Software Design

Ac. yr. 2019./2020.

Festival organization

Documentation, Rev. *2*

Group: *Festival organisation*
Coordinator: *Bartol Bilic*

Turn-in Date: *16 . 01 . 2020.*

Supervisor: *Hrvoje Nuic*

Contents

1. Documentation Change Log

Rev.	Change Description/Appendix	Authors	Date
0.1	Template uploaded to git.	Bilic	20.10.2019.
0.2	Project Description written.	Ceple	6.11.2019.
0.3	Functional Requirements written.	Ceple	6.11.2019.
0.4	Class Diagram	Strbad	12.11.2019.
0.5	Use Cases finished	Ceple	13.11.2019.
0.6	Database description and diagrams added	Fribert	13.11.2019.
0.7	Architecture description added	Bilic	14.11.2019.
0.8	Sequence Diagrams added	Ceple	15.11.2019.
1.0	First version	Bilic, Strbad, Rey Sparem-blek, Ceple	15.11.2019.
2.0	Everything else	Everyone	16.01.2020.

2. Project Description

2.1 General Idea

The idea of this app is to enable a low to mid size festival organisation in a relatively simple and straight-forward manner that would be easily accessible and understandable even to non-technically educated Users.

The application would be open-source, and would run on the Android OS - a native mobile app.

2.2 Position in the market - the competition

Mainly, the competition consists of either high-profile professional apps, or non-native(non-mobile) apps. Thus, the point of this app is to fill that gap - it's supposed to be a native app, that's relatively simple to use, and very portable and easily deployable.

This would also imply, and goes hand in hand, with the fact that the application would be easy to use and easily accessible to a wide range of people - from highly-trained professional IT Users all the way to non-IT savvy amateur/inexperienced Users.

Because of the low difficulty, and relatively ad rem employability, this app would be more suitable to the lower skill level(entry to mid-level) Users, as it is likely that Pro Users would require a larger scale App for, probably, larger caliber Festivals that they deal with. With that in mind however, this app can be used as a mini, mobile device reminder version of whatever Pro tool is used for Festival organisation.

In the market there is presence of both event organisation, and user-event interface apps. This app is focused on organisation, and not festival-goers. Therefore, from a marketplace standpoint, there will be no impact on the demand of the application due to the selected specialisation.

2.3 In-Depth Description

Application would be used for Festival organisation - music festivals, film festivals, library events, food and alcohol festivals, parties, birthdays, and other kinds of meet-ups. By scope and complexity it would be used for smaller and medium scale Events/Festivals.

The system would be run and moderated by Administrators. The system/platform would allow concurrent usage by multiple Users. These Users are: Administrators, Creators, Organisers, Workers.

From here on, unregistered users and Users who aren't logged in will be referred to as textitGue

2.3.1 Account Data

Prior to registration, Guests must fill in a Registration form. This form consists of:

- Username
- Email
- Password
- Password Verification
- Phone Number
- Name
- Surname
- Desired Role
    1. Leader
    2. Organiser
    3. Worker

Aside from this data, Users can also upload their Profile Picture. In case they don't, a default anonymous one will be used. These Profile Pictures will be put on Users' Cards that serve as Festival entrance tickets. Users can change some of this data. Specifically, they can alter:

- Email
- Password
- Phone Number
- Profile Picture

Users aren't allowed to modify their Usernames in order to prevent abuse and misuse. If Users want their Usernames, Names or Surnames changed, they can contact the Administrator, and if the appeal makes sense the Administrator can change those values for the User.

2.3.2 User roles

2.3.3 Administrators

Maintain and organize the platform. They curate the Creators. They have complete transparency and access to all data. They can veto and issue bans on: Festivals, Jobs, User registrations, comments, etc... Basically they have complete control over the platform.

2.3.4 Creators

Need to be verified by one of the Administrators. Have the ability to create and edit Festivals, as well as have complete transparency of the Festivals they have created. Can undertake certain actions regarding the details of their Festivals(and Jobs).

They appoint and verify Organisers. To these Organisers they assign Festivals that need to be organised - delegation of Festival planning and execution. Organisers cannot organise concurrent Festivals.

2.3.5 Organiser

Organisers are appointed by the Creators. They manage, plan and execute Festivals on both a macro and micro/detail scale. They are appointed to Festivals by Creators.

They can organise multiple Festivals, and be appointed to those Festivals by multiple Creators - it is only important that none of those Festivals are concurrent.

They organise and manage Jobs and Activities that need to be done for the Festival. Jobs are performed by Workers. Organisers also have the ability to write and read comments on the Worker's walls as to better organise these Jobs.

