Software Requirements Specification

For

College Festival Management Web Application

Prepared by

[1]Aviral Pandey (21CS10089)

[1]Harsh Sharma (21CS30023)

[1]Ishan Raj (21CS10032)

[1]Prasanna Paithankar (21CS30065)

[1]Utsav Dhanuka (21CS10087)

[1]Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur

Submitted on 3rd March 2024
for CS39202 – Database Management Systems Laboratory

Version no.: 1.0

This page has been intentionally left blank

Table of Contents

1. Introduction

1.1 Purpose

This SRS specifies the software requirements of the College Festival Management Web Application, henceforth referred to as CFM. This is the first release of the software, and the SRS provides the full description of the product.

1.2 Document Convention

The document follows the IEEE Std 830-1998 standards which lays recommended practice for Software Requirements Specifications.

1.3 Intended Audience and Reading Suggestions

The SRS has been targeted towards the development and maintenance community for active suggestions and prototyping scope. The readers are advised to jump to their target of interest after navigating through the Table of Contents. For the readers wanting to have basic description of the software, we recommend them to go through Section 2 and 4.

1.4 Production Scope

The software and subordinates have not been envisioned to be deployed commercially by the authorships. No formal production and release will be carried out. Readers interested in usage are supposed to contact the authors and developers involved.

1.5 References

[1] Dr. Johnn C. Klensin, (2008). Simple Mail Transfer Protocol. RFC Editor, RFC 5321.

[2] IEEE-SA Standards Board. IEEE Recommended Practice for Software Requirements Specification. IEEE Std 830-1998.

2. Overall Description

2.1 Product Perspective

The SRS specifies the CFM's features, a new, self-contained product. The system hopes to create a smooth platform for the event participants and the event management to execute the event in an organized and secure manner.

2.2 Product Functions

The web-based system for managing a university cultural festival accommodates diverse user roles and functionalities. External participants can seamlessly create accounts, browse and search events and schedules, register for events, and stay updated on event winners in real-time. Additionally, they can access logistics like accommodation and food through the same platform. Students, on the other hand, can browse and search event schedules, register for events, and even volunteer for various activities. Organizers have the capability to create accounts, access event details, designate winners, set Food menu and send mails and Notifications to the Participants and Volunteers. The Administrator holds the authority to add or remove users, events, and assign roles to users as deemed necessary.

2.3 User Classes and Characteristics

We define five broad user classes in our software implementation

a. Students

The website primarily serves students who interact with the system solely through its user interface (UI). Within the system, students may assume various roles, including user, organizer, admin, or volunteer, each entailing specific privileges and responsibilities tailored to their involvement in the university cultural festival. The UI is meticulously designed to be intuitive and user-friendly, ensuring that students, regardless of their prior experience, can navigate the system effortlessly. While students can access various features and functionalities, they are not authenticated to the system's backend, which stores sensitive information, including Administrator credentials. This ensures that only authorized personnel can access critical data, safeguarding the system's confidentiality and integrity.

b. External Participants

External Participants are offered a seamless registration process, where they can provide essential details such as college accommodation preferences, food choices, and other relevant information. With full user functionalities, they can browse and register for events of interest. Additionally, they benefit from timely email notifications from event organizers, ensuring they stay informed about upcoming events and festival updates.

c. Volunteers

The Students have the flexibility to register as Volunteers for events, enabling them to participate in the festival's operations actively. Additionally, they benefit from receiving timely email and notification updates from event organizers, providing essential event schedules and details.

d. Administrator

The Administrator gains access to comprehensive database information, including the complete lists of students, participants, organizers, volunteers, roles, food, notification details, and associated events. The administrator holds the authority to add or remove users, events, and assign roles to users as deemed necessary.

e. Organizers

As organizers, Students can seamlessly manage event logistics, including sending emails and notifications to participants and volunteers regarding upcoming events. Moreover, they possess the capability to designate event winners, set food menus and retrieve Volunteer details, facilitating efficient event coordination and execution.

2.4 Operating Environment

The operating environment for any user is that they should have an internet-connected device and valid credentials to log in. However, users can explore the landing and Events page without logging in. The web application is compatible with all standard web browsers and operating systems.

