EVENT MANAGEMENT SYSTEM ‘EventLog’

Design Document

[**1 Introduction**](#_2l3ya6d18oup) **3**

[1.1 Purpose of this document](#_y879z3st7rkj) 3

[1.2 Identification](#_268rx0yhvd4r) 3

[1.3 Document Scope](#_e9s715omqcub) 3

[1.4 Intended Audience](#_yyeqqizhgjy8) 3

[1.5 Key Stakeholders](#_x0l73cui0biq) 4

[1.6 Languages and Tools](#_x3d65jqex8zm) 4

[1.7 Reference](#_25nr12bhkngo) 4

[**2 General Overview And Design Consideration**](#_6ox1lha4w2p3) **5**

[2.1 General Overview](#_mz8xhegwwz7h) 5

[2.2 Current](#_tt625koziamq) 5

[2.2.1 Proposed Solution - Statement of Need](#_j9jgqw4eeju4) 5

[2.3 System Assumptions](#_tt625koziamq) 6

[2.4 System Constraints](#_tt625koziamq) 6

[2.5 System Dependencies](#_tt625koziamq) 6

[**3 Design Considerations**](#_vujegiaz1c3c) **7**

[3.1 Goals and Guidelines](#_57yfpviwl4a6) 7

[3.2 Operational Environment](#_57yfpviwl4a6) 7

[3.3 Development Methods & Contingencies](#_lrs54q90ljdh) 8

[3.4 Architectural Strategies](#_57yfpviwl4a6) 9

[3.4.1 Infrastructure On Demand](#_gtkt7lxn7qgy) 9

[3.4.2 Cloud Computing](#_v9uirj1oqwm) 9

[3.4.3 Disaster Recovery](#_6pylwa1xgua) 10

[3.4.4 Mobility](#_yj07pco1wisj) 10

[3.4.5 Scalability](#_pi6i66b02uxp) 10

[3.4.6 Platform](#_k6nd4ywxq7oz) 10

[3.4.7 Development Environment](#_5x6tnjylb696) 10

[3.4.8 Open API Centric](#_o5ruwkcu1lof) 11

[3.5 Performance Engineering](#_57yfpviwl4a6) 11

[**4 System Architecture and Architecture Design**](#_fcjoyp8tuwmz) **12**

[4.1 Topology Diagram](#_v71n4wgu4z8i) 12

[4.2 System Architecture Diagrams](#_zg1kuqh4rxpw) 12

[4.2.1 External Systems Diagram](#_bblg7p2viebl) 13

[4.2.2 Website Architecture Diagram](#_rzqnigt5nlj3) 13

[4.3 Software Architecture](#_juwoffd002p0) 14

[4.3.1 Software Architecture Diagram](#_ej40e7ifaa7d) 14

[4.3.2 Software Element](#_hkkkokwtmwjr) 15

[**5 System Design**](#_rzm8nukcbcbp) **16**

[5.1 Business Requirements](#_kg573j8gngx7) 16

[5.2 Database Design](#_634ve7fglrdm) 17

[5.3 User Interface Design](#_9u3m5ekx7fgw) 18

[5.3.1 Organizer Interface](#_yqafy1itv1sp) 18

[5.3.2 User Interface](#_9ukue9mqibss) 19

[**6. Risk Analysis**](#_b3w6z8e079mi) **22**

[6.1 Feasibility Study Activity](#_47h78tmvujiv) 22

[6.2 Requirements Elicitation Activity](#_5ao31t5bgue5) 22

[6.3 Requirements Analysis Activity](#_an1ylxyvh49) 22

[6.4 Requirements Validation Activity](#_gyrs0escx3v8) 23

[6.5 Requirements Documentation Activity](#_z4y7esk9m0le) 23

[6.6 Examining the Requirements Document (RD) Activity](#_r1t7x3floe0o) 23

[6.7 Choosing the Architectural Design Method Activity](#_b3razhdmlvq) 23

[6.8 Choosing the Programming Language Activity](#_emy0s53q7t8v) 23

[6.9 Constructing the Physical Model Activity](#_abggdldufmqx) 24

[6.10 Verifying Design Activity](#_r5gl769w4aom) 24

[6.11 Specifying Design Activity](#_a7mdyj3ko5ri) 24

[6.12 Documenting Design Activity](#_4ib1tqw66se6) 25

[6.13 Coding Activity](#_piblapz36077) 25

[6.14 Unit Testing Activity](#_1isclle8bq1y) 26

# 

# 1 Introduction

## 1.1 Purpose of this document

* The purpose of this Software Design Document is to provide a description of the design of a system fully enough to allow for software development to proceed with an understanding of what is to be built and how it is expected to be built.

## 1.2 Identification

The software would be using Amazon Web Services which is an on-demand cloud computing platform and APIs to individuals. For the database design SQLite will be used which would also help in creating the required queries.

## 1.3 Document Scope

This design document provides a description of the technical design for EventLog. This document provides an architectural overview of the system to depict different aspects of the system. This document also functions as a foundational reference point for developers. It presents a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system. The primary intended audience of this document are system designers and system builders.

## 1.4 Intended Audience

This project is intended to be used by a particular University that includes faculty, students, staff, organizer of events, and so on. This SRS is intended for several audiences, including the customer, as well as the project manager, designers, developer and tester.

