Summer Fellowship Report

On

FOSSEE Web Development

Submitted by

Manurbhav Arya

Under the guidance of

Prof.Kannan M. Moudgalya
Chemical Engineering Department
IIT Bombay

July 24, 2024

Acknowledgment

First and foremost, I would like to express my sincere thanks to my project head, Prof.Kannan M. Moudgalya, who gave me this valuable opportunity to work in such a learning environment at FOSSEE.

I would like to extend my heartfelt gratitude to my project mentors, Mr. Akshay Thakur, Ms. Shashi Rekha, and the entire Web team. Your guidance, constant supervision, and the wealth of knowledge and expertise you have imparted have been invaluable throughout this journey. Thank you for your unwavering support and encouragement.

Finally, I am deeply thankful to my parents and teachers who helped and inspired me throughout this fellowship.

Sincerely

Manurbhav Arya,

IIT Tirupati

Contents

Chapter 1

Introduction

FOSSEE (Free/Libre and Open Source Software for Education) is an initiative by the Indian Institute of Technology Bombay aimed at promoting the use of open-source software in education. FOSSEE supports various open-source software and provides resources, workshops, and training programs to help students and professionals adopt and use these tools effectively. The initiative focuses on reducing the dependency on proprietary software and fostering a collaborative and inclusive learning environment.

1.1 Django

Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. Developed by experienced developers, it takes care of much of the hassle of web development, allowing developers to focus on writing their app without needing to reinvent the wheel. Django emphasizes reusability and ”don’t repeat yourself” (DRY) principles. It includes a robust ORM (Object-Relational Mapping) system, a powerful templating engine, and a comprehensive admin interface. Django is well-suited for building complex, data-driven websites and web applications, thanks to its built-in features like authentication, URL routing, and form handling.

1.2 Astro

Astro is a modern JavaScript web framework that excels in creating fast, content-driven websites. It is designed with a strong focus on performance, simplicity, and flexibility, making it an excellent choice for developers who want to build high-performance static sites with minimal effort.

Astro adopts a server-first approach to enhance website performance. Instead of relying heavily on client-side JavaScript, Astro renders components on the server side. This results in sending lightweight HTML to the browser, drastically reducing the amount of JavaScript that needs to be processed on the client side. This server-side rendering approach ensures that web pages load quickly and efficiently, providing a smooth and fast user experience without unnecessary JavaScript overhead.

One of Astro's standout features is its content-driven design. It is built to work seamlessly with various content sources. Whether your content is stored in markdown files, retrieved from an external API, or managed through a content management system (CMS), Astro can integrate with these sources effortlessly. This flexibility makes Astro ideal for a range of content-rich websites, including blogs, documentation sites, and marketing pages.

Astro also prides itself on being customizable. It allows developers to extend its functionality by incorporating their preferred tools and technologies. You can bring in your own JavaScript UI components, integrate CSS libraries, apply custom themes, and use various integrations to tailor Astro to fit your specific project needs. This high degree of customization ensures that Astro can adapt to diverse development workflows and project requirements.

In essence, Astro combines server-side rendering, content flexibility, and extensive customization options to offer a robust framework for building fast and efficient websites. Its approach to delivering minimal JavaScript and optimizing performance makes it a compelling choice for developers looking to create modern, high-performance web experiences.

1.3 Astro vs Django

While both Django and Astro have their strengths, Astro might be considered better than Django for certain use cases, particularly when performance and static site generation are key priorities. Here's why:

- Performance: Astro's approach of compiling down to static HTML means that websites built with Astro are incredibly fast. By sending only essential code to the browser, Astro significantly reduces load times, making it ideal for static sites where performance is critical.

- Minimal JavaScript: Astro ensures that only the necessary JavaScript is shipped to the client. This minimalistic approach helps in creating faster-loading pages, which can be crucial for improving user experience and SEO.