Job and Activity management is done via Auctions. First Workers apply to these Auctions, and their entries need to be verified by the Organisers. And then the Verified Entries can compete to receive the Job Offer. Should they deem it necessary, Organisers can extend Auctions by one day.

2.3.6 Worker

They perform Jobs and Activities. For these Jobs they first have to apply to the selection process(Auctions) during which their entries are curated by the Organisers(or eventually Administrators or Creators). Upon verification, their entries compete in order for the Worker to receive the Job Offer.

Some Jobs are necessary to be done by multiple Workers. Workers can specify in their Auction Entry how many more Workers would the Job require. Jobs can be done concurrently(parallelised), and Workers can perform Jobs with maximum freedom, so long as these Jobs aren't concurrent - in case they are, the system will immediately veto such entries.

Workers' accounts have certain specifics compared to other accounts. Their profiles contain:

1. Field of specialisation
2. Basic account data and information
3. Former Job information
4. Comment section on the Worker's profile wall – intended to allow the Worker's co-workers, boss, ... to comment the Worker's performance, characteristics, their satisfaction of working with the Worker, etc...

2.3.7 Functions and happenings of the system

Here some inner elements of the system will be defined, along with their attributes and characteristics.

2.3.8 Festival

Festivals are events that are being organised. Their attributes are:

1. Name
2. Description
3. Location
4. Planned start-end time

2.3.9 Auction

Auctions are part of the process where Workers apply to Jobs. First Workers need to turn in their entries, which are then verified by the Organisers/Creators, and

moderated by Administrators. Upon successful verification/moderation, Workers' entries finally enter the Auction competition where the lowest bid ends upon the expiration of the Auction period. If need be, Organisers and Creators can extend the Auction period by one day.

Auctions have the following attributes:

1. Price
2. Comment
3. Number of Workers needed(Workforce Quantity)
4. Estimated time to Completion(ETC)

2.3.10 Timelines

Timelines are intended to provide an easy and graphical way of organising Jobs in the time-domain – this is done in a way that the timeline itself is actually a flowchart organised against a time axis. The liable User can add, remove and manipulate the Objects located in the Timeline.

The beginning time, end time and the duration of a Timeline Object can be manipulated by moving and resizing the Object's corresponding box. This provides a visual way of organising Objects, and makes it easy to parallelise them.

The ability to parallelise implies the existence of multiple branches. Branches can be created, deleted, and moved.

There are 2 types of Timelines:

- Festival Job Timeline
- Job Auction Timeline

2.3.11 Job

Jobs are performed by Workers. They are constituted of Activities that need to be done in order for the Job to be completed. Jobs can require multiple Workers. The lowest bid Worker is assigned the Job. Auctions usually last 1 day, but if necessary Organisers and/or Creators can extend them by an additional day.

Jobs are to be given a sequential order of execution. They can be parallelised as well. Multiple Workers cannot work on concurrent Jobs.

2.3.12 Activity

Jobs consist of multiple Activities that need to be done in order for the Job to be completed - as already defined above. Activities help break down Jobs into manageable, and easily organised chunks. Since Jobs can be done by multiple Workers, as well as be parallelised, Activities can help with that task as they would be the elementary particle, the smallest unit of work that needs to be done and distributed throughout the network of Workers that would be performing the given Job.

2.3.13 Cards

Participants in the organisation of the Festival would each receive a **SINGLE** card - with an exception of multiple Workers working on the same Job - that will be explained further below. Workers also have another specificity - their Cards feature some additional information.

A single Card can be printed for a single User and a single Festival. Therefore, the same User is able to download and print multiple Cards for multiple Festivals.

2.3.14 Card Format

The cards can be downloaded in a .pdf format - with the dimensions 10x7cm

Everyone's Cards feature the following information:

1. User picture
2. Name and Surname
3. Name and Logo of the Festival
4. QR code(MD5 Hash):
   (a) User's Name
   (b) Name of the Festival

Workers' Cards contain **ADDITIONAL** information:

1. Time and Location of the Job
2. QR code(MD5 Hash):
   (a) ID of the Job
   (b) ID of the User

In case a Job needs multiple Workers - multiple Cards will be printed for all the Workers.

2.3.15 System implementation targets

The Platform is meant to enable **MULTIPLE** Users **CONCURRENT** access to, and usage of the Platform. This would make festival organising a dynamic and fast environment. As such, a modern object-oriented programming language will be used for implementation.

Since the application will be developed for Android OS, we have chosen Java as the programming language, and SQLite has been chosen for the database software.

2.3.16 Project Scope and Targeted Users

The software would be targeted towards people organising a festival and/or participating in its execution - administrators, technicians, investors, musicians, other kinds of artists, influencers, ...

