# 📌Project Name - FolioHub

## Developed by - Komal Kiri

## 💻Code and Deployment

- GitHub Repository URL
  - https://github.com/Komal-7/folioHub-server ( Backend )
  - https://github.com/Komal-7/folioHub ( Frontend )

- Live Website URL
  - https://foliocore.netlify.app/ (INACTIVE currently)

## 📚Project Overview

- This web application enables users to **create**, **customize**, and **deploy** personal portfolio websites through a user-friendly visual editor.
- Portfolios are stored in S3 for persistence but **served through dynamic routes** on the **main application's domain** (https://foliocore.netlify.app), allowing users to have **custom portfolio links** like:

  https://foliocore.netlify.app/portfolio/:username

## 🔧Backend Structure

### Backend Implementation

- **Framework**: Node.js + Express for backend, DynamoDB for database, and AWS S3 for storage.
- **Backend Folder Structure**: The backend is modular and divided into:
  - `routes/:` Handles routing logic
  - `controllers/`: Handles HTTP requests.
  - `services/`: Contains reusable business logic.

    - `repositories/`: Handles DynamoDB connection and operations.
    - `s3/`: Manages interactions with AWS S3, including signed URLs.
    - middlewares/ : Handles middleware duties like authenticating users before routing.

- **AWS SDK Integration**:
    - **DynamoDB**: Connected using AWS.DynamoDB.DocumentClient for simplified data operations.
    - **S3**: Interfaced using AWS.S3 for file storage and retrieval.

- **Authentication**:
    - Utilizes **JWT** for session management.
    - Tokens are stored in **HTTP-only cookies** to enhance security

- **Signed URLs**:
    - Generated for accessing private S3 buckets to ensure secure, time-limited access to resources.

- **Deployment**:
    - Instead of exposing S3 URLs, deployment metadata is saved in DynamoDB, and the backend dynamically serves HTML files from the `foliohub-user-deployments` bucket when a visitor accesses a user portfolio URL.

## Database Used

- **Amazon DynamoDB** serves as the primary database, managing:
    - **Users**: Stores user credentials and profile information.
    - **Templates**: Contains metadata for global templates, including S3 keys for template JSON and preview images.
    - **User_Projects**: Holds user-specific project data, including references to S3-stored project JSON files and deployment metadata.

- **AWS S3 Buckets -** The application utilizes multiple S3 buckets for organized storage:
    - **foliohub-templates**
        - Stores global template JSON files and preview images.
        - Accessed via signed URLs for security.
        - DynamoDB stores references (S3 keys) to these files.
    - **foliohub-user-projects**
        - Stores individual user project JSON files.
        - Accessed via signed URLs for security.
    - **foliohub-user-assets**
        - Hosts user-uploaded assets (e.g., images).
        - Files are publicly accessible to enable embedding in portfolios.
    - **foliohub-user-deployments**
        - Contains deployed HTML files for user portfolios.
        - Accessed by the backend to serve content through the main application domain.

## API Endpoints

The backend exposes the following RESTful API endpoints:

- **Authentication Routes**
    - POST /register: Register a new user.
    - POST /login: Authenticate user and issue JWT.
    - GET /user: Retrieve authenticated user's information.
    - POST /logout: Invalidate user session.
- **Template Routes**
    - GET /templates: Fetch list of global templates.
    - GET /template/:id: Retrieve specific template metadata to load in editor.

- **Asset Management Routes**
    - POST /assets: Upload user assets.
    - GET /assets: List user's uploaded assets.
    - DELETE /assets: Delete specified user assets.
- **Project Routes**
    - GET /projects: Retrieve all projects for the authenticated user.
    - POST /save-project: Save or update a user project.
    - POST /deploy: Deploy a user project to generate a public portfolio URL.
- **Portfolio Route**
    - GET /portfolio/:sitename: Serve the deployed portfolio corresponding to the provided site name.

## 🌟Core Features

- **User Authentication**: Secure registration and login system.
- **Template Browsing**: Search and preview global templates.
- **Visual Editor**: Drag-and-drop interface for customizing templates.
- **Asset Management**: Upload and manage personal assets within the editor.
- **Project Management**: Save, view, and manage multiple projects.
- **Deployment**: Publish portfolios to a public URL under the main application domain.
- **User Dashboard**: Access account details and manage deployed projects.

## 📝Feature Details

### User Authentication

- **Functionality**: Allows users to create an account, log in, and maintain a secure session.

- **User Interactions**:
    - Users register with an email and password.
    - Upon login, a JWT is issued and stored in an HTTP-only cookie.
    - Authenticated users can access protected routes and features.
- **Technical Details**:
    - Passwords are hashed using bcrypt before storage in DynamoDB.
    - JWTs are managed using the jsonwebtoken library.
    - Sessions are maintained via secure, HTTP-only cookies.

**Template Browsing**

- **Functionality**: Enables users to explore and select from a collection of global templates.
- **User Interactions**:
    - Users can search templates by keywords.
    - Preview images are displayed for each template.
    - Selecting a template loads it into the editor for customization
- **Technical Details**:
    - Template metadata is stored in the DynamoDB Templates table.
    - Template JSON and preview images are stored in the foliohub-templates S3 bucket.
    - The frontend fetches template data via the /templates and /template/:id endpoints.

**Visual Editor (GrapeJS Integration)**

- **Functionality**: Provides a drag-and-drop interface for users to customize selected templates.
- **User Interactions**:
    - Users can add, remove, and modify components within the template.
    - Styles and content can be adjusted in real-time.
    - Assets can be uploaded directly into the editor.
- **Technical Details**:
    - GrapeJS is integrated into the frontend to facilitate visual editing.
    - Custom configurations are applied to support asset management and template loading.
    - Edited content is serialized into JSON for storage and deployment.

**Asset Management**

- **Functionality**: Allows users to upload and manage personal assets for use within their portfolios.
- **User Interactions**:
    - Users can upload images and other assets through the editor interface.
    - Uploaded assets are listed and can be inserted into the template.
    - Assets can be deleted when no longer needed.
- **Technical Details**:
    - Assets are uploaded to the public foliohub-user-assets S3 bucket.
    - Metadata and references are managed within the user's session.
    - The backend provides endpoints for uploading (POST /assets), listing (GET /assets), and deleting (DELETE /assets) assets.

**Project Management**

- **Functionality**: Enables users to save and manage multiple projects.
- **User Interactions**:
    - Users can save their current work as a project.
    - A list of saved projects is accessible from the dashboard.
    - Projects can be reopened for further editing or deployment.
    - Deployed Portfolio URL shown here.
- **Technical Details**:
    - Project metadata is stored in the DynamoDB User_Projects table.
    - Project content is saved as JSON files in the private foliohub-user-projects S3 bucket.
    - Access to project files is secured via signed URLs.

**Deployment**

- **Functionality**: Allows users to publish their customized portfolios to a public URL.
- **User Interactions**:
    - Users can deploy any one project at a time.
    - A unique URL is generated for the deployed portfolio.
    - Deployed portfolios are accessible to the public.
- **Technical Details**:
    - Deployed HTML files are stored in the foliohub-user-deployments S3 bucket.
    - The backend serves these files through the /portfolio/:sitename route.
    - Access to deployment files is managed via signed URLs to ensure security.

**User Dashboard**

- **Functionality**: Provides users with details of their account.
- **User Interactions**:
    - Users can view their account details.
- **Technical Details**:
    - Account information is retrieved from the DynamoDB Users table.
    - The dashboard is a protected route, accessible only to authenticated users.