* The customer will use this SRS to verify that the developer team has created a product that is acceptable to the customer.
* The project manager of the developer team will use this SRS to plan milestones and a delivery date, and ensure that the developing team is on track during development of the system.
* The designer will use this SRS as a basis for creating the system’s design. The designer will continually refer back to this SRS to ensure that the system they are designing will fulfill the customer’s needs.
* The developer will use this SRS as a basis for developing the system’s functionality. The developer will link the requirements defined in this SRS to the software they create to ensure that they have created software that will fulfill all of the customer’s documented requirements.
* The tester will use this SRS to derive test plans and test cases for each documented requirement. When portions of the software are complete, the tester will run his tests on that software to ensure that the software fulfills the requirements documented in this SRS. The tester will again run his tests on the entire system when it is complete and ensure that all requirements documented in this SRS have been fulfilled.

## 1.5 Key Stakeholders

* All school (SEAS, SCS, AMSOM, etc) student (Main user)
  + Students would be the users of the application as it would provide an interface to access all the information regarding the club activities.
* Core Committee member of all club (Main user)
  + Club’s would use this application to broadcast their activities around the university.
* Program office head
  + The professional talks and workshops which are organised by the university would be added by the admin offices of the respective school.
* Rahul sir (IT Manager)
  + Maintenance of the software would be the responsibility of the IT manager.

## 1.6 Languages and Tools

* Language
  + Node Js
  + React Js
  + SQLite
* Tools
  + VisualParadigm
  + Creately
  + CSS
  + DB Browser

## 1.7 Reference

* Development Standards and Guidelines2 General Overview and Design Approach

# 

# 2 General Overview And Design Consideration

## 2.1 General Overview

* The current way of communication for all events, workshops at ahmedabad university to the student is email and our way of communication through software called “EventLog”.
* The Event Management System, “EventLog” is a Web application which helps students and organizers belonging to a University to facilitate and manage the events in a more organized way. This web application should be free and accessible from any browser on desktop. This software includes all events occurring in the university and gives permission to the users to view and register through it. An Organizer also uses the web-portal in order to administer the system and manage events. It can also access all the functionalities of a user.
* EventLog uses Google OAuth for login authentication. EventLog is hosted on Ahmedabad University Server.

## 2.2 Current

The Statement of Need explains why the system is being developed, what purpose it serves, and why it is necessary. The reason for this web application is to

provide an alternative for emails. In university life every student's inbox is

flooded with mails, from administration mails to activities taking place around the university. Keeping one up to date to all those activities is too much of a task for the students. In university everyday there are 10's of activities taking place. The issue as conveyed above is all in keeping one up to date for the events.

### 2.2.1 Proposed Solution - Statement of Need

The solution we require must take over emails. Thus the solution we came up with included creating a web application. It would allow the students and the organizers to add and modify events. Through the application the organizers would be given the authority to post about the events. For the students, they would have the rights to view the events as per their choices from the registered events to their top priorities. Thus this would work as an alternative for the flooding emails.

## 2.3 System Assumptions

* The main assumption is that all events and talks permission and the location have already been taken by the clubs from the authorities before adding into the system. Also, all clubs member’s registration is done offline.

## 2.4 System Constraints

* Verification of the event details like location and authorization of the event is the main constraint of the system. Also, the payment gateway is not provided by the system for the event registration.

## 2.5 System Dependencies

* Google OAuth (Login)
* Material-UI (Components)
* React-Router
* React-Router-DOM

# 

# 3 Design Considerations

## 3.1 Goals and Guidelines

3.1.1 Architecture

* The proposed solution to the problem must satisfy all the functional needs stated by the user, and is required to be developed considering all the non-functional requirements. It is supposed to be adaptive to further changes like support to additional features, functionalities and use cases.

3.1.2 Development Environment

* The application development is supposed to stay consistent.Adapting the technological advances this project aims to build the application in the latest versions of technology with backward compatibility for better performances.

3.1.3 Ease of Use

* The features of the application must be user-friendly for strong user experience. Since the application is specific to University events it is supposed to have easy and fast access.

3.1.4 Extensibility

* The application is aimed to be extensible to adding new features, also providing backward compatibility.

3.1.5 API Enabled

* Regional coordination support is a key driver of the project. The application must be API centric and support an open and published API architecture.

3.1.6 RESTful Framework

* The application and underlying architecture must be a REST framework

## 3.2 Operational Environment

* SQLite
* Git version control
* GitHub repository
* NodeJS
* ExpressJS
* ReactJS
* Material-UI

Functional goals of the proposed system includes :

* Developing a web application with advanced features
* Providing with best of application performance
* Sharing and coordinating data with light and compatible data-structure.
* Completing the one click all events motto

