GSoC’ 22: Proposal

Name:

Email:

Phone:

Github: *https://github.com/noman2002*

Linkedin:

University:

Degree:

About me:

What exists already? What is the identified need? What can you reuse, and what needs to be replaced?

.
.
..
.
.
.
.
.
.
.
.
.
.
..
..
.

Deliverables: (As per the idea list)

1..

2.

3.

4.

5.

6.

7.

Design and Description of Work:

1. Create ways for volunteers and/or attendees to have checked in for events for better coordination.

Overview:

Just having the registration for an event doesn't mean that the person attended the event or not which makes it tough for the organizer/admin to keep a record of people present at the event. Having a check-in feature would be a boon, where the admin can track the attendees and/or volunteers present at the event.

Solution:

Currently, [project] has the feature to register for the event only which makes it difficult for the admin to manage the attendees who actually attended the event. For that, I propose to implement a checkbox in front of every name on the event info page which is visible to admins only through which they can check-in a person for the event. Which makes it a lot more convenient for the admins to manage things easily through the app itself. These checkboxes will provide ease to the admin and such a huge problem can be fixed through minimal yet perfect feature. refer to fig. 1.1

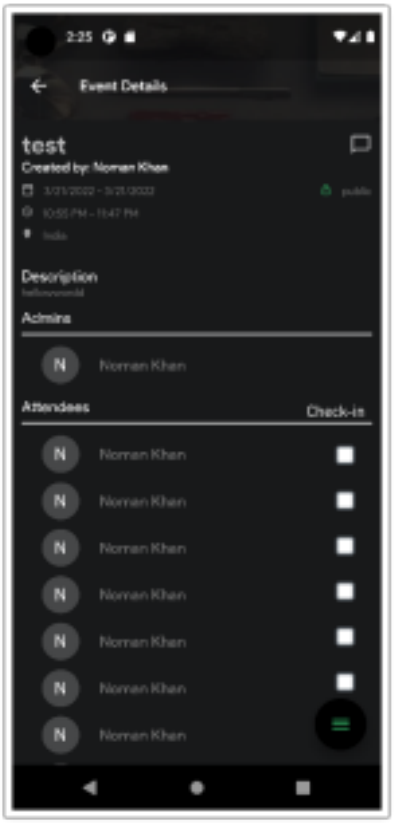
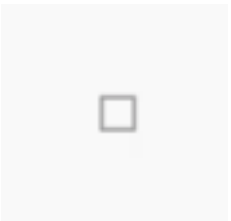


fig 1.1

For implementing this feature we can use the checkbox class which is a material design checkbox. The checkbox does not keep track of its own state.

Instead, the widget invokes the onChanged callback when the state of the checkbox changes. Most checkbox-based widgets will listen for the onChanged function and rebuild the checkbox with a new value to update the visual appearance of the checkbox. refer to fig 1.2 for a better understanding of implementation through the code snippet.



gif 1.1

```
bool isChecked = false;

 @override
 Widget build(BuildContext context) {
     Color getColor(Set<MaterialState> states) {
      const Set<MaterialState> interactiveStates =
<MaterialState>{
     MaterialState.pressed,
     MaterialState.hovered,
```

```
        MaterialState.focused,
        };
        if (states.any(interactiveStates.contains)) {
        return Colors.green;
        }
        return Colors.grey;
        }

        return Checkbox(
        checkColor: Colors.white,
       fillColor: MaterialStateProperty.resolveWith(getColor),
        value: isChecked,
        onChanged: (bool? value) {
        setState(() {
              isChecked = value!;
        });
        },
        );
}
```

fig 1.2

2. Implement ways to limit the number of event attendees.

Overview:

While organizing an event sometimes it becomes very difficult for the organizers to manage the audience as sometimes we get an overwhelming number of registrations and we can't say no to any attendee who already registered for the event. We can tackle this problem by limiting the number of attendees as per the convenience of organizers and venue while creating the event or we can further edit it.

Solution:

For solving the above-mentioned issues I am proposing to implement a slider for limiting the number of registrations while creating an event that the admin can edit further. The maximum number on the slider can be determined by the maximum capacity of the venue. Refer to fig 2.1 for a better understanding of how I'm planning to implement it in the app.

