TEAM D5 REVIEW

Architecture and Detailed Design Specification Template

SE-D5 → ConfHub

Revision History

Date	Description	Author	Approver	Comments
<date>	<Version 1>	<Your Name>	<Approver's Name>	<First Revision>

Document Approval

The following Software Requirements Specification has been accepted and approved by the following:

Signature	Printed Name	Title	Date

Contents

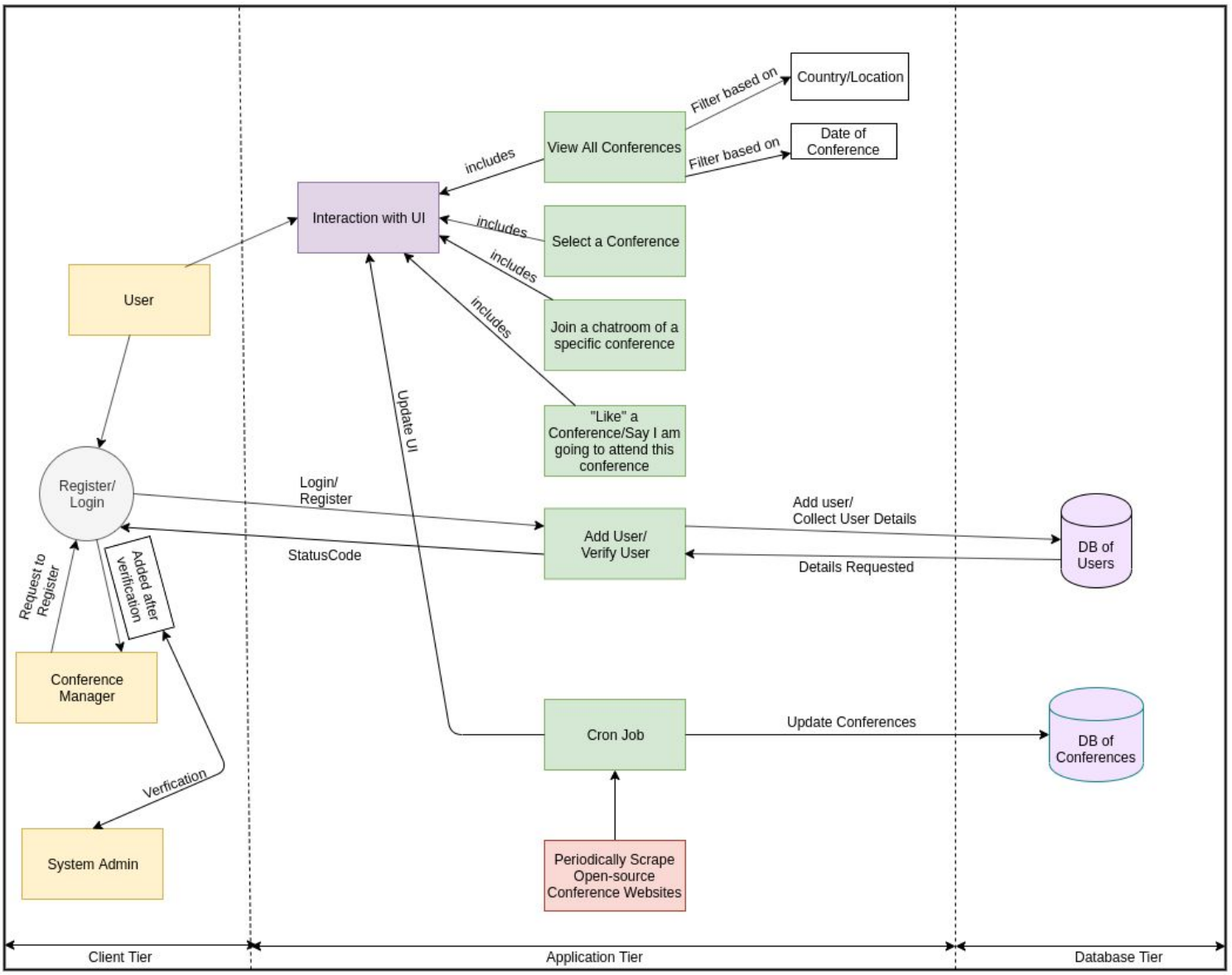
1.0 Architectural and component-level design