- Flexibility with Front-End Frameworks: Astro allows developers to use multiple front-end frameworks like React, Vue, and Svelte in a single project. This flexibility can be advantageous for teams that have expertise in different frameworks or want to leverage specific features from each

- Static Site Generation: For use cases like blogs, documentation sites, and marketing pages, Astro's static site generation is a clear winner. It simplifies the deployment process and reduces server load since the pages are pre-rendered and served as static files.

- Modern Development Experience: Astro offers a modern development experience with features like hot module replacement (HMR), an intuitive component-based architecture, and easy integration with modern tools and workflows.

In contrast, Django is a full-fledged web framework suited for building dynamic, data-driven web applications. It excels in scenarios where server-side processing, complex database interactions, and robust backend functionality are required. However, for static content-focused sites where performance and simplicity are key, Astro provides a more streamlined and efficient solution.

Chapter 2

Initial Task - Reconstructing a Django Site with Astro

2.1 Pages

- Home Page
- Resources Page
- Schedule Page

2.1.1 Home Page

Code Overview:

- The Home Page uses a layout component to structure the page and include various components.
- It imports and incorporates the following components: TopBar, Navbar, Header, IconBar, HomeBody, and Footer.
- The page is set up to display the main content of the website with a structured layout.

Features:

- Displays the main content area with a combination of header, navigation, body, and footer sections.
- Utilizes a responsive layout to ensure proper display on different devices.
- Integrates various components to create a cohesive and functional homepage.

2.1.2 Resources Page

Code Overview:

- The Resources Page employs the Layout component and includes TopBar, Navbar, IconBar, and Footer.
- It specifically uses the ResourcesBody component to manage the main content, focusing on resources-related information.

Features:

- Provides a dedicated space for resources content, separated from other page types.
- Continues the consistent layout and design pattern with a focus on resource management.

2.1.3 Schedule Page

Code Overview:

- The Schedule Page also uses a layout component for structure.
- It imports the TopBar, Navbar, Header, IconBar, and Footer components.
- The main content is handled by the HomeBody component, indicating that the schedule page might have a similar content structure to the home page.

Features:

- Similar to the Home Page, it includes the core components for consistent page layout and navigation.
- Displays schedule-related content, potentially utilizing the same structure as the home page for a unified look and feel.

2.2 Common Components

- Topbar
- Navbar
- Iconbar
- Footer

2.2.1 Topbar

This component is responsible for displaying a set of banner images in the top bar of the website.

Code Overview:

- Displays a set of banner images at the top of the website.
- Initially hardcoded data for the banner images, later updated to use dynamic Markdown files for easier content management.

Features

- Responsive design ensures it looks good on different screen sizes.
- Includes links wrapped around banner images, directing users to specified URLs.

2.2.2 Navbar

This component handles the main navigation menu of the site, including a responsive hamburger menu for mobile views.

Code Overview:

- Manages the main navigation menu, including a responsive design that adapts to mobile screens with a hamburger menu.
- Initially hardcoded navigation routes, later updated to use Markdown files for dynamic route management.

Features

- Displays navigation links and includes a hamburger menu for mobile views.
- JavaScript functionality for toggling the mobile menu and highlighting active links.

2.2.3 Iconbar

This component displays social media icons that link to various social media platforms.

Code Overview:

- Shows social media icons that link to various platforms.
- Initially hardcoded social media links, later made more modular for easier updates.

Features

- Uses SVG icons for scalability and quality.
- Each icon links to a respective social media page and opens in a new tab.

2.2.4 Footer

The Footer component provides essential information and links typically found at the bottom of the webpage.

Features

- Displays essential information such as copyright notices, links to terms of service, privacy policy, and social media icons.
- Initially, the content was hardcoded directly into the component. Later, you switched to using Markdown files for more flexible and manageable content.

2.3 Main Components

- Accordion
- Carousel
- Header
- HomeBody
- ResourcesBody
- ScheduleBody

2.3.1 Accordion

This component is responsible for displaying expandable/collapsible sections with headings and content.

