Module Guide for Software Engineering

Team 4, EvENGage
Virochaan Ravichandran Gowri
Omar Al-Asfar
Rayyan Suhail
Ibrahim Quraishi
Mohammad Mahdi Mahboob

January 21, 2026

1 Revision History

Date	Version	Notes
November 13, 2025	1.0	Rev -1
January 21, 2025	1.1	Revision 0 finished. Updated Sections 10,11,12
January 21, 2025	1.1	Finished Abbreviations table.

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
DAG	Directed Acyclic Graph
M	Module
MG	Module Guide
R	Requirement
SRS	Software Requirements Specification
UC	Unlikely Change
MES	McMaster Engineering Society
SMS	Short Message Service (Text Message)
UI	User Interface
ORM	Object-Relational Mapping

Contents

List of Tables

List of Figures

3 Introduction

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team (Parnas et al., 1984). We advocate a decomposition based on the principle of information hiding (Parnas, 1972). This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules layed out by Parnas et al. (1984), as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is implemented in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (Parnas et al., 1984). The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.
- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility, and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section 4 lists the anticipated and unlikely changes of the software requirements. Section 5 summarizes the module decomposition that was constructed according to the likely changes. Section 6 specifies the connections between the software requirements and the modules. Section 7 gives a detailed description of the modules. Section 8 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section 9 describes the use relation between modules.

4 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 4.1, and unlikely changes are listed in Section 4.2.

4.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: The user interface of the app. Modifications may be made to improve usability, accessibility, or better represent the use case (eg. MES).

AC2: The form builder functionality. The structure and logic of registration and feedback forms may change to support new fields, features, or requirements.

AC3: The user authentication and authorization policy. The security and user requirements may evolve, requiring the introduction of new authentication techniques (eg. custom login, university single sign-on). There may also be a need to add new user/admin roles, or modify the permissions process based on the use case.

AC4: Backend functionality. The database handling or entry relationships may change to support scalability, or custom event workflows.

AC5: Notification delivery methods. Communication channels (email, SMS, push notifications) and message formats may improve to meet updated communication needs for event reminders, updates, or confirmations.

AC6: Security and encryption network. The app's security standards may require updated encryption algorithms, expanded input validation, and refined logging detail depending on the use case to ensure compliance.

AC7: External integration methods. The app may be required to connect with specialized APIs, external databases, or other third-party systems (eg. payment gateways). The data formats and protocols used in these integrations may evolve over time.

4.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: Changes to the primary application of events, registration, tickets, and feedback. The development is focused solely on event management and is not expected to change to accomodate other uses.

UC2: Migrating to a different technology stack. The current architecture is designed around an existing framework. A different technology stack would require reimplementing nearly all modules.

UC3: Transitioning from a centralized database model. The platform relies on centralized data storage for all event management, and applies role-based access to ensure users are only viewing permitted information. Moving to a different model (eg. decentralized) would conflict with the functional requirements, and require rethinking the entire user interaction model.

5 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 1. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Main System Module

M2: User Authentication Module

M3: User Authorization Module

M4: Form Template Module

M5: Form Submission Module

M6: Event Management Module

M7: Event Notification Module

M8: Registration Module

M9: Attendance Tracking Module

M10: Report Generation Module

M11: Analytics Module

M12: Database Access Module

M13: Audit Module

Level 1	Level 2
Hardware-Hiding Module	No Modules
Behaviour-Hiding Modules	M1: Main System Module M2: User Authentication Module M3: User Authorization Module M4: Form Template Module M5: Form Submission Module M6: Event Management Module M7: Event Notification Module M8: Registration Module M9: Attendance Tracking Module M10: Report Generation Module
Software-Decision Modules	M11: Analytics Module M12: Database Access Module M13: Audit Module

Table 1: Module Hierarchy

6 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Tables 2 to 10.

7 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by Parnas et al. (1984). The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. *Software Engineering* means the module will be implemented by the Software Engineering software.

Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented.

7.1 Hardware Hiding Modules

There are no hardware components for this system.

7.2 Behaviour-Hiding Module

7.2.1 Main System Module (M1)

Secrets: Defines the business logic which allows for the communication between the different modules and with external users
Services: Manages and stores the application states and context and serves as the central entrypoint to the system and which modules to communicate with.
Implemented By: React and NodeJS
Module Type: Abstract Object Module.

7.2.2 User Authentication Module (M2)

Secrets: The internal methods used for verifying user identities within the system.
Services: Authenticates users by validating credentials and provides access to the system.
Implemented By: Backend JavaScript API.
Module Type: Abstract Object Module.

7.2.3 User Authorization Module (M3)

Secrets: Rules defining access levels and user roles. Defines which features are to be accessed by each role.
Services: Provides the system with a ruleset on what each roles have access to and limits/grants access to different parts of the system to users based on credentials.
Implemented By: Backend JavaScript API.
Module Type: Abstract Data Type Module.

7.2.4 Form Template Module (M4)

Secrets: The structure and layout of form templates used in surveys and event creation.
Services: Provides reusable blueprints for constructing event or survey forms. Allows for the creation of new forms and editing of preexisting forms.
Implemented By: React Frontend, Backend PostgreSQL Database Schemas.
Module Type: Abstract Data Type Module.

7.2.5 Form Submission Module (M5)

Secrets: Validation rules of the user input data.
Services: Provides mechanism for user to answer and submit forms. Receives and validates user-submitted forms and stores them in database.

Implemented By: Backend form processing JavaScript API.
Module Type: Abstract Object Module.

7.2.6 Event Management Module (M6)

Secrets: Event data structures and scheduling rules.
Services: Enables creation, editing, and cancellation of events by administrators.
Implemented By: React Frontend with Backend event processing JavaScript API.
Module Type: Abstract Object Module.

7.2.7 Event Notification Module (M7)

Secrets: Notification delivery logic and timing rules.
Services: Sends alerts to users regarding new events, updates, or cancellations.
Implemented By: Third-Party Messaging Apis.
Module Type: Library.

7.2.8 Registration Module (M8)

Secrets: Mapping between users, events, and registration states.
Services: Allows users to register, modify, or cancel event participation. Provides validation and confirmation of registration within the event. Allows for admins to view users who have registered for each event and the stage of the process they are in.
Implemented By: Backend JavaScript APIs.
Module Type: Abstract Data Type Module.

7.2.9 Attendance Tracking Module (M9)

Secrets: Methods for recording attendance and validating entry codes.
Services: Tracks event attendance and verifies participant access. Allows for admins for to view
Implemented By: Backend JavaScript APIs.
Module Type: Abstract Data Object Module.

