UNIVERSITY OF LEICESTER

School of Computing and Mathematical Sciences

CO7201 Individual Project

Final Report

DanceFlow: Personalized Dance Sequence Builder and Scheduler

Dinesha Bungatavula
dlb29@student.le.ac.uk
239058881

Project Supervisor: Dr Paula Severi
Principal Marker: Dr Zedong Zheng

Word Count: 9616
(excluding code & table of contents)
16/05/2025

DECLARATION

All sentences or passages quoted in this report, or computer code of any form whatsoever used and/or submitted at any stages, which are taken from other people's work have been specifically acknowledged by clear citation of the source, specifying author, work, date and page(s). Any part of my own written work, or software coding, which is substantially based upon other people's work, is duly accompanied by clear citation of the source, specifying author, work, date and page(s). I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this module and the degree examination as a whole.

Name: Dinesha Bungatavula

Date: 16/05/2025

Abstract

Dance Flow web application is an innovative platform where the users are provided with a structured and engaging interface. There are many dancers who face the problems in staying consistent and organized to develop their skills, this web application comes with the various features which includes to create, manage and track the progress of their own sessions.

The Dance Flow offers the users to explore the variety of dance moves and personalize their own sequences with the preferred dance moves and schedule them for their later practice using the calendar. This scheduling helps the users to plan their practice sessions according to their convenience.

Progress tracking is one of the main functionalities of this system, which tracks the user's activity and their performance. Users can see the progress of their dance sessions and set the new goals according to their performance levels and improve their dance skills. This application also incorporates an intelligent recommendation system which suggests the users with the practice sessions based on the user's activity and community space for interaction and sharing the sequences and sessions with other users. This Dance Flow web application leverages modern web technologies to provide user-friendly interface to the dancers with different skill and performance levels.

Table of Contents

LIST OF FIGURES

1. Introduction

In the era of digital transformation, the arts and education are rapidly adapting to the interactive virtual platforms [14]. Dance Flow is a digital platform that has been designed to full-fill all the necessary requirements of dancers and it bridges the gap between the traditional practice methods and innovative digital platforms [15]. The Dance Flow web application is integrated with the library of dance moves containing images and videos [16], sequence customization, calendar for scheduling and progress tracking features which allows the users to stay organized, schedule and track the progress of their dance practice sessions along with the community space to interact with the other users.

1.1 Aim

The main aim of this project is to develop a Dance Flow Web application that provides the users with intuitive and flexible platform. In this platform the users are allowed to create and customize their dance sequences. With the help of this personalized dance sequence builder users will be able to develop and refine their dancing skills. This platform is suitable for all the dancers with different skill levels. They can choose and create their own sequence according to their dancing skills with the dance moves. The calendar has been integrated into this platform which helps the users to schedule the dance practice sessions on their own at their convenience. Additionally, the integration of progress tracking features helps the users to monitor the progress of their dance practice sessions at any time, set new goals and focus on the areas where they need improvement. This web application makes it easier for the users to organize and finish their sessions on time.

1.2 Objectives

The main objectives of the Dance Flow Web application include

- **Secure user authentication:**
  Implementing the JWT-based secure user authentication system that allows the users to create and login with their accounts securely. This system also includes the role-based access control which makes sure a clear distinction between the admins and the regular users.

- **Develop a user-friendly interface:**
  Designing a clean, interactive and easy-to-navigate interface for the users to easily explore the dance moves, create and personalize the sequences of their own choice. This also focuses on providing a great experience for both the beginners and advanced users.
- **CRUD operations:**
  Enabling the full CRUD (create, read, update and delete) functionalities for both the users and admin. Users can create and modify their sequences as needed whereas admin can perform these functions for the dance moves in the database which improves the flexibility of this system.
- **Integration of interactive Calendar:**
  Integrating a dynamic calendar system that allows the users to schedule their own dance practice sessions and manage them according to their preferences effectively. This feature also allows the users to stay organized and set reminders for their practice sessions to maintain the consistency of their dance routines.
- **Progress tracking:**
  Enabling the progress tracking system to store and visualize the user's activity and performance which helps the users to monitor their progress and set the new goals for their practice sessions. This also levels up the motivation for the users to practice their sessions.
- **Auto recommendation system:**
  Implementing the auto recommendation system to suggest the users with the next recommended session in the calendar based on the user’s past scheduled and completed sessions. This helps the users to maintain their consistency in the dance learning.
- **Community features:**
  Designing a community space where users can share sequences and sessions with the other users in the application and communicate with others via threads and posts. This also allows the user to create a challenge in which the other users can participate and do comments in the challenge to communicate with the other participants.

2. Literature Review

2.1 Introduction

Dance Flow, a web application aimed to provide the dancers with optimized dance practice scheduling system and enhance the personalized and interactive learning experience of the users. This literature review explores the key aspects of this project which includes the interactive and personalized learning experience, virtual dance education, personalized scheduling system, progress tracking and community engagement for the users.

2.2 Personalized and Interactive learning

There are huge number of dance learning platforms which provide the users with pre-recorded instructional videos and structured sessions among different dance styles for dance learning, but they all lack with the personalized and interactive learning which is the main feature for the users to enhance their learning experience. Research on personalized learning highlights the importance of interactive learning experiences [1]. The increase of web-based learning has created a room for all the learning sectors including dance. This web based personalized learning helps the users to develop the skills on their own and it leads to improved learning outcomes when supported by content adaptation [2]. Interactive learning emphasizes the user engagement with the content and the environment. The interactivity in multimedia learning environment improves the learner's engagement with the content and demonstrates higher cognitive retention [3]. The integration of all the necessary features will allow the users to improve their experience with personalized and interactive learning.

2.3 Virtual Dance education

Virtual dance education refers to the digital learning platforms which have a rapid growth and evolution. Especially, during and after the COVID-19 pandemic the virtual dance education has changed as a crucial mode of dance learning for all the dancers and choreographers. Virtual dance learning platforms had a significant growth during the pandemic as all the dancers migrated to the digital platforms [4]. These platforms provide the flexibility and accessibility to have an

interactive learning environment [3]. The main core of the virtual dance learning platform is the content of the dance moves and the flexibility to access them. The online dance learning is also growing in styles and categories which includes ballet training and many others [5]. The growth in particular styles and categories will help the dancers to learn what they prefer.

2.4 Personalized Scheduling System

The use of personalized scheduling system in the web application is more important for time management and learning consistency. This always allows the users to schedule their sessions on their own without missing any sessions. This scheduling system can be done by the integration of calendar libraries such as FullCalendar.js and Tui.Calendar due to their API rich environment and real-time synchronization. This visual representation of upcoming sessions in the calendar increases the user engagement and improves their learning experience. This not only improves the usage of the system but also the adaptive learning experience of the users [6]. Moreover, the integration of personalized scheduling system will provide a potential for data collection and suggesting the optimal sessions for the users based on the user's activity. This recommendation of sessions to the users will enhance the interaction with the system [7].

2.5 Progress Tracking

Progress tracking is the feature in the digital platform that allows the users to monitor their learning progress and performance. This feature is implemented based on session history and activity of the users.This stored session history of users in database is retrieved and shown for the users by using libraries such as chart.js. This progress tracking also helps the users in continuous improvement and to stay motivated with their work [8]. By adapting this progress tracking functionality, the digital platforms offer creative and gamified incentives like badges and awards for performance [9]. These gamified incentives will level up the user's activity and performance in order to achieve them and set new goals to gain those incentives. This progress tracking will help all the levels of users to track their progress, improve the skills and level up their performance.

2.6 Community engagement

Community features such as sharing the sequences, adding comments and having chats are important for the modern learning platforms as they provide the engagement to the users with other members [10]. This communication helps them to have shared learning experiences and, the users will have the space to learn new things from other users. This shared learning helps the low skilled users to engage with the high skilled, learn and increase their level of performance. Research shows that the digital platforms with strong social skills and easy communication will directly boost the participation and usage [11]. This shared learning experience will significantly enhance the motivation and confidence of the users [12].

3. Requirements

This section has all the necessary and specific requirements of the project. These requirements are divided into the following three categories which are essential, recommended and optional.

3.1 Essential Requirements

The essential requirements are the important ones that should be met to achieve the base of the project. The following are the essential requirements:

- User registration: The users should be able to register and login to their securely. All the user information must be stored safely, and the passwords of the users should be hashed perfectly before storing them. The admin and regular must be taken to their respective dashboards without any mismatch of accessing the data with the help of role-based access control.
- Admin role: The admin role has the ability to create, read, update and delete the dance moves that are stored in the database and these changes should reflect the user dashboard. This role is critical to keep the content up to date ensuring that the learners have access to all the resources and to manage the user accounts.
- Database: A well-structured database that stores all the information of the users, dance moves, sequences and the scheduled dance practice sessions in their respective collections.

- Sequence creation: The users will be able to create and customize their own sequence by combining multiple dance moves that are available in the database for their personalized learning. These sequences are stored in their profiles and can access them whenever they login to their accounts.
- Progress tracking: This functionality will track the progress of the users based on their activity and the sessions they have created and practiced. This will give a visual representation of the progress to the users to keep the track of their dance practice sessions.
- Community features: Users will be able to share their personalized sequences with their preferred members who are existing in this application.

3.2 Recommended Requirements

The recommended requirements are the extended functionalities of the project that need to be done. The following are the recommended requirements:

- Role-based access control: This ensures that the admin and the users are directed to their own dashboards to make sure that the admin data is not accessed by the regular users. This will have the appropriate permissions for different roles. This separation allows the system to manage and access the content securely and efficiently.
- CRUD Operations:
  The CRUD operations are essential for both the admin and user to manage and interact with the content of the application effectively.
  - Admin role: The admin role has the CRUD operations to make the changes in the dance moves and makes sure the content of the dance moves is up to date.
  - User role: The CRUD operations for the user role is to create, read, update and delete their sequences and sessions as they only have the access to their own personalized sequences and sessions.
- Advanced features: The advanced search features like keywords and filter-based search options for easy access to the content. It will not only enhance the user experience but also saves time and makes navigation efficient mainly for the users with specified goals and preferences.
- Calendar Integration: The integration of calendar into the system will allow the users to schedule their dance practice sessions with their preferred sequence for their later practice. This will visually display the upcoming

sessions of the users and help them to be prepared for that. This also allows the user to set remainders for their dance practice sessions without missing any of them.

- Notifications: The system will be automatically sending the notifications to the user about their scheduled sessions as a remainder before the start date of the session.
- Auto recommendations: The system tracks the user's activity and the sessions history, considers if any gap between the sessions of the users and sends the recommended sessions to the users based on them.

3.3 Optional Requirements

The Optional requirements are the additional requirements that improve the project. The following are the optional requirements:

- Profile customization: The users will be able to customize and update their profile and details whenever they want, including the name and password. They can also reset their password before logging into their account if they forget it.
- Offline mode: Users have the option to download their sequences, if they want to practice them later which is more flexible.
- Voice Commands: Voice commands for the dance practice sessions like Start and Stop if the user wants to do in the middle of the dance practice session.

4. Technical Specifications

Frontend Technologies:

- HTML5: Used for building a structured, easily accessible and visually clear pages which enhances the navigation experience for both admin and user interfaces.
- CSS: Used for styling of the pages and to implement visually attractive and responsive web pages with interactive elements such as forms, buttons and modals which enhance the user interactivity and usability.
- JavaScript: Used for managing the dynamic web page behaviours such as event-handling, DOM manipulations, content loading and real-time updates.

Backend Technologies:

- Node.js: Used to run the server-side JavaScript and it ensures efficient server-side operations and seamless client-server interaction.
- Express.js: The framework that has been used to serve the static files, it simplifies the API endpoint creation, routing and middleware handling.

Database:

- MongoDB: Used for storing the data of users, dance moves, sequences and sessions. It has been chosen for it's scalability and flexibility as it supports the complex document structures.
- Mongoose: Used for the connection between the backend node.js and the database MongoDB.

Authentication:

- JWT Authentication: Used for secure user authentication with token-based verification and ensuring the security and data privacy.
- bcryptjs: Used for secured password hashing and encrypting them safely.

Libraries:

- Nodemailer: A node.js module that will send the emails easily. It has been integrated to send the email notifications and for the forgot password functionality.
- Multer: A middleware for node.js and express.js for easy handling of file uploads. This will handle all the image files uploaded by the admin for the dance moves.
- tui.calendar: It has been used to have an interactive calendar for the user to schedule their dance practice sessions and set the remainders.
- Choices.js: A lightweight JavaScript library that allows the advanced searchable dropdowns and multiple selections. It has been used to have the searchable dropdown of dance moves for users while creating a sequence.
- Chart.js: This has been used for the visual representation of progress of the users in the form of bar charts.

Testing:

- Postman: This is used for testing all the backend routes of the project to check the correct behavior.

5. System Design and Architecture

The Dance Flow web application adopts a layered, modular architecture which has Model-View-Control (MVC) design pattern. This ensures the maintainability and scalability of the system. It consists of three main layers Data layer, Presentation layer, Application layer and are supported by the backend logic and APIs for the client-server communication.