2.5 Design Implementation and Constraints

The software's efficiency is dependent on the manual work of maintaining the catalogue of movies to be done by the staff. The software is also constrained by the limited working scale, hence requiring smaller databases of movies.

2.6 User Documentation

The software will come with a user manual that will guide users in properly using the website.

2.7 Assumptions and Dependencies

We assume that a student can only organize a single event. Furthermore, the organizer can't participate in an event.

3. External Interface Requirements

3.1 User Interfaces

Our User Interface will be implemented using Flask 3.0.2. Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. Our backend will be implemented in Python 3.10.x, and the front-end and back-end will communicate seamlessly over a local server that Flask helps us to host (deployment using Apache HTTP Server has been detailed later).

The user interface can be divided into multiple major sections:

a. Landing Page

The landing page serves as the initial point of entry for all users visiting the website for the first time. Upon arrival, visitors are presented with various options, including logging in as different stakeholder classes or browsing the landing page anonymously. Here, users can conveniently access browsing controls for the events page, utilize the search bar, view their profile, login, register, and explore the events schedule.

b. Login/Sign-Up Page

Users clicking the login button on the landing page are brought here. This page is implemented using security-enabled forms that Flask provides using its sister extensions. Users will be given the option to sign in as one of the stakeholder classes. User ID/Name/E-mail ID and Passwords will be asked for. Further, the interface will have the option to go to the register page and also the option to reset the password. The forgot password page will just take the input email and first check in the database if the user exists; if it does, the database will reset the password and send it using the SMTP mail interface to the corresponding user mail. The temporary password would be alphanumeric of length 10. Email will be also sent after registration with a welcome message and accommodation information.

c. Profile / Edit-Profile

The "View Profile" option in the navigation bar directs users to a dedicated page their personal information such as name, email, and department. Additionally, users have the convenience of editing their profile directly from this page, allowing them to update their details with ease

d. Events

The Events page enables users to browse through both their registered events and other events they wish to participate in. Users can access more detailed information about an event by clicking on it.

e. Specific Event

Students have the capability to access comprehensive event information such as date, description, and winners. Additionally, they can register for events either as participants or volunteers.

f. Food Menu

The Food Menu page displays the available food options for both vegetarian and non-vegetarian preferences, along with their descriptions, which are provided by the organizers.

g. Events Schedule

This Page shows the Event name and their dates for all the events upcoming or finished and the user can click on them to see further details about them.

h. Admin Panel

This page is exclusively accessible to the administrator upon login. Here, the administrator gains access to comprehensive database information, including the complete lists of students, participants, organizers, volunteers, roles, food, notification details, and associated events. The administrator holds the authority to add or remove users, events, and assign roles to users as deemed necessary. We use the CRUD modelling in the admin page to handle the database activities.

i. Organizer Panel

This page is for the Organizer who can see his event details like winners and date, description. He can send mail and notifications to the participants and volunteers of the event, informing them about the event details. Moreover, he possess the capability to designate event winners, set food menu and retrieve Volunteer details, facilitating efficient event coordination and execution.

j. Volunteer Panel

This page is for displaying messages received from organizers of various events to the corresponding volunteer.

3.2 Hardware Interfaces

The application does not raise any non-standard hardware interrupts and faults. Hence the design philosophy has not incorporated the hardware module design and interface.

3.3 Software Interfaces

Our backend software will run on Python 3.10.x. We have adhered to the object-oriented programming paradigm as much as possible. Further, our backend will communicate with both the Flask server for frontend and PostgreSQL server for databases. Whenever a user logs in, an object of the class is instantiated with required data fields fetched from the SQL databases.

Weightage has been given to clean and readable code with industry-standard linting and code formatting. We make use of Git version control, Flake8, autopep8 and isort to regulate our workflow and standardization.

3.4 Communication Interfaces

The software interacts with users through both text and graphical user interfaces (GUI). Based on the functionality expected from the user, it redirects them to the appropriate web address. The search bar intelligently displays relevant events or volunteers based on the user's query. Upon selecting an event from the list, users are directed to the corresponding event page for registration. New users are seamlessly redirected to the login page after registration. Registration for events is restricted to logged-in users only; without valid credentials, users cannot register. Similarly, unauthorized access to the admin page redirects users to the login page. Organizers are restricted from registering for events, ensuring fair participation. Furthermore, all volunteers associated with an event receive notifications and emails from the organizer, providing them with essential event details and updates. This ensures effective communication and coordination among event participants.