7.2.10 Report Generation Module (M10)

Secrets: Formatting logic for report generation and provides static data structure of the report.
Services: Converts analytics data into exportable human-readable reports.
Implemented By: Third Part Apis with JavaScript and React.
Module Type: Abstract Data Object Module.

7.3 Software Decision Module

7.3.1 Analytics Module (M11)

Secrets: Algorithms for aggregating and calculating statistics.
Services: Computes summaries based on the data provided for set statistics and performs data analysis.
Implemented By: Javascript Backend Library.
Module Type: Library.

7.3.2 Database Access Module (M12)

Secrets: Database schema design and access.
Services: Provides database operations to query and insert to database through a unified database layer.
Implemented By: Drizzle ORM with JavaScript APIs.
Module Type: Abstract Data Object Module.

7.3.3 Audit Module (M13)

Secrets: Policy and data structure for recording administrative actions.
Services: Provides methods to track user and admin activities to ensure traceability and compliance.
Implemented By: Backend Javascript API with PostgreSQL Database.
Module Type: Abstract Data Type Module.

8 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	Modules
FR-1	M1,M3, M4
FR-2	M1,M9, M10
FR-3	M1,M7, M5
FR-4	M1,M7
FR-5	M1,M5
FR-6	M1,M11
FR-7	M1,M11, M3, M4
FR-8	M1,M9, M10
FR-9	M1,M3, M4
FR-10	M1,M5
FR-11	M1, M2

Table 2: Trace Between Functional Requirements and Modules

Req.	Modules
AR-1	M1
AR-2	M1
AR-3	M1, M10, M11
SR-1	M1
SR-2	M1, M4, , M6, M10

Table 3: Trace between Look and Feel Requirements and Modules

Req.	Modules
ER-1	M4, M5, M8
ER-2	M1, M2
ER-3	M1, M4, M10, M11
PI-1	M4
PI-2	M1, M9, M11
LR-1	M1
LR-2	M1, M4
UR-1	M1
UR-2	M1
UR-3	M1, M12, M13
UR-4	M1
AC-1	M1

Table 4: Trace between Usability and Humanity Requirements and Modules

Req.	Modules
SL-1	M1, M5, M8, M12
SL-2	M9, M11, M12
SC-1	M1, M12, M13
PA-1	M11
FT-1	M1, M5
FT-2	M1, M5, M12
FT-3	M1, M4, M12
FT-4	M1, M6, M12
FT-5	M11
CR-1	M1, M12
CR-2	M12
SE-1	M12
SE-2	M1, M11, M12
LG-1	M1
LG-2	M1, M12, M10

Table 5: Trace between Performance Requirements and Modules

Req.	Modules
PE-1	
IR-1	M1
IR-2	M1
PD-1	M1
PD-2	M1
RR-1	

Table 6: Trace between Operational & Environmental Requirements and Modules

Req.	Modules
MT-1	
SU-1	M1
AD-1	M1
AD-2	M1

Table 7: Trace between Maintainability & Support Requirements and Modules

Req.	Modules
AC-1	M2
AC-2	M2, M3
IG-1	M1, M5, M12
IG-2	M1
PV-1	M1, M12, M13
PV-2	M1, M2, M12
PV-3	M12, M13
AU-1	M13

Table 8: Trace between Security Requirements and Modules

Req.	Modules
CL-1	M1

Table 9: Trace between Cultural Requirements and Modules

Req.	Modules
LG-1	M12, M13
ST-1	M1

Table 10: Trace between Compliance Requirements and Modules

AC	Modules
AC1	M1, M4, M6, M10
AC2	M4, M5
AC3	M2, M3
AC4	M1, M6, M8, M12
AC5	M7, M10
AC6	M2, M12, M13
AC7	M1, M7, M10, M12

Table 11: Trace Between Anticipated Changes and Modules

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Parnas (1978) said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure 1 illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

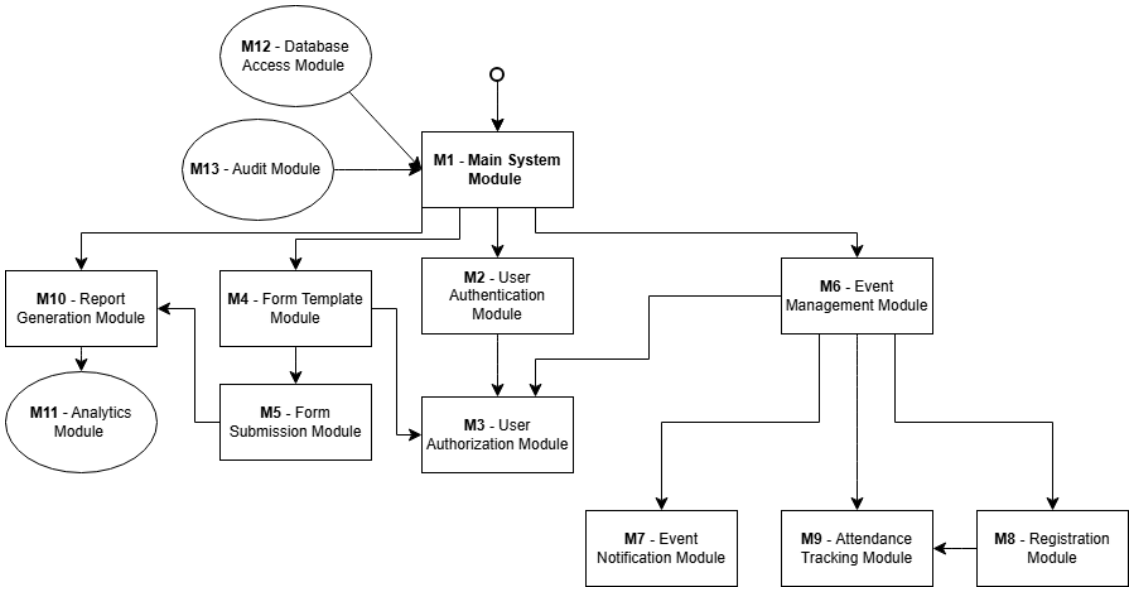

Figure 1: Use hierarchy among modules

10 User Interfaces

The platform offers a web application with supporting two distinct user views, general users and administrators.

10.1 Attendee Interface

Initially, users are presented with a sign-up/log-in Authentication screen (Figure **??**). Upon authentication, users are taken to a Homepage that serves as the foundation of the user app (Figure **??**). From this page, users can access the Events and Surveys sections through clearly labeled tabs, as well as profile information and the logout option.

The Events page displays a list of all available events (Figure **??**), each of which can be expanded to reveal detailed event information such as date, location, and description (Figure **??**). Users may register for an event directly from this page, after which the event would appear in the Registered Events tab (Figure **??**). In this tab, users can select registered events to view the generated QR code ticket for event check-in (Figure **??**).