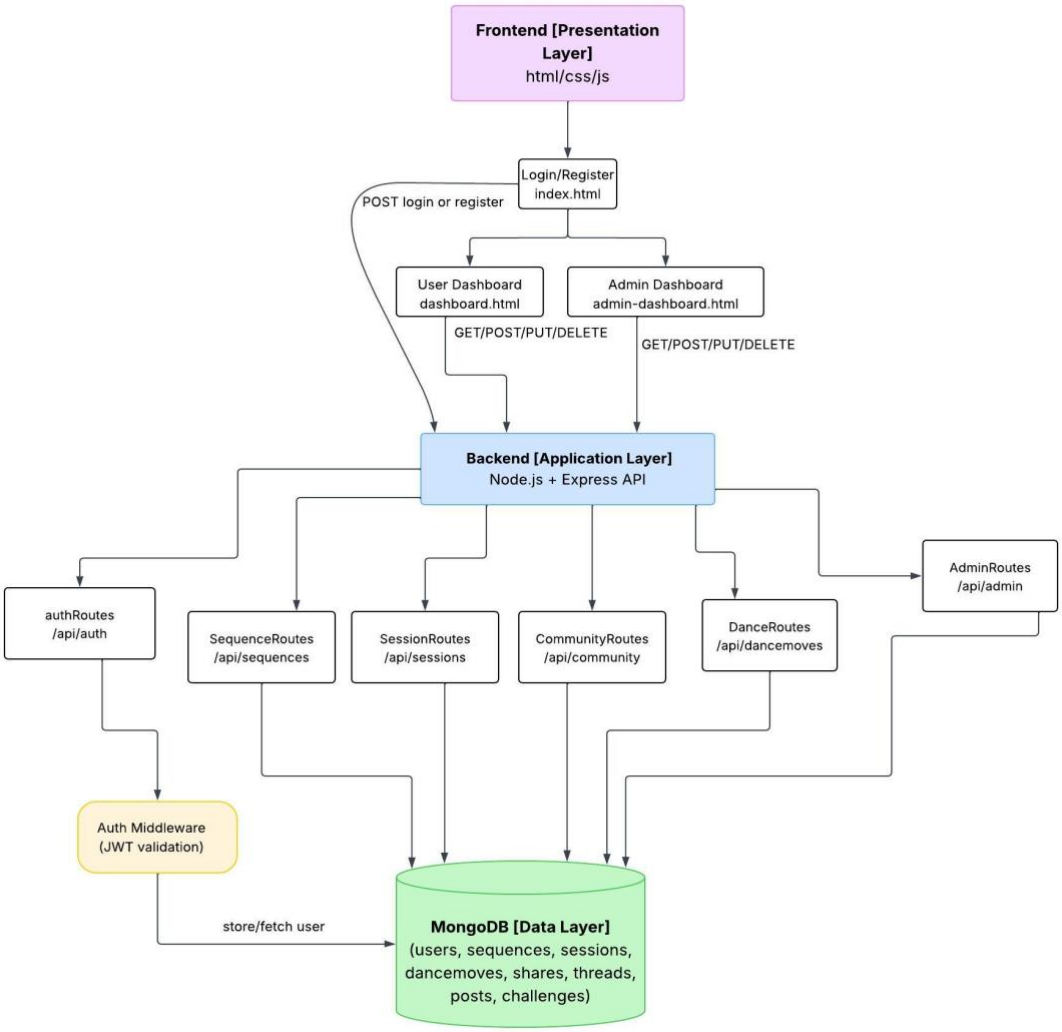

Figure 1: System Architecture Diagram

5.1 Presentation Layer (View)

The Presentation layer is the place where user interacts directly with the system. This layer is built using HTML, styled with the responsive CSS and the JavaScript to create the user interface (UI) and to manage how the content is displayed and being interacted with the users. This layer is separated into two different roles: user and admin; with their respective custom dashboards and functionality. To get the users with their correct dash boards role-based access control is implemented.

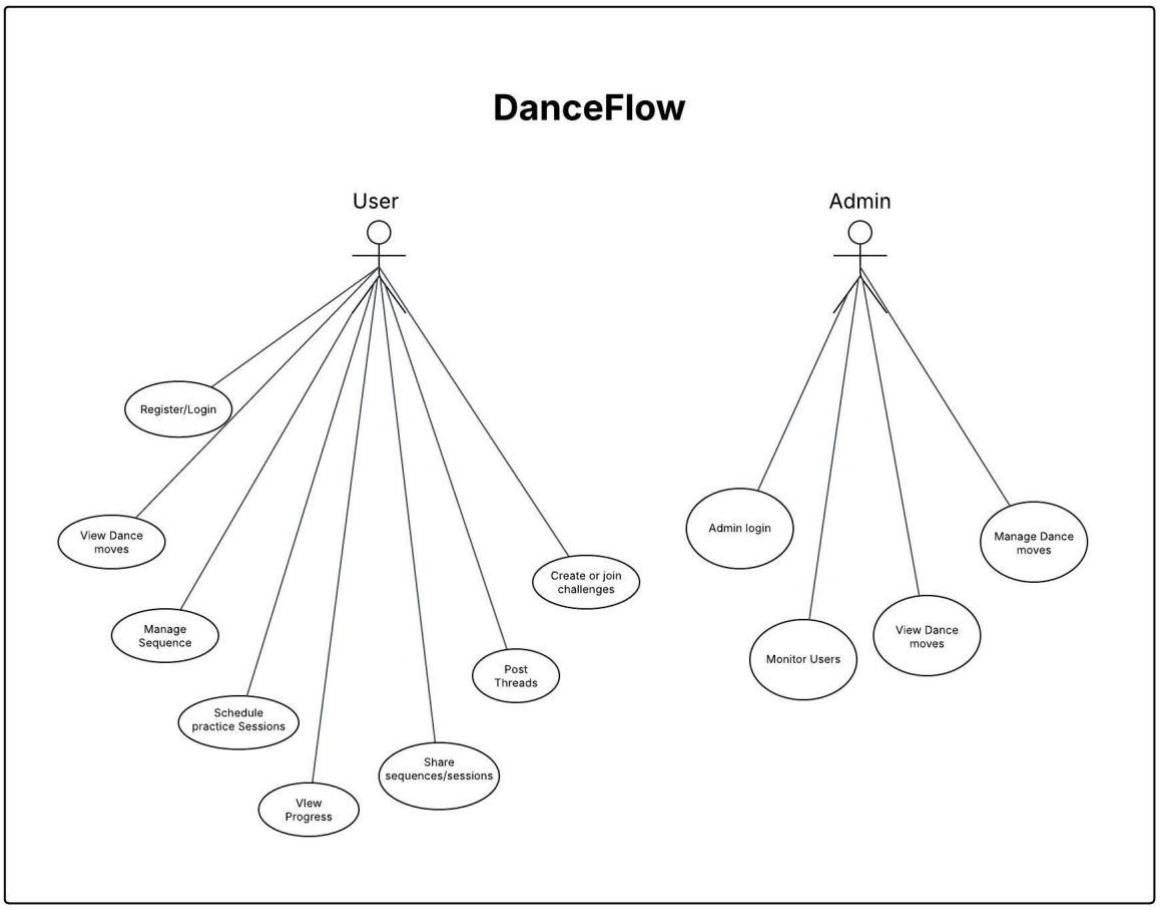

Figure 2: Use case Diagram

Admin Dashboard:

The Admin Dashboard is a management interface that is tailored for the administrative tasks like managing the dance moves content that displays on the user dashboard and to monitor the user accounts. The admin dashboard is designed with the following sections:

- All Users: This section displays all the users with their name, email and role to the admin with a searchable table. When the admin clicks the "ALL USERS" section in the side navigation bar, the frontend sends the request and gets the data from the backend. This section allows the admin to monitor all the user accounts.
- Add Dance Move: This section allows the admin to add the dance moves with their name, category, description, video and image via Multer. Once admin fills all the fields in the "Add Dance Moves" form and uploads it, the dance move is then can viewed by the users in their dashboard.
- All Dance Moves: This is the section where the admin will be able to view all the dance moves added and this section also allows the admin to edit and delete the dance moves by searching and filtering them by category. If the admin wants to edit any of the dance moves the edit button will take to the "Edit Dance Move" form where any detail of the dance moves can be edited. The delete button allows the admin to delete the dance move.
- Edit Profile: In this section admin can update their personal details such as name and password.
- Logout: This button on the navigation bar allows the admin to log out of their account.

Logic Flow:

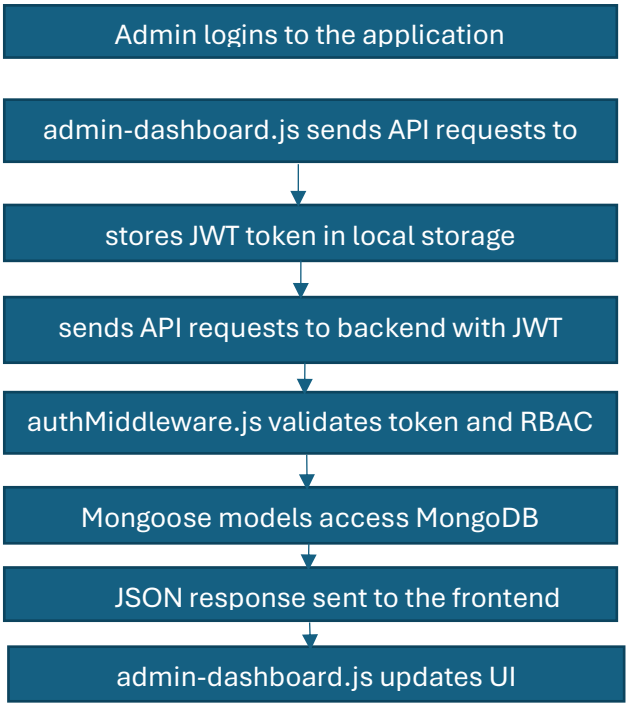

Figure 3: Workflow of Admin Dashboard

User Dashboard:

The User dashboard is the place where the regular users will interact with the system. This dashboard allows the users to do all their necessary activities in the process of learning the dance. In this the users can manage their profile details, explore the dance moves, build their custom and personalized sequences with the dance moves and save them for later practice. They can also schedule their own sessions with the in-built calendar available in the dashboard and keep the track of their learning progress as well. The user dashboard has the following sections to perform each functionality:

- Profile: This section will display the details of the user such as name, email and role once they are logged in.

- Dance Moves: This is the section where the users will be able to explore all the dance moves with images and videos by searching them with keywords and filtering them by category.

- Dance Sequences: This section displays all the sequences created by the users with a keyword and filterable search by dance move and the sequences in this section can be edited and deleted by the user. The edit button in the table will be giving the edit form after the click on it, in which the user can edit all the details of their custom and personalized sequence. The delete button in the table will delete the sequence. The users can also share the sequences with the other users in the application and download them for the offline practice.

- Create Sequence: The create sequence section will open the “Create New Dance Sequence” in which the user must fill the details of the sequence such as name, description and add some dance moves from the dropdown to create their own custom and personalized sequence.

- Calendar: The Calendar section helps the users to schedule their own dance practice sessions with the sequence created by them. This calendar will allow the users to mark their sessions as done and the users can edit the sessions created by them. They can also share the sessions with the other users in the application. The system will get the user’s activity of sessions

and gives them the recommended sessions in the calendar. Overall, the users can visually manage their own dance practice sessions in the calendar.

- My Progress: This section visually displays the progress of the user's dance practice session in a table format. This section has the date format in which the user can search their session by entering from and to dates. They can also search their scheduled dance practice sessions with the sequences and status of the session.

- Community: This section has all the shared information of the user. The user can see the shares done by them with the other users and they can check what the other users have shared with them. This section also contains the threads and challenges. In threads the users can post, and all the other users can see and reply to the posts. In the challenges section users can post the challenges and join them, they can also do the comments for the joined challenge.

- Edit Profile: In this section the user will be able to change their personal details such as name and password of their account at any time.

- Logout: This logout button on the navigation bar will allow the user to get logged out of their account.

Logic Flow:

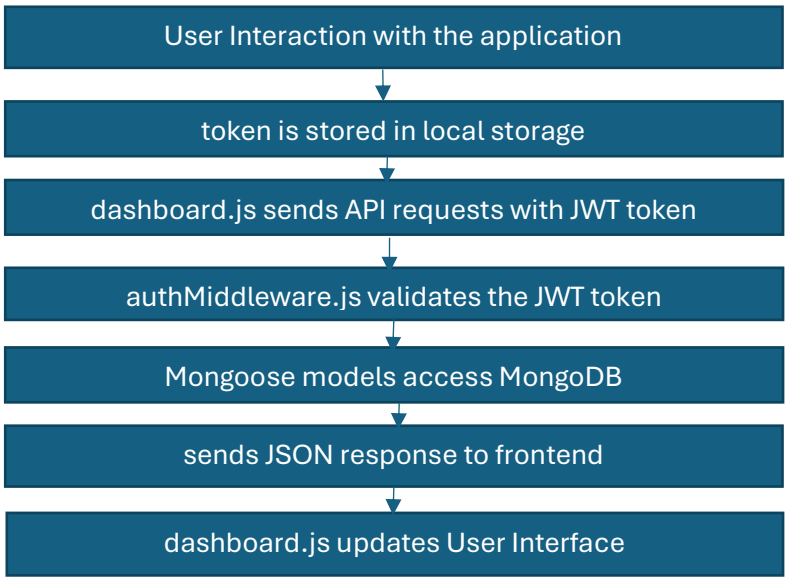

Figure 4: Workflow of User Dashboard

5.2 Application Layer (Control):

The Application Layer is the main layer of the Dance Flow system. This layer processes the requests from the frontend, applies the logic and interacts with the data layer to get the requested data and sends the result to the frontend. It is built and runs on Express.js. It consists of express route handlers, authentication with JWT, role-based access control, CRUD operations, admin only controls and the server setup.