We are following the 3-tier web services architecture. The three tiers include the client layer, the application tier and the database tier. The client tier consists of the users, the conference managers and system administrators. Upon elaboration, there is a REST client, WebSocket Client, and the Conference info/analytics interface. The REST client provides a user interface through which the user interacts with the systems. The websocket client is required to maintain the open connection with the system so that the client can receive push notifications. There is even a higher level of abstraction where the conference data and analytics gets displayed on the client side with the REST client acting as an intermediate module. Next, the application tier, which is where the back-end code is written and this is where all the front-end requests are handled. In case, the client requests for a DataBase(DB) access, then the application tier intercepts the requests, contacts the DB, verifies say the username and password and sends back a status code or success or failure to the client. This layer also handles requests like filter the conferences based on location or date of conference. Furthermore, application tier consists of the script that performs the periodic scraping of websites, to keep our whole application up-to-date. This layer is also responsible for processing the script to send out push notifications to the client based on their interests and other recommendations that we come up with. Lastly, the DB layer is where the databases are constituted and all the required information to run our application are stored here.

IT IS CLEARLY WRITTEN. (Furthermore, application tier consists of the script that performs the periodic scraping of websites, to keep the whole application up-to-date. : It is a very good idea)

2.0 System Structure

2.1 Architecture diagram



2.2 Description for Component

2.2.1 Conference data scraping service: This component is responsible for collecting information about the upcoming conferences by scraping a set of relevant websites, and periodically updating the repository of conference information to account for changes or addition of new conferences. The data first needs to be cleaned and normalized into the required format before adding it to the repository of conference data. This service runs periodically (week/month) to collect and update conference related information.

2.2.2 Alert management service: This service runs in the background and accepts new entries from users to let them be notified about the deadline for registration in a conference or the day of arrival of the conference. It also lets users delete the scheduled alerts if the user wishes. When the time for sending a scheduled alert is encountered, the service sends the same reminder to the user.

This service receives the input from a user via the REST web service, and it sends alerts to the users by accessing the API of the websocket service which sends a push notification containing the alert to the user's client app.

2.2.3 REST-based web service: This service is at the centre of the ConfHub app which is the main source of communication between the user's client app and the back-end. This service provides functionalities such as user registration, user authorization and authentication, providing the list of conference details (obtained from the repository created by conference data scraping service), providing user profile data, allowing users to subscribe to a conference and manage alerts, allowing conference organizers to add and update information about the conference, and social features such as allowing users to add a rating or review about a conference.

2.2.4 Websocket service: This component keeps a persistent but lightweight connection to the client app open, in order to allow push notifications initiated by the server. It exposes an internal API used by the alert management service to send alerts to the users.

2.2.5 Conference query processing: This component provides a set of functionalities consumed by the REST web service in order to process user queries on the conference data. It allows filtering and sorting of the conferences based on factors such as location, date or topic of the conference, or filtering the conferences which are in a given radius of a user's current location or from any given arbitrary location. It also provides processed information such as the density of conferences in a given month or at a given place, and other more complex information such as distribution of number of conferences in a time period around the world. The functionalities provided by this component are requested by and returned to the web service, which provides the client app an interface to request for as well as receive such information.

2.2.6 REST client: This component lies on the client side and provides a web interface to the user to interact with the app. It exposes the functionalities of the REST web service to the user, such as signing up, logging in or out, displaying the list of conference details, subscribing to a conference, displaying and providing interface for adding ratings and reviews for the conferences.

