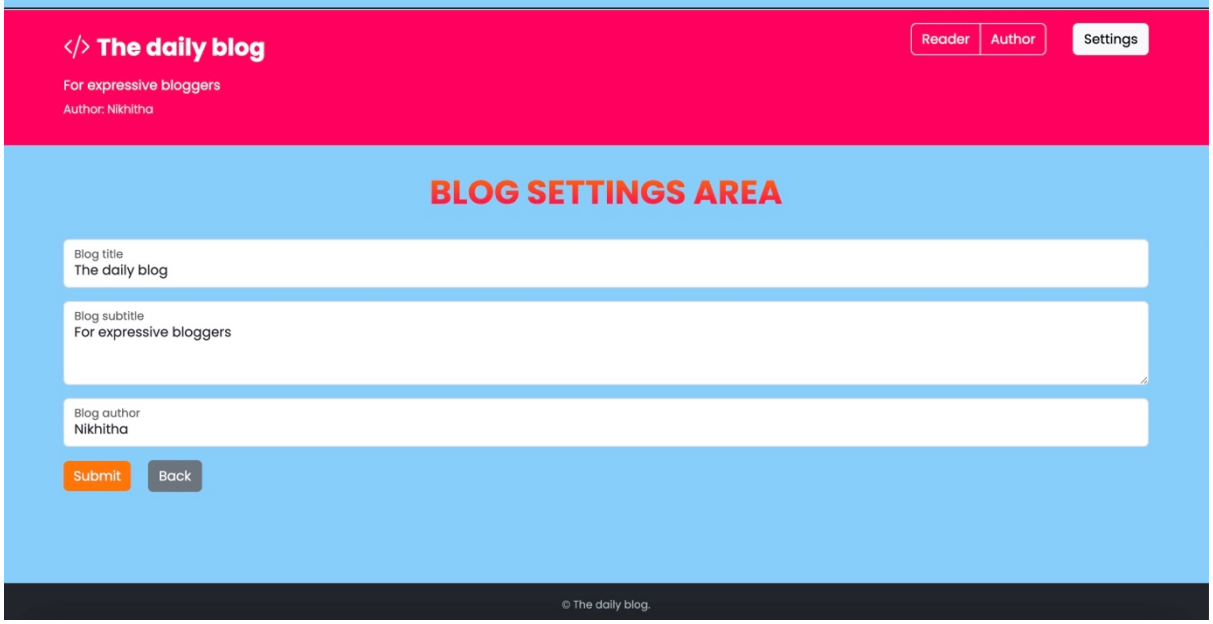


Views > reader > .ejs files



Categorized each new function so that the code would be more organized.

Authors page output: -



</> The daily blog

For expressive bloggers

Author: Nikhitha

Reader | Author | Settings

EDIT ARTICLE

Article title
Aliquam vulputate ullamcorper magna. Sed eu eros. Nam consequat

Article subtitle
massa. Integer vitae nibh. Donec est mauris,

Article content
tristique ac, eleifend vitae, erat. Vivamus nisi. Mauris nulla. Integer urna. Vivamus molestie dapibus ligula. Aliquam erat volutpat. Nulla dignissim. Maecenas ornare egestas ligula. Nullam feugiat placerat velit. Quisque varius. Nam porttitor scelerisque neque. Nullam nisl. Maecenas malesuada fringilla est. Mauris eu turpis. Nulla aliquet. Proin velit. Sed malesuada augue ut lacus.

Article created at
2023-07-24 09:51:39

Article updated at
2023-07-24 09:51:39

Submit | Back

</> The daily blog

For expressive bloggers

Author: Nikhitha

Reader | Author | Settings

ARTICLE MANAGEMENT AREA

Draft Articles

Create New Draft

Title	Published On	Actions
Aliquam vulputate ullamcorper magna. Sed eu e Nam consequat	2023-07-24 09:54:43	Publish

Delete Article

Are you sure you want to delete this article?

Delete | Cancel

Published Articles

Title	Author	Likes	Created At	Updated At	Published On	Actions
sociis natoque penatibus et magnis dis parturient	Graiden Chapman	14	2023-07-24 09:51:39	2023-07-24 09:51:39	2023-01-15 23:52:34	Draft

</> The daily blog

For expressive bloggers

Author: Nikhitha

Reader | Author | Settings

ARTICLE MANAGEMENT AREA

Draft Articles

Create New Draft

Title	Published On	Actions
Aliquam vulputate ullamcorper magna. Sed eu e Nam consequat	2023-07-24 09:54:43	Publish

