**Advanced Web Programming**

**Project Documantation**

For this semester's final project, I have developed a messaging application named ScriptChat. Based on your request, the application utilizes MongoDB for its database, ensuring efficient and robust data management. The interface has been designed to be both simple and highly functional, with a strong focus on delivering an intuitive and user-friendly experience. ScriptChat aims to provide a smooth and reliable communication platform for users.

To enhance the user-friendliness of the frontend, Intellisense and DaisyUI have been integrated into the project. These tools greatly simplify the process of writing CSS, making it more intuitive and efficient. Additionally, the use of DaisyUI significantly increases the interactivity of the site, providing a richer and more dynamic user experience. By incorporating these technologies, the development process becomes smoother, and the end product is more polished and responsive.

During the development of my project, I took a meticulous and careful approach to organizing the file directories. By maintaining a well-structured and clear directory hierarchy, I ensured that locating specific elements and variables was straightforward and efficient. This attention to detail in the organization not only streamlined the development process but also made it easier to manage and navigate the codebase, enhancing overall productivity and reducing potential errors.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

In this directory, there is initially a control file (js) that manages the core functions of the website, such as registration, login, and logout. Subsequently, there are files that handle the control statuses of users and messages. To better understand this structure, let's examine the auth.controller.js file:

metin, ekran görüntüsü, menü içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu

This code connects to a MongoDB database using Mongoose. It starts by importing Mongoose and then defines an asynchronous function connectToMongoDB. Within this function, it attempts to connect to the database using the URL stored in the MONGO\_DB\_URL environment variable. The connection attempt is wrapped in a try-catch block to handle any errors gracefully. If the connection is successful, it logs "Connected to MongoDB"; otherwise, it logs an error message. Finally, the function is exported as the default export of the module for use in other parts of the application.

metin, ekran görüntüsü, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

This code defines a middleware function `protectRoute` to secure routes using JWT in a Node.js application. It imports `jsonwebtoken` and the `User` model. The function retrieves the JWT from request cookies, verifies it, and decodes it to get the user ID. If the token is missing, invalid, or the user is not found, a 401 Unauthorized error is returned. If the user is valid, their information is added to the request object, and the middleware calls `next()`. Errors are logged, and a 500 Internal Server Error is returned if any issues arise. The middleware is exported for use elsewhere in the application.

metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu

The `models` directory contains the schemas used in the project, including `conversation`, `message`, and `user.model.js` files. Now, let's take a look at the `conversation.js` file for an overview. This code defines a Mongoose schema for conversations. Each conversation includes an array of participant IDs that reference the User model and an array of message IDs that reference the Message model. Timestamps are automatically added to each document. The schema is then used to create a Mongoose model named Conversation, which is exported for use in other parts of the application.

metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu

This code sets up an Express router for handling authentication routes. It imports the signup, login, and logout functions from the auth.controller.js file and maps these functions to the corresponding HTTP POST routes. The /signup route calls the signup function, the /login route calls the login function, and the /logout route calls the logout function. The router is then exported for use in the main application.

metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu

This code defines a function generateTokenAndSetCookie that creates a JWT using a user's ID and sets it as a cookie in the browser. The JWT is signed with a secret key from environment variables and is valid for 15 days. The cookie is configured to be secure, HTTP-only, and to prevent cross-site scripting (XSS) and cross-site request forgery (CSRF) attacks. The function is then exported for use in other parts of the application.

metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu

And this code sets up an Express server with middleware to handle JSON parsing and cookie parsing. It imports routes for authentication, messages, and users from respective files. The server listens on the specified port (or default port 5000) and connects to MongoDB upon startup. Environment variables are loaded from a .env file using dotenv.config(). The server is then started, and a message is logged to the console indicating the server is running.

**Yusuf Ataş**

**20212022071**