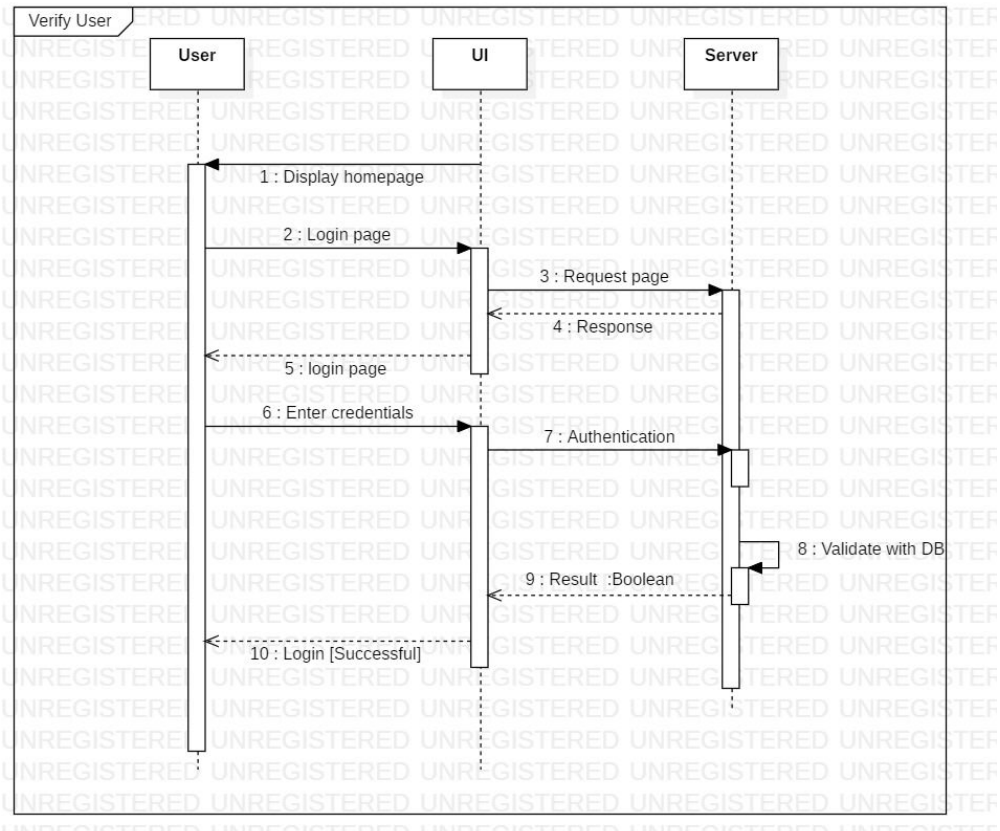
2.2.7 Websocket client: It maintains a long-running persistent and lightweight connection with the websocket service on the back-end in order to receive push notifications from the back-end and show the notifications to the user. On clicking a push notification, the client app is opened for the user and the details of the notification are displayed.

2.2.8 Conference analytics and querying: This component lies on the client side and provides the user with analytical functionalities to sort and filter the list of conferences, find conferences near their physical location or at an arbitrary location, and find processed analytics about conferences. The information is presented in a graphical manner making use of tables, different charts, heatmaps and other analysis and visualization tools. This component obtains these set of information by sending requests to the back-end using the REST client as the intermediary for communication.