JWT Authentication (authMiddleware.js):

The authentication system will ensure the secure login of the users to their accounts. This is done in the Dance Flow system like the below:

- Login Flow: When the user logs in to the application using their email and password, the details are checked by interacting with the database and creates a JWT token if the user exits. The token is then sent to the front end and stored in the local storage. Every future request will include this token in the authorization header.
- Token Verification: This is the way to keep the application secure. This will check the token every time the users make a request like sequence creation, Session requests etc., Once the front end sends token with every request that needs protection, the backend reads the token form the header and sends the data of the user based on request from the data base. If the token is invalid or expired the route denies the access.

Role-based access control:

The role-based access control will make sure that different roles have their appropriate permissions. There are two different roles in Dance Flow: Admin and User. The admin will have the access to manage the global content like the dance moves and to monitor the user accounts. The user will have the permissions to access and manage their own sequences, scheduling sessions and the profile. This Role-based access is implemented as below:

- When the user logs in to their account, the role is included in the JWT token.
- After verifying the token, the users are taken to their respective dashboards.

- If the regular user tries to access the admin dashboard, it prevents saying “Admin access only”.

CRUD Operations:

In the Dance Flow Web application, the CRUD operations are implemented for both the users and admin for different sections. The CRUD operations for admin to manage the content and for the user to manage their own practice sessions, sequences, threads and challenges.

Admin role (adminRoutes.js):

The allows the admin to manage all the content of dance moves. The admin can create the new dance moves by uploading all the details of it and then it is stored in the database. The admin can update the dance move by editing it in the edit form and can also delete it from the database.

User role:

The user role has this CRUD operations for creating their custom and personalized sequence and to schedule their dance practice sessions using calendar.

Sequence Creation (SequenceRoutes.js):

The authenticated user can create and customize their own sequences. The users can create, read, update and delete their sequences, the user ID and the dance moves are then stored in the database along with created sequence in the sequences collection.

Scheduling Session (SessionRoutes.js):

Users can schedule their dance practice sessions using the calendar in their dashboard. The users can also view, update and delete their practice sessions. The sessions created are stored in the Sessions collection of data base with the user ID, selected sequence, date, description and duration of the session.

Community features (CommunityRoutes.js):

The Community section for the users has the information about their shares of sessions and sequences with the other users and they can view all the threads and challenges posted by them and the other users. The user can only delete the

thread or challenge posted by them while others can only view or do replies and comments for it.

Server Setup (server.js):

The Server is the main part of the system which is the entry point to the backend. This helps to run the whole application, connects it to the database and links all the routes.

- The server connects to the database to run the application
- Each route is linked to its specific route handler where all the API logic is defined.
- Server static files like HTML, CSS and JS.
- Starts the server and confirms the server is running.

Logic Flow:

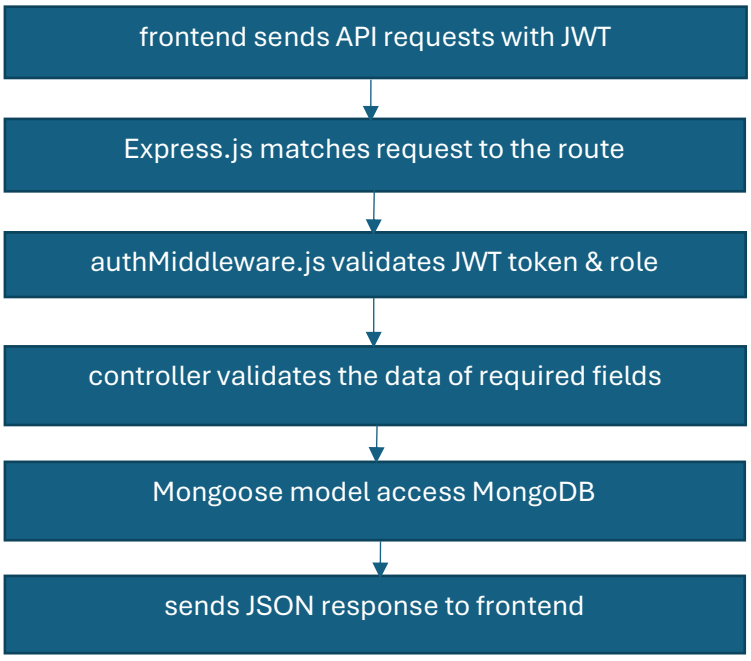

Figure 5: Workflow of Application layer

5.3 Data Layer (Model)

The data layer consists of all the models required for the Dance Flow Application and this is implemented using Mongoose which defines the clear structure for all the core entities in the application. This ensures the consistency, querying and easy validation across all the database collections. This data layer uses MongoDB a scalable, document-oriented NoSQL database. The data layer consists of the following models:

User Model (User.js):

This schema defines the structure for both the user and admin with the following fields

- name: the full name of the user added by them.
- email: the email id of the user which should be unique and used by the user to login.
- password: The password to be hashed by using bycrypt.js for the security.
- role: This defines the access and permissions to all the user whether it is admin or user.
- resetToken/resetTokenExpiry: These are used for the password reset of the users.
- timestamps: The time stamps are for getting the created and updated data of the user accounts.

Dance Move Model (DanceMove.js):

This schema holds all the data of dance moves content added by the admin in the admin dashboard. This stores the data of dance moves including the following fields:

- name: to store the name of the dance move.
- category: to have the category or the type of dance move.
- description: to provide a textual explanation and describe the dance moves.
- image: to store the uploaded image of a dance move and present it to the users.

- video: to store the video of the dance move with the URL link.
- createdBy: this refers to the admin that has created the dance move.
- timestamps: this is for tracking the updates of the dance moves content.

Sequence Model (Sequence.js):

This model represents the custom sequence created by the user and stores it with the respective user id. This has the following fields in it:

- name: The name of the sequence created by the user
- description: the description of the sequence that has been added while creating the sequence.
- moves: It is an array of the dance moves that are selected by the user from the existing dance moves to create a sequence.
- timestamps: this to track when the sequence is created and updated.

Session Model (Session.js):

This schema is to store the individual practice sessions scheduled by the user. This has the following fields to have the session data of the users.

- user: this is to get the details of the user who created the session.
- sequence: this for linking the session with the sequence that will be practiced in that session.
- date: the date of the session when it is scheduled in the calendar.
- description: the description of the session that should be added by the user when creating a sequence.
- completed: this stores the completion status of the session.
- createdAt: this is to track when the session was created.
- duration: it is for the user to set the duration of their session; the system will set 30 min of duration by default if the user doesn’t set any.

Share Model (Share.js):

The share model is for the users to share the sequences and sessions with the other users in the application. This has the following fields in it

- from: the user Id of the user who has shared the content.
- to: the user Id of the user who received the shared content.
- type: it describes what is shared (sequence/session) in the share.
- reference: it stores the Id of the item that has shared (sequence Id/session Id).
- caption: this is an optional text message that user can ass while sharing.

Thread Model (Thread.js):

This schema is for storing the threads started by the user in the Dance Flow Community. This stores the following required information of the threads.

- title: the title of the thread given by the user while creating it.
- createdBy: it stores the user Id of the user who created the thread.
- createdAt: this records the date of the thread when it was created.

Post Model (Post.js):

This model holds individual replies posted by the user for the specific thread and this has the following required fields

- thread: this stores the Id of the specific thread in the reply has been posted by the user.
- author: it has the user Id of the user who has posted the reply for the thread.
- content: it stores the content of the reply post for the thread.
- createdAt: this records the date when the post was done.

Challenge Model (Challenge.js):

This is the schema to store the information about the challenges in the community section. This allows the users to create and join the challenges with the following required fields

- name: it has the name of the challenge created by the user.
- description: the description of the challenge added by the user while creating.

- endsAt: a date should be given by the user that when the challenge expires.
- creator: it stores the user Id of the user who created the challenge.
- participants: it stores the user Ids of all the participant users who joined the challenge.
- comments: this has an array of objects with the user Id and the content of comments posted by the user for the challenge.

Logic Flow:

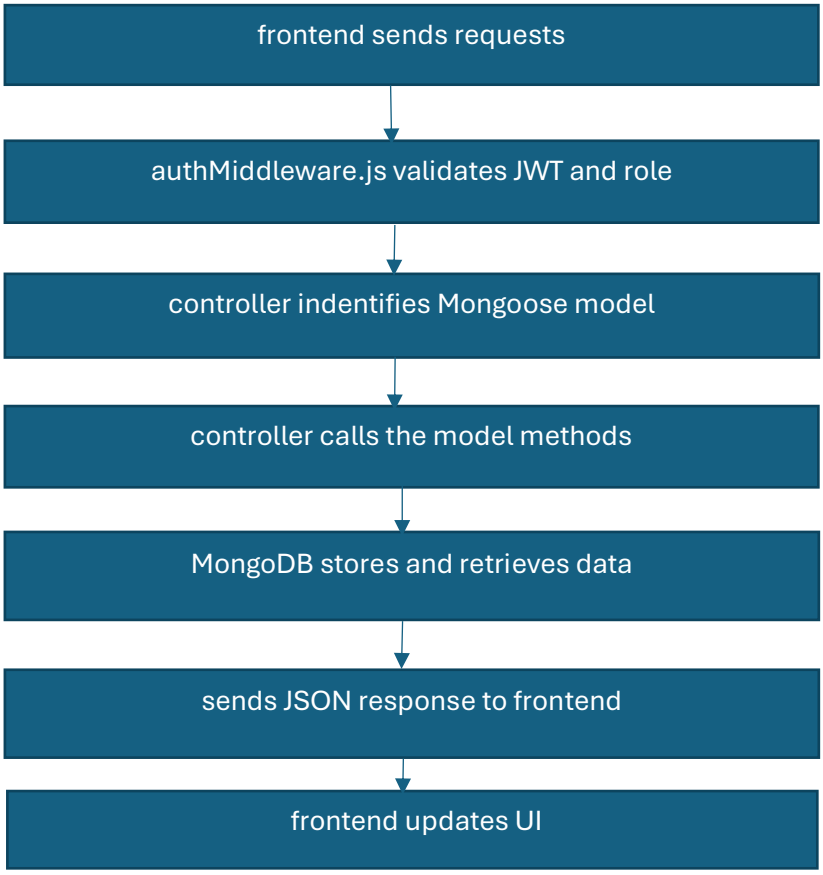

Figure 6: Workflow of Data layer

5.3.1 Entity Relationship Diagram:

The Entity Relationship diagram represents the data model of the Dance Flow application. This shows the relationship between core entities. It's main purpose it to illustrate the primary entities, attributes of each entity and the relationship between the entities.

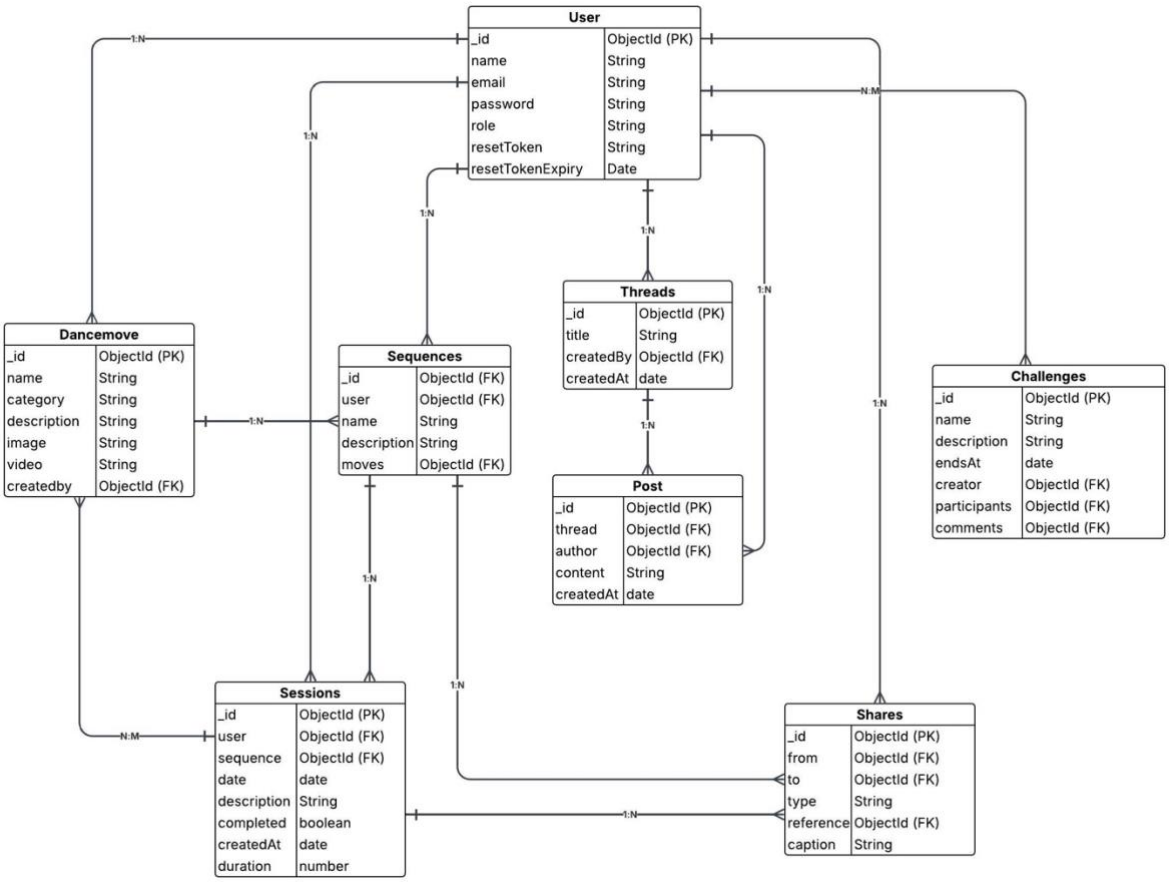

Figure 7: Entity Relationship (ER) Diagram

User Collection:

This is the core collection that stores all the data of both Admin and regular users. This has the following relations with the other collections

1: N with Dance Move - A user (admin) can create multiple dance moves.