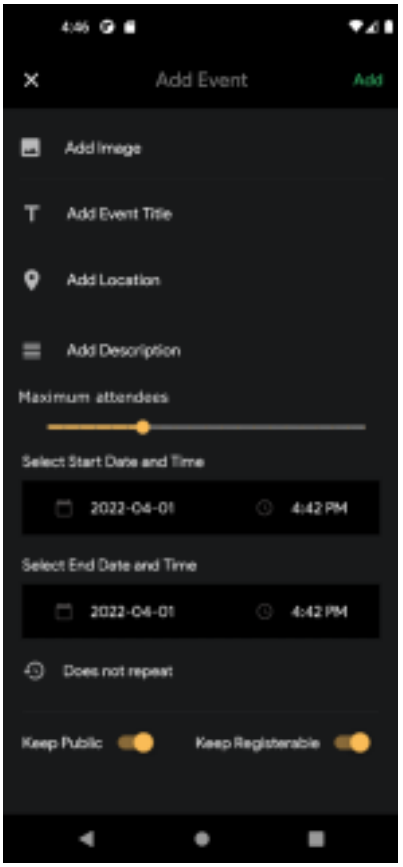


fig 2.1

Slider Class will be used for implementing it, which is a material design slider used to select a range of values. Refer to fig 2.2 for the code snippet of implementation.

gif 2.1

```
double _currentSliderValue = 100;

@override
Widget build(BuildContext context) {
return Slider(
      value: _currentSliderValue,
      max: 500,
      divisions: 10,
      label: _currentSliderValue.round().toString(),
      inactiveColor:Colors.grey,
      activeColor:Color(0XFFFABC57),
      onChanged: (double value) {
      setState(() {
```

```
        _currentSliderValue = value;
        });
        },
);
 }
```

fig 2.2

3. Allow venues to be reserved from being used for other events.

Overview:

The app includes a feature that allows users to organize and manage events, but this is insufficient for an organization to conduct and manage events. It requires various other means to conduct an event smoothly, such as a system for adding venues and reserving them for another event to avoid a collision.

We can fix this by creating a section for venues and a calendar similar to Google Calendar that allows us to organize events at specific venues.

Solution:

I propose to implement a tab bar for the venue and events through which we can switch between venues and events. In the venues section, we have the list of venues and a floating action button to add a new venue with the required information like name, maximum capacity, location, etc similar to add event form, we can reuse it with some modifications refer to fig 3.1. On tapping a specific venue we can have a calendar like a google calendar in which we can see the events at that particular venue on upcoming dates and times.