## 

## 3.3 Development Methods & Contingencies

* Scalability
  + Ensure that the architecture can be scaled horizontally, across multiple servers and across multiple regions. That means that once your traffic goes up, you should be able to add and remove new servers as the solution requires.
* Availability
  + The architecture should support a high availability environment. Infrastructure redundancy is required. This ensures the solution is available if multiple servers or an entire data center fail. The current availability of the solution per the hosting providers service level agreement is 99.999% availability.
* Security
  + Solution architecture should expose only the minimal amount of code possible. Most of the back-end pieces should be hidden away. In addition to that, security of each system should be multi-layered.
* Extensibility
  + Architecture must be able to swap out modules, change layers, and add pieces to the application without having to worry about the underlying data contracts in place
* Separation of responsibility
  + System should be modular enough that each piece of code has a set of responsibilities and not more. The back-end should not create front end code nor should the front-end code include business logic
* Restful Framework
  + The reason for a Restful API is plain and simple flexibility. Framework does not want to be tied or dependent on a specific programming language and architecture. Architecture needs to be able to replace each layer independently and even use different languages that might be better suited for a certain layer
* Functionality
  + The software is capable to provide functions which meet stated and implied needs when the software is used under specified conditions (what the software does to fulfil needs)
* Reliability
  + The software is capable to maintain its level of performance under stated conditions for a stated period of time
* Usability
  + The software is capable to be understood, learned, used and attractive to the user, when used under specified conditions (the effort needed for use)
* Efficient
  + The software is capable to provide appropriate performance, relative to the amount of resources used, under stated conditions
* Maintainability
  + The software is capable of being modified. Modifications may include corrections, improvements or adaptations of the software to changes in the environment and in the requirements and functional specifications (the effort needed to be modified)
* Portability
  + The software is capable of being transferred from one environment to another. The environment may include organizational, hardware or software environment

## 3.4 Architectural Strategies

### 3.4.1 Infrastructure On Demand

* In a non-cloud environment:

(i) infrastructure assets require manually configured

(ii) capacity requires manual tracking

(iii) capacity predictions are based on the guess of a theoretical maximum peak

(iv) deployment can take weeks.

* Within the cloud, these building blocks that represent the Infrastructure are not only provisioned as required, following actual demand and allowing pay-as-you-go, but can also be programmed and addressed by code.
* Infrastructure can be automated through code, allowing for greater self-service and more automated delivery of desired business and technical outcomes.

### 3.4.2 Cloud Computing

* Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.
* Cloud computing has become the primary engine driving IT as a service. With cloud computing, you don’t need to make large upfront investments in hardware and spend a lot of time managing that hardware. Instead, you can provision exactly the right type and size of computing resources you need to power your newest bright idea or operate your IT department.
* AWS offers global infrastructure available to Customers on a pay-as-you-go model, allowing for more flexibility in meeting requirements for Data Protection and Disaster Recovery.

### 3.4.4 Mobility

* When talking about mobility it has changed the angel of viewing any software. When the software supports mobility one can access all kinds of information from multiple devices in real time. The accuracy is through the roof. Furthermore combining cloud with mobility allows productivity from any location.

### 3.4.5 Scalability

* Ensure that the architecture can be scaled horizontally, across multiple servers and across multiple regions. That means that once your traffic goes up, you should be able to add and remove new servers as the solution requires.

### 3.4.6 Platform

* Google Developer Console:Google Developers Console is a site that is used by the developers for managing and viewing traffic data, authentication and billing information for Google APIs.

### 3.4.7 Development Environment

|  |  |
| --- | --- |
| **Software** | **Description** |
| SQLite | Database |
| Github | Version control repository |
| NodeJS | Programing Language at server |
| ReactJS | Programming Framework for Web |
| ExpressJS | Backend Framework |
| Material-UI | UI Framework / Theme |
| Amazon Web Services | For hosting website |

### 3.4.8 Open API Centric

* APIs allow for the creation of a minimum interface which is relatively stable that can be used by other software systems to access or manipulate the underlying systems or data. This allows for enhancements to the underlying systems or data without disturbing the software systems that use the API Usually implemented using here JSON. Third party application and database integration is simplified as long as all others support the published API.

## 3.5 Performance Engineering

* AWS provides multiple options to configure and procure related services to eliminate potential performance issues.

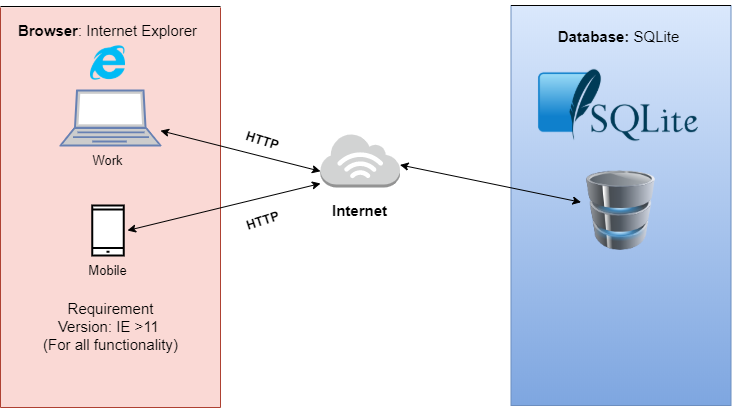
# 

# 4 System Architecture and Architecture Design

* This section outlines the system architecture design of the system.

## 4.1 Topology Diagram:

A **network topology diagram** shows how the elements of a computer **network** are arranged. It allows you to visualize how different nodes are connected and how they communicate.