The Surveys page operates with a similar structure. Available surveys are listed and can be selected to open a survey completion interface (Figure **??**). Surveys consist of various question types presented sequentially, including multiple choice, short answer, and scale ratings (Figure **??**). After submission, completed surveys are moved to the Completed Surveys tab, where users can review surveys they have already filled out (Figure **??**).

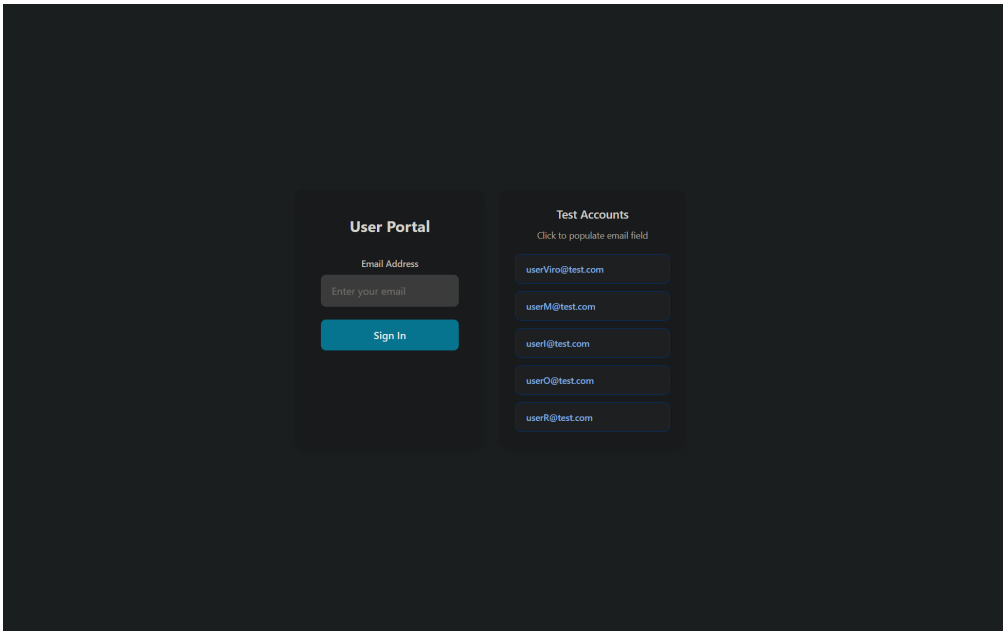

Figure 2: User - Authentication Page

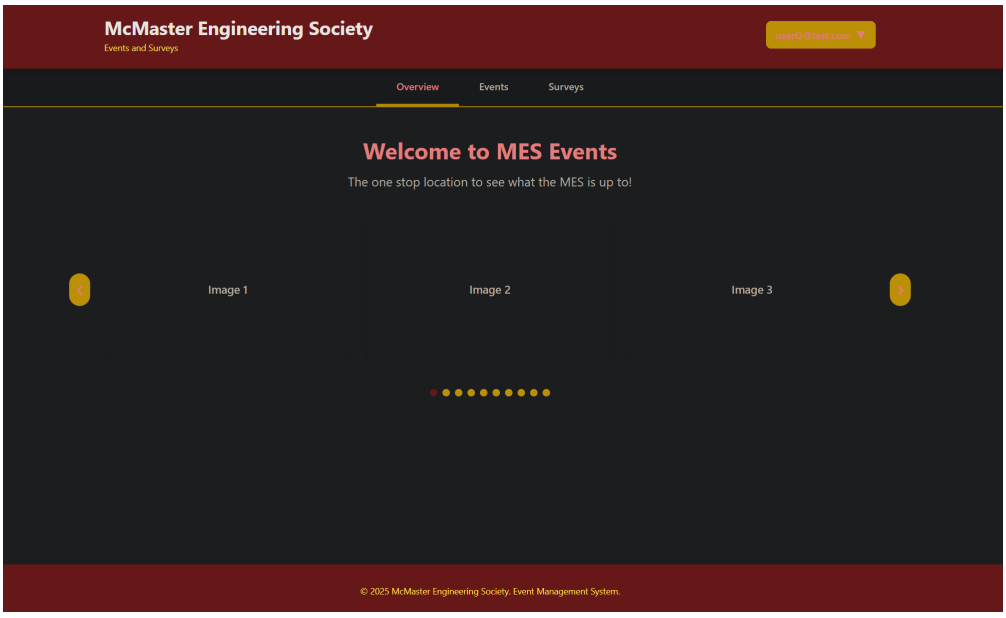

Figure 3: User - Homepage

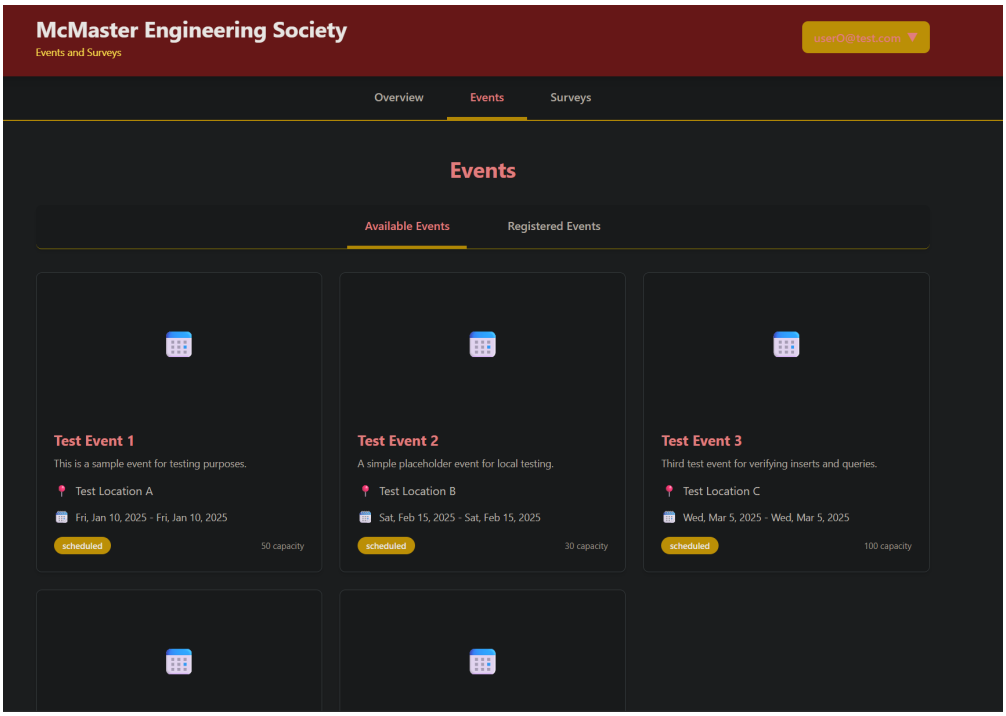

Figure 4: User - Available Events Page

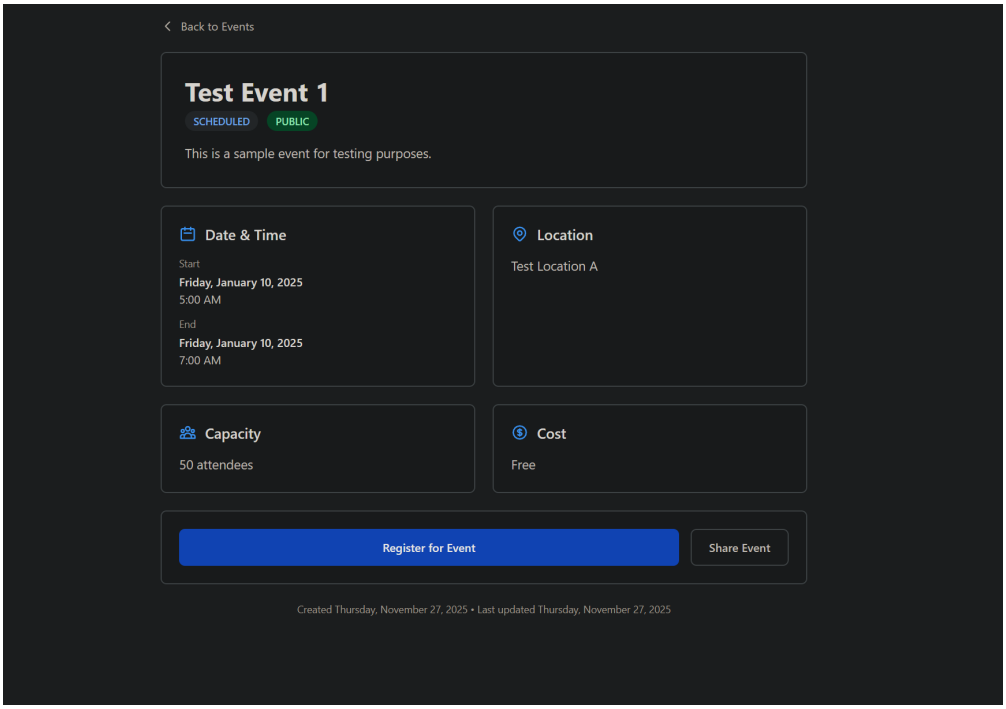

Figure 5: User - Event Details Page

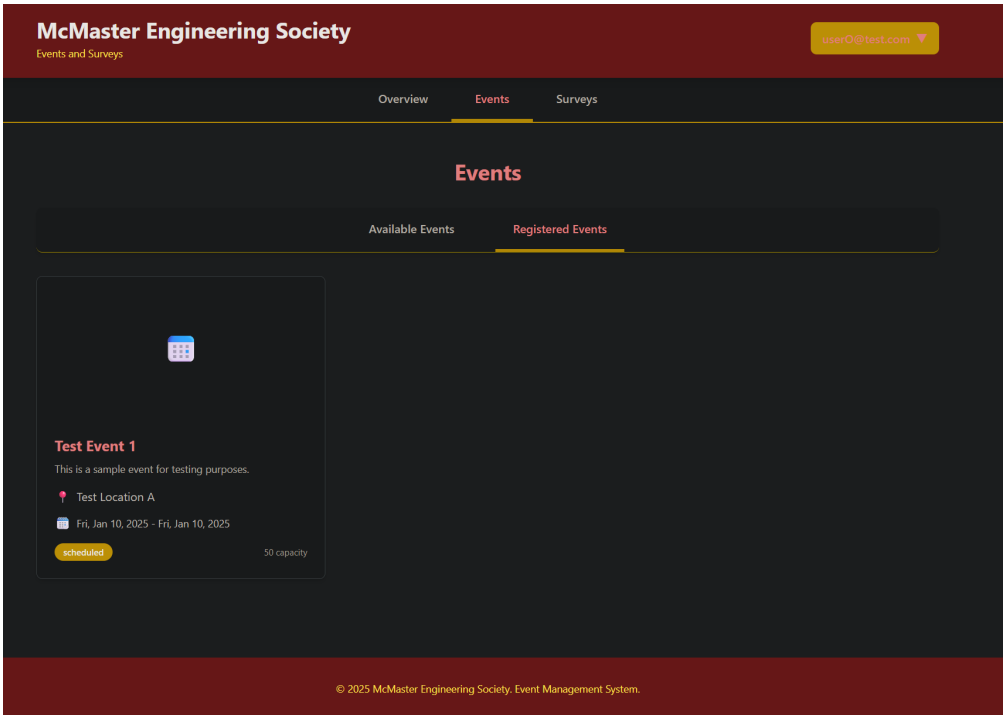