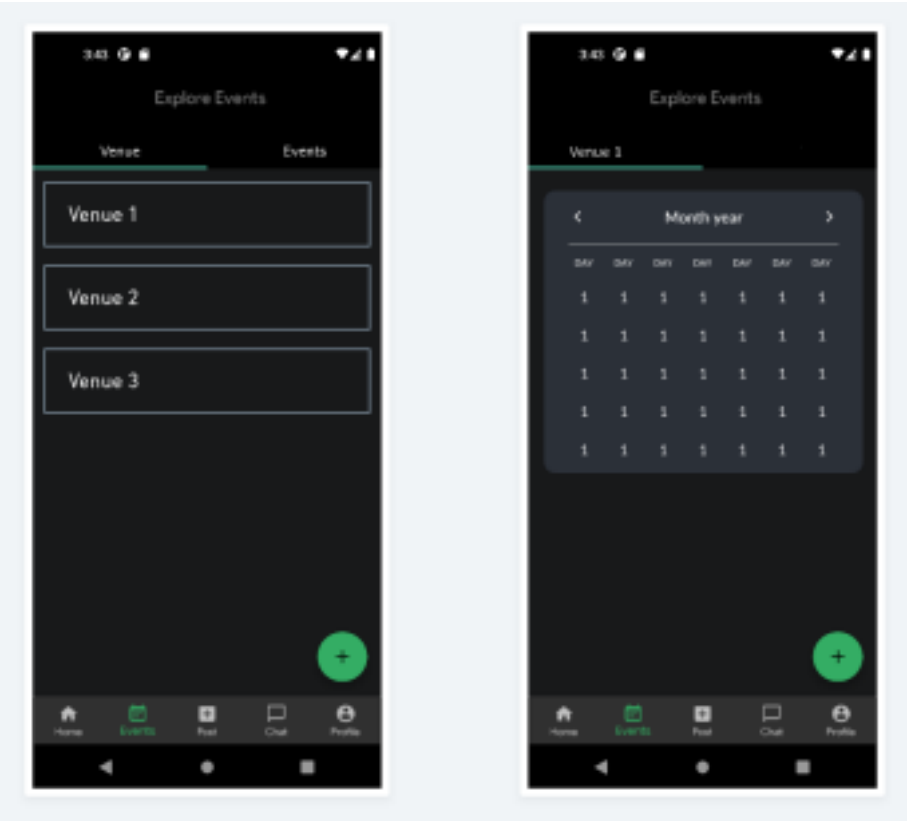


fig 3.1

The tab bar is already present in the app in the chat section (chat_list_screen.dart) so I can reuse it with some modifications. I can use syncfusion_flutter_calender for the implementation of the event calendar. It offers built-in customizable views for scheduling and representing events efficiently, including day, week, work week, month, schedule, timeline day, timeline week, timeline workweek, and timeline month. refer to fig 3.2 for a better understating of implementation through the code snippet.

```
@override
  Widget build(BuildContext context) {
    return Scaffold(
        body: SfCalendar(
      view: CalendarView.month,
      dataSource: MeetingDataSource(_getDataSource()),
      // by default the month appointment display mode set as
Indicator, we can
      // change the display mode as appointment using the
appointment display
      // mode property
      monthViewSettings: const MonthViewSettings(
          appointmentDisplayMode:
MonthAppointmentDisplayMode.appointment),
    ));
  }
```

```
List<Meeting> _getDataSource() {
  final List<Meeting> meetings = <Meeting>[];
  final DateTime today = DateTime.now();
  final DateTime startTime =
      DateTime(today.year, today.month, today.day, 9, 0, 0);
  final DateTime endTime = startTime.add(const Duration(hours:
2));
  meetings.add(Meeting(
      'Conference', startTime, endTime, const
Color(0xFF0F8644), false));
  return meetings;
}
```

fig 3.2

Disclaimer: This is a commercial package. To use this package, we need to have either a Syncfusion commercial license or [Free Syncfusion Community license.]

We can use free license as the eligibility criteria for free license state “Companies and individuals with less than $1 million USD in annual gross revenue”.

Now coming to the event tab, We can have a calendar icon button on the event screen as shown in fig 3.3 through which we can navigate to our event calendar where we can see all the events scheduled irrespective of venues as it is in the previous screen.

The implementation for the event calendar will be the same as stated above.