Code Overview:

- Displays a set of collapsible sections, each with a heading and expandable content.
- Initially hardcoded content, later updated to use dynamic Markdown files for easier content management.

Features

- Responsive design ensures proper display on different screen sizes.
- JavaScript handles the expansion and collapse of sections with smooth animations.
- Content can be dynamically updated through Markdown files.

2.3.2 Carousel

This component is responsible for displaying a horizontal gallery of images.

Code Overview:

- Shows a series of images in a horizontal scrollable view.
- Initially hardcoded image data, later updated to use dynamic Markdown files for easier content management.

Features

- Responsive design ensures it adapts to various screen sizes.
- Includes support for scrolling through images using mouse wheel or navigation buttons (currently commented out).
- Images can be dynamically updated through Markdown files.

2.3.3 Header Component

This component is responsible for displaying the navigation and branding elements at the top of the website.

Code Overview:

- Displays the website's branding and navigation menu.
- Initially used static data for navigation items, later updated to use dynamic Markdown files for flexible content management.

Features

- Responsive design for optimal display on various devices.
- Includes navigation links to different sections of the website.
- Dynamic content updates using Markdown files for menu items and branding details.

2.3.4 HomeBody

This component serves as the main content display for a hackathon or event.

Code Overview:

- Displays various sections including introduction, significance, purpose, registration, partners, FAQ, a nd core team.
- Initially hardcoded data for content, later updated to use dynamic Markdown files for easier content management.

Features

- Responsive design ensures all elements are properly displayed on different devices.
- Includes sections for event introduction, significance, purpose, registration links, partner logos, FAQs, and team details.
- Content and team information can be dynamically updated through Markdown files.

2.3.5 ResourcesBody

This component handles the display of resources or content relevant to the website or application.

Code Overview:

- Displays a list of resources, such as documents, links, or other content.
- Initially hardcoded resources, later updated to fetch content dynamically from Markdown files for easier management.

Features

- Responsive design to ensure content is accessible on different devices.
- Includes links and descriptions for each resource.
- Resources can be updated dynamically using Markdown files.

2.3.6 ScheduleBody

This component handles the display of the schedule or agenda for an event, conference, or any time-based content.

Code Overview:

- Displays a detailed schedule or agenda, including timings, sessions, and speakers.
- Initially used static data, later updated to use dynamic Markdown files for flexible schedule management.

Features

- Responsive design to ensure the schedule is readable on various devices.
- Includes detailed timings and session information.
- Schedule content can be dynamically updated through Markdown files.

Chapter 3

Extended Functionality and Improvements

3.1 New Page for Results Announcement

3.1.1 Results Page

Code Overview:

- The Results Page utilizes the Layout component for structuring the page.
- It incorporates several components: TopBar, Navbar, IconBar, ResultsBody, and Footer.
- The page includes a script to handle date comparison:
  - Imports a configuration file to get the result date.
  - Calculates the target time based on the result date from the configuration.
  - Compares the current date with the target time.
  - Redirects to the /announce page if the current date is past the target time.

Features:

- Displays the results-related content through the ResultsBody component.
- Uses a consistent layout with other pages, integrating navigation and footer components.
- Includes a date-based redirection feature to ensure timely access to results.
- Provides a responsive and user-friendly interface for viewing results.

3.2 Modularizing Content in JSON Files

3.2.1 Data Integration and Fetching

Code Overview:

- The task involved moving data management from hardcoded values to dynamic JSON files.
- Created JSON files to store data for various components, such as banners, schedules, resources, and results.
- Updated components to fetch data from these JSON files rather than using hardcoded values.
- Implemented data fetching logic in components to dynamically render content based on the data from the JSON files.

Features:

- Modular Data Management:
  - Improved maintainability by separating data from the component logic.
  - Enabled easier updates and modifications to data without altering the component code.
- Dynamic Content Rendering:
  - Components now dynamically render content based on the data fetched from JSON files.
  - Ensures that changes to the data are reflected in the UI without needing code changes.
- Enhanced Flexibility:
  - Allowed for easier integration of new data and features.
  - Facilitated the addition of new banners, schedules, resources, and results by simply updating JSON files.