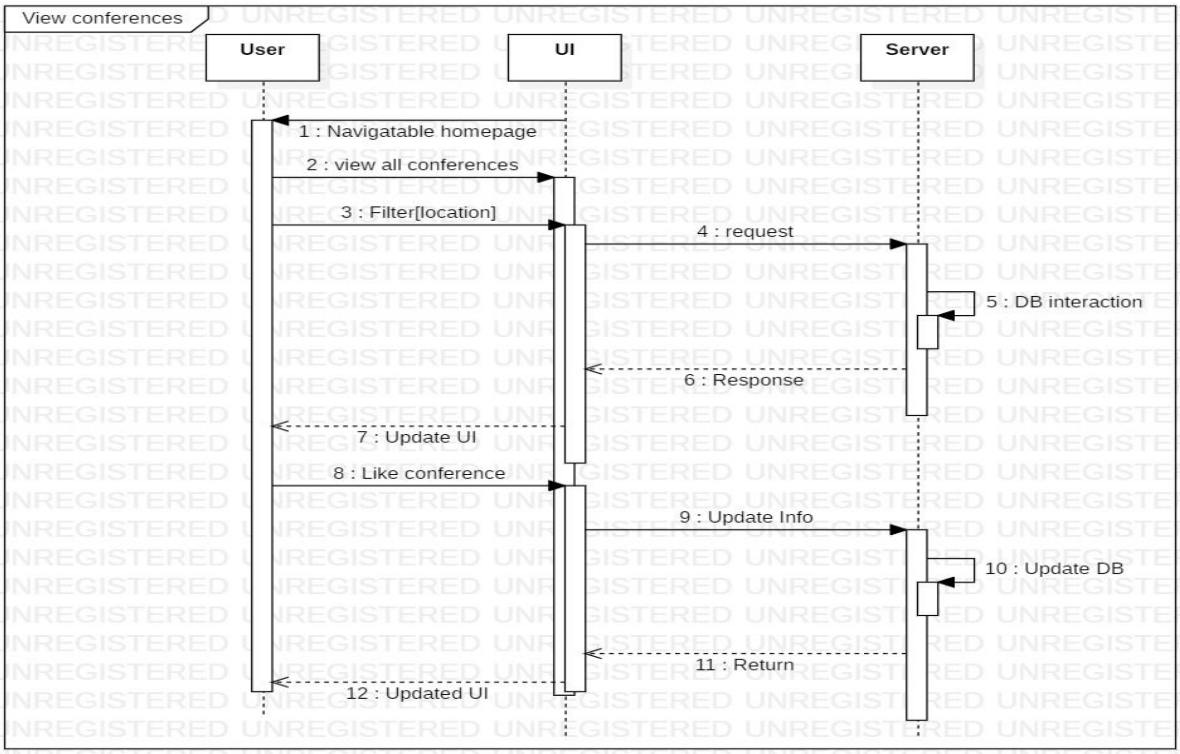
2.2.5 and 2.2.8 almost similar!!
2.2.2 where is it in the architecture diagram? (if it is the CRON job, it should be made explicit)

2.3 Interaction Diagrams

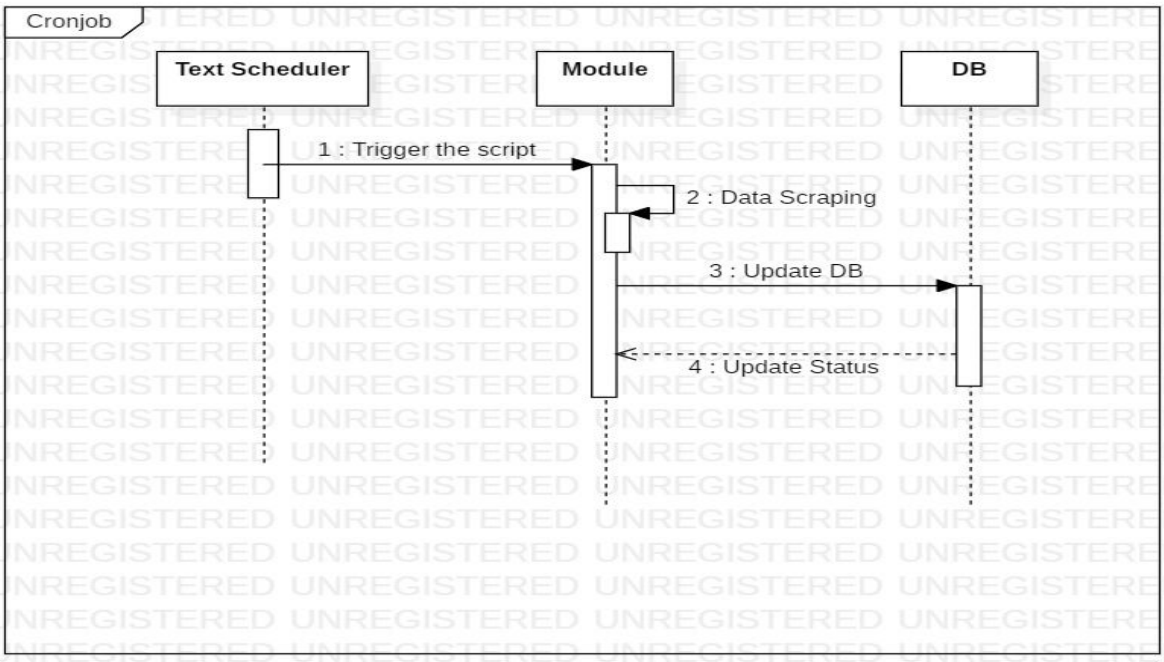
2.3.1 : Verify User



2.3.2 : View conferences and other UI interactions



2.3.3: Cronjob



2.3.1 and 2.3.2 is clear (the application of a cronjob for automation is a good idea)

2.3.3 trigger based on? And isn't the cron job a loop based on seconds,minutes, hours, days, weeks, or months?

