

# EVENT MANAGEMENT WEBSITE

GET2GETHER

SPECIALIZE FOR SUNWAY'S STUDENT

BY KAREN, YI SHIEN, YUKKI,  
DANIEL, AISYAH





# PROBLEM STATEMENT

The current approach of relying on scattered information across social media platforms like Instagram makes it troublesome for students to explore, join and manage their participants in university events. The fragmented system leads to reduced awareness, limited exposure for clubs, and a lower motivation for students to actively participate in campus activities.

Problem 1: Fragmented Information Landscape

Problem 2: Limited User Motivation and Engagement



# PROPOSED SOLUTION

An event management website that promotes Sunway Education Group Club and Society activities and encourages freshmen and seniors to participate and make their trip memorable. This website will offer features for Event Holders and Participants. Event Holders can broadcast their event and collect participant data, while Event Participants can explore future events and register for those they like.

# FUNCTIONAL REQUIREMENT (ROLES: USER)

- ☐ CREATE AN ACCOUNT
- ☐ LOGIN
- ☐ CREATE EVENTS
- ☐ VIEW CREATED EVENTS
- ☐ MANAGE CREATED EVENTS
- ☐ CANCEL CREATED EVENTS

- ☐ SEARCH EVENT AVAILABILITY
- ☐ VIEW AVAILABLE EVENTS
- ☐ REGISTER EVENT
- ☐ MANAGE USER PROFILE
- ☐ VIEW CLUB DETAILS
- ☐ SUBSCRIBE INTERESTED CLUB
- ☐ RECEIVE NOTIFICATION





GET2GETHER

# QUALITY ATTRIBUTE

ID	Name	Description	Roles
01	Performance	Users using the system to register event during normal operation and the system should process the user's registration with a latency average of three seconds.	User
02	Security	An unauthorized user with an invalid email address tries to login the website during operation. The system should not allow user from accessing the website within 2 minutes of detection.	User
03	Usability	A regular user with no technical background visits the event management website during runtime and is able to use each function easily within 2 minutes of usage.	User
04		An organizer would like to make changes to the event details during normal operation. The organizer can make changes to the event within a minute with no issues.	User
05	Modifiability	During the system evolution phase, the developer wishes to add a notification feature to the system. Within a day, the feature is added to the system and tested.	Developer, Architect
06	Testability	The system tester awaits the completion of the error validation code during development and proceeds to perform a test by entering invalid inputs. The results capture those errors with 100% rate of detection within 10 seconds.	Tester


# SYSTEM ARCHITECTURE

## MODEL-VIEW CONTROLLER

MVC was chosen for a student event management system because it met project requirements. This design facilitates modular development and maintainability for dynamic web applications and ensures smooth integration of future changes. The MVC architecture's clear organization and functions make it ideal for a robust and scalable student event management systems.

## CLIENT-SERVER



Students and event organizers who use the website are clients. They use computers or cellphones as clients to connect to the server hosting the web application, which hosts the event booking system's website, databases, and application logic. Event organizers and attendees use this server for system services.



# SYSTEM ARCHITECTURE

## PUBLISH-SUBSCRIBE

A publish-subscribe design is beneficial for event management websites because it loosely couples components. Without subscribers, event organisers can publish events and let different components display them. Add new features without changing the publisher for scalability and fast feature development. Analytical and rate-limiting issues can be effortlessly incorporated, simplifying dependencies and communication flows and improving the platform's flexibility and efficiency.



# CHOICE OF TECHNOLOGY

VISUAL STUDIO CODE (VSCODE)

HTML5 AND CSS3

XAMPPS

PHP

GITHUB & GIT

PHPMYADMIN

FIGMA






# SE KNOWLEDGE - REQUIREMENT ENGINEERING (DIFFERENTIATING BETWEEN FUNCTIONAL REQUIREMENTS AND QUALITY ATTRIBUTES)

The team differentiated between two main types of requirements: functional requirements, which describe the specific functions and services the website should perform, and quality attributes, which set benchmarks for aspects like usability, performance, and reliability.


By clearly separating these two categories of requirements, the team gained a deeper understanding of the website's workflow and identified any gaps that needed improvement. This strategy also helped determine which features to exclude to prevent functional overlap.



# SE KNOWLEDGE - REQUIREMENT ENGINEERING (USING REQUIREMENT ARTIFACTS TO ALIGN STAKEHOLDERS)

Identifying requirements artifacts like sequence diagram and use case diagrams was important for finalizing the system's requirements. Setting overall goals ensured the website's functionality aligned with those aims, while use case diagrams provided visual clarity into how the system would meet user needs.


These artifacts enabled effective communication between stakeholders and developers, ensuring user requirements were integrated into the design. Negotiating requirements through ongoing stakeholder meetings also facilitated alignment and consensus among team members with diverse perspectives.



# SE KNOWLEDGE - SOFTWARE TESTING (RISK-BASED PRIORITIZATION)

Critical functionalities and workflows were prioritized for testing based on their risk and impact to users and clubs & societies. This allowed the team to focus testing efforts on areas that posed the biggest risk.


Techniques like decision tables and state transition diagrams were used to model and test high risk login workflows and registration system states. Finding issues early through these models reduced redundant work.



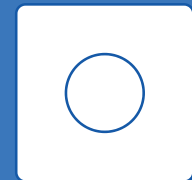
# SE KNOWLEDGE - SOFTWARE TESTING (ITERATIVE, REQUIREMENTS-BASED TESTING)

Testing was conducted iteratively throughout development lifecycles using test-driven development. Each iteration concluded with a review to improve subsequent testing.

Testing tied directly to requirements specifications, ensuring test coverage aligned to expected functionality. Techniques like component testing, regression testing and user acceptance testing focused on validating software against requirements.

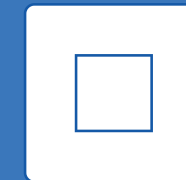


# SE KNOWLEDGE - SOFTWARE ARCHITECTURE (MODEL-VIEW-CONTROLLER)




## SEPARATION OF CONCERNS

MVC neatly separates the data layer (Model), presentation layer (View), and application layer (Controller). This segmentation of duties allows for modular development, easier maintenance, and ability to swap out components.



## ADAPTABILITY

The loose coupling between the components allows them to be modified and replaced without affecting the rest of the system. This makes it easy to improve and scale the system by working on one component at a time.



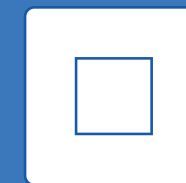


# SE KNOWLEDGE - SOFTWARE ARCHITECTURE (CLIENT-SERVER)



## CENTRALIZED DATA STORAGE

The server acts as a centralized repository for data and application logic. This ensures consistent data access and sharing between clients.



## DEFINED ROLES

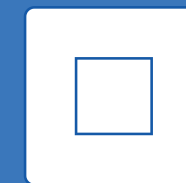
The architecture defines distinct roles for the client (event organizers and participants) and server. This role separation streamlines usage for different users.

# SE KNOWLEDGE - SOFTWARE ARCHITECTURE (PUBLISH-SUBSCRIBE)



## LOOSE COUPLING

Publishers of events are unaware of subscribers. This allows easy addition of new event sources, displays, apps without affecting rest of flow.



## SCALABILITY

Additional subscriber components can be added to introduce new features without making adjustments to publishers or existing subscribers. This model is inherently scalable.



GET2GETHER

THANK YOU