1: N with Sequences - A user can create multiple sequences.

1: N with Sessions - A user can schedule multiple sessions.

1: N with Shares (from) - A user can send multiple shares.

1: N with Shares (to) - A user can receive multiple shares.

1: N with Threads - A user can create multiple threads.

1: N with Posts - A user can do multiple replies (posts) on a thread.

N: M with Challenges - Users can participate in multiple challenges and challenges can have multiple participants.

Dance Move Collection:

The dance move collection holds all the data of dance moves created by the admin including multimedia files and it has the following relations

1: N with Sequences – A single move can be added to multiple sequences.

Sequences Collection:

This collection stores the data of the sequences created by the user using the dance moves. The relations of this with the other entities are

1: N with Sessions – One Sequence can be added to multiple sessions.

1: N with Shares – One Sequences can be shared multiple times to multiple users.

Sessions Collection:

This collection has the data of the scheduled sessions by the users for their own practice. This collection has one indirect collection with the dance move

N: M indirect with Dance move – this relation is via sequences.

1: N with Shares – One Session can be shares multiple times.

Shares Collection:

The shares collection holds the data of the shares of sequences/sessions between the users. This collection acts as a bridge for sharing of sequences and sessions between the users.

Threads Collection:

The threads collection stores the data of the threads created by the user with all the required fields and it has only one relation with the other collections which is 1: N with Posts – A single thread can have multiple posts.

Posts Collection:

This collection stores the data of each post done by the user for the threads in Dance Flow Community. This belongs to one thread and one user.

Challenges Collection:

This challenges collection handles the data of the challenges created in the community section and it has

N: M with User – Challenges can have multiple participants and users can join multiple challenges.

6. Implementation:

This section provides the overview of the implementation of Dance Flow Web application. This gives step-by-step process of the implementation which includes the system requirements like server setup, user authentication, role-based access control, dance moves and sequence creation, session scheduling, progress tracking, recommendation system and community features. To implement these requirements Node.js and Express.js were selected for backend development due to their scalability and MongoDB for database due to it's flexibility and maintainability. The frontend stack includes HTML, CSS, JavaScript and the other libraries.

6.1 Server Setup

The Server in the Dance Flow is the core part of the application which maintains the connection between user interface and the database. The Server was implemented using Node.js and Express.js. The sever is initialized in the server.js and this acts as the entry point for the backend. The server helps in establishing the connection with the database, handling the API routes, serving the static frontend files, error handling and starting the HTTP requests.

```
connectDB()
    .then(() => console.log("MongoDB Connected Successfully!"))
    .catch((err) => {
        console.error("MongoDB Connection Error:", err);
        process.exit(1);
    });
```

```
const PORT = process.env.PORT || 3000;

app.listen(PORT, () => {

    console.log(`Server running on http://localhost:${PORT}`);

});
```

6.2 User authentication

User authentication is essential for securing the Dance Flow web application. This user authentication process involves the user registration, login, password hashing and token generation. This is implemented with the combination of JWT validation and password encryption.

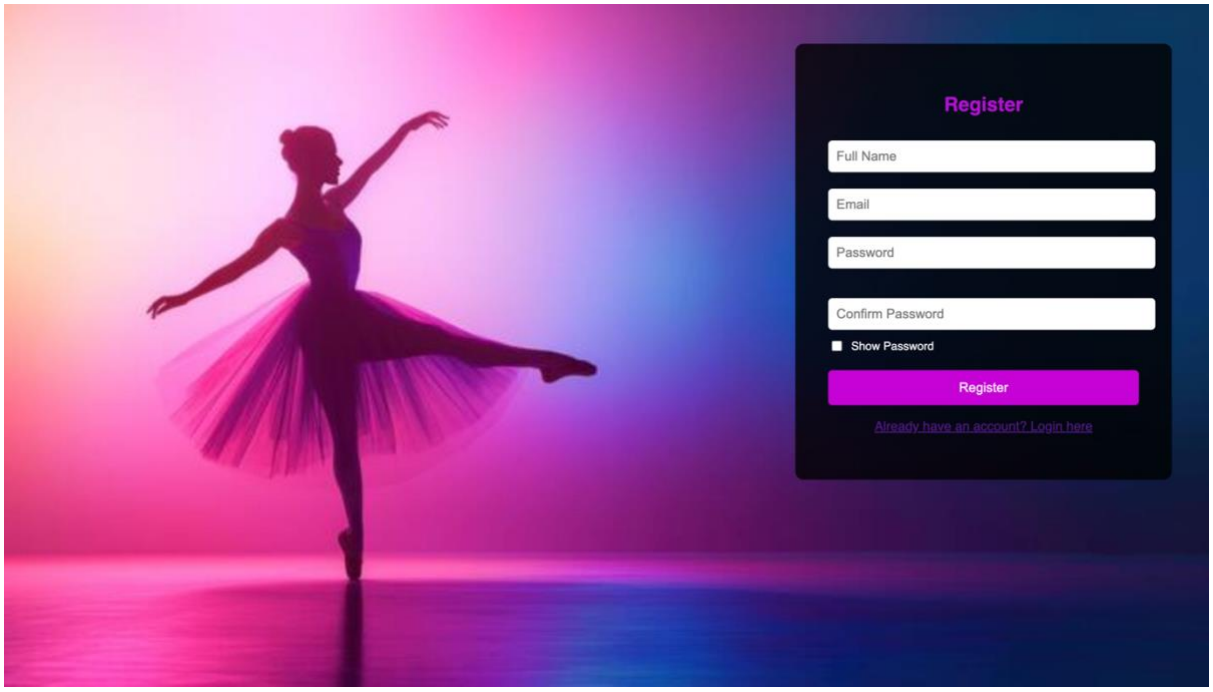

Figure 8: Index page of DanceFlow Web application

- Register:

In the index page of Dance Flow web application, the new user can register with their name, email and password. Once the user clicks on register button, before saving it to the database the password is hashed using bcrypt to prevent the storing of plain text passwords in the database.

```
  registerForm?.addEventListener("submit", async (e) => {

    e.preventDefault();

    const name = document.getElementById("name").value.trim();
```

```
    const email = document.getElementById("email").value.trim();
    const password = passwordInput.value;


    const res = await fetch("/api/auth/register", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ name, email, password }),
    });
    const data = await res.json();
    alert(data.message || data.error);
    if (res.ok) {
      switchToLogin();
    }
  });
```

```
const hashedPassword = await bcrypt.hash(password, 10);
        user = new User({ name, email, password: hashedPassword,
            role: "user" });
        await user.save();
```

- Login:

Once the user is registered into the application, they can login by using their credentials such as email and password in the index page. At login the system checks the credentials, the submitted password is compared to the hashed password stored in the database. If the login is successful JWT token is generated with the user id and role.

```
loginForm?.addEventListener("submit", async (e) => {
    e.preventDefault();
    const email = document.getElementById("loginEmail").value.trim();
    const password =
document.getElementById("loginPassword").value.trim();
```

```
    const res = await fetch("/api/auth/login", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ email, password }),
    });

    const data = await res.json();
    if (res.ok && data.token) {
      localStorage.setItem("token", data.token);
      localStorage.setItem("userId", data.userId || data._id);
      window.location.href = data.role === "admin" ? "admin-
dashboard.html" : "dashboard.html";
    } else {
      alert(data.error || "Login failed");
    }
```

```
const isMatch = await bcrypt.compare(password, user.password);
        if (!isMatch) return res.status(400).json({ error: "Invalid
credentials" });
        const token = jwt.sign({ _id: user._id, role: user.role },
process.env.JWT_SECRET, { expiresIn: "12h" });
```

The token is then sent to the frontend which will be stored in the local storage. This token can be later used for future requests.

- Forgot Password:

The user can reset their password if they forget it from the forget password link in the index page. Once the user clicks on the forgot password link it sends request to the backend, and the backend finds the user by email id and generates a reset token with expiry and sends an email to the registered email id with the reset password link in which the user can reset the password by clicking on it and sends the JSON response to the frontend. The email is sent to the user with the help of node mailer library [17].

```
const response = await fetch("/api/auth/reset-password", {
            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({ token, newPassword })
        });
```

```
const resetToken = jwt.sign({ userId: user._id },
process.env.JWT_SECRET, { expiresIn: "1h" });
```

```
const resetLink = `http://localhost:3000/reset-
password.html?token=${resetToken}`;
        await transporter.sendMail({
            from: process.env.EMAIL_USER,
            to: user.email,
            subject: "Password Reset Request",
            html: `<p>Click the link below to reset your
password:</p><a href="${resetLink}">${resetLink}</a>`,
        });
        res.json({ message: "Password reset email sent!" });
```

- JWT validation:

All the protected routes in the application need a valid token. The middleware (authMiddleware.js) acts as security to check the incoming requests and validate the JWT tokens. If the token in the request is missing, invalid or expires it blocks the access and returns an error message to the frontend. This makes sure the system remains secure and safe from unauthorized access.

```
const actualToken = token.split(" ")[1];
    const decoded = jwt.verify(actualToken, process.env.JWT_SECRET);
    const user = await User.findById(decoded._id);
```

6.3 Role-based access control

Role-based access control (RBAC) is implemented to have the access based on roles for the dashboards. This prevents the regular users from accessing the

admin only content. This is designed to provide the access based on the role of the user embedded within the token.

Admin role:

The admin role has access to the admin dashboard in which they can perform admin only tasks like monitoring the users, create, edit and delete the dance moves which will be reflected in the user dashboard.

User role:

The user role has the access to the user dashboard in which they can view all the dance moves and perform the task on their own.

```
const isAdmin = (req, res, next) => {
  if (req.user.role === "admin") {
    next();
  } else {
    return res.status(403).json({ error: "Admin access only." });
  }
};
```

6.4 CRUD Operations

The CRUD operations are implemented for both the admin and user role through which admin can manage the dance moves and users can manage their own personalized sequence.

CRUD Operation for dance moves (admin)

Dance moves can only be managed by an admin through adminRoutes.js. This route is protected by middleware. It checks the role of the user before giving the access. The admin can create, read, update and delete a dance move in the database. The admin uploads required data of dance moves along with image and video via Multer [18].

Frontend

```
const formData = new FormData();
    formData.append("name", name);
    formData.append("category", category);
```

```
    formData.append("description", description);
    formData.append("image", imageFile);
    formData.append("video", video);


    try {
      const res = await fetch("/api/admin/add-dance", {
        method: "POST",
        headers: {
          Authorization: `Bearer ${token}`,
        },
        body: formData,
      });
      const data = await res.json();
      if (res.ok) {
        alert("Dance move added!");
        e.target.reset();
        loadDanceMoves();
```

Backend

```
router.post("/add-dance", auth, isAdmin, upload.single("image"), async
(req, res) => {
    try {
      const { name, category, description, video } = req.body;
      const imagePath = req.file ? `/uploads/${req.file.filename}` :
null;
      const newMove = new DanceMove({ name, category, description,
image: imagePath, video });
      await newMove.save();
```

The dance moves can be read by both the admin and the user; they can also search and filter them by category. Any data of the dance moves like name, description, category, image and video can be edited only by the admin and the deletion of dance moves can also be done by the admin.

```
router.get("/dancemoves", auth, isAdmin, async (req, res) => {
  try {
    const dances = await DanceMove.find();
    res.json(dances);
```

```
router.put("/dances/:id", auth, isAdmin, upload.single("image"), async
(req, res) => {
  try {
    const { name, category, description, video } = req.body;
    const { id } = req.params;
    const updateData = { name, category, description, video };
    if (req.file) {
      updateData.image = `/uploads/${req.file.filename}`;
    }
    const updatedDanceMove = await DanceMove.findByIdAndUpdate(id,
updateData, { new: true });
    if (!updatedDanceMove) {
      return res.status(404).json({ error: "Dance move not found" });
    }
    res.json({ message: "Dance move updated successfully",
updatedDanceMove });
```

```
router.delete("/dances/:id", auth, isAdmin, async (req, res) => {
  try {
      const { id } = req.params;
      const deletedDanceMove = await DanceMove.findByIdAndDelete(id);
      if (!deletedDanceMove) {
          return res.status(404).json({ error: "Dance move not
found" });
      }
      res.json({ message: "Dance move deleted successfully" });
```

CRUD Operation for Sequences (User)

Users can manage their own personalized dance sequences through sequenceRoutes.js. Users can create, read, update and delete their own sequences. A sequence can be created by the user with the required fields like name, description and the selection of dance moves. The dance moves for the sequence's creation will be populated in the dropdown of the sequence creation form and it has been implemented using choices.js library [20]. The user can also update any detail of the sequence and delete their own sequences. They can download their sequences and practice them offline as well.