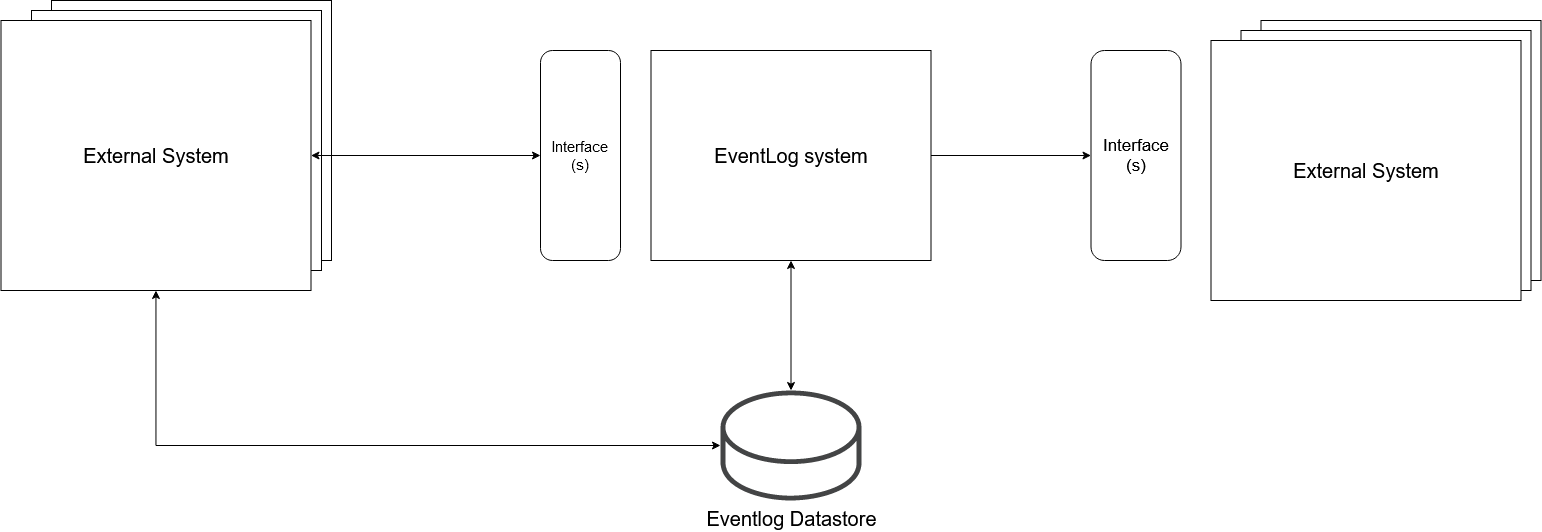
## 4.2 System Architecture Diagrams

* This section provides the conceptual view of the system and its functionality.
* Simply Get There currently provides the following major components.
  + View events
  + Register event
  + Add event
  + Modify event
  + Delete event

### 

### 4.2.1 External Systems Diagram:

The **objective** of the **system** context **diagram** is to focus attention on **external** factors and events that should be considered in developing a complete set of **systems** requirements and constraints. **System** context **diagrams** are **used** early in a project to get agreement on the scope under investigation.



### 

### 4.2.2 Website Architecture Diagram:

**Website** information **architecture uses** a hierarchy structure to visualize a **website's** overall framework, from which you can see the directory structure of the **web** pages and organization of **website** content.

### 

## 

## 4.3 Software Architecture

### 4.3.1 Software Architecture Diagram:

**Architecture diagrams** can help system designers and developers visualize the high-level, overall structure of their system or **application** for the **purpose** of ensuring the system meets their users' needs. You can also use **architecture diagrams** to describe patterns that are **used** throughout the design.

### 

### 

### 4.3.2 Software Element

|  |  |  |
| --- | --- | --- |
|  | SYSTEM | NOTE |
| Programming Language | JavaScript | JavaScript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities. |
| Frontend Framework | Node/React JS | ReactJS is a JavaScript library used for building reusable UI components. React can also render on the server using Node, and it can power native apps using React Native. |
| Backend Framework | Express JS | Express. js is a Javascript framework based on Node. js, that supports development both on the server side and the user side. Express is a very fast, essential, assertive, and moderate web framework of Node. |
| DATABASE | SQLite | SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. It is a database, which is zero-configured, which means like other databases you do not need to configure it in your system. |