The software, as said before, is aimed at primarily festivals of smaller sizes, but could be employed for festivals up to certain 'medium' size. Especially with improvements, it could be a viable free, open-source and easily protable alternative for both medium-sized festivals and small-sized festivals.

2.3.17 Changes, upgrades, adaptability of the Application

The application as it stands currently has some unfortunate restrictions on the freedom that Administrators, Creators and Organisers. It would be possible to modify these restrictions as to allow a greater freedom of Festival organisation, but still keeping some in place as to prevent abuse and misuse. Certain functionalities could also be added in order to make the app more useful to mid-sized Festivals.

A few changes that would probably help tremendously is to implement an intra-platform messaging service, feedback system, and support/ticked service. This would allow the communication within the App between Users. It would also provide a way for the Developers to easily diagnose, track and reproduce bugs, as well as see what changes, ideas and updates Users would like to see in the App. Finally, a support/ticket system would help alleviate any frustration that Users could suffer due to bugs and/or failures of the platform.

A list of changes(actively tracked) to eventually, if time and will permits, be implemented:

- Email confirmation

- Email reset

3. Software Specification

3.1 Functional Requirements

Stakeholders:

1. Developers and Maintainers
2. Festival-goers
3. Festival investors and sponsors
4. Administrator
5. Creator
6. Organiser
7. Worker

Actors and their functional requirements:

1. Unregistered/Guest User(initiator) can:
   (a) Register a new account - fill in the form
      i. Username
      ii. Email
      iii. Password
      iv. Password Verification
      v. Phone Number
      vi. Name
      vii. Surname
      viii. Desired Role
         A. Leader
         B. Organiser
         C. Worker
   (b) Log in - fill in the form
      i. Username or Email
      ii. Password

2. Administrator can:
   (a) Access the list of Users
   (b) Verify Creators
   (c) Moderate Users' details and/or ban them as to alleviate abuse/misuse
   (d) Access the list of Festivals
      i. Access the list of Jobs
         A. Access the list of corresponding Activities
         B. Access the list of Workers
      ii. Moderate Festivals, Jobs and Activities and/or veto/delete them as to alleviate abuse/misuse
3. Leaders manage Festivals:
   (a) Create [multiple] Festivals
   (b) Modify or delete their Festivals
   (c) Inherit Organiser functionalities for their own Festivals
   (d) Appoint Organisers to their Festivals(check if the selected Organiser is organising any possibly concurrent Festivals)
   (e) Access the Jobs, Activities, Workers and other details of their Festival
4. Organiser organises the concrete Festival workflow:
   (a) Job management
      i. Select which Jobs need to be done - open their corresponding Job Auctions
      ii. Job Sequence - order and parallelise Jobs - Festival Job Timeline
      iii. Ability to extend Job Auction lifetime by 1 day
      iv. View and modify Jobs
         A. Access Workers' profiles, details, comments, ...
         B. Access Job description, Time and Location - modify them as needed
         C. View each Job's list of Activities
   (b) Organise a Festival – Ability to organise multiple Festivals - check for concurrency!
5. Workers perform specific Jobs. They can:
   (a) Select their fields of specialisation
   (b) Apply to Job Auctions

(c) Perform Job - can perform multiple Jobs - check for concurrency
(d) Fill out Job information sheet
    i. Job Description
    ii. Job Location and Time
    iii. Form a list of Activities that need to be done - ability to modify, add and/or delete the entries in this list

3.1.1 Use Cases

Use Cases Description

UC1 - Registration

- **Main Stakeholders:** Unregistered/Guest Users
- **Goal:** Register a new User Account
- **Stakeholders:** Database
- **Conditions:** Must not be logged in, and must be located at the Login screen.
- **Event flow description:**

1. The User is located at the login screen, and taps the 'Create one' button, located next to the 'No account yet?' label
2. A new Screen pops up - Guest fills the Registration Form
3. Upon tapping 'Create Account' a new account is created and stored in the Database

2.a Illegal input

1. An error message is displayed informing the User that they have entered illegal input
2. The notification asks the User to re-format and re-enter the input. The expected format and rules are displayed
3. The steps above are repeated if new input isn't accepted either

3.a Data not successfully sent and parsed on the Server

1. An error message is displayed informing the User that an error has occurred
2. A 'Retry' button is displayed on the screen - upon clicking it the form is resent and parsing tried again
3. The User is notified upon success

UC2 - Log-In

- **Main Stakeholders:** Guest Users who have a registered account
- **Goal:** Log into the platform
- **Stakeholders:** Database
- **Conditions:** Must not be logged in, and must be located at the Login screen.
- **Event flow description:**