Frontend

```
const form = document.getElementById("createSequenceForm");

  form?.addEventListener("submit", async (e) => {

    e.preventDefault();

    const token = localStorage.getItem("token");

    const name = document.getElementById("sequenceName").value;

    const description =
document.getElementById("sequenceDescription").value;

    const selected = moveChoices.getValue();

    const moves = selected.map(item => item.customProperties?._id);

    try {

      const res = await fetch("/api/sequences", {

        method: "POST",

        headers: {

          Authorization: `Bearer ${token}`,

          "Content-Type": "application/json",

        },

        body: JSON.stringify({ name, description, moves }),

      });

      if (res.ok) {

        alert("Dance Sequence created Successfully!");

        form.reset();

        loadUserSequences();
```

Backend

```
router.post("/", auth, async (req, res) => {
  try {
    const { name, description, moves } = req.body;
    const newSequence = new Sequence({
      user: req.user._id,
      name,
      description,
      moves,
    });
    await newSequence.save();
    res.status(201).json({ message: "Sequence created", sequence:
newSequence });
```

```
router.get("/", auth, async (req, res) => {
  try {
    const sequences = await Sequence.find({ user:
req.user._id }).populate("moves");
    res.json(sequences);
```

```
router.put("/:id", auth, async (req, res) => {
  try {
    const { name, description, moves } = req.body;
    const updatedSequence = await Sequence.findOneAndUpdate(
      { _id: req.params.id, user: req.user._id },
      { name, description, moves },
      { new: true }
    );
    if (!updatedSequence) {
      return res.status(404).json({ error: "Sequence not found or
unauthorized" });
    }
    res.json({ message: "Sequence updated", sequence:
updatedSequence });
```

```
router.delete("/:id", auth, async (req, res) => {

  try {

    const sequence = await Sequence.findOneAndDelete({ _id:
req.params.id, user: req.user._id });

    if (!sequence) {

      return res.status(404).json({ error: "Sequence not found or
unauthorized" });

    }

    res.json({ message: "Sequence deleted successfully" });
```

6.5 Advanced search features

The advanced search features with keyword and filter-based search will allow the users to search and find the content easily and fast. This keyword and filter search is implemented for both the dance moves and sequences. The admin dashboard contains these search features for all Users section and the dance moves section.

The keyword and filter-based search is implemented for both the admin and user dashboard for dance moves. They can search for dance moves by typing the keywords matching the name and description. The filter dropdown gives them the list of dance categories from which they can select the dance moves. The dance move search functionality is efficiently handled at the frontend level, after loading all the dance moves from the backend without need of any additional backend routes of search.

```
function applyDanceFilters() {

  const searchInput =
document.getElementById("danceMoveSearchInput").value.toLowerCase();


  const activeCategory = document.querySelector(".category-
option.active").dataset.category;


  const filtered = allDances.filter(dance => {

    const matchesCategory = activeCategory ? dance.category ===
activeCategory : true;
```

```
    const matchesSearch =
dance.name.toLowerCase().includes(searchInput);

    return matchesCategory && matchesSearch;

  });

  renderDanceCards(filtered);

}
```

The search is also implemented for the sequences in the user dashboard through which users can search their sequences by giving the keywords as name, description and moves. The sequences also have the filter-based search in which it gives the dropdown of dance moves. The search for sequences is the same as dance moves and is handled efficiently at the front end without need of any backend routes.

```
function applySequenceFilters() {

  const query =
document.getElementById("sequenceSearchInput").value.toLowerCase();


  const filtered = allSequences.filter(seq => {

    const nameMatch = seq.name.toLowerCase().includes(query);

    const descMatch = seq.description.toLowerCase().includes(query);

    const movesMatch = formatMoves(seq.moves ||
[]).toLowerCase().includes(query);


    const moveFilterMatch = activeSequenceMoves.size > 0

  ? (seq.moves || []).some(move => {

      const name = typeof move === "string" ? move : move?.name;

      return activeSequenceMoves.has(name);

    })

  : true;

    return (nameMatch || descMatch || movesMatch) && moveFilterMatch;

  });

  renderSequenceList(filtered);

}
```

The admin dashboard has search input for all users section, in this the admin can search for the user by typing the keywords like name, email and role of the user.

```
function setupSearch() {
  const searchInput = document.getElementById("userSearchInput");
  if (searchInput) {
    searchInput.classList.add("search-input");
    searchInput.addEventListener("input", (e) => {
      loadUsers(e.target.value);
    });
  }
```

6.6 Calendar Integration

The calendar is integrated into the Dance Flow web application which allows the users to schedule the dance practice sessions. This calendar for users was implemented using Toast UI calendar to provide an interactive and visually attractive scheduling experience for the users [19]. This calendar allows the user to create, read, update and delete the dance practice sessions. The purpose of the calendar in the Dance Flow application is to provide the users with the ability to schedule a dance session with their personalized sequences for their practice.

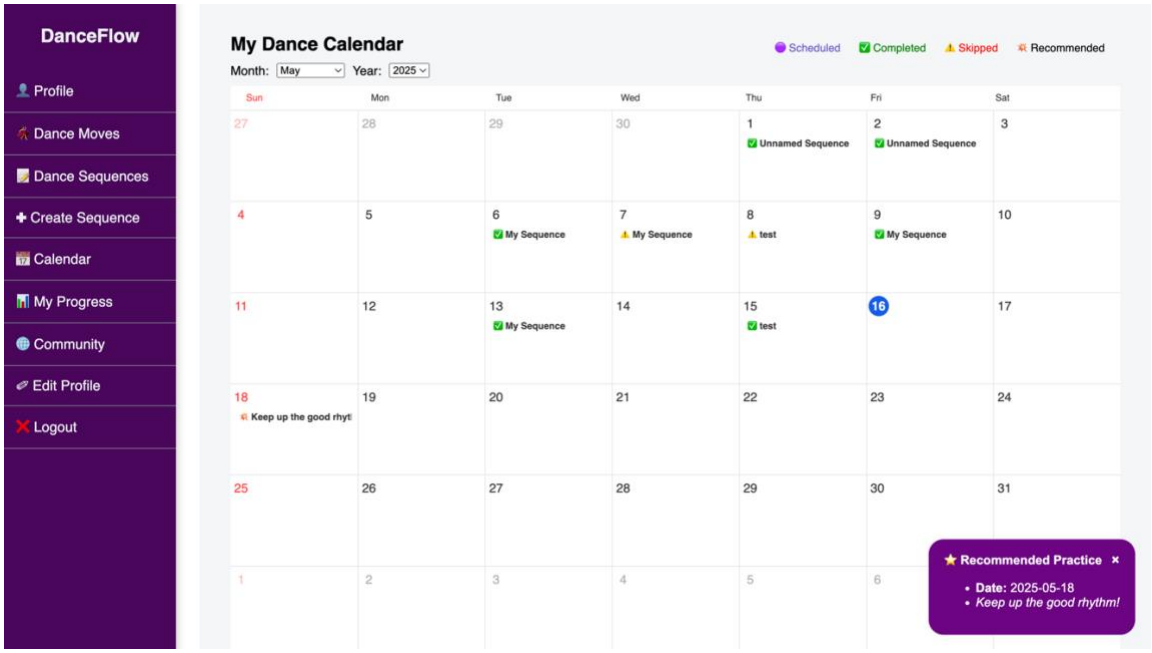

Figure 9: Calendar in User dashboard

Frontend Implementation:

Once the page loads, the calendar is initialized, and the system fetches the current user's sessions data from the backend and displays it to the users in the calendar as events. Users can view all the scheduled sessions of their own. The users can create a session clicking on the date, it will show them a create session modal to select a sequence, write a description and give duration for their session. The user can also edit the scheduled session. When the form if filled, all the session details are collected and if the Session Id exists the frontend sends the PUT request to edit the session details, otherwise it sends the POST request to backend to create a dance practice session. It also uses the JWT token for secure authorization from the local storage in the header of the request.

```
document.getElementById("createSessionForm").addEventListener("submit",
async (e) => {
    e.preventDefault();


    const form = e.target;
    const sessionId = form.dataset.sessionId;
    const sequenceId = document.getElementById("sessionSequence").value;
    const description =
document.getElementById("sessionDescription").value;
    const date = document.getElementById("sessionDate").value;
    const duration = document.getElementById("sessionDuration")?.value;


    const payload = {
      sequence: sequenceId,
      date,
      description,
      duration
    };


    const url = sessionId ? `/api/sessions/${sessionId}` :
"/api/sessions";
    const method = sessionId ? "PUT" : "POST";
```

```
    try {
      const response = await fetch(url, {
        method,
        headers: {
          "Content-Type": "application/json",
          Authorization: `Bearer ${localStorage.getItem("token")}`
        },
        body: JSON.stringify(payload)
      });
```

The user can mark the session as completed on today's date but won't be able to mark the session as completed on the future dates. Once the session is marked as completed, the label of the session changes to green to represent visually for the user. The user can also delete a session of their own.

```
document.getElementById("markDoneSessionBtn").addEventListener("click",
async () => {
    const sessionId =
document.getElementById("createSessionForm").dataset.sessionId;
    const sessionDateStr =
document.getElementById("sessionDate").value;
    const sessionDate = new Date(sessionDateStr);


    const today = new Date();
    today.setHours(0, 0, 0, 0);


    if (sessionDate > today) {
      alert("You can't mark future sessions as done.");
      return;
    }
    if (sessionId) {
      await completeSession(sessionId);
    }
  });
```

```
document.getElementById("deleteSessionBtn").addEventListener("click",
async () => {

    const sessionId =
document.getElementById("createSessionForm").dataset.sessionId;

    if (sessionId && confirm("Are you sure you want to delete this
session?")) {

      await deleteSession(sessionId);

    }

  });
```

Backend Implementation:

The backend implementation has the secure routes for CRUD operations for scheduling calendar sessions. Each route in the backend is protected by middleware for the security. Once the user schedules a practice session via calendar interface the POST endpoint is called, it uses the “auth” middleware to make sure the user is authenticated and stores the session data with the user Id and sends the JSON response to the frontend.

```
router.post("/", auth, async (req, res) => {
  try {
    const { sequence, date, description, duration } = req.body;
    const session = new Session({
      user: req.user._id,
      sequence,
      date,
      description,
      duration,
    });
    await session.save();
    res.status(201).json({ message: "Session created", session });
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: "Failed to create session" });
  }});
```

The user will be able to view their all-scheduled dance practice sessions with the name of the sequences, edit any detail of their practice sessions and can also delete any session that they have scheduled. All these happens through the GET, PUT and DELETE routes in the backend.

```
router.get("/", auth, async (req, res) => {
  try {
    const sessions = await Session.find({ user:
req.user._id }).populate("sequence");
    res.json(sessions);
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: "Failed to fetch sessions" });
  }
});
```

```
router.put("/:id", auth, async (req, res) => {
  try {
    const { date, description, completed, duration } = req.body;
    const updated = await Session.findOneAndUpdate(
      { _id: req.params.id, user: req.user._id },
      { date, description, completed, duration },
      { new: true }
    );
    if (!updated) return res.status(404).json({ error: "Session not
found" });
    res.json({ message: "Session updated", session: updated });
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: "Failed to update session" });
  }
});
```

```
router.delete("/:id", auth, async (req, res) => {
  try {
    const deleted = await Session.findOneAndDelete({ _id:
req.params.id, user: req.user._id });
    if (!deleted) return res.status(404).json({ error: "Session not
found" });
    res.json({ message: "Session deleted" });
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: "Failed to delete session" });
  }});
```

6.7 Progress Tracking

The progress tracking is implemented in the Dance Flow application to track the user's session activity, and the status of the user's dance practice sessions whether completed or scheduled. This tracking of the progress is visually represented in the bar chart as completed or scheduled for the users along with filters such as date range (from date - to date), sequences and status to easily get the progress of their dance practice sessions. This visual representation of progress is implemented using chart.js [21].

On the page load all the progress of the user is displayed in the bar charts and all the sequences are populated in the filter. Once the user applies any of the filters the chart updates instantly.

```
statusChart = new Chart(statusCtx, {
      type: 'bar',
      data: {
        labels: ['Completed', 'Scheduled'],
        datasets: [
          {
            label: 'Completed',
            data: [completed, 0],
```

```
            backgroundColor: barChartColors.completed.background,
            borderColor: barChartColors.completed.border,
            borderWidth: 1
          },
          {
            label: 'Scheduled',
            data: [0, scheduled],
            backgroundColor: barChartColors.scheduled.background,
            borderColor: barChartColors.scheduled.border,
            borderWidth: 1
          }
        ]
      },
```

6.8 Auto recommendation of sessions

The auto recommendation feature provides the users a personalized recommendation of the next sessions based on the past sessions' activity. This helps the users to stay consistent with their dance practice. This is implemented with the simple logic of sessions activity, without using any complex AI algorithms.