.

3.3 Authentication

3.3.1 Lucia: Simplifying Authentication

Lucia is a robust authentication library designed to streamline the implementation of secure user authentication in web applications. Here are its key features and components:

- Session Management: Handles session creation, validation, and invalidation, ensuring users stay authenticated across sessions.
- Password Security: Uses strong hashing algorithms like Argon2 to securely store and verify user passwords, protecting against unauthorized access.
- Flexible Integration: Easily integrates with various databases for storing user credentials and session data.
- Error Handling: Provides built-in mechanisms for handling common authentication errors, ensuring a smooth user experience.
- Session Cookies: Manages session cookies to maintain user authentication state between requests.

3.3.2 SQLite3: Lightweight Database Management

SQLite3 is a compact, serverless SQL database engine. Key features include:

- Single File Storage: Stores data in a single file, simplifying deployment.
- Zero Configuration: Requires no server setup or management.
- SQL Support: Provides full SQL query capabilities for data manipulation.
- Lightweight: Suitable for embedded applications and small to medium-sized projects.

3.4 Authentication Pages and Components

3.4.1 Login Body Component

This component handles the display and functionality of the admin login form on the login page.

Code Overview:

- Displays a login form for admin users to access the results before the public announcement.
- Handles form submission and displays errors if login fails.
- Redirects to the admin page if login is successful.

Features:

- Clean and user-friendly UI for the login form.
- Client-side validation and error handling for better user experience.
- Includes a signup button for new admin users.

3.4.2 Signup Body Component

This component handles the display and functionality of the signup form on the signup page.

Code Overview:

- Displays a signup form for new admin users to create an account.
- Handles form submission and displays errors if the signup fails.
- Redirects to the admin page if the signup is successful.

Features:

- Clean and user-friendly UI for the signup form.
- Client-side validation and error handling for better user experience.
- Includes a sign-in button for existing users to navigate to the login page.

3.4.3 Login Page

This page is responsible for handling user authentication for accessing the results before the fixed time.

Code Overview:

- Checks if the user is already logged in; if so, redirects them to the results announcement page.
- If the user is not logged in, displays the login form.

Features:

- Ensures only authorized users (admins) can access the results before the specified time.
- Simple and secure login mechanism.

3.4.4 Signup Page

This page includes a signup form for new admin users to create an account.

Code Overview:

- Imports necessary components such as Layout, TopBar, Navbar, Footer, IconBar, and SignupBody.
- Uses a structured layout to maintain consistency across pages.
- Provides a clear structure for the signup process, ensuring ease of use and navigation.

Features:

- Implements a responsive design with components organized for clarity and functionality.
- Integrates client-side validation and error handling within the signup form.
- Includes navigation links for seamless user experience between pages.

3.5 API Endpoints

3.5.1 Login API

- **Endpoint:** `/api/login`
- **Method:** POST
- **Description:** Authenticates a user by verifying their username and password. If the credentials are valid, a session is created, and the user is redirected to the admin page. If authentication fails, an error message is returned.
- **Request Body:**
  - `username`: The username of the user.
  - `password`: The password of the user.
- **Response:**
  - **200 OK**: Redirects to the admin page if login is successful.
  - **400 Bad Request**: Error message if the username or password is invalid or incorrect.
- **Validation:**
  - Username must be between 3 and 31 characters, and only consist of lowercase letters, numbers, hyphens, and underscores.
  - Password must be between 6 and 255 characters.

3.5.2 Logout API

- **Endpoint:** `/api/logout`
- **Method:** POST
- **Description:** Logs out the current user by invalidating their session. A blank session cookie is set to clear any existing session data.
- **Response:**
  - **200 OK**: Successfully logs out the user and clears session data.
  - **401 Unauthorized**: If no session exists for the current request, indicating that no logout action can be performed.
- **Session Handling:**
  - Invalidates the current session if it exists.
  - Sets a blank session cookie to remove any existing session data.