4. System Features

4.1 The Search Engine

Our program would provide a robust interface to search for events if the current user is not participant and events as well as volunteers if the current user is organizer. We execute the search in the titles and the event description and ignore the case. The search engine attempts to index the input as a substring in the given context. We have demonstrated two strategies to search: using sqlalchemy and javascript. The sqlachemy approach is used to carry the events search and the search result is found by the server, while the javascript approach is dynamic and does not involve any form. The js approach is used for volunteer searching.

4.2 SMTP Mailing Interface

When they register or edit their profiles, emails are sent to students and participants. Moreover, the organizer of an event can send emails to the participants and volunteers of his event. The mailing interface is done by using a SMTP server, the configuration is provided by Flask-Mail extension. The receiver SMTP server is smtp.google.com

4.3 Notification Modelling

We have implemented functionality for the organizer to send notifications to the volunteers in his respective events. This enables an efficient communication interface between the organizer and volunteers. A schema for notification has been initialized in order to do so.

4.4 Information & System Security

Special attention has been given to the security and privacy integrity of the web application architecture. Our application handles a vast amount of data like personal data, making it a prime target for malicious actors. We need to implement security features to safeguard from data breaches, unauthorized access, injection attacks and cross-site scripting. This not only protects but also preserves the integrity and reliability of the application itself. In the following subsections, we highlight how we are implementing data security policies in our web framework.

4.4.1 User Authentication

Users are authenticated before every request they make to the server using secure cookies. They are explicitly authenticated at login, register and edit profile pages using HTML forms. These forms are protected using CSRF tokens to prevent corresponding attacks. The passwords are hashed and stored in our database, which provides data security in case of breach.

4.4.2 Session Management and Cookies

We implement user sessions that are set as secure cookies (the ones that can only be sent back over an HTTPS connection to the server. We set a total of 4 cookies in a session instance as follows:

1. user_id: It stores the primary key of that user. This serves personalization of content and also authentication.
2. role: This token effectively stores the user type, which can be user, external, admin, volunteer or organizer. This aids the backend to provide respective services to each user.
3. CSRF_token: This cookie that is a nounce and will be used to protect against CSRF attacks. They are set when the user interacts with a form. The details will be discussed shortly.

We integrate security by setting finite idle session time (1800 seconds) and Flask inherently provides us with SecureCookie implementation. Moreover, we have abided by the standard web browser rules to keep a number of cookies to a minimum. All the cookies are just essential cookies that aid in the normal functioning of the website and no other cookies have been set respecting the privacy of the user.

4.4.3 SQL Injection Attacks

The dreaded SQL Injection attacks have been managed by sanitizing the query strings.

4.4.4 Cross Site Request Forgery (CSRF) Attacks

CSRF attacks are commonly carried out in applications that involve form. A malicious site can send a POST request to our website to make unintended changes when a user session is active. To prevent this, we introduce a CSRF token, which is a nounce that is integrated as a hidden field in every form that can't be replicated by the attacker. This eliminated almost all the CSRF attacks.

4.4.5 Cross Site Scripting (XSS) Attacks

XSS or CSS attacks happen when an attacker attempts to introduce malicious code into our website. We have attempted to sanitize the data that comes from the user side at all steps and also minimize the use of such inputs in server-side functions.

4.4.6 Privilege Setup

When the application is deployed as highlighted in a later section, appropriate permissions are set in the server file system in order to prevent higher privileges in case of breach, mitigating any further exploit.

5. Queries & Triggers

5.1 Insert/Delete Queries

The basic SQL DML commands are used in order to INSERT and DROP rows from any table.

Insert data into *model* :
INSERT INTO *model* VALUES (…);

Delete data from *model* :

DROP FROM *model* WHERE *condition* ;

Deleting records from the tables is only carried from the admin page, while we insert records at multiple endpoints, e.g., user registrations, user participating or volunteering for an event, etc.