This first gets an array of all the sessions data of current user, then splits the session array into past sessions whether completed or skipped and future sessions whether any sessions scheduled in the future, if they are any future sessions no recommendation is needed. From the past sessions it sorts the sessions by most recent date first and calculates the gap from the last session to today to give the next recommended session for the user. This recommended session is then visually displayed in the calendar.

```
lastDate.setHours(0, 0, 0, 0);
  const gapDays = Math.ceil((today - lastDate) / (1000 * 60 * 60 *
24));


  if (gapDays > 3) {
    return {
```

```
      suggestedDate: addDays(today, 1),
      note: "You've been away for a while. Let's get back!"
    };
  } else if (gapDays <= 2) {
    return {
      suggestedDate: addDays(today, 3),
      note: "Keep up the good rhythm!"
    };
  } else {
    return {
      suggestedDate: addDays(today, 2),
      note: "Maintain consistency!"
    };
  }
}
```

6.9 Community features

The Community in the Dance Flow web application helps the users to share a session or sequence with the other users in the application. They can also post the threads and challenges in which other users can participate and add posts or comments. All these community features are safely managed by backend API routes with the support of the frontend and gives the smooth experience for the users to share and communicate with the other users in the application.

Shares

To share a sequence or session with the other users, the system looks up for the recipient's email id in the database and if present the user will be able to share the content of sessions and sequences with that existing user in the application. All the shares are stored with the sender, recipient, type of the share, either sequence or session and the optional caption in the shares collection of the database. Users can then see the shares sent by them in the “My Shares” section and the received shares in the “Shared with me” section.

Frontend

```
// — My Shares —
      let res = await fetch("/api/community/shares?mine=true", {
        headers: { Authorization: `Bearer ${token}` }
      });
      const mine = res.ok ? await res.json() : [];
      renderShares(mySharesList, mine, "You haven't shared any
sequences yet.");
// — Shared With Me —
      res = await fetch("/api/community/shares?mine=false", {
        headers: { Authorization: `Bearer ${token}` }
      });
      const incoming = res.ok ? await res.json() : [];
      renderShares(sharedWithMeList, incoming, "No one has shared a
sequence with you.");
```

Backend

```
router.post("/share", auth, async (req, res) => {
  try {
    let { type, reference, caption, toUser } = req.body;
    type = type.charAt(0).toUpperCase() +
type.slice(1).toLowerCase();
    const share = await Share.create({
      from:      req.user._id,
      to:        toUser,
      type,
      reference,
      caption:   caption || undefined
    });
    res.status(201).json(share);
```

```
router.get("/shares", auth, async (req, res) => {
  try {
    const mine = req.query.mine === "true";
    const filter = mine ? { from: req.user._id } : { to:
req.user._id };
    let shares = await Share.find(filter)
      .populate('from', 'name email')
      .populate('to', 'name email')
      .sort({ createdAt: -1 });
    await Promise.all(shares.map(async (share) => {
      if (share.type === 'Session') {
        await share.populate({ path: 'reference', model:
'Session' });
      } else if (share.type === 'Sequence') {
        await share.populate({ path: 'reference', model:
'Sequence' });
      }
    }));
    const validShares = shares.filter(s => s.reference);
    res.json(validShares);
```

Threads and Posts

A user can create a thread which all the other users in the application can view and do posts for it. All the threads in the application are stored with the title and user Id of the creator of the thread and the posts are stores ad well with the content and the user Id of the user who posted it. The user will be able to delete a thread which they have created.

Frontend

```
  newThreadForm?.addEventListener("submit", async e => {
    e.preventDefault();
    const title = e.target.threadTitle.value.trim();
    if (!title) return;
    await fetch("/api/community/threads", {
```

```
      method: "POST",
      headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${token}`
      },
      body: JSON.stringify({ title })
    });
    e.target.reset();
    await loadThreads();
  });
```

```
postForm?.addEventListener("submit", async e => {
    e.preventDefault();
    const content = postContentInput.value.trim();
    if (!content || !currentThreadId) return;

    const res = await
fetch(`/api/community/threads/${currentThreadId}/posts`, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${token}`
      },
      body: JSON.stringify({ content })
    });

    if (res.ok) {
      postContentInput.value = "";
      await loadPosts(currentThreadId);
    } else {
      alert("Failed to post.");
    }
  });
```

Backend

```
router.post("/threads", auth, async (req,res)=>{

  const thread = new Thread({ title:req.body.title,
createdBy:req.user._id });

  await thread.save();

  res.status(201).json(thread);

});
```

```
router.get("/threads", async (req,res)=>{

  const threads = await Thread.find().populate("createdBy","name");

  res.json(threads);

});
```

```
router.delete("/threads/:id", auth, async (req, res) => {

  const thread = await Thread.findById(req.params.id);

  if (!thread) return res.status(404).json({ error: "Thread not
found" });

  if (!thread.createdBy.equals(req.user._id)) return
res.status(403).json({ error: "Unauthorized" });

  await thread.deleteOne();

  res.json({ success: true });

});
```

```
router.post("/threads/:id/posts", auth, async (req,res)=>{

  const post = new Post({ thread:req.params.id, author:req.user._id,
content:req.body.content });

  await post.save();

  res.status(201).json(post);

});
```

```
router.get("/threads/:id/posts", async (req,res)=>{
  const posts = await Post.find({ thread:
req.params.id }).populate("author","name");
  res.json(posts);
});
```

Challenge and Comments

The user can post a challenge in the community section which can be viewed and joined by the other users who are interested. The users can also do comments for the challenge once they join it. The challenge is stored with the name, description, ends at date, user Id of the creator and the participants who joined the challenge in the challenge collection of data base. The user will be able to delete the challenge posted by their own.

Frontend

```
if (newChForm) {
    newChForm.addEventListener("submit", async (e) => {
      e.preventDefault();
      const name = e.target.challengeName.value.trim();
      const endsAt = e.target.challengeEnds.value;
      const desc = e.target.challengeDesc.value.trim();
      if (!name || !endsAt) return;
      await fetch("/api/community/challenges", {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
          Authorization: `Bearer ${token}`
        },
        body: JSON.stringify({ name, description: desc, endsAt })
      });
      e.target.reset();
```

```
      await loadChallenges();
    });
  }
```

```
if (e.target.matches(".join-challenge")) {
    await fetch(`/api/community/challenges/${challengeId}/join`, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${token}`
      }
    });
    await loadChallenges();
  }
```

```
await fetch(`/api/community/challenges/${challengeId}/comment`, {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
          Authorization: `Bearer ${token}`
        },
        body: JSON.stringify({ content })
      });
```

Backend

```
router.post("/challenges", auth, async (req,res)=>{
  const challenge = new Challenge({
    name: req.body.name,
    description: req.body.description,
    creator: req.user._id,
    endsAt: req.body.endsAt,
```

```
  });
  await challenge.save();
  res.status(201).json(challenge);
});
```

```
router.post("/challenges/:id/join", auth, async (req,res)=>{
  const c = await Challenge.findById(req.params.id);
  if(!c.participants.includes(req.user._id)){
    c.participants.push(req.user._id);
    await c.save();
  }
  res.json(c);
});
```

```
router.post("/challenges/:id/comment", auth, async (req, res) => {
  const challenge = await Challenge.findById(req.params.id);
  if (!challenge) return res.status(404).json({ error: "Challenge not
found" });
  const isJoined = challenge.participants.some(p =>
    p.toString() === req.user._id.toString()
  );
  if (!isJoined) {
    return res.status(403).json({ error: "You must join the challenge
to comment" });
  }
  challenge.comments.push({
    user: req.user._id,
    content: req.body.content
  });
  await challenge.save();
  res.json({ success: true });
});
```

7. Testing

The testing phase of Dance Flow web application focused on evaluating that the application is working as expected with all the requirements. To systematically test all the backend APIs of the application Postman was used [22].

Functional Testing:

The functional testing is the core part of testing in this application. This verifies that all the functionalities and requirements of the application are working correctly. This includes all the requirements such as user register, login, scheduling sessions and community features. The testing process was followed with a structured approach in which all the routes are tested independently to check the working of each functionality in the Postman.

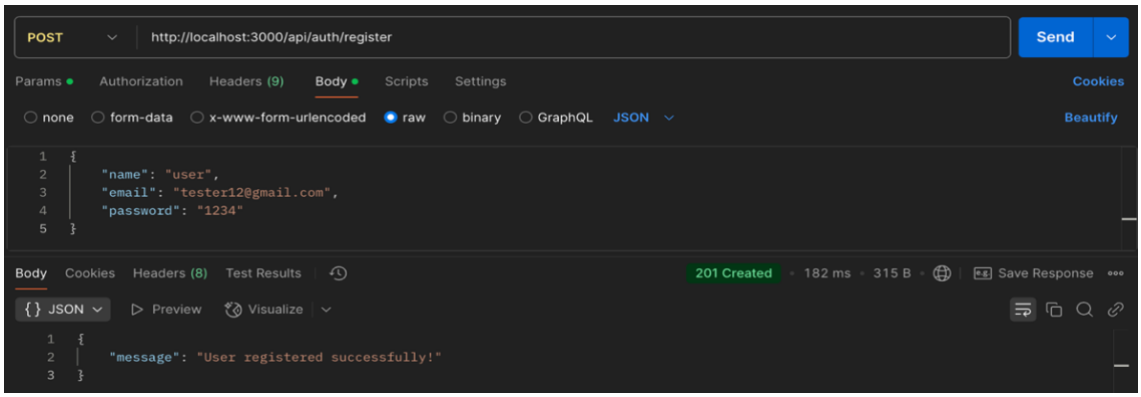

Figure 10: register route testing

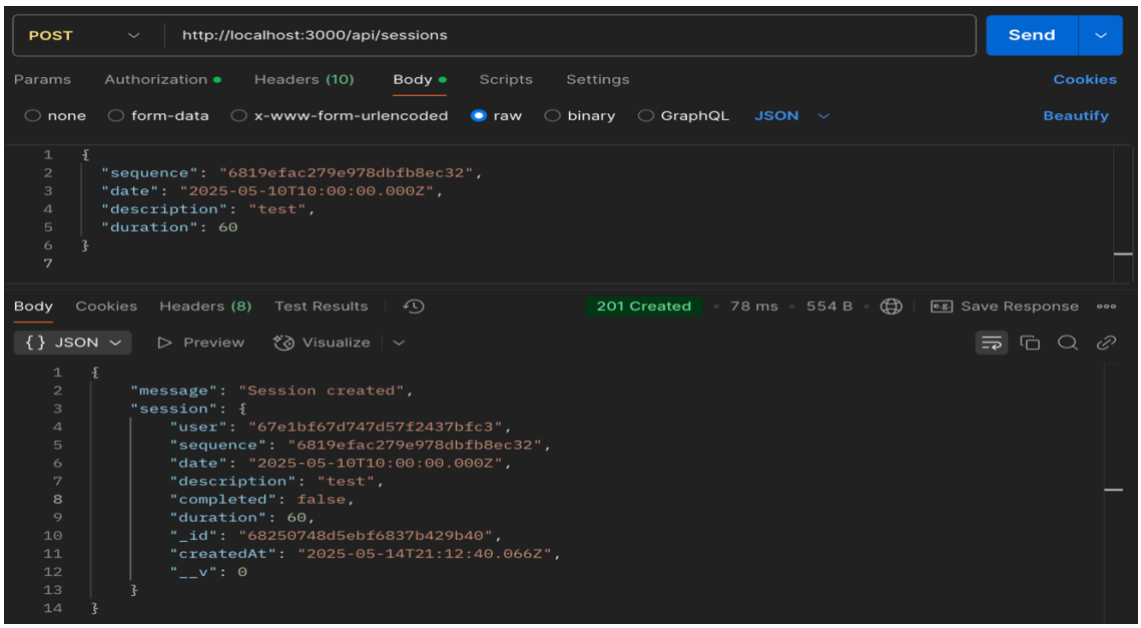

Figure 11: Sessions route testing

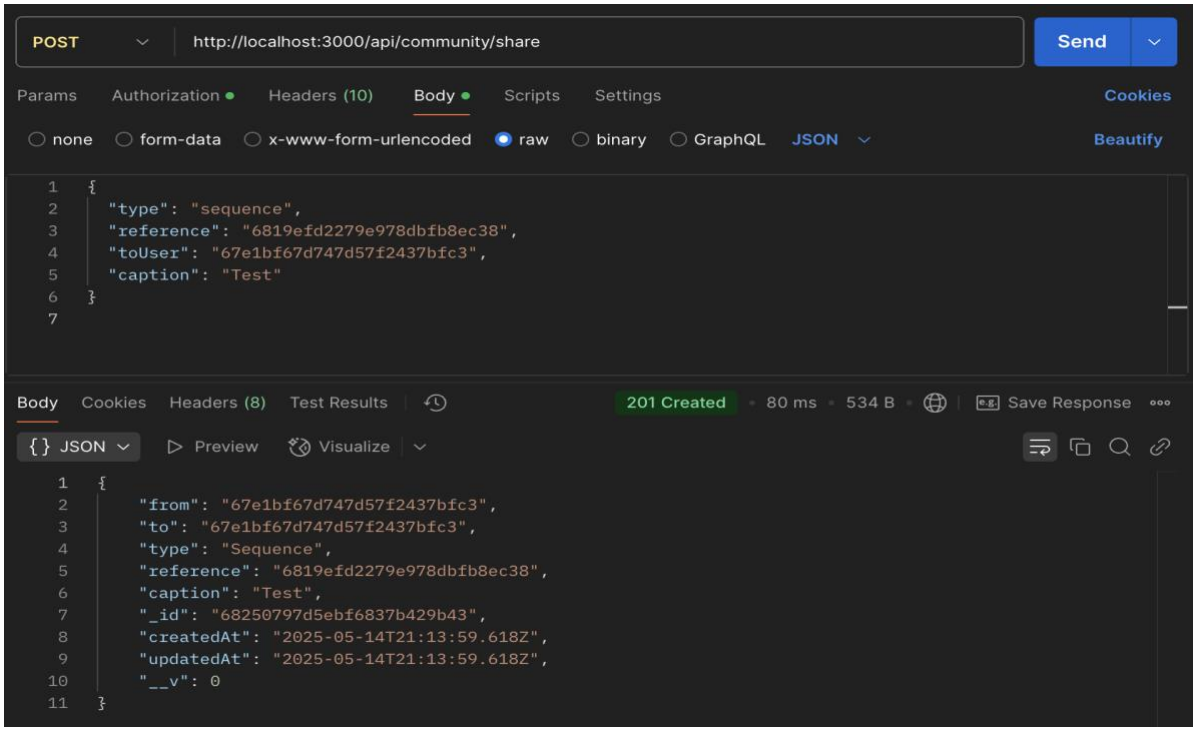

Figure 12: Community share route testing

CRUD operations testing:

The CRUD operations of dance moves, sequences and sessions were tested with the backend APIs in the postman. All the models with CRUD operations were tested successfully for the correct behaviour. The CRUD (create, read, update, delete) testing was performed to make sure the system handles all the operations correctly to manage the dance moves for the admin and to manage the sequences and sessions for the user.

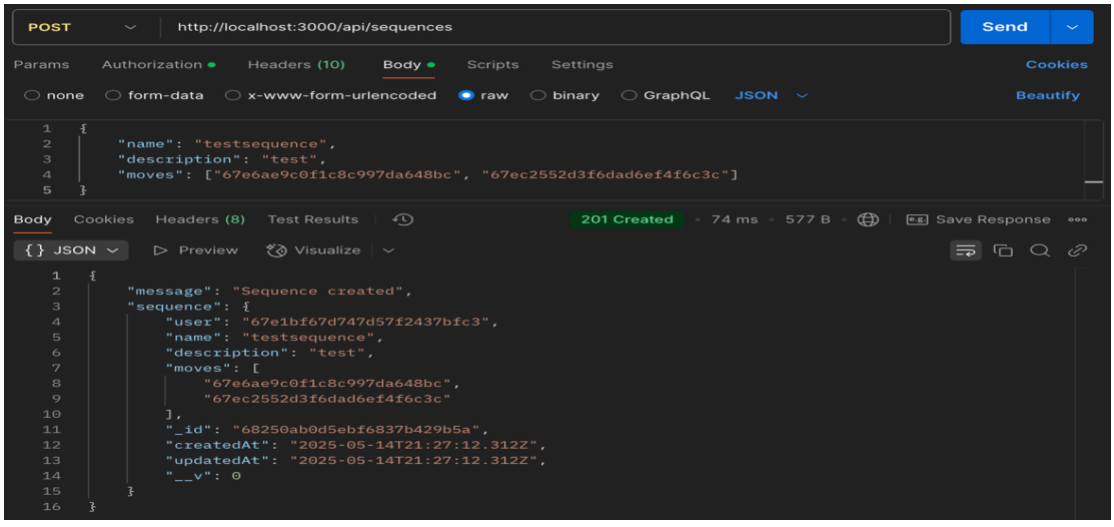

Figure 13:Post route of sequence

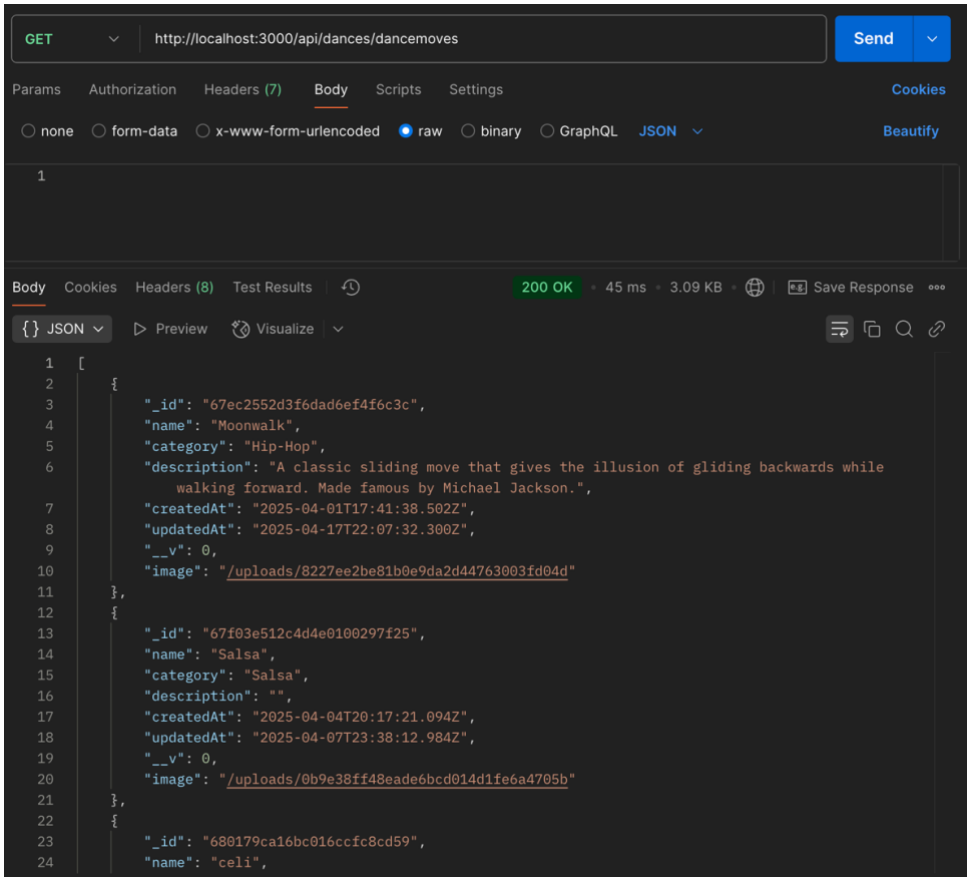

Figure 14: Get route of dance moves

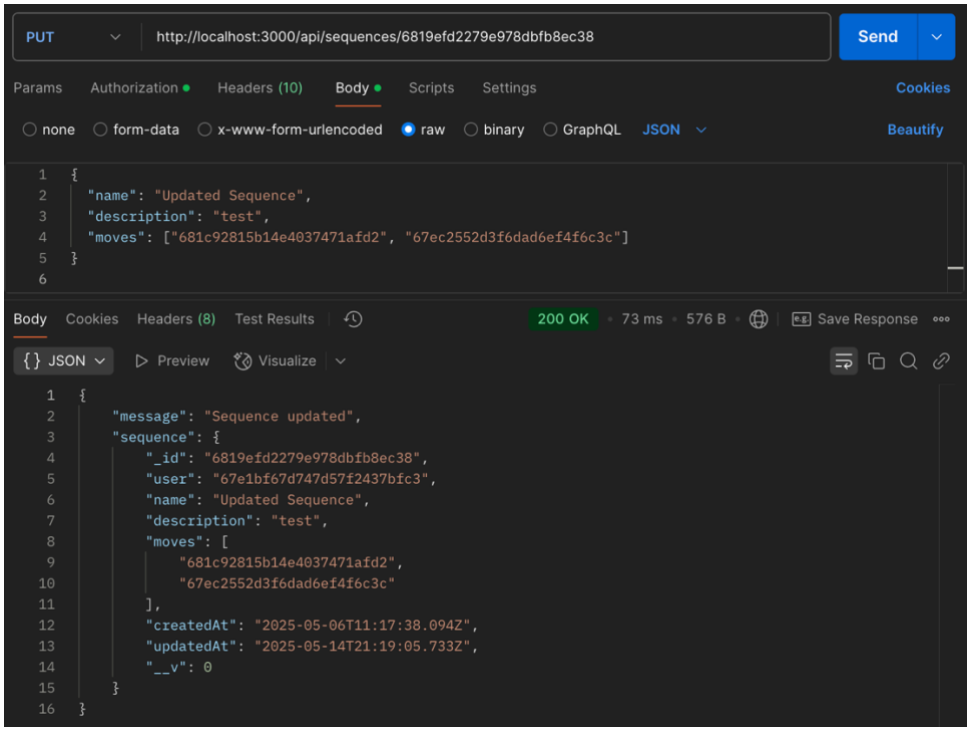

Figure 15:Put route of sequence

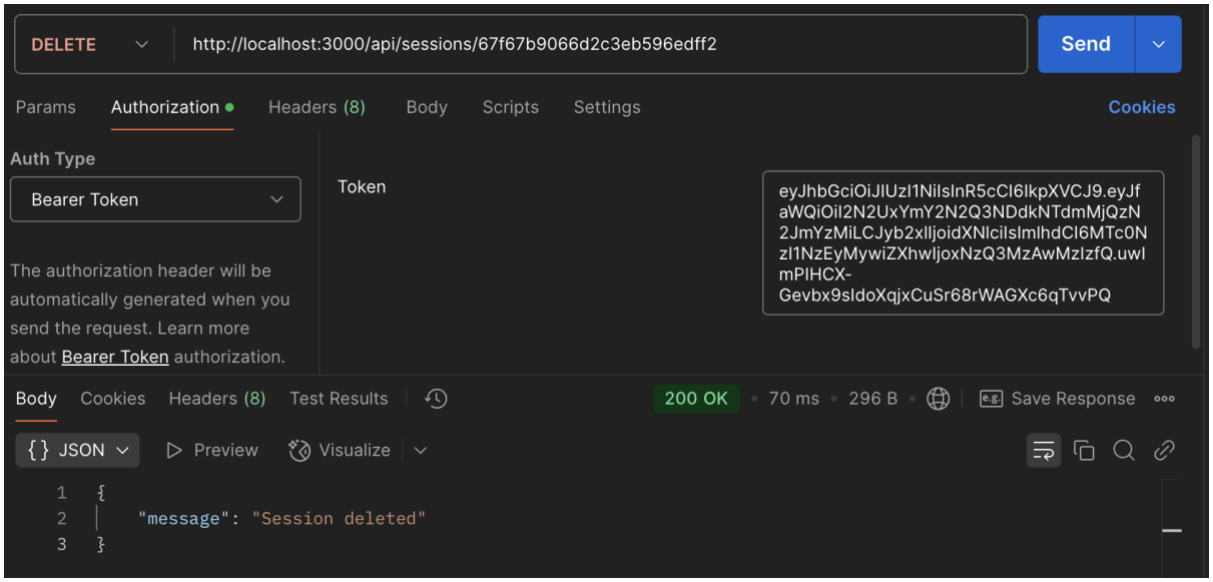

Figure 16: Delete route of sequence

Role-based access testing:

To test whether the admin and user are given the access to their correct dashboards and content the role-based access testing is performed in Postman with the backend API routes. To test the admin only routes, the regular user token was attempted, and it correctly returned the error message. The admin and user roles were tested and they correctly directed to their respective dashboards and the access to their respective content was given correctly. This proves the admin only content cannot be accessed by regular users, same as the users can only access their own content.

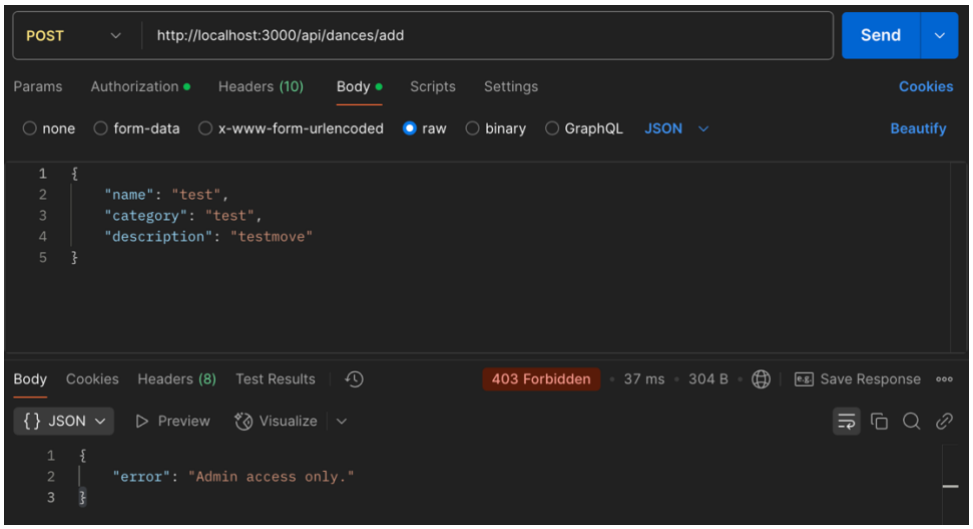

Figure 17: Role-based access testing

Security testing:

Security is one of the most important requirements for any of the web applications. To check the security of the Dance Flow web application, the JWT authentication was tested successfully, the reset token was also tested to maintain the secure application. The reset tokens were also verified that they are unique and can be used only for one request. All the protected routes were tested for security with JWT tokens.