3.5.3 Signup API

- **Endpoint:** `/api/signup`
- **Method:** POST
- **Description:** Registers a new user by creating an account with the provided username and password. On successful signup, a session is created, and the user is redirected to the admin page. If the registration fails due to an existing username or other issues, an error message is returned.
- **Request Body:**
  - `username`: The desired username for the new account.
  - `password`: The password for the new account.
- **Response:**
  - **200 OK**: Redirects to the admin page if signup is successful.
  - **400 Bad Request**: Error message if the username is already in use or if the request data is invalid.
  - **500 Internal Server Error**: Indicates an unknown error occurred during signup.
- **Validation:**
  - Username must be between 3 and 31 characters, and only consist of lowercase letters, numbers, hyphens, and underscores.
  - Password must be between 6 and 255 characters.
  - If the username is already used, a unique constraint violation error is returned.

3.6 Middleware for Authentication

This middleware handles authentication and request validation:

- **Request Method Check:** Validates that the request method is GET. For non-GET requests, it checks the 'Origin' and 'Host' headers to ensure the request is from an allowed origin.
- **Session Management:** Retrieves the session ID from cookies and validates it. If the session is valid and fresh, it updates the session cookie. If not, it sets a blank session cookie.
- **Local Variables:** Sets `context.locals.session` and `context.locals.user` based on the validated session and user data.

3.7 Library Components and Database Management

3.7.1 databaseUpdates.ts: Database Operations and User Management

- **Description:** Manages database operations and user interactions. Includes functions to print all users and delete a specific user by ID. Handles foreign key constraints to maintain database integrity during delete operations.
- **Functions:**
  - `printAllUsers()`: Fetches and displays all user entries from the `user` table.
  - `deleteUserById(userId)`: Deletes a user from the `user` table based on the provided user ID. Temporarily disables foreign key constraints during the delete operation.

3.7.2 db.ts: Database Initialization and Schema Management

- **Description:** Initializes and updates the database schema. Ensures that the `user` and `session` tables are created if they do not already exist. Defines the structure of user data with the `DatabaseUser` interface.
- **Tables Created:**
  - `user`: Stores user information including `id`, `username`, and `password`.
  - `session`: Manages session data with fields for `id`, `expires_at`, and `user_id`. Implements foreign key constraints to link sessions to users.
- **Interface:**

  - `DatabaseUser:` Defines the user data structure with `id`, `username`, and `password`.

3.7.3 auth.ts (Lucia Configuration): Lucia Authentication Setup

- **Description:** Configures the Lucia authentication library with SQLite integration using the `BetterSqlite3Adapter`. Manages session creation, secure cookie attributes, and user attribute retrieval.
- **Key Features:**
  - `Session Management:` Handles session creation and validation, ensuring secure and persistent user sessions.
  - `Password Security:` Utilizes hashing algorithms to securely store and verify user passwords.
  - `Integration:` Easily connects with SQLite database for storing user credentials and session information.
  - `Error Handling:` Provides mechanisms for managing authentication errors and session issues.

3.8 Overview of Python Script for Markdown Conversion

The Python script automates the conversion of JSON data into Markdown files to enhance readability and accessibility for non-developers. The script performs the following key operations:

- **Reads JSON Data:** Opens and loads JSON data from a specified input file.
- **Creates Output Directory:** Ensures the output directory exists or creates it if necessary.
- **Generates Markdown Files:** Iterates over each entry in the JSON data, formats it into Markdown, and writes it to individual files.
- **Completion Message:** Prints a success message indicating the Markdown files have been created.