Figure 6: User - Registered Events Page

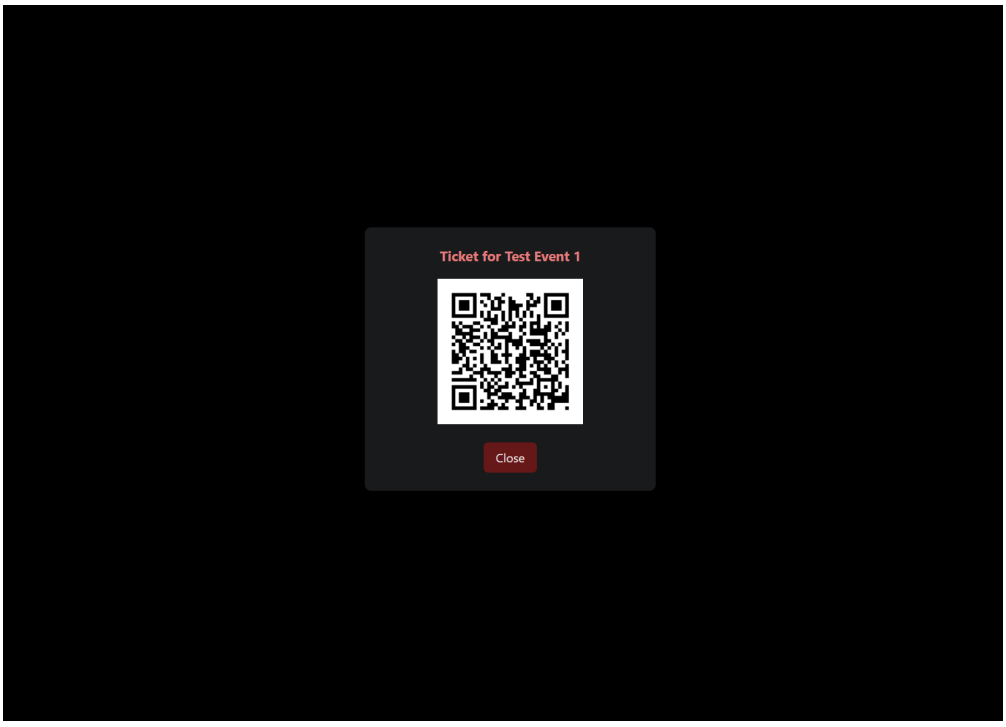

Figure 7: User - QR Code Ticket

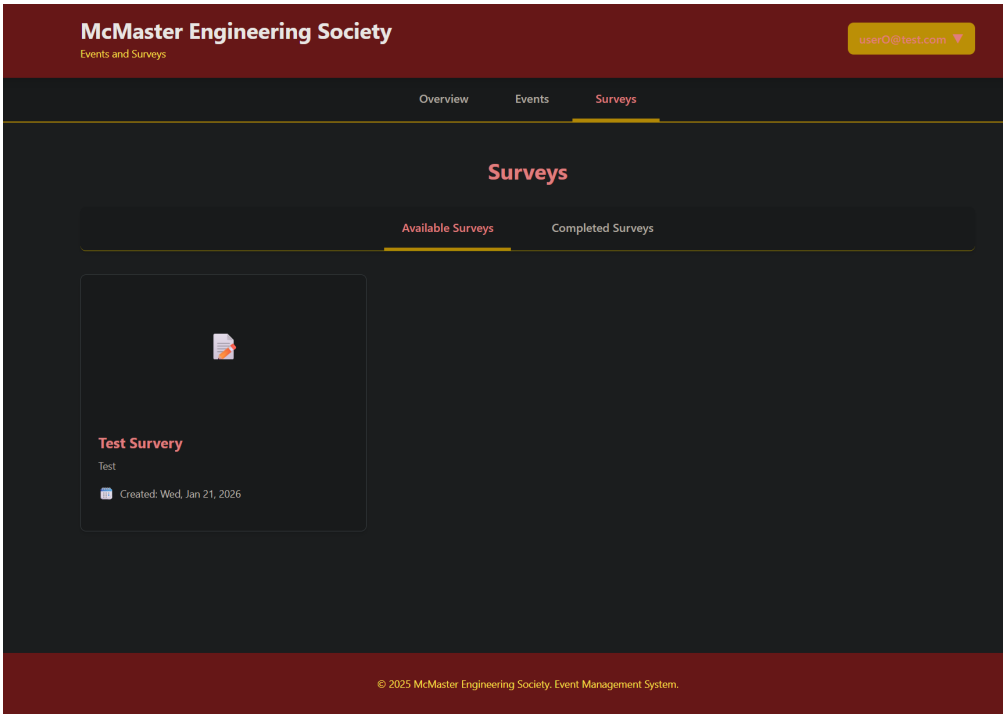

Figure 8: User - Available Surveys Page

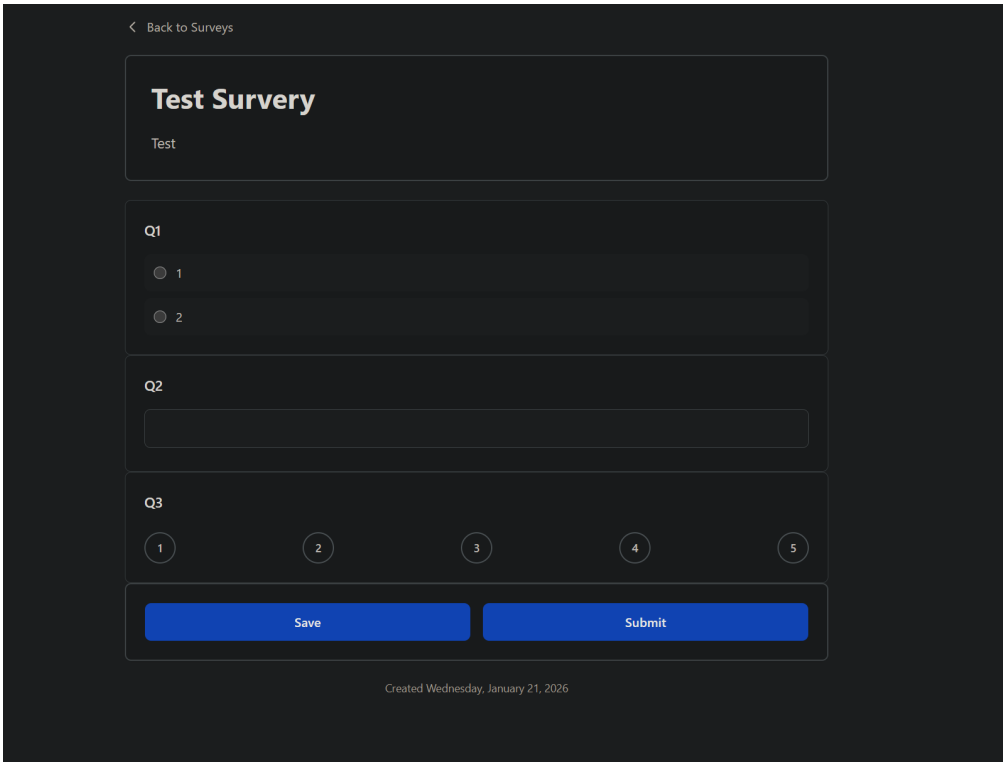

Figure 9: User - Survey Question Page

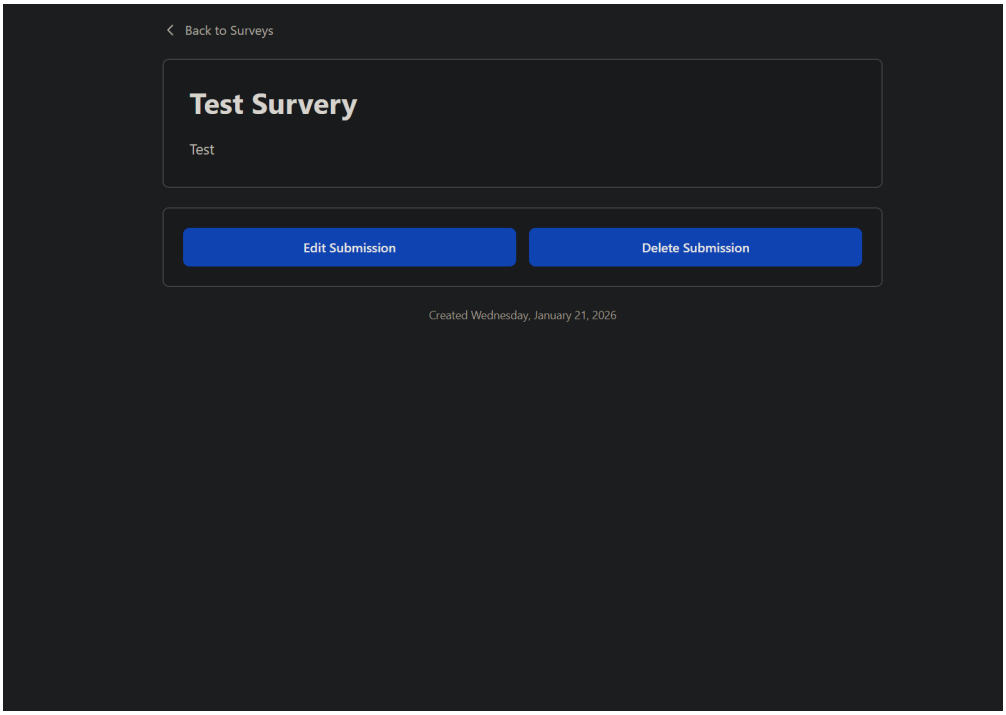

Figure 10: User - Completed Surveys

10.2 Admin Interface

Administrators access a separate interface designed for event and survey management. They start with a similar sign-up/log-in authentication process (Figure **??**). This sends the user to the Admin Dashboard which allows for navigation between different tabs (Figure **??**).

In the main dashboard view, the admin can view all existing events with key details, and create new ones. Clicking on existing events opens a page listing all registered users (Figure **??**). Admin can also create events from the dashboard, opening a modal requesting event details such as name, time, capacity, and description (Figure **??**). Upon submission, the event becomes immediately available to users in the aforementioned user Available Events Page.

Admin can also use the Form Builder to create and manage surveys (Figure **??**). When creating a survey, admin can add questions and customize surveys by selecting from the different supported question types (Figure **??**). Completed surveys appear in the Available Surveys user list. Finally, Form Analytics allows admin to observe data and survey metrics, such as number of submissions, and response statistics. This data can be exported to a .csv file through the interactive button (Figure **??**).