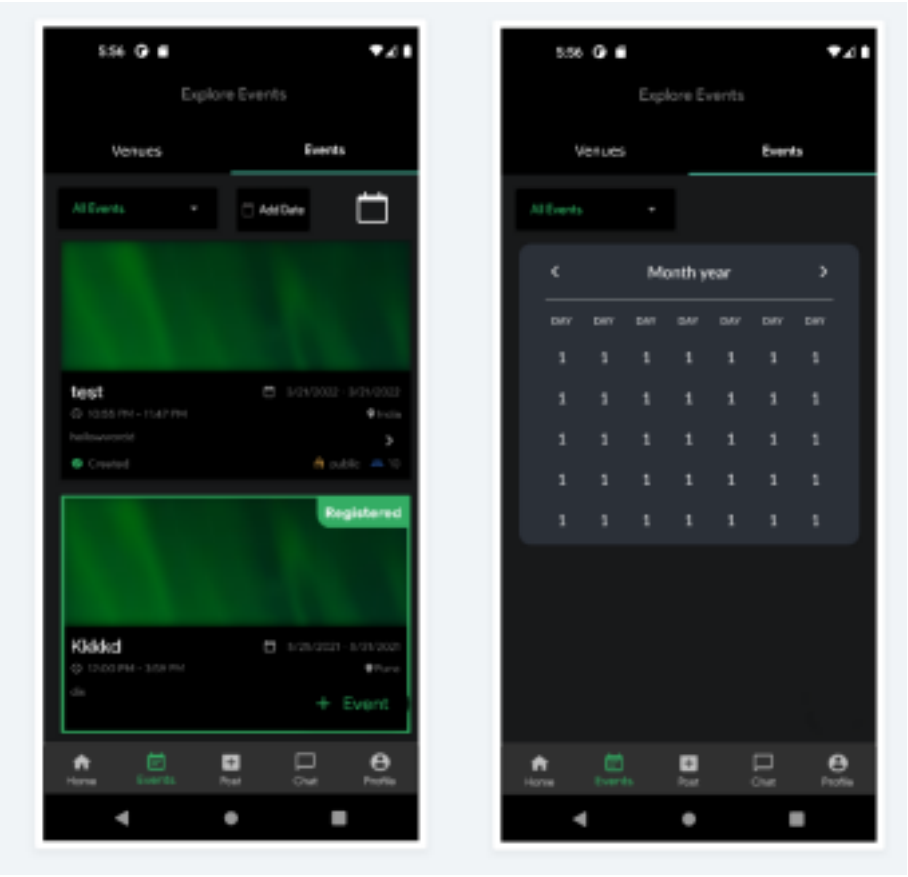
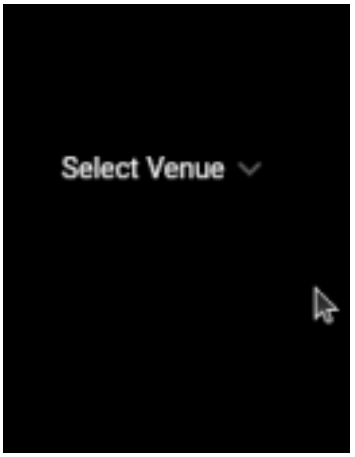


fig 3.3

For integrating the venue thing we also need some extra fields to store data about the venue on the server. I planned to add a dropdown button for the list of venues. As shown in fig 3.5 and gif 3.1.



gif 3.1

Code snippet for dropdownbutton:

```
String dropdownValue = 'Select Venue';

  @override
  Widget build(BuildContext context) {
      return DropdownButton<String>(
      value: dropdownValue,
      icon: const Icon(Icons.expand_more),
      elevation: 8,
      style: const TextStyle(color: Colors.white),
      dropdownColor: Colors.grey[800],
      underline: Container(
      height: 0,
      ),
      onChanged: (String? newValue) {
      setState(() {
            dropdownValue = newValue!;
      });
      },
      items: <String>['Select Venue', 'Venue 1', 'Venue 2', 'Venue 3']
                .map<DropdownMenuItem<String>>((String value) {
      return DropdownMenuItem<String>(
            value: value,
            child: Text(value),
      );
      }).toList(),
      );
  }
```

fig 3.4

For this feature we also need to make changes to the [project]-API in which we have to add models for the venue, venue mutations like add venue, delete venue, edit venue, etc.