2.4 Describe usage scenarios and how you would test that

- Check Login Functionality - different usage scenarios would include
  - Username already exists
  - Invalid Password
  - Registration failed - server with the DB is down
- Check DB security for users - make sure the passwords are encrypted, also store user information for easy login not using "document.cookie" but "HTTPCookies" so that they are protected from easy hacking
- Check User Reminders/Notification Functionality - work with the browsers, local storage and other variables on the client and test each scenario
- Test each REST API communication initially using insomnia/Postman, and later directly make the calls from the front-end to realize and test the overall communication from the front-end to the back-end

UI update functionality could be added and also about search filter function which is based on location, date and subject of the conference

They have talked about conference organizer whereas no usage functionalities about conference organizer has been mentioned.

How to test is WELL WRITTEN in a compact manner

2.5 Architectural Styles and Patterns considered

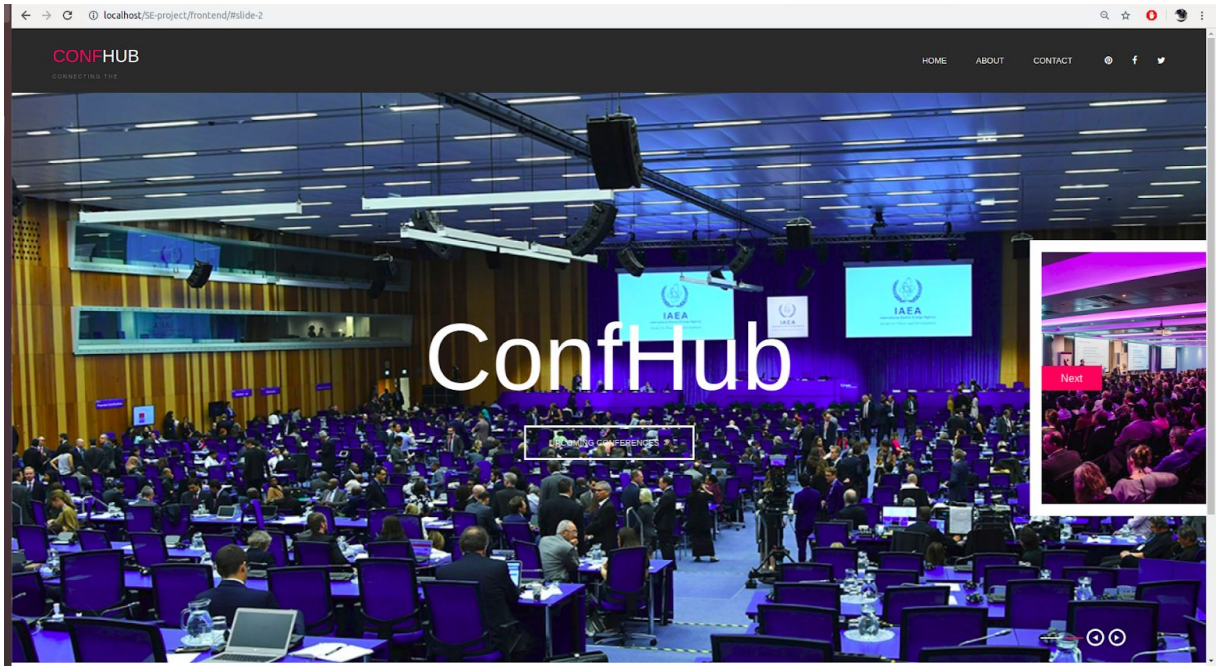
S.No	Architectural Styles/ Pattern	Intent of this pattern	Rationale for choosing or not choosing
1.	Architectural pattern: Model View Controller(3 tier architecture)	Divides the related program logic into three interconnected elements.	Designing visually appealing UI requires different skill set compared to what is needed for developing complex business logic. So it's good to separate the development effort of these two parts.
2.	Architectural Style: Service Oriented Architecture	Separates functions into discrete units or services.	As services are independent of each other they can be updated and modified easily without affecting other services.