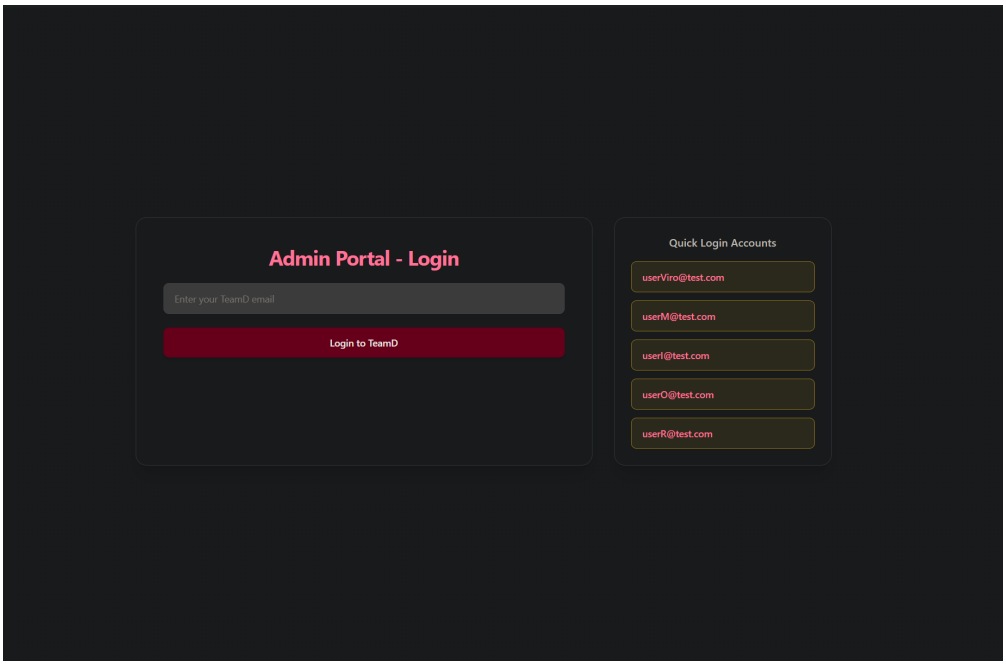

Figure 11: Admin - Authentication Page

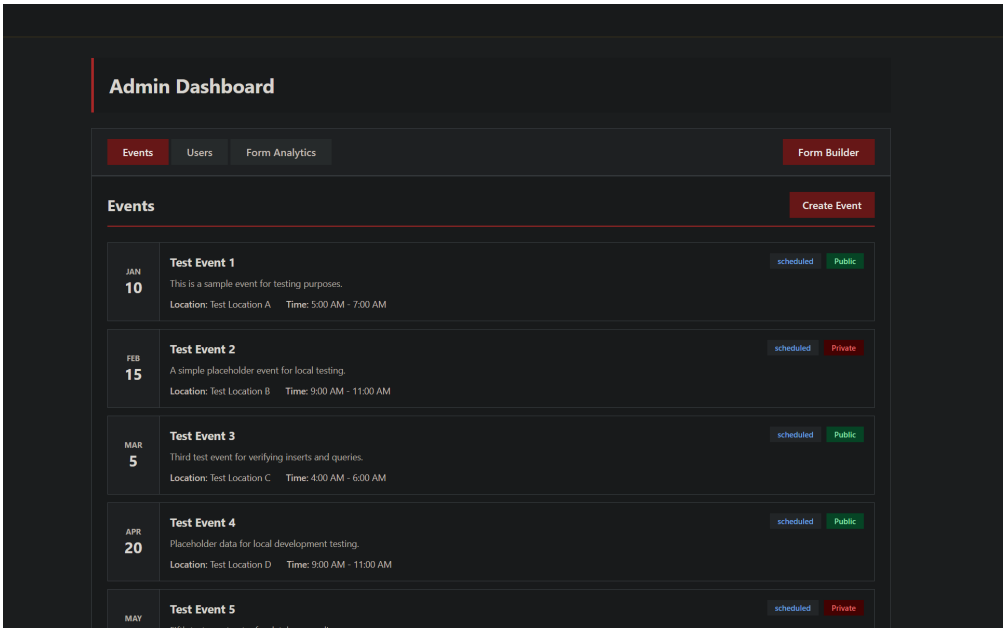

Figure 12: Admin - Dashboard

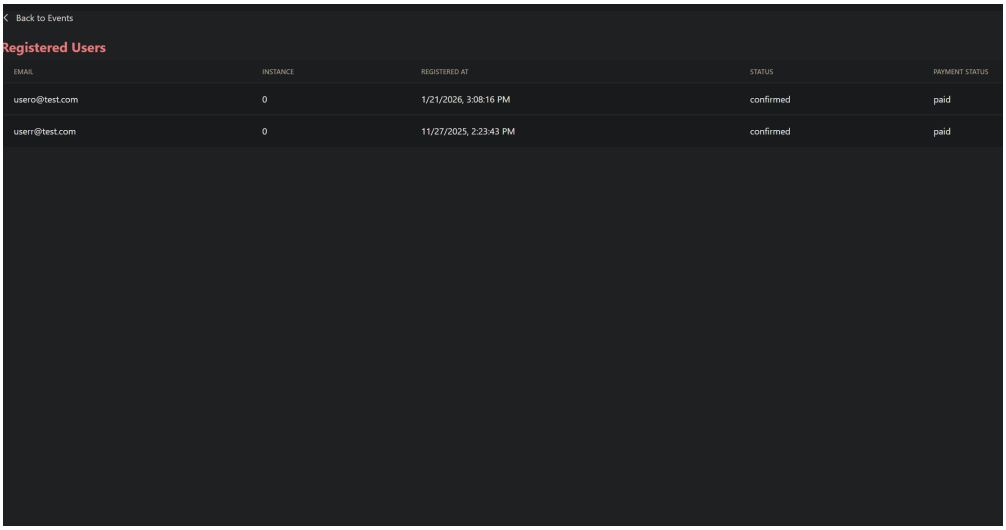

Figure 13: Admin - Event Registrations Page

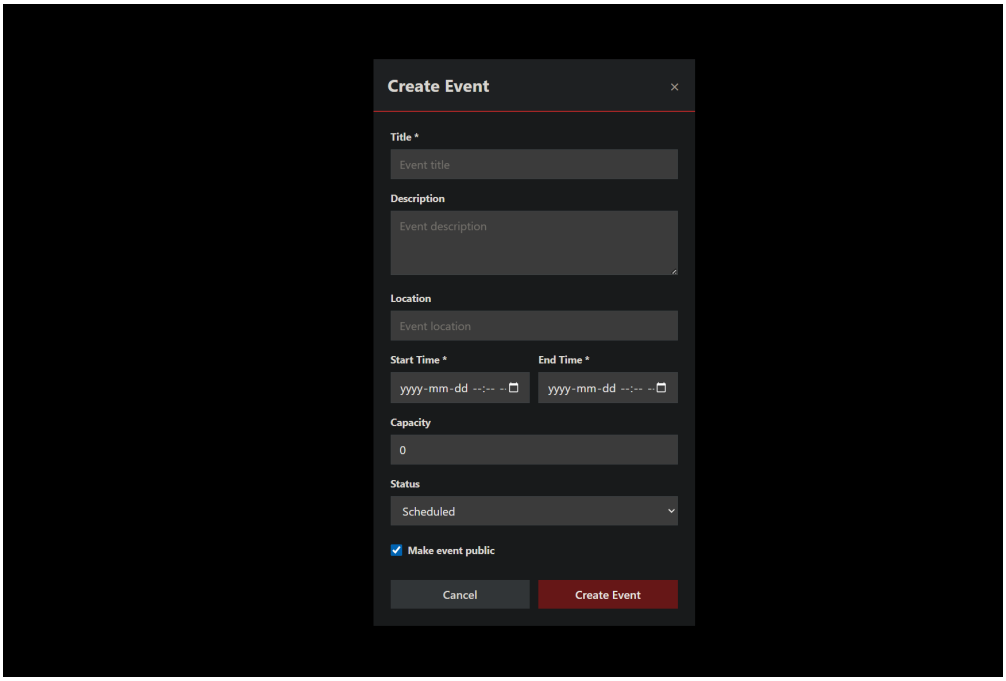

Figure 14: Admin - Event Creation Page

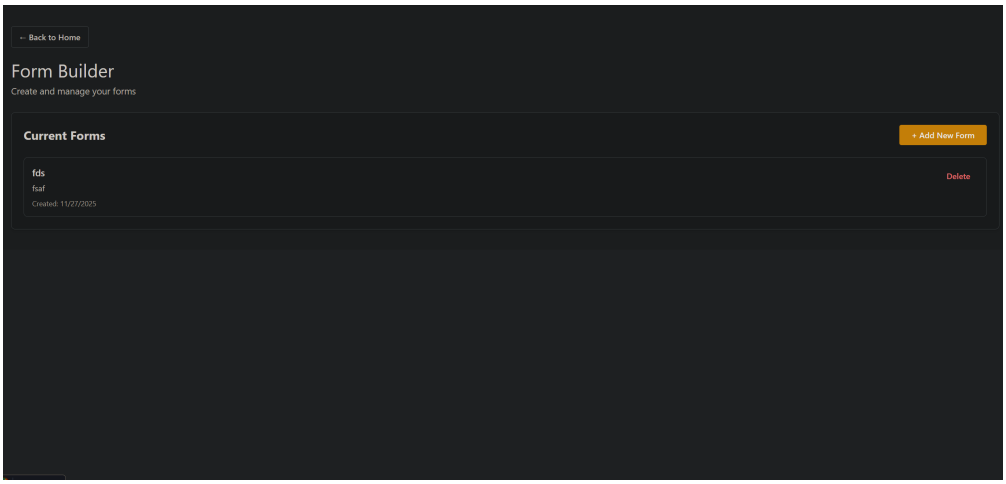

Figure 15: Admin - Form Builder

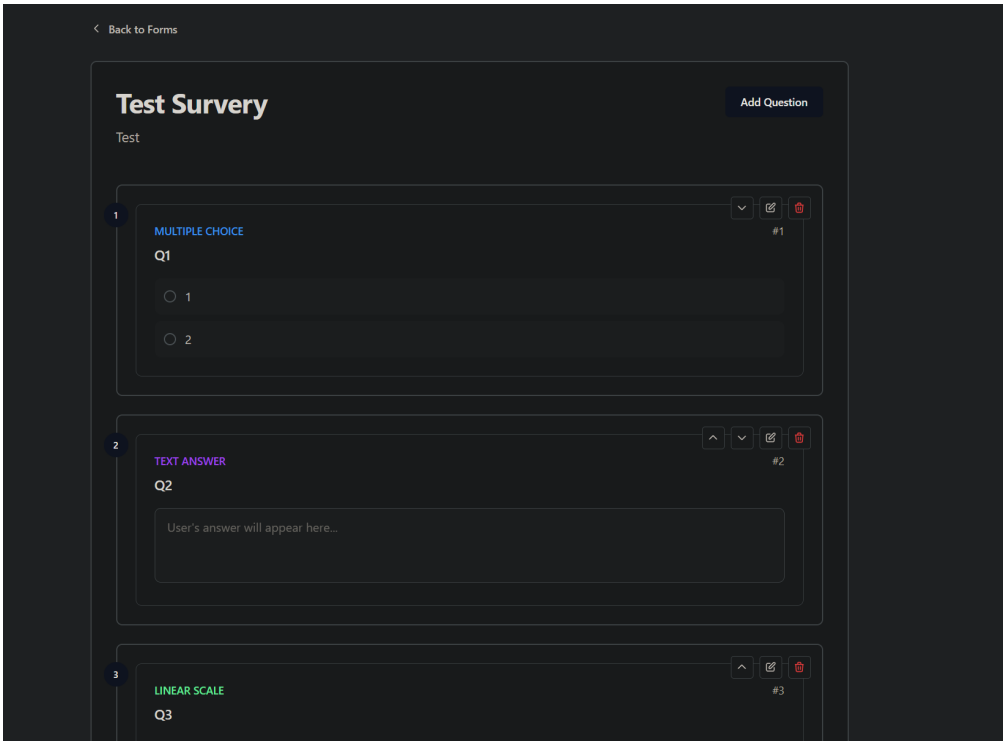

Figure 16: Admin - Survey Creation Page

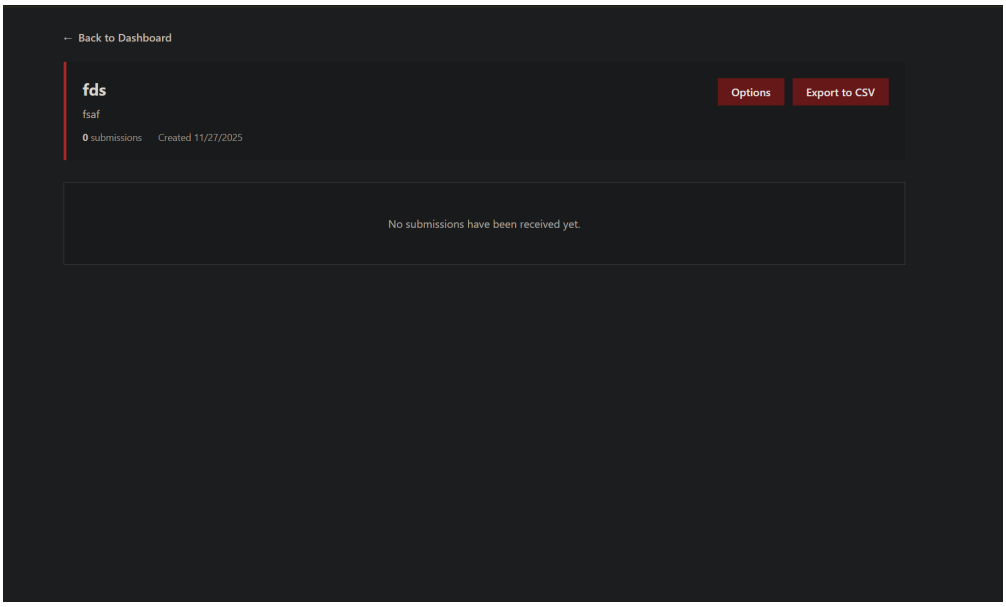

Figure 17: Admin - Form Analytics Page

11 Design of Communication Protocols

No relevant communication protocols are neccesary for this project.

12 Timeline

The timeline shows the development of our platform from december after our POC. It describes the deadlines for implementation of the modules and who will be taking ownership of the module. Some modules may have to be updated based on implementations of other modules but this deadline represents implementation of base feature set.

Week 1 (December 1 - 7):

- M1 - Main System Module (Ibrahim): Set up the basic framework of the application with the various servers and portals for admin and users. Define interface and connection with other modules.
- M12 - Database Access Module (Virochaan): Set up the Database to work with the Main Application. Create various schema relating to the different stored data in the database.

Week 2 (December 8 - 14):

- M2 - User Authentication Module(Mohammad): Set up basic user authentication with the server. Ensure only allowed accounts can sign up and enter the platform. Create various test accounts for development and testing.