5.2 Update Queries

The UPDATE SQL DML is used to update table record instances as follows

UPDATE *model* SET col1 = val1, col2 = val2 WHERE *condition* ;

5.3 Triggers

The need for explicit triggers in Flask framework is unnecessary, as we could handle such functionalities in Flask itself before. Though to demonstrate the usage of triggers we have explicitly defined two triggers as follows:

1. **notification_trigger_listener**
   Trigger Event Check: The function begins by checking if the trigger event is an INSERT. This is determined by examining the connection.dialect.name attribute, specifically looking for the PostgreSQL dialect.

   Setting Time: If the event is an INSERT, it sets the time attribute of the Notification instance (target) to the current date. This ensures that the time field is automatically populated with the current timestamp when a new notification is inserted.

   Additional Actions or Validations: It then performs additional actions or validations related to the sender and receiver of the notification. It queries the Student table to check if both the sender and receiver exist by searching for their respective Roll values. If either the sender or receiver is not found (not sender_student or not receiver_student), it raises an IntegrityError. The IntegrityError is caught in the except block, and a ValueError is raised with the message 'Invalid sender or receiver'. This indicates that the sender or receiver specified in the notification does not exist in the Student table, and the transaction should be rolled back.

Attach Trigger Function: The trigger function is attached to the Notification model using event.listen. This ensures that the function is executed before the insertion operation on the Notification table.

2. **Fooddatabase trigger**

   The provided code sets up a trigger function (modifyfood_listener) to be executed before inserting a new record into the food table. The trigger function is designed to update certain columns based on the values of the new record.

   Additionally, a DDL statement is defined to create a trigger function (modifyfood_trigger) in Python directly. This trigger function is then attached to the 'after_create' event of the food table, ensuring it is executed when the table is created.

   The print statement at the end is more of an informational message and does not provide dynamic information about the trigger event. It might be more useful if placed inside the trigger function or if it provides more context about the trigger event being observed.

6. Self-Hosting on Apache HTTP Server

6.1 Apache HTTP Server

We have self-hosted the web application using Apache HTTP as a web server. We selected Apache as it is a popular, robust and scalable web server sufficing our needs. The server address is http://10.145.95.65/, and it is hosted on the campus' internal network. The *.conf* has been appropriately configured. We attempted to use Certbot to provide SSL certification to transpile the application to HTTPS, but it failed as the port has not been forwarded to the eternal internet.

6.2 Web Server Gateway Interface (WSGI)

To establish communication of the Flask applications with the Apache server, a *.wsgi* file was configured using the mod_wsgi module.

6.3 Linking to https://cse.iitkgp.ac.in/ domain

To link the internal IP to our cse domain, we create a *.htaccess* file in the public_html folder of the sftp server.

7. Other Non-functional Requirements

7.1 Performance Requirements

Our CFM is built up on 3 major modules namely the backend, the frontend and the databases. The frontend is chosen to be hosted as a web application because it allows seamless cross-platform usage. Further the entire premise of a College Festival Management Application depends on it being online and accessible from anywhere and anytime. Python and PostgreSQL are chosen to work with, given their proven performance benefits and seamless compatibility mutually.

7.2 Safety and Security Requirements

Databases are crucial for our application. We implement all the security standards both for our databases by encrypting them and our frontend forms using Flask's built-in form encryption. Further details can be found in the Secure Login System Feature section.