3.9 Modularization of Data into Markdown Format

The script modularizes JSON data into Markdown format to make it more accessible. Key steps include:

- **Input and Output Paths:** Defines paths for input JSON data and output Markdown files.
- **Data Extraction:** Extracts relevant fields from the JSON data, such as name, position, and institute.
- **Markdown Formatting:** Formats the extracted data into a Markdown template, with fields like `Number`, `Description`, `Content` and other fields depending on the file data.
- **File Creation:** Saves each entry as a separate Markdown file with a sequential filename.

This approach ensures that the data is presented in a readable format, facilitating easier access and management.

3.10 Animation and Redirection Enhancement

To enhance user engagement before transitioning to the result announcements page, the following features were implemented:

- **Animated Congratulations Message:** Implemented a bounce animation for each letter in the "Congratulations" message, creating a dynamic and visually appealing effect.
- **Confetti Animation:** Added a confetti effect that starts shortly after the page loads, providing a celebratory atmosphere. The confetti continues for a few seconds before stopping.
- **Automated Redirection:** Configured the page to automatically redirect users to the result announcements page after the animations are complete, ensuring a smooth transition.

Chapter 4

Codebase Optimizations and Enhancements

4.1 Code Documentation and Cleanup

- **Description:** This task focused on enhancing the readability and maintainability of the codebase by:
  - **Adding Comments:** Included comments throughout the code to explain functionality and logic, making it easier for developers to understand and work with the code.
  - **Removing Console Logs:** Cleaned up unnecessary console log statements used during development, improving code clarity and reducing clutter in the production environment.
  - **Updating Documentation:** Revised the README file to provide clear instructions and guidelines for other developers, including details on how to edit Markdown files and navigate the codebase.

4.2 Component Publishing to npm

- **Description:** To improve efficiency and consistency, reusable components were published to npm (Node Package Manager). This process involved:
  - **Component Creation:** Refactoring common UI elements into reusable components.
  - **Publishing:** Configuring and publishing these components as an npm package.
  - **Integration:** Updating the project to use the npm-published components instead of duplicating code.
- **Benefits:**
  - **Consistency:** Uniform design across the application.

    - **Efficiency:** Simplified updates and maintenance.
    - **Maintainability:** Easier management of common components.

4.3 Enhancements to Website Functionality

- **Active Link Styling in Navbar:**
    - **Description:** Enhanced the navigation bar by adding active link styling to indicate the currently selected page. This improvement provides a clearer visual cue for users, making it easier to identify their current location within the site.
    - **Implementation:** Updated the CSS to include styles for the active link state. Added logic to dynamically apply the active class based on the user's current page, ensuring that the corresponding link is visually distinguished.
- **Countdown Timer on Results Page:**
    - **Description:** Added a countdown timer to the results page to provide users with a visual countdown to an upcoming event or deadline. This feature helps in creating anticipation and keeping users informed about important time-related details.
    - **Implementation:** Integrated a JavaScript-based countdown timer into the results page. Configured the timer to display the remaining time and updated the page dynamically as the countdown progresses.

Chapter 5

Minor Task

5.1 Modularizing Animate 2024 Website Content

- **Description:** This task involved restructuring the content of the Animate 2024 website to improve manageability and readability by converting it into Markdown files.
  - **Content Extraction:** Extracted all text and data from the existing website to serve as the source material for modularization.
  - **Markdown Conversion:** Converted the extracted content into Markdown format, creating individual Markdown files for different sections or components of the website. This included text content, headings, lists, and other relevant data.
  - **File Organization:** Organized the Markdown files in a structured directory layout, making it easier to update and maintain specific sections without affecting other parts of the site.
  - **Benefits:** This approach simplifies content management, enhances readability for non-developers, and facilitates easier updates and maintenance by separating content into discrete, manageable files.

Reference

- Astro Docs : Astro Docs
- Npm package Docs : Npm package Docs
- Lucia Auth Docs : Lucia Auth Docs

Work is Available at

- FOSSEE Site : FOSSEE
- ANIMATE2024 Site : ANIMATE2024
- Npm package : Npm Package