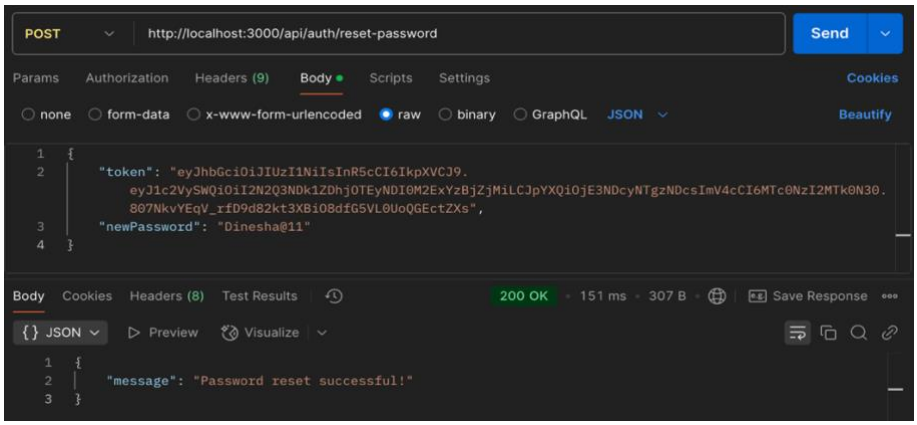

Figure 18: Security testing

User feedback:

In addition to all the testing, user feedback was collected to check the usability of the application, the end user satisfaction and the perspective of them while and after using the application. A feedback form for the application has been created and given to the group of users who used the application to collect some valuable feedback from them. They were provided with the application and asked to use the application either as regular user or admin and later asked to give some feedback on their experience with the Dance Flow web application.

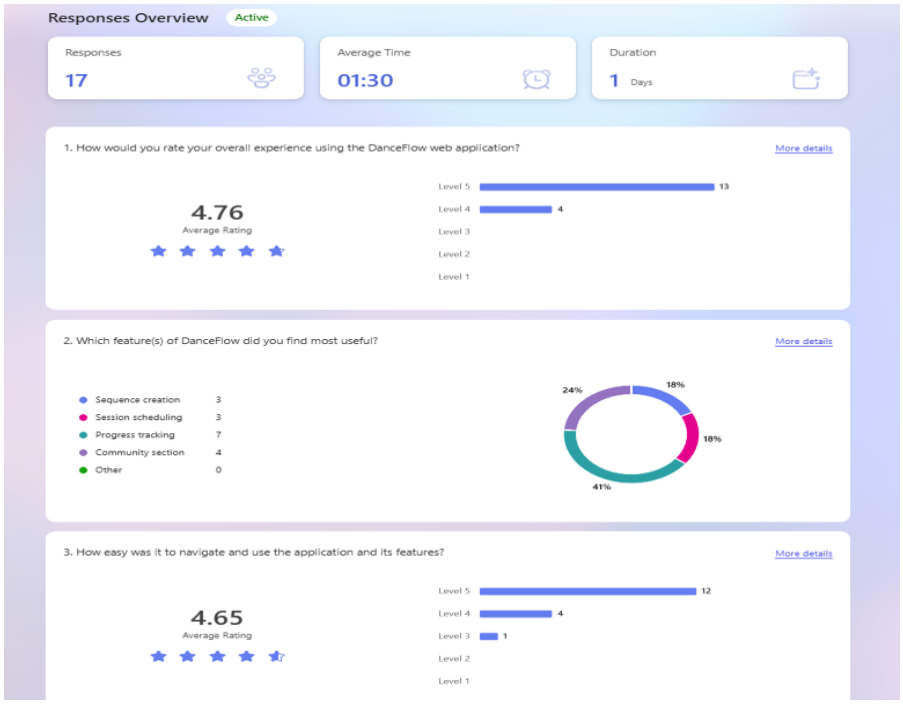

Figure 19: User feedback form (1)

Figure 20: User feedback form (2)

9. Discussion

This section discusses the development and outcomes of the Dance Flow web application with the comparison of existing virtual dance practice platforms. Also highlights the unique features of Dance Flow Web application, challenges faced during the development and scope and limitations of the project.

9.1 Overview of existing platforms

There are several dance learning platforms existing. Each of them are provided with different types of features to support the dancers to learn dance virtually. Most of the dance learning platforms are mainly focused on providing the instructional videos to help the users to learn the dance moves or on the administrative needs for the professional dance studios.

STEEZY studio is one of the leading virtual dance learning platforms [13]. This platform offers a huge number of dance classes among various styles of dances such as Hip-Hop, Ballet, K-pop, Salsa etc., with the key features like adjustable video speed. Like this there are many other virtual dance learning platforms but they all lack with the some of the important features that user needs to practice a dance session which are personalizing their sequence of dance moves and scheduling them according to their preference. The existing dance platform doesn't provide the users with a track of their progress, which is another important feature for the users to improve their dance learning. All the existing platforms provide the valuable features, but they still lack in providing the personalized learning for the users.

9.2 Distinctive features of Dance Flow

The Dance Flow web application is developed to clear the identified gap in the existing platforms by offering an interactive and comprehensive platform which includes the personalized learning, scheduling their practice, tracking their progress and having the community space. The personalized learning improves the learning progress of the users as they can create their own sequence by selecting the pre-defined dance moves of their interest and scheduled them at their own convenience using the integrated calendar in the application. This application also tracks the user's progress of dance practice sessions and visually represents it in a bar chart for the user's monitoring. The Community space in the Dance Flow application will provide the users to share the sequences and sessions with the others in the application and the users can also

communicate with all the other users in the application through threads and posts. They can also create a challenge in which any of the users in the application can join and participate and do comments in the joined challenge to communicate with the other participants.

9.3 Challenges faced

This project has been developed by facing the several challenges which includes the role-based access control for the users and admins to have the correct access to their respective dashboards without any mismatches as this is one of the key elements of this project. Initially, implementing the secure user authentication to protect the user data which consists of personal information and session histories has been crucial. Additionally, integrating the calendar has been the huge challenge, as this project first used the FullCalendar.js library to have the calendar in user dashboard for scheduling the sessions but later due to limitations in flexibility and adaptability it was switched to more flexible and developer friendly Tui.Calendar. This replacement needed to be performed carefully to ensure the data integrity and reinstallation of the other library.

9.4 Scope and limitations

This project aims to design and develop a user-friendly and interactive web application that helps in assisting the users to organize and manage their content and practice sessions. This application mainly focuses on two areas. Firstly, custom sequence creation where the users can create their own sequence and personalize them. Secondly, scheduling and the progress tracking system where the users can schedule and track a practice session. It also has a community space for the users to share their sequences and interact with others. This application is modular and extensible for future enhancements. However, the implementation of all the core features was met partially in the final system due to the time constraints and some of these limitations are listed below,

- The auto recommendation system is currently basic, and the AI-driven recommendations were not implemented in this system due to lack of time for the development.
- The calendar for now has been designed to schedule a dance practice session, advanced functionalities like rescheduling and different time zones are not supported.

10. Conclusion

The Dance Flow web application was developed to bridge the gap in the virtual dance learning platforms. The existing applications of dance learning platforms provide users with all the required advanced learning tutorials and instructional videos including the features of administrative needs for the professional dance studio platforms. Those existing dance platform often miss out to provide the users with personalized learning, ability to track their progress and having some community space to communicate with the dancers. The current Dance Flow provides the users with interactive and personalized learning experience through which they can improve their quality of learning dance skills. Though the Dance Flow web application is a full working prototype with the positive feedback there are some areas which might need improvement in the future. This application has provided some useful insights into how the virtual dance learning can help the dancers to improve their personalized learning and maintain their consistency and make it more enjoyable. As this is the strong start, future work can make the application even more interactive and more useful for all the dancers.

10.1 Future Enhancement:

The current version of Dance Flow web application has met the necessary requirements of the user to have a scheduled dance practice session with their personalized sequence of dance moves including the community area. There are several enhancements which can be made in the future to develop the system and improve the user experience.

The Voice Commands in this current application has been partially integrated. This initial prototype allows users to use simple voice commands like start and stop. Those commands can be added for the video of the dance moves which will be very useful to the users and improve their experience. As of now the application is only web based but, in the future, it can be developed as a mobile application.

The current application has only calendar date to schedule a dance practice session but in the future, it can be added with the time which will be more specific for the user to schedule a dance practice session, and it will also be helpful to send the early remainder based on the time for the users about their practice sessions.

The community section of the Dance Flow can also be developed in the future with more advanced features like adding the images and some small videos while creating the threads. There can also be a space for the users to message or communicate privately with the other users in the application. It can also be expanded to have share option outside of the application by using social media apps like Facebook, Instagram etc., The challenges in the community section can be improved with some advanced features to be more interactive for the users.

11. References

[1] P. Brusilovsky and E. Millán, “Adaptive learning systems: From data-driven personalizations to AI-powered recommendations,” ACM Comput. Surv., vol. 54, no. 3, pp. 1-35, 2021.
[2] J. F. Pane, E. D. Steiner, M. D. Baird, L. S. Hamilton, and J. D. Pane, Informing Progress: Insights on Personalized Learning Implementation and Effects. Santa Monica, CA, USA: RAND Corp., 2017.
[3] R. E. Mayer, Multimedia Learning, 2nd ed. Cambridge, U.K.: Cambridge Univ. Press, 2009.
[4] Z. Li, “Creativity and opportunity: How COVID-19 fosters digital dance education,” Digital Creativity, vol. 32, no. 3, pp. 188-207, 2021, doi: 10.1080/14626268.2021.1967406.
[5] Ballet with Isabella, “The benefits of online ballet training with Isabella.” [Online]. Available: https://balletwithisabella.com/posts/the-benefits-of-online-ballet-training-with-isabella/ . Accessed: Apr. 11, 2025.
[6] H. Xie, H.-C. Chu, G.-J. Hwang, and C.-C. Wang, “Trends and development in technology-enhanced adaptive/personalized learning: A systematic review of journal publications from 2007 to 2017,” Computers & Education, vol. 140, Art. no. 103599, 2019.
[7] C.-M. Chen and Y.-L. Li, “Personalized e-learning system using item response theory,” Computers & Education, vol. 55, no. 4, pp. 1625-1633, 2010.
[8] M. Khalil and M. Ebner, “Learning analytics: Principles and constraints,” in Proc. Int. Conf. E-Learning & E-Technologies, 2016, pp. 15-22.
[9] S. Deterding, D. Dixon, R. Khaled, and L. Nacke, “From game design elements to gamefulness: Defining ‘gamification’,” in Proc. 15th Int. Academic MindTrek Conf., 2011, pp. 9-15.
[10] D. H. Jonassen, “Objectivism versus constructivism: Do we need a new philosophical paradigm?,” Educ. Technol. Res. Dev., vol. 39, no. 3, pp. 5-14, 1991.
[11] J. Preece, Online Communities: Designing Usability, Supporting Sociability. New York, NY, USA: Wiley, 2000.
[12] F. Wang and M. Hannafin, “Design-based research and technology-enhanced learning environments,” Educ. Technol. Res. Dev., vol. 53, no. 4, pp. 5-23, 2005.
[13] STEEZY Studio, “STEEZY Studio online dance classes,” 2024. [Online]. Available: https://www.steezy.co/ . Accessed: May 14, 2025.
[14] J. Li and M. A. Ahmad, “Evolution and trends in online dance instruction: A comprehensive literature analysis,” Frontiers in Education, vol. 10, May 2025, doi: 10.3389/feduc.2025.1523766.
[15] M. Parrish, "Toward transformation: Digital tools for online dance pedagogy," *Arts Education Policy Review*, vol. 117, no. 3, pp. 168–182, 2016, doi: 10.1080/10632913.2016.1187974.
[16] Y. Xuan and C. Della, "Assessing the impact of interactive multimedia learning platforms on dance education outcomes," *Int. J. Adv. Appl. Sci.*, vol. 11, no. 9, pp. 7–16, 2024, doi: 10.21833/ijaas.2024.09.002.
[17] “Nodemailer,” Nodemailer.com, 2025. [Online]. Available: https://nodemailer.com/

[18] “expressjs/multer — Node.js middleware for handling multipart/form-data,” GitHub README, 2025. [Online]. Available: https://github.com/expressjs/multer
[19] “TOAST UI Calendar Documentation,” nhn.github.io/tui.calendar, 2025. [Online]. Available: https://nhn.github.io/tui.calendar/latest/
[20] “Choices.js — Configurable select box/text input plugin,” Choices-js/Choices README, GitHub, 2025. [Online]. Available: https://github.com/Choices-js/Choices
[21] “Chart.js Documentation,” Chartjs.org/docs, 2025. [Online]. Available: https://www.chartjs.org/docs/
[22] Postman, "Postman API Platform," *Postman, Inc.*, [Online]. Available: https://www.postman.com/