Neatly written and given example for choosing the style and pattern.

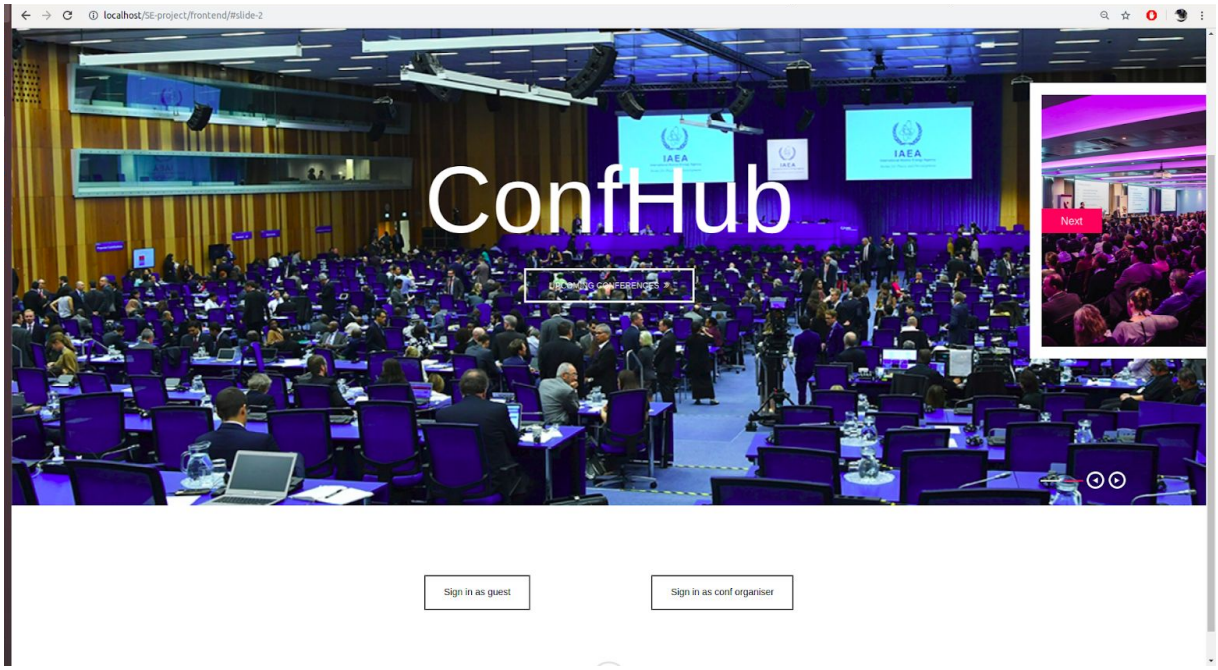
Could have added what is not going to be used just to get an idea of what makes the chosen styles or patterns better.