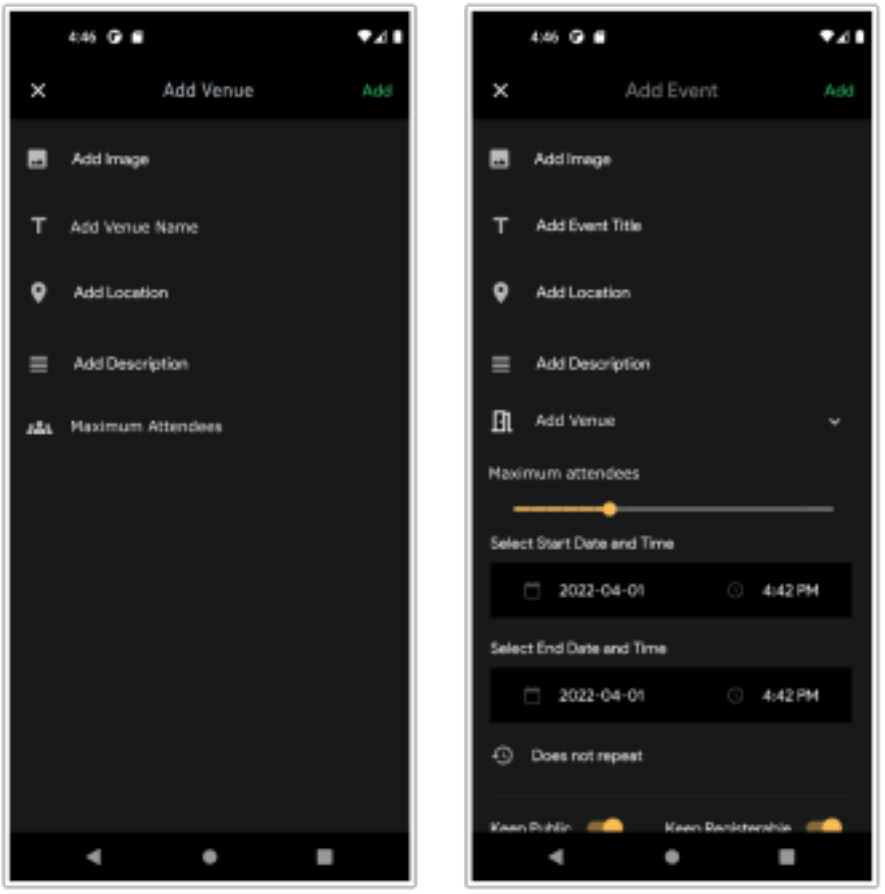


fig 3.5

We need an add_venue_form which contains Image, Venue Name, Location of the venue, a short description about the venue (might be the address of venue) and at last the maximum number of attendees to be accommodated.

To inculcate the venue thing in the existing events we need to have one more field for the venue in create_event_form and edit_event_form in the form of a dropdown button where the options of dropdown can be fetched from the API/backend.

4. Create ways for attendees to register for events with or without an invitation.

Overview:

The app supports registrations without invitation by simply going to the event and clicking on the register button and it's done but there is no method through which can invite someone to register for the same event. For that, I am proposing to implement an invite/share button on the event info page through which we can share the event or invite others to the event.

Solution:

I propose to implement a feature to invite users similar to every other app present on the app store. Where we create an embedded link for that particular event that we want to invite and share that link with a customized message through various other sharing apps and email.

After a user is invited to our application, an e-mail is sent with a link, which when selected should follow this flow.

Case 1: The guest user doesn't have the app installed.

The link should redirect to the app store that corresponds App Store/Play

Store. Case 2: The user does not open the link from the mobile.

The link should redirect to a page with links to download the

application. Case 3: The invited user has the app installed.

The link should redirect to the app and after that redirect to the invitation screen.

Dynamic Link parameters:

```
final dynamicLinkParams = DynamicLinkParameters(
  link: Uri.parse("https://www.example.com/"),
  uriPrefix: "https://example.page.link",
  androidParameters: const AndroidParameters(
      packageName: "com.example.app.android",
      minimumVersion: 30,
  ),
  iosParameters: const IOSParameters(
      bundleId: "com.example.app.ios",
      appStoreId: "123456789",
      minimumVersion: "1.0.1",
  ),
  googleAnalyticsParameters: const GoogleAnalyticsParameters(
      source: "twitter",
      medium: "social",
      campaign: "example-promo",
  ),
  socialMetaTagParameters: SocialMetaTagParameters(
      title: "Example of a Dynamic Link",
      imageUrl: Uri.parse("https://example.com/image.png"),
  ),
);
final dynamicLink =
      await FirebaseDynamicLinks.instance.buildShortLink(dynamicLinkParams);
```

fig 4.1

Create Dynamic links from parameters

```
final dynamicLinkParams = DynamicLinkParameters(
  link: Uri.parse("https://www.example.com/"),
```

```
  uriPrefix: "https://example.page.link",
  androidParameters: const AndroidParameters(packageName:
"com.example.app.android"),
  iosParameters: const IOSParameters(bundleId: "com.example.app.ios"),
);
final dynamicLink =
      await
 FirebaseDynamicLinks.instance.buildLink(dynamicLinkParams)
```