- M4 Form Template Module (Rayyan): Develop the framework to create forms including modular question types and form behaviours. Create form rendering and UI components.
- M5 Form Submission Module (Virochaan): Handle form submission, saving and editing behaviour as well as connecting the form submission with the Database.

Week 3, 4, 5 (December 15 - January 4):

- M10 - Report Generation Module(Omar): Develop UI components to read form data and display it in an easy to read manner. Provide buttons for exporting to varying file format. Develop functionality for users to select what data they want to see and export based on predefined categories.
- Testing Form Modules (Rayyan): Test full functionality of the form modules to ensure no errors. Ensure form modules are integrated well with the database and the database access module.

Week 6 (January 5 - 11):

- M6 - Event Management Module (Mohammad): Develop the framework to create and edit events. Develop functionality for updating of details regarding events as they change. Create Event rendering and UI components.
- M8 - Event Registration Module (Ibrahim): Allow for users to register to events and submit details. Develop UI components that allows for the user to view their tickets and admins to see registered users. Integrates the Events with the Database.

Week 7 (January 12 - 18):

- M3 - User Authorization Module (Virochaan): Develop various role types to enable feature based access control for users. Allows for admins to authorize other users to access different components of the platform.
- M11 - Analytics Module (Omar): Develop the various types of data analysis and process that will be done on the form data. Integrate with the report generation module to allow for export of the data.

Week 8 (January 19 - 25):

- M7 - Event Notification Module (Omar): Develop functionality to push notifications related to upcoming and registered events to the user.
- M11 - Attendance Tracking Module (Ibrahim): Interface for admins to view the status of registered users for each event. Provide admins with functionality to contact specific users and request more data.

Week 9 and Beyond (January 26 -):

- M13 - Audit Module (Mohammade): Develop UI components for admins to view who performed privileged actions.
- Testing (Rayyan, Virochaan): Perform integration testing to ensure all modules are working as intended. Ensure modules have sufficient unit testing to ensure correct functionality.
- Feedback Implementation (Entire Team): Implement feedback based on our supervisor's comments and through stages of usability testing and general improvements for the platform.

References

David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):1053–1058, December 1972.

David L. Parnas. Designing software for ease of extension and contraction. In *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, pages 264–277, Piscataway, NJ, USA, 1978. IEEE Press. ISBN none.

D.L. Parnas, P.C. Clement, and D. M. Weiss. The modular structure of complex systems. In *International Conference on Software Engineering*, pages 408–419, 1984.