# 

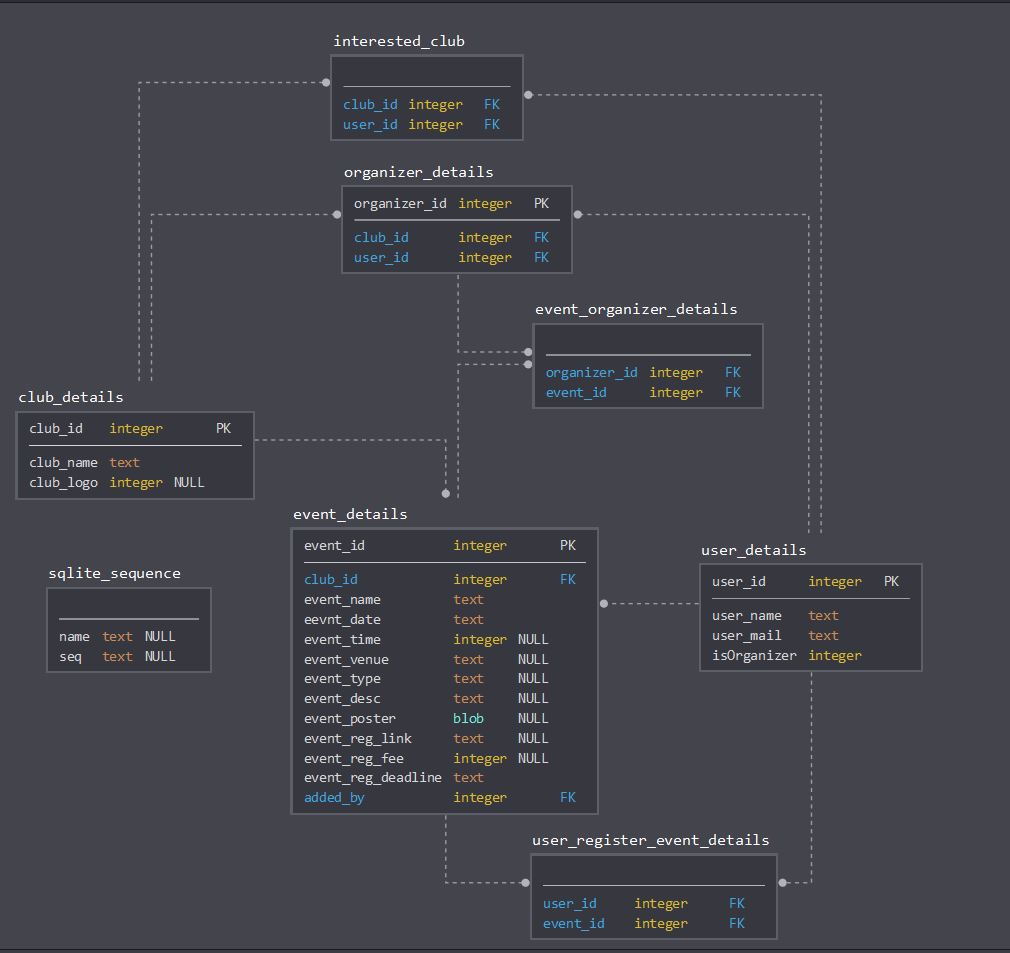
# 5 System Design

The proposed system tries to provide a framework to register, organize any event with easiest interfaces for the user.Proposed design may be developed to incorporate modular components that plugs into the application.Technical support and maintenance will be required for the transactional and mission critical components of the system. The system will try to provide strong security and credentialing methods to ensure privacy and system security.

## 5.1 Business Requirements

* For the proposed system the main requirements would be
  + Providing the main list of all the events in a day
  + List of activities to be done by each club individually
  + List of events for which the user has registered
    - Allow the user to see the number of seats which has already been allocated
    - Registration happens through the link provided in the activity description
  + Access to every event taking place in the university