Sharable link

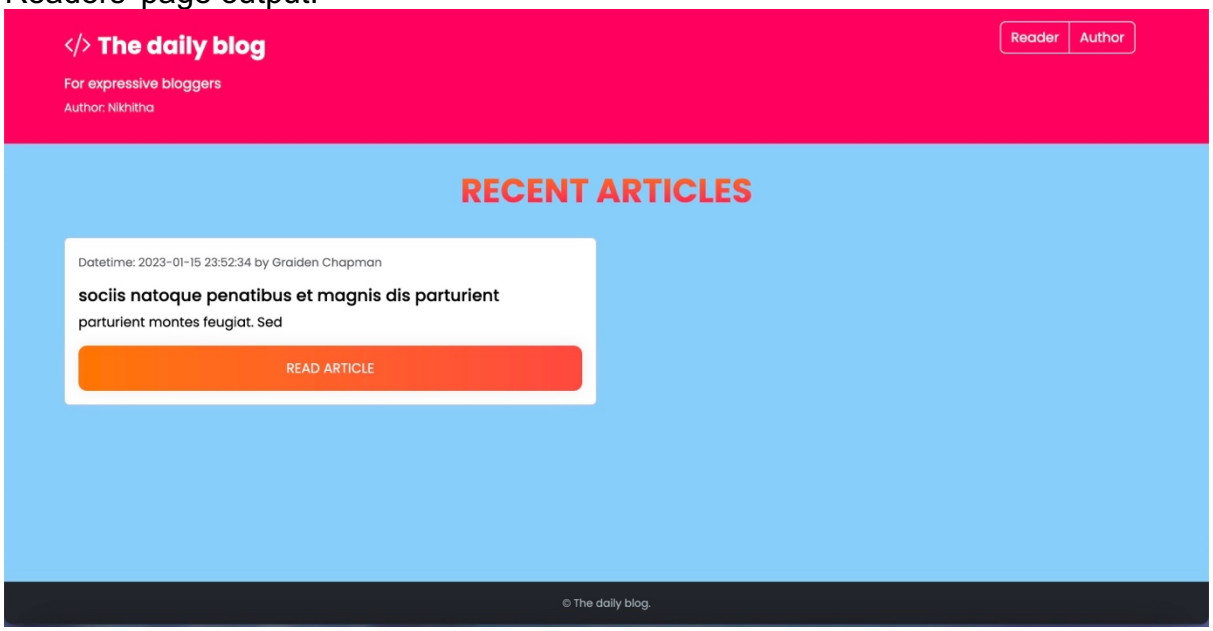
localhost:3000/article/2 | Copy

Open | Close

Published Articles

Title	Author	Likes	Created At	Updated At	Published On	Actions
sociis natoque penatibus et magnis dis parturient	Graiden Chapman	14	2023-07-24 09:51:39	2023-07-24 09:51:39	2023-01-15 23:52:34	Draft

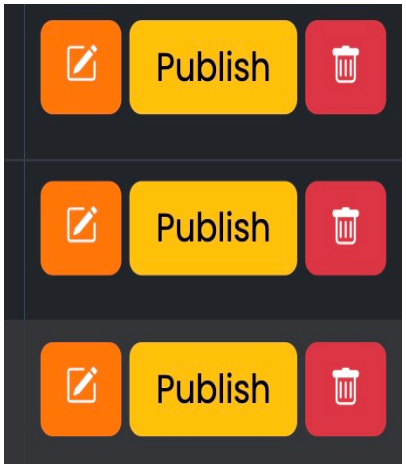
Readers' page output: -

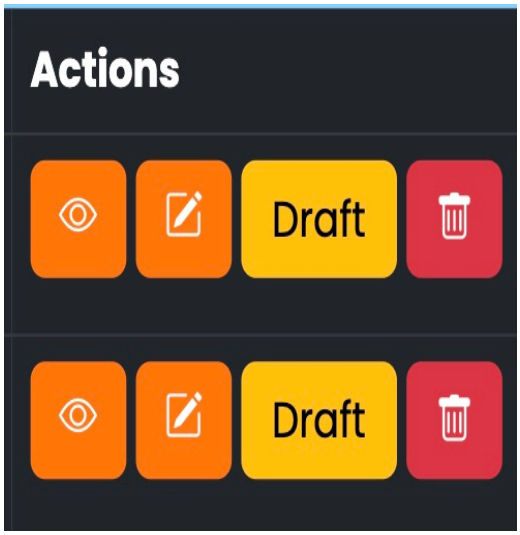


After clicking on read article: -

Extension: - Front-end styling and GUI, I Used CSS styling to create a modern professional look and feel for my application.

I want the main focus of my users to be the vibrant colour scheme that is visually appealing and modern along with icons and the overall design of the website. I have written my CSS in a way that it gives a colourful and bright UI. This CSS code sets styles for the body, title, buttons, forms, button hovering effects and other elements on a web page. It uses Flexbox for layout, applies gradients for visual effects, and includes media queries for responsive design.