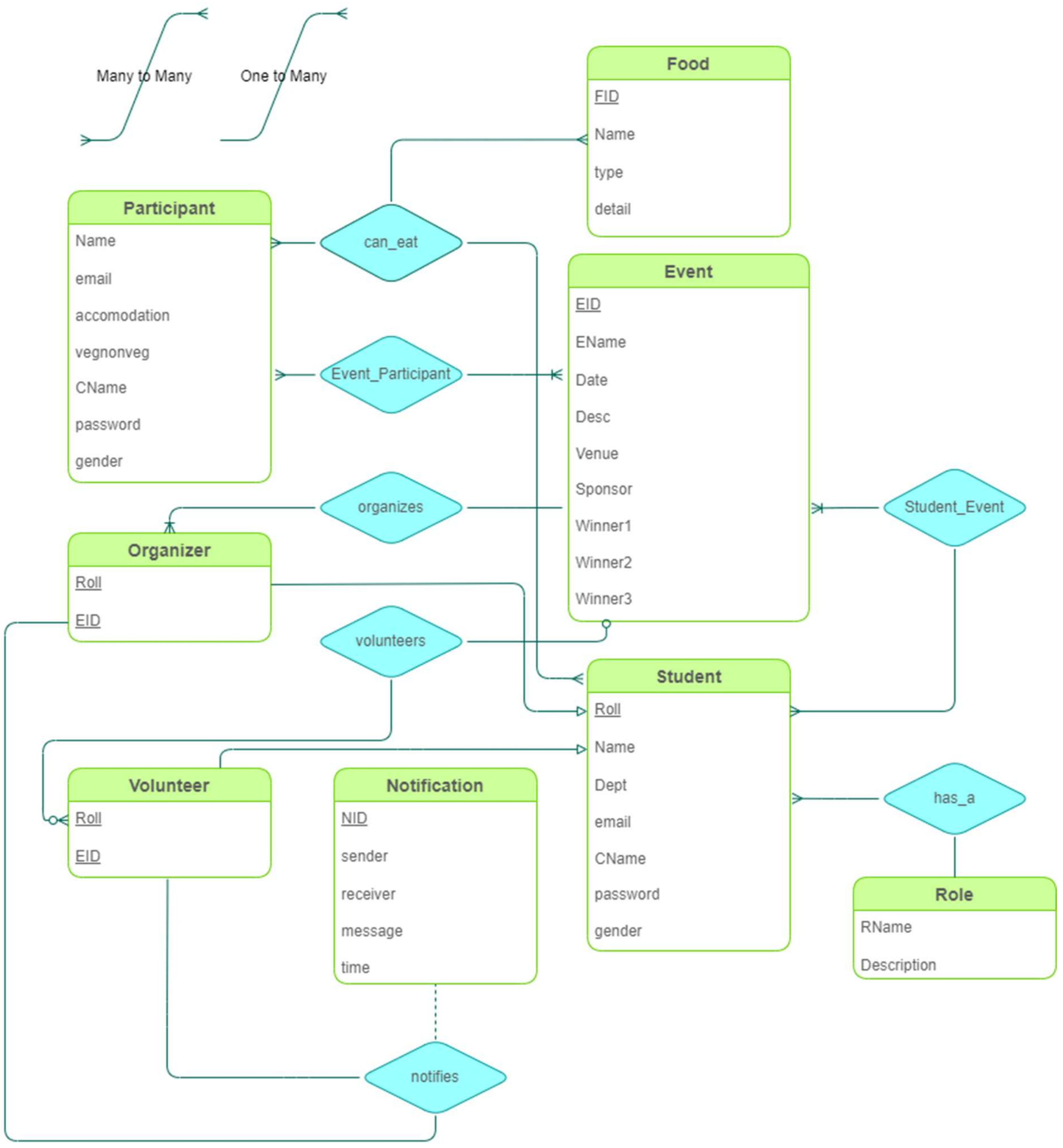
7.3 Software Quality Attributes

Our CFM promises the utmost quality and robustness. Both ease of usage and functional completeness have been ensured at each step. There is a clear distinction between rights of the different stakeholder classes which ensures no misuse/loss of business and data. The separate hosting of the movies and auditing allows for seamless changes in the library. Further, the admin is given rights to delete entries and change rights of other classes as well. All these features combined make our CFM complete on all fronts of quality.

7.4 Business Rules

The Software during development and after deployment is not meant to be monetized. The authors agree to publish the software under the guidelines of open-source Creative Commons after final evaluation.

Appendix A: ER Diagram



Appendix B: Code Maintenance

The code files are hosted in a GitHub repository:
https://github.com/PrasannaPaithankar/DBMS-Laboratory-Spr-24

Appendix C: Glossary

Python	Programming language
Flask	python API framework to seamlessly integrate frontend and backend
GUI	Graphical User Interface
Server	A computing node that provides service to other client nodes

Appendix D: Conflict of Interest

The authors share no conflicts of interest and the authorship list follows the Hardy-Littlewood Rule.