3.0 User interface design

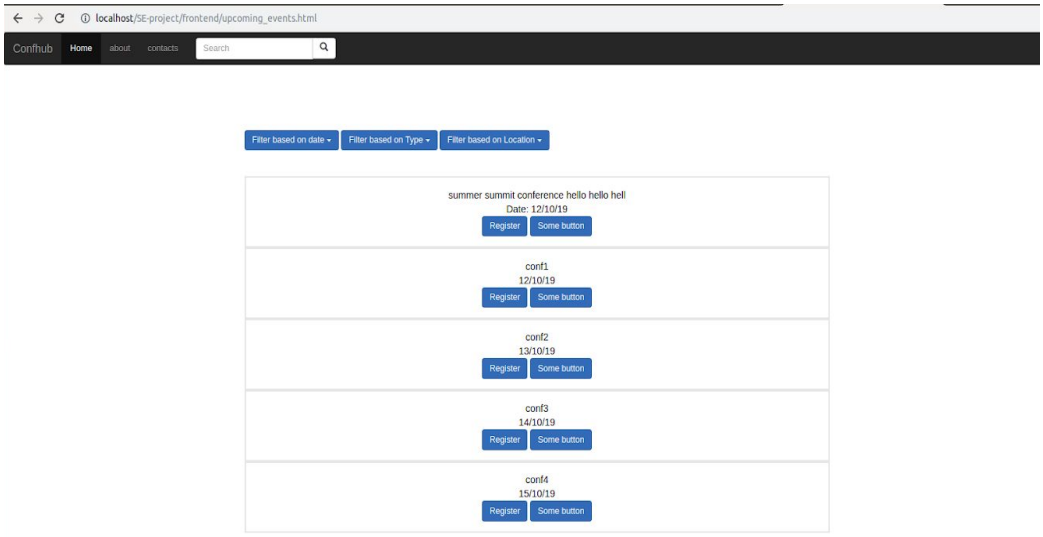
Screenshot 1



Screenshot 2



Screenshot 3



Brief Description

- Each page of the UI will have the title of the website i.e. “Confhub” in the beginning followed by a set of navigable pages.
- User interface mainly divided into two parts i.e. Guest page, conf list page (after login)
- The guest page will have a button to view list of upcoming conferences and options to choose to continue as attendant or to continue as conference organizer
- The conf list page will have a search bar for the user to search for specific conferences along with filter options below the search bar
- On the opposite side of the search bar there are buttons to choose from ‘conferences’, ‘analytics’ and ‘conferences near me’ which varies in the computation done to display the list of conferences
- There is also a button below the search button to the right to view the user details such as past conferences, upcoming conferences
- The rest of the page is followed by the list of conferences
- The footer of the page contains copyrights and team information

USER INTERFACE DESIGN IS FINE.

The pictures selected for the home page might hinder the visibility of the text

4.0 Detailed Design Approach

Booch Method: The Booch method is a method for object-oriented software development in the analysis and design phases. An iterative object-oriented development process.

Macro process: includes the first step is to establish the requirements from the customer perspective. This analysis step generates a high-level description of the system's function and structure, developing a model of the desired behaviour, creation of an architecture, implementation and maintenance.

Micro process: includes identification of classes and objects for each component or service, identification of their semantics, relationships, specification of their interfaces and implementation.

4.1 Design patterns considered and for what reason

S.No	Design Pattern	Intent of this pattern	Rationale for choosing or not choosing
1.	Structural decomposition pattern	Breaks down a large system into subsystems.	Distribution of work roles in a team is easier and organised.Tasks can be divided between role specific groups in a team.
2.	Observer/Publish /subscribe (structural design pattern)	Object maintains a list of its dependents and notifies them automatically of any state changes, usually by calling one of their methods.	Whenever data changes for the list of conferences and there are new upcoming events, the display elements are notified with new data and they display the latest data accordingly. In addition, we have alert subscription too.
3.	Factory pattern	Derived classes figure out what to instantiate and decouple client from instantiated class.	We are not using factory pattern because in a dynamically typed language like Python, the client can use the interface provided by the resultant object without knowing the type of the object.
4.	Prototype	We don't create the objects of a class directly but clone the existing object and change the state of the object as needed.	We have a database class, the constructor sets up the database for the class. Now for each new user logging to the system once the system is up, we don't setup the database but just clone the first object and change the user specific details like user name / password to validate the user.

Well written!!

One suggestion, with thousands of conferences, Object Pool - a creational design - pattern could have been considered for recycling of resources (not sure how out-of-date conferences are being handled currently)

5.0 Requirement Traceability Matrix

Sl. No	Req Id	Brief Desc	Architecture Ref Section	Design Ref	Code File Ref	Unit Test Cases	Function/ System test cases
1	1	User registration	FR-1	2.2.3			
2	2	User login	FR-2	2.2.3			
3	3	User logout	FR-2	2.2.3			
4	4	Presentation of list of conferences	FR-3	2.2.7			
5	5	Managing user reminders	FR-3	2.2.2			
6	6	Search and filter over conference data	FR-3	2.2.5			
7	7	Presentation of conference analytics data	FR-3	2.2.8			
8	8	Listing conferences near the user	FR-3	2.2.8			
9	9	Periodic scraping and cleaning of conference data	FR-3	2.2.1			