CT30A3204 Advanced Web Applications 27.2.2025

AWA-Project-Work Documentation

Simon Ihegbu

Introduction

An implementation of a Kanban board app that lets users register, log in, and create cards on the board's columns. Only authenticated users can create and manage cards, while non-authenticated users cannot. The user can create as many columns and cards as they wish. This Kanban board is called ColabBoard.

Technology choices

The following technologies were used for the project's implementation.

MongoDB: This was mainly used for data management, such as storing user registration and login information. The database holds other related data, such as columns and cards that are created on the board. NodeJS and Express: NodeJS is a JavaScript run-time environment, while Express is a backend framework for building and managing APIs. TypeScript framework is for backend and front-end applications, while React is used for front-end applications. Other implemented services/dependencies include react-router-dom for managing frontend react component routing, Mongoose for helping with the backend routing, and Nodemon for monitoring changes to the application. The respective TypeScript definitions for each dependency were installed to enhance the working of the dependencies. Material UI was implemented on the React side to enhance the user experience.

The selected code editor was the VS code. The VS code is transformed into an Independent Development Environment (IDE) by installing the relevant extensions. Git and GitHub were implemented for version management. Git skills are important even in small projects or when working on bigger or distributed projects.

Features

The following features/requirements were implemented.

- Add columns to the Colab-board
- Add cards to the columns
- Delete cards from columns
- Edit cards,
- Add comments to cards,
- Delete a column from the ColabBoard
- Edit the column title by double-clicking it
- Set the card's title and content's background colors
- Search for cards,
- Save the Board's content to the database

Feature declaration

Features	Points expected
Basic features with well-written documentation	25
Utilization of a frontside framework, such as React	3
Set the card's title background color	1
Set the card's content background color	1
Double-click and edit any edible content, like a column's title, etc.	4
Cards can have comments in them, one or many	3
Provide a search that can filter cards that have the searched keyword	3
Cards and comments have visible timestamps when they were created and updated	4
Cards have an estimated time when the work is done	1
Cards have a progress bar comparing Done vs. Logged vs. Estimated time	3
The user can register/log the time spent with the task/card	1
Test software for accessibility and usability with keyboard/voice command and screen readers	3
Total points	**52**

Enhanced features

The app was implemented and tested for responsiveness. Thus, it can be used on mobile devices (e.g., mobile phones).

A progress control (progress bar) was implemented, which indicates the status of the work with four colors: green, blue, red, and the default color, gray; blue indicates that the task is going on, and it is noticed when the user clicks the "Done" button. Green indicates that the work is completed. Thus, the "Done" hours would be equal to the estimated hours or equal to the logged hours. Red indicates that the task took longer than estimated, and this is noticed when the user clicks the "Done" button when the "Done" hours exceeded the estimated hours. The board, with its related data, is saved in the database.

Testing for accessibility with keyboard/voice command and screen reader

Goal: To ensure that the web page is navigable using a keyboard, voice command, and screen reader

Resources:

Using the local keyboard and a popular screen reader, NonVisual Desktop Access (NVDA). This NVDA was downloaded free of charge from the internet. An alternative screen reader would be Job Access With Speech (JAWS). However, JAWS is paid.

Keyboard testing

1. Used Tab to move through buttons and fields
2. Used Shift and Tab to move back and forth
3. Used the Enter key to update messages when a text was entered in the field
4. Used Space bar to access buttons
5. Used arrow keys to navigate back and forth to access buttons

Result: Pass (Everything worked well).

Accessibility Testing Using NVDA screen reader

Goal: To ensure that the webpage is navigable and readable using automated software (NVDA)

Summary of the test procedure with the expected result

Step	Action/Input	Expected result	Status
1	The app (NVDA) was downloaded from https://www.nvaccess.org/ and launched	After launch, NVDA started reading the screen content	Pass
2	Started the webpage (ColabBoard) in a new browser such as Chrome, Firefox and Edge	NVDA started pronouncing the title of the webpage(app)	Pass
3	Pressed Tab key to navigate through the elements	NVDA pronounces the elements such as buttons, texts, fields, etc.	Pass
4	Pressed Shift and Tab to navigate back and forth	NVDA follows suit and pronounces the elements	Pass
5	Clicked a button using the enter key, or enter a text in a text field and press enter key	When the button was activated, NVDA announced it, and when a text was entered, NVDA announced it	Pass

User manual

A new user registers through the registration page and is then directed to the login page; the user logs in and is authenticated; if the authentication succeeds, the user can start using the board, such as creating cards. A returning users log in through the login page and is authenticated; if the authentication succeeds, the user can start using the ColabBoard.

Installation guide

Frontend installation guide

React was used for implementing the front-end. React is a JavaScript library, and can be installed through Vite, and the developer can install the latest version of it using npm. Create a frontend root folder, and in the root folder, enter npm, create vite@latest, and select your preferences. For this project, React and TypeScript were selected as the main library and framework, respectively. The frontend folder was initiated by navigating into it and entering npm init, which initiates node modules and other files such as package.json, tsconfig.json, and vite.config.ts. All the installed dependencies and future dependencies were viewed and managed through package.json file.

Backend installation

A couple of resources were needed for the backend: the backend hosts the server and the routing. For that, create a root directory and install Nodejs. For installing the Express server, which was used in this project, initiate the directory by running npm init, initiate TypeScript if the TypeScript framework is desired, and run the following (npm install express, npm install path, npm install morgan, npm install nodemon, nodemon can be installed globally by adding the -g flag. Install MongoDB if desired, and install mongoose, a microservice for MongoDB. Install the respective TypeScript support/helper for each of the above resources, such as npm install - -save-dev @types/express, npm install - -save-dev @types/morgan, etc.

Adjusting the tsconfig.json file at the backend

Locate the “compilerOptions” and inside it, uncomment “target” and change its resource to “ESNext”. Similarly, uncomment “module”, “outDir” and “noEmitOnError”. Add “./dist” to “outDir”.

Working with MongoDB

- install MongoDB to your local machine
- npm install mongoose
- npm run start [To run the server]

Full Stack installation

Move the front-end directory to a root directory that holds both the front-end and the backend directories. In the package.json file in the backend directory, adjust the content of the “scripts” to listen to both front and back ends. Also, implement the CorOptions in the server file to enable the backend to share resources with the front end. Install other necessary microservices/dependencies, and these dependencies can be viewed and managed in the package.json files both at the front and backend sides.

AI Declaration

Chatgpt was used to get ideas and learn how some methods were implemented. Chaptgpt was also used for code proofreading, where minor corrections to the code were suggested. However, these minor defects did not originally affect the application's functionality.