; fig 4.2

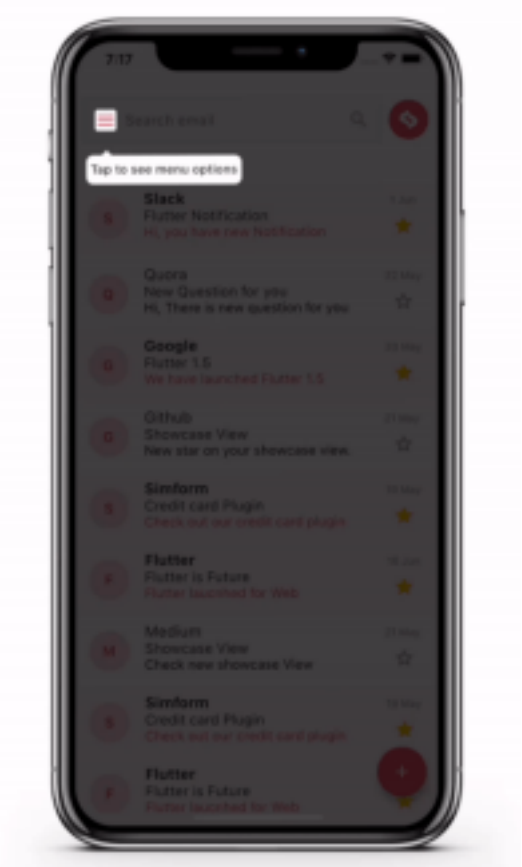
5. Adding an app guide for onboarding users

Overview:

Despite having a simple and easy to use user interface it is very important to have a basic app guide through which, when the user opens the app for the first time he will be prompted with basic info of what is where this will make the user familiar with the UI and ultimately lead to a better user experience.

Solution:

I propose to implement the app guide feature where users will be presented with the info about important components that are important for them to operate the app, like different options in the bottom bar, what different buttons do, etc. For this implementation, we can use ShowCaseView. It is a Flutter package to Showcase/Highlight widgets step by step which requires a very basic setup, refer to gif 5.1 to get an idea of how it looks. A simple showcase can be implemented using the code snippet shown in fig 5.1 and fig 5.2.



gif 5.1(Click here to view gif)

```
GlobalKey _one = GlobalKey();
GlobalKey _two = GlobalKey();
GlobalKey _three = GlobalKey();

...

Showcase(
  key: _one,
  title: 'Menu',
  description: 'Click here to see menu options',
  child: Icon(
      Icons.menu,
      color: Colors.black45,
  ),
),
```

Fig 5.1

```
Showcase(
  key: _two,
  title: 'Profile',
  description: 'Click here to go to your Profile',
  disableAnimation: true,
  shapeBorder: CircleBorder(),
  radius: BorderRadius.all(Radius.circular(40)),
  showArrow: false,
  overlayPadding: EdgeInsets.all(5),
```

```
    slideDuration: Duration(milliseconds: 1500),
    tooltipColor: Colors.blueGrey,
    blurValue: 2,
    child: ...,
),
```

Fig 5.2

6. Testing the Implemented features.

Testing: I believe that tests can very well explain the intention of the programmer for a piece of code, better than comments can do as writing tests ensures that developers catch the regressions at an early stage.

There are already so many code coverage issues created on GitHub. Writing tests for the new feature will reduce the number of issues to be created and improve code coverage. This will help in making the app production-ready and bug-free. With the implementation of the new features, I will contribute to testing the existing and the implemented ones simultaneously.

To **unit test** any particular class in isolation, any external dependencies (classes) need to be mocked so no external noise affects our tests. The mocked object would simulate the behavior of a real object but knows in advance what's supposed to happen in the test and its intended behavior. I'll use the **mockito** package to add mock classes.
For **widget testing** of all files in the UI, I would use WidgetTester along with the pump() method for testing Stateful Widgets and would refer to this article.

At the end of the development process, I'll add the rest of the tests like **integration tests** to ensure proper **performance profiling.** Integration test mimics user behaviors. The goal would be to verify that all the widgets and services being tested work together as expected. This will also check if the ScrollView behavior is working smoothly and the application is free of 'jank', i.e, no frame is being skipped.

To maintain proper testing standards, I'll refer to the flutter cookbook.