CSS coding: -

```
body {
  background-color: #87CEFA;
  font-family: 'Poppins', sans-serif;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  justify-content: space-between;
  width: 100%;
  color: black;
}

.title-container {
  width: 100%;
  display: flex;
  justify-content: center;
}

#page-title {
  margin-top: 2rem;
  margin-bottom: 2rem;
  color: #1a1a1a;
  text-transform: uppercase;
  font-weight: bold;
  background: -webkit-linear-gradient(#ff7600, #ff005d);
  -webkit-background-clip: text;
  -webkit-text-fill-color: transparent;
}

.fit {
  display: inline-block;
  white-space: nowrap;
}

.article-container {
  margin-top: 2rem;
  margin-bottom: 2rem;
  color: black;
}

.comment-form {
  margin-top: 2rem;
  margin-bottom: 2rem;
  border: unset;
}
```

```
.bottom-footer {
  background-color: #1a1a1a;
  color: #f8f8f8;
  padding: 1rem;
  text-align: center;
  font-size: 0.8rem;
  /* position: absolute; */
  /* bottom: 0; */
  margin-top: 2rem;
  width: 100%;
}

.btn-read-article {
  background-image: linear-gradient(to right, #ff7600 0%, #ff005d 100%);
  padding: 15px 45px;
  text-align: center;
  text-transform: uppercase;
  transition: 0.5s;
  background-size: 200% auto;
  color: #f8f8f8;
  box-shadow: 0 0 20px #eee;
  border-radius: 10px;
  display: block;
  border: unset;
}

.btn-read-article:hover {
  background-position: right center; /* change the direction of the change here */
  color: #fff;
  text-decoration: none;
}

.purple-bg {
  background-color: #ff005d;
  color: #f8f8f8;
}

.btn-primary {
  background-color: #ff7600;
  border: unset;
}
```

```
.btn-danger {
  border: unset;
}

@media (max-width: 991.98px) {
  .mx-w-100 {
    width: 100% !important;
    color: #1a1a1a
  }
}

/* forms */
.form-floating > textarea.textarea {
  height: 100px;
}
```

Partial > head.ejs code: -

```
<head>
  <meta name="viewport" content="width=device-width" />
  <title><%= blog_settings.blog_title %></title>

<!-- Google's poppins font -->
<link rel="preconnect" href="https://fonts.googleapis.com" />
<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin />
<link
  href="https://fonts.googleapis.com/css2?family=Poppins:ital,wght@0,100;0,200;0,300;0,400;0,500;0,600;0,700;0,
  rel="stylesheet"
/>

<!-- Bootstrap Lib and Icons -->
<link
  href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css"
  rel="stylesheet"
  integrity="sha384-GLhlTQ8iRABdZLl6O3oVMWSktQOp6b7In1Zl3/Jr59b6EGGoI1aFkw7cmDA6j6gD"
  crossorigin="anonymous"
/>
<link
    rel="stylesheet"
    href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.10.3/font/bootstrap-icons.css"
  />
<script
  defer
  src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.bundle.min.js"
  integrity="sha384-w76AqPfDkMBDXo30jS1Sgez6pr3x5MlQ1ZAGC+nuZB+EYdgRZgiwxhTBTkF7CXvN"
  crossorigin="anonymous"
></script>

 <!-- Axios -->
 <script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>

 <!-- fitty lib is used to auto resize the page title width -->
 <script src="/libs/fitty.min.js"></script>

 <link rel="stylesheet" href="/css/style.css" />
</head>
```

- Google's Poppins Font: This section includes a link to Google Fonts to load the Poppins font family with various weights and styles.

- Bootstrap Lib and Icons: I have used icons to enhance my UI. These links load the Bootstrap CSS framework and Bootstrap Icons, enabling the use of Bootstrap styles and icons in the web page.

- <script defer src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.bundle.min.js" integrity="sha384-w76AqPfDkMBDXo30jS1Sgez6pr3x5MlQ1ZAGC+nuZB+EYdgRZgiwxhTBTkF7CXvN" crossorigin="anonymous"></script>: This script includes Bootstrap's JavaScript bundle (including jQuery) for handling various interactive components and behaviors.

- Axios: This script includes Axios, a popular JavaScript library for making HTTP requests, which enables easy communication with a server or API.

- Fitty Lib: This script includes the Fitty library, which is used to automatically resize the width of the page title to fit within its container.

- <link rel="stylesheet" href="/css/style.css" />: This link loads a custom CSS file named style.css from the /css directory of the web server, allowing for additional custom styling.

In summary, the <head> section of the web page sets up essential meta tags, includes external resources like fonts and CSS frameworks (Bootstrap), and scripts for handling interactivity and making HTTP requests (Axios). Additionally, it includes a custom CSS file (style.css) for further styling.