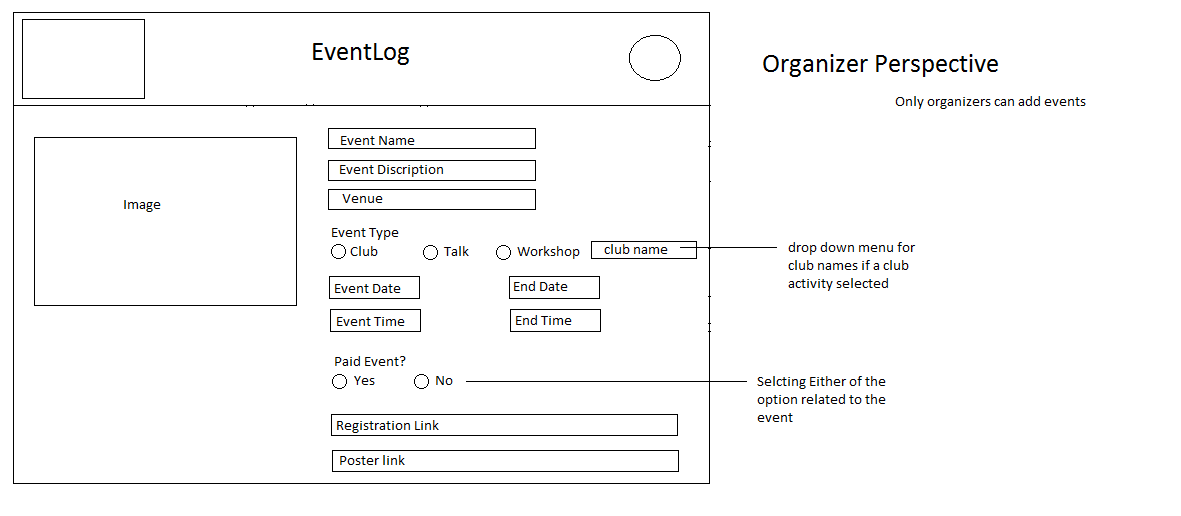
## 5.2 Database Design



## 

## 5.3 User Interface Design

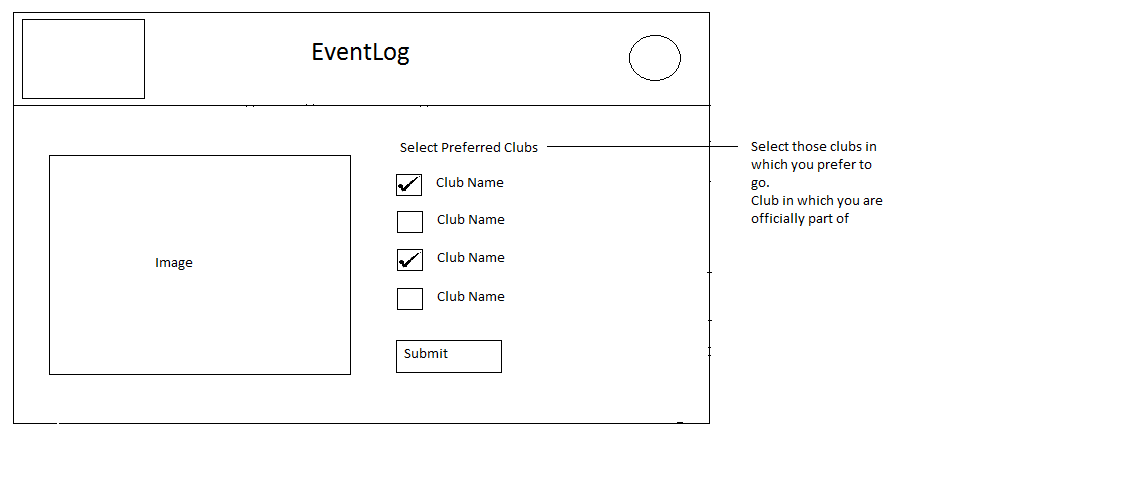
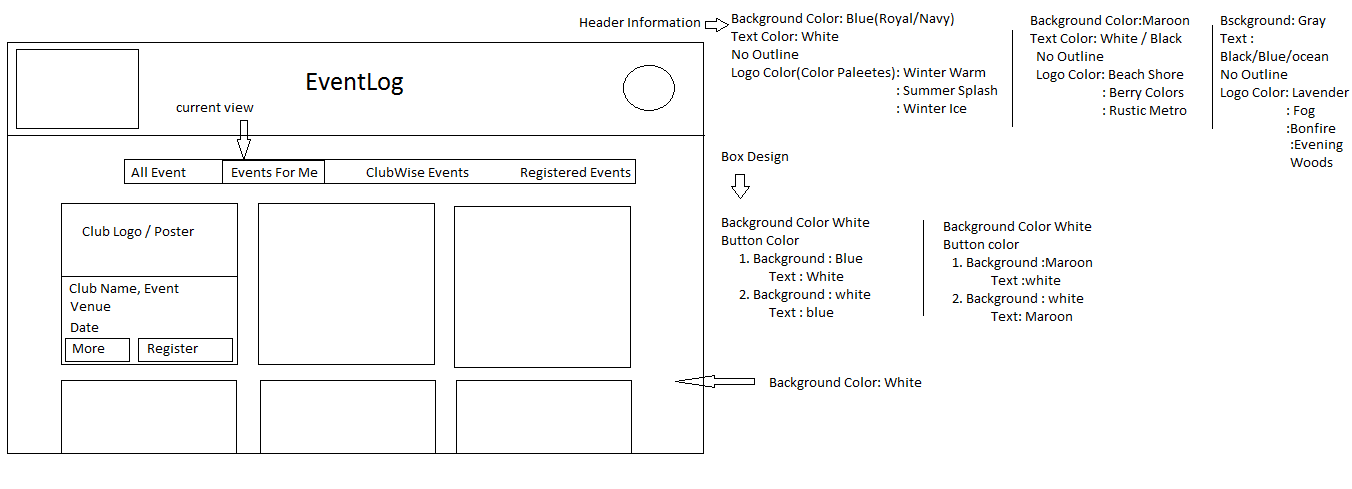
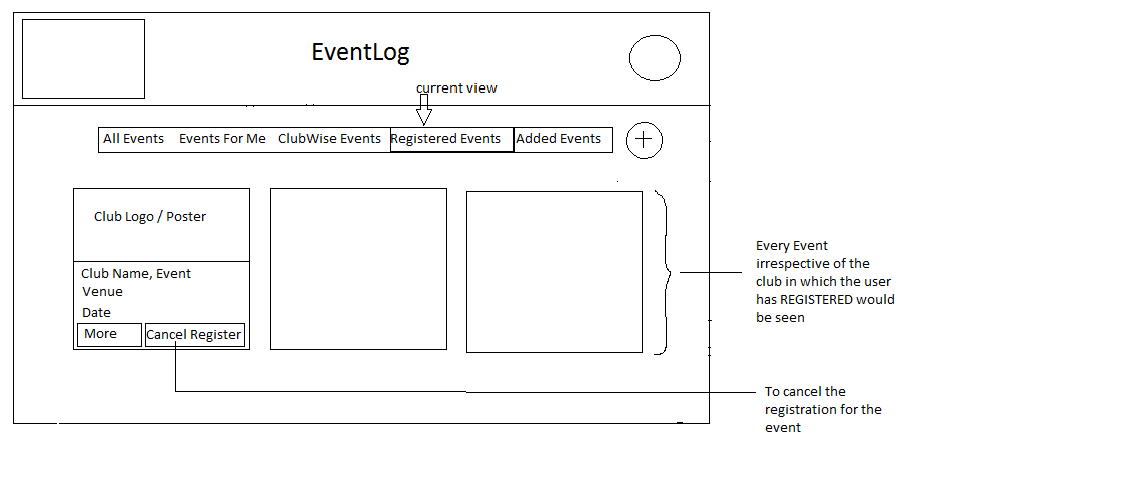
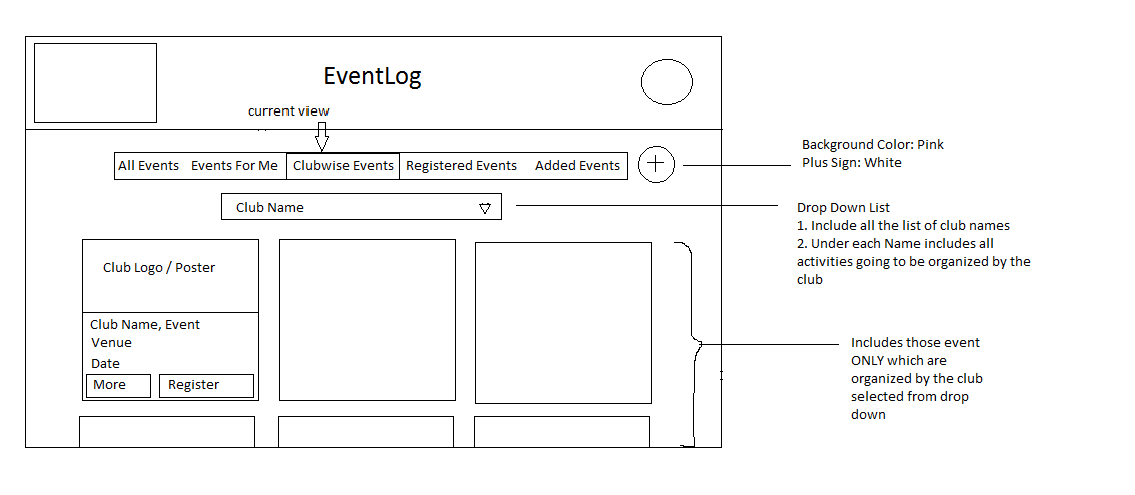
### 5.3.1 Organizer Interface