7. Signing the APK and Playstore release.

This would be the final stage where the main problem to solve is to test the app against all the checks and add more strong checks if required, meanwhile, if any debuggable area is encountered then fixing it in place and finally heading towards the release which includes generating the artifacts or APK in this case and releasing the beta version to playstore. If time permits then we can also do security checks on APK using mobSF, DexGuard, etc.

Timeline:

Timeline	Tasks
May 20 - June 12	**Community Bonding Period** ➢ Interact with the mentor and other community members and discuss how I plan to implement the ideas and ask for suggestions. Get a deeper understanding of the codebase. Fix minor bugs and write tests to reach 100% code coverage.
June 13	**Coding Officially begins !!**
June 13 - June 18 (Week 1)	**Adding the feature to check-in attendees.** ➢ Adding the checkbox class in the existing ListTile. ➢ Adding required parameters in API with the help of mentor. ➢ Writing test for the implemented feature.
June 20 - June 25 (Week 2)	**Implementing ways to limit the number of attendees.** ➢ Add a slider to the UI using the slider class. ➢ Discuss the parameters of the slider with a mentor and modify them accordingly. ➢ Creating a parameter to store the value and restrict the registration after reaching the limit. ➢ Testing the above-implemented feature.
June 27 - July 09 (Week 3 + Week 4)	**Creating a way to manage venues and reserve them for events.** ➢ Creating a tab bar for venue and event. ➢ Creating add_venue_form for adding a new venue. ➢ Discuss the parameters for the venue with mentor and create ListTile for the same. ➢ API and database modification with the help of mentor for the venue data to be stored.

July 11 - July 16 (Week 5)	**Creating a Calendar View for both event and venue.** ➢ Make the required modification to it and make it compatible with the current UI. ➢ Add events to the calendar by fetching them fromthe server using API requests.
July 18 - July 23 (Week 6)	Fix any potential bugs, if raised, after implementing the above-discussed features and blog on all the work done.
July 25	**Phase 1 evaluation**
July 25 - July 30 (Week 7)	**Continued refactoring and performance enhancements** ➢ Updating all the dependencies to their latest stable

	version if any. Removing unnecessary import statements. ➢ Testing for the above-mentioned features and views.
August 01 - August 06 (Week 8)	**Creating embedded links to invite attendees to an event.** ➢ Creating the invite sliding_up_pannel. ➢ Discuss mentor and create the embedded link. ➢ Testing the implemented feature.
August 08 - August 20 (Week 9 - Week 10)	**Adding app guide/ walkthrough for onboarding users.** ➢ Adding required dependency for implementing this feature i.e. ShowCaseView. ➢ Creating the simple guide sequence for Dashboard with relevant data and information. ➢ Taking suggestions from mentors regarding info and UI. ➢ Expanding the sequence to multiple screens. ○ Event view ○ Add post view ○ Chat view, etc. (as suggested by mentor)
August 22 - August 27 (Week 11) August 29 - September 03 (Week 12)	Testing and fixing UIs for multi-screen sizes. The app currently works fine with most of the generic smartphone screen sizes, but requires testing on larger tablet screens(>7”). Fix any potential bugs, if raised, after implementing the above-discussed features and blog on all the work done. Running all checks and generating artifacts/signed apk for beta release.
September 05	**Final Evaluation**

Contributions to Organisation:

Issue:

Pull Requests(Merged):

What other commitments do you have this summer?

I don’t have any active commitments this summer. Completing this project is my primary goal and I assure to prioritize this with utmost sincerity and dedication. However, I'd like to mention that it takes about 8 hours to reach my college hostel or back to my hometown, for which I'd be unavailable if I have to travel, usually once per semester.

Why are you the best person to execute this proposal?

.
.
.
.
.
.
.

Other contributions to different organizations:

Community Engagement:

.
.
.
.
.
..

Post GSoC:

If there are things left unimplemented (like documenting the codebase or testing or anything other feature), I would strive to complete them post-GSoC and will keep contributing to the enhancement of this project.

Gratitude towards the organization:

The mentors and members of the organization have been so kind and generous since I started contributing. I am grateful enough while writing this proposal that I will get an amazing opportunity to work with a great community and learn immensely through it. I am looking forward to all the feedback and am willing to learn and change accordingly.

Thanks