* Adding an event
  + 
  + Whenever any organizer wants to add any event in the list of events, he must first clarify all the permissions required from the university authorities, from getting a go on the event till allocating a venue.
  + If all the permissions are cleared, then an organizer can add the event in the list.
  + For adding an event on clicking the button provided for adding an event, a form would be presented which would include the parameters such as the name of the event, the location, time. Meaning all the required parameters of any event.
  + The organizer will have to provide the form for registration link too in the event log.
  + Also a parameter for adding a poster for the event would be provided to the organizer.

* Modifying an event
  + Whenever there are any changes in the added parameters value for any event the organizer can modify those details.
  + The reason for modifying may change in venue or time, or any-other reason.
* Deleting an event
  + Whenever due to any reason the event needs to be deleted the organizer has the authority to delete the events which were added by them.
  + The authority can delete those events and ONLY those which were added by them.

### 5.3.2 User Interface

Whenever the user logins onto the web page there are a lot of options provided to him for the viewing of events.

* Selecting the clubs
  + 
  + The user when first logins have to select various clubs which they are interested in.
  + These details are stored under the user details individually. Every user would have this list of clubs selected.
  + Those selections will affect one of the viewing patterns.
  + Those selected clubs would be the clubs in which the user is officially registered.
* Viewing all the events
  + 
  + Irrespective of the clubs all the events taking place around the university are placed in this list.
  + It would just be a list which would include events taking place today and every other event planned in the future days.
* Viewing registered events
  + 
  + The user would always register for the events from the club activity, talks or any other event in university.
  + Those registered events would be visible in this list.
  + This list is important as users can easily access the registered events, and keep themselves informed of any changes.
  + Every user in the database would have a list of registered events under their respective id.
* Club-wise event list
  + 
  + Every club has events taking place every week.
  + In this list the user can view the events under every club name.
  + The importance of this list is that , if the user just wants to see the events of a particular club then they can just view them from this list.
  + The main purpose is that the user can see all the upcoming events of the particular club.
  + For e.g.
    - Photography club has planned 2 events this week, and 2 events next week
    - In the list, under the club Photography club all 4 events would be visible.

# 

# 6. Risk Analysis

|  |  |
| --- | --- |
| 6.1 Feasibility Study Activity | |
| 1. Unclear Project Scope | To manage project scope (i.e. size, goals and requirements) is the most important task for the successful project manager. Project managers, usually, find it difficult to determine what the project is supposed to do exactly, this may cause many core functionalities to be missed and other extra ones to be taken into consideration, in both cases the project failure is an expected outcome. |
|  | |
| 6.2 Requirements Elicitation Activity | |
| 1. Unclear Requirements | The requirements are unclear if they are not understandable  by analysts and developers |
| 2. Inaccurate Requirements | The requirement is inaccurate if it does not reflect the real  user needs |
| 3. Incomplete Requirements | The requirements are incomplete if they are missing some  of the user needs, constraints and other requirements |
| 4. Ignoring the Nonfunctional requirements | Usually analysts and developers focus on what the system should do and ignore how the system should be (i.e. usability, maintainability, scalability, testability, etc.). Non-functional requirements are essential to project success as much as the functional requirements |
| 5. Conflicting user requirements | When the system has different users, their needs from the system may be not only different but also conflicting  Thus, this will lead to inconsistency when analyzing requirements. |
|  | |
| 6.3 Requirements Analysis Activity | |
| 1.Non-verifiable Requirements | The requirement is non-verifiable if there is not a finite cost  effective process (i.e. testing, inspection, demonstration or analysis) with which we can check that the software meets the requirements |
| 2.Inconsistent Requirements | The requirement is inconsistent if it contradicts any other  requirement in the project |
| 3. Unrealistic Requirements | If the requirements are clear, verifiable, accurate, consistent,  complete and feasible then they are realistic to be put in the  requirements document and then implemented |
|  | |
| 6.4 Requirements Validation Activity | |
| 1. Mis-expressing user requirements in natural  language | Natural language is a good, but not the best, way for  expressing requirements, since many users may use  different NLs and different conventions. Besides; many  expressions, terms and needs cannot be expressed this way,  and need a more formal way for expressing and  documenting |
|  | |
| 6.5 Requirements Documentation Activity | |
| 1. Non-modifiable Requirement Document | Sometimes, while documenting the requirements,  structuring the document with maintainability in mind is  not considered, what makes it difficult to modify the data  in the requirement document without rewriting it. |
|  | |
| 6.6 Examining the Requirements Document (RD) Activity | |
| 1. RD is not clear for developers | If developers were not involved in the requirements analysis  and definition phase, then the requirements document may  be not understandable by them. Hence, they will be unable to start their design on a solid knowledge of the system  requirements, and thus they may develop a design for a  system other than the intended one |
|  | |
| 6.7 Choosing the Architectural Design Method Activity | |
| 1. Improper AD method choice | There is no standard architectural design method. For any  project, you can choose the most suitable design method  depending on the project’s need. If a wrong choice was  made, then the system implementation will not be completed successfully and problems in the integration may arise later. Even if it was implemented and integrated successfully, the architectural design may not work on the current machine. Furthermore, the choice of the architectural design method may affect the choice of the programming language (Board for Software Standardization and Control, 1995b). If this was not considered, then the developers may choose a language  that does not support the architectural design method in  use |
|  | |
| 6.8 Choosing the Programming Language Activity | |
| 1. Improper choice of the PL | The improper choice of the programming language can  affect the development process in many different aspects.  The wrong choice of programming language may not  support the applied architectural design method . It may  reduce the system’s maintainability and portability too. |
|  | |
| 6.9 Constructing the Physical Model Activity | |
| 1. Too much complex system | If the software system to be developed was too much large  and complex, then the developers well get lost and  confused and do not know from where to start and how to  decompose the system into its main components. |
| 2. Complicated Design | If the system was too much complex, and the developers  do not have the enough skills and experience to manage  this complexity, then they will create a complicated not  understandable design which will, while being implemented, suffer from different difficulties. |
| 3. Less reusable components than expected | If an inaccurate estimate about the available reusable  components was made in the analysis phase, then these  components have to be developed from scratch. Thus time  schedule and budget may be under-estimated and the  developers will be surprised that much of the code that was  considered ready and available to reuse has to be re-written  from scratch what will cause project delay and budget  over-run. |
|  | |
| 6.10 Verifying Design Activity | |
| Many feasible solutions | When verifying design, it might be discovered that many  alternatives to the same design problem may exist. Which  one to choose depends on the system itself and its nature |
| Incorrect Design | When verifying the design, it might be found that the  design does not match some, or even all, of the  requirements. Worse, it might be different design another  than the intended one. |
|  | |
| 6.11 Specifying Design Activity | |
| 1. Difficulties in allocating functions to components | If the system was not decomposed correctly and the  components were not defined well, then developers may face difficulties in assigning functions to each component and defining its objectives. Moreover, if the requirements in the requirements documents were not clearly defined, it also may threaten the allocation activity since the components’  functionalities are derived from the functional requirements in  the requirements document (Board for Software  Standardization and Control |
| 2. Large amount of tramp data | When system’s components are organized hierarchal, data  needs to be passed through these components. Sometimes,  this passing data is not used (tramp data); it passes only to be passed to another component to be used there. If this data was not managed carefully, it can reduce readability and lead to confusion. |
|  | |
| 6.12 Documenting Design Activity | |
| 1. Incomplete Design Document | The design document must be detailed enough to allow the  programmers to work independently. If the design  document lacks these important details then the  programmer may not work independently |
| 2. Unclear Design Document | If the components in the design document are not clearly  defined; their inputs, outputs, functions and relationships  were not stated properly. Moreover, if the design document was written in an uncommon natural language,  then the design document is unclear and might be non  readable by developers |
|  | |
| 6.13 Coding Activity | |
| 1. Non-readable Design Document | If the design document was large, unclear then it might be  non-readable nor understandable by programmers, and thus  they will be unable to decide what to code |
| 2. Programmers cannot work independently | If the design document was incomplete, then programmers  will not be able to work independently since they have to  make their own decisions to fill the gaps in the design  document, which may affect the programmers working on  other components. |
| 3.Modules are developed by  different programmers | In large projects, development team usually has more than  one programmer. These programmers may work on  different components, and each may follow his own way of  thinking and coding, this will lead to inconsistent, complex  and ambiguous code. Moreover, if they work on the same  component, then different versions for the same component  will result. |
| 4. Developing the wrong user  interface | Designing a good user interface is a very important aspect;  it helps make the system more understandable and usable,  which results in a greater user acceptance. Otherwise, the  project could fail. Developing the correct user interface  requires a good understanding of user needs and detailed  specification in the design. |
| 5. Modules are developed by different programmers | In large projects, development team usually has more than  one programmer. These programmers may work on  different components, and each may follow his own way of  thinking and coding, this will lead to inconsistent, complex  and ambiguous code. Moreover, if they work on the same  component, then different versions for the same component  will result |
| 6. Large amount of repetitive code | In some projects types, specific pieces of code have to be  rewritten repeatedly. If this was done manually, it will  consume time, effort and budget |
|  | |
| 6.14 Unit Testing Activity | |
| 1. Testing is monotonous, boring and repetitive | if the testing process were not  automated, it will be monotonous and boring and will  continue to fail to produce results. |
| 2. Poor documentation of test cases | Test cases have to be documented automatically while doing  the testing for effective future use for similar cases |
| 3. Data needed by modules  other than the under testing  one | In unit testing, each unit is tested individually. The module  being tested might need data from another module or send it  to another module; this is solved by coding drivers and stubs |
| 4. Poor Regression Testing | Regression test in unit testing aims to rerun all the already  successful run affected test cases when a change is made to  an existing code. Although regression testing  saves time and money, it might do the opposite if most or all  of the original test cases were selected and the time is limited. |