1. User enter the email or username and their password

2. User taps the ‘Log-in‘ button
3. Database checks the data and if login is successful the user is logged in

3.a Email/Username and Password combination is wrong and the User isn’t logged in.
   1. The fields Email/Username and Password are reset
   2. An error message is displayed
   3. User needs to enter the log-in data again

3.a Data not successfully sent and parsed on the Server
   1. An error message is displayed informing the User that an error has occurred
   2. A ’Retry’ button is displayed on the screen - upon clicking it the form is resent and parsing tried again
   3. The User is notified upon success

UC3 - Account Data Overview

- **Main Stakeholders:** User
- **Goal:** View the account data
- **Stakeholders:** Database
- **Conditions:** Must be logged in
- **Event flow description:**

   1. User taps the corresponding button, and views his profile information
   2. Data is retrieved from the Database
   3. A screen depicting Account data appears

   2.a Data not successfully retrieved
      1. An error message is displayed informing the User data couldn’t be retrieved
      2. A ’Retry’ button is displayed on the screen - upon clicking it the app attempts to fetch data again
      3. Upon successful data retrieval(If ever) the button and error message are removed and then the data is displayed

UC4 - Verifying a Leader

- **Main Stakeholders:** Administrators
- **Goal:** Check, and if all is in order, verify a Leader
- **Stakeholders:** Database, Leader

- **Conditions:** Must be logged in. There is a Leader awaiting confirmation.
- **Event flow description:**

  1. The list of Leaders is displayed to the Administrator
  2. Data is fetched from the Database
  3. Administrator decides whether to verify or deny the Leader

  2.a Data not successfully retrieved
    1. An error message is displayed informing the User data couldn't be retrieved
    2. A 'Retry' button is displayed on the screen - upon clicking it the app attempts to fetch data again
    3. Upon successful data retrieval(If ever) the button and error message are removed and then the data is displayed

UC5 - Create a Festival

- **Main Stakeholders:** Leader
- **Goal:** Create a Festival, and open it up to Workers so that they can start working
- **Stakeholders:** Database
- **Conditions:** Must be logged in
- **Event flow description:**

  1. The Leader opens the Screen for creating a Festival
  2. The Leader fills in all the necessary info required for creating a Festival
  3. The Leader submits the form and a Festival is created

  2.a Illegal input
    1. An error message is displayed informing the User that they have entered illegal input
    2. The notification asks the User to re-format and re-enter the input. The expected format and rules are displayed
    3. The steps above are repeated if new input isn't accepted either

  3.a Data not successfully sent and parsed on the Server
    1. An error message is displayed informing the User that an error has occurred
    2. A 'Retry' button is displayed on the screen - upon clicking it the form is resent and parsing tried again

3. The User is notified upon success

UC6 - Approve an Organiser to the selected Festival

- **Main Stakeholders:** Leader
- **Goal:** The Organiser is appointed and begins carefully managing and organising the Festival
- **Stakeholders:** Database, Organiser
- **Conditions:** Must be logged in, a Festival requires an Organiser
- **Event flow description:**

1. The Leader opens the Screen featuring their festivals
2. The Leader selects one of the Festivals
3. The Leader taps the button to approve organizers
4. The Leader selects one of the Organisers, and appoints them to the selected Festival
5. This data is sent to the Server, which updates the Database

4.a Data not successfully sent and/or parsed on the Server

1. An error message is displayed informing the User that an error has occurred
2. The Leader can try resending the form
3. The Leader is notified upon success

UC7.a - View the list of all the active Jobs

- **Main Stakeholders:** Worker
- **Goal:** View the list of all the active Jobs
- **Stakeholders:** Database, Leaders, Organisers
- **Conditions:** Must be logged in
- **Event flow description:**

1. The User opens the Screen for viewing the list of active Jobs
2. The data is fetched from the Database
3. The list is displayed to the Worker

1.a Data not successfully retrieved

1. An error message is displayed informing the User data couldn't be retrieved
2. A 'Retry' button is displayed on the screen - upon clicking it the app attempts to fetch data again

3. Upon successful data retrieval(If ever) the button and error message are removed and then the data is displayed

Worker taps the entry

1. The Job details are opened to the Worker. There he can further inspect the details of this Job

UC7.b - View the list of all the completed Jobs and Worker's Specializations

- **Main Stakeholders:** Worker
- **Goal:** View the list of all the completed Jobs and this Worker's Specializations
- **Stakeholders:** Database, Leaders, Organisers
- **Conditions:** Must be logged in
- **Event flow description:**

1. The User opens his profile screen
2. The data is fetched from the Database
3. The lists are displayed to the Worker

1.a Data not successfully retrieved

1. An error message is displayed informing the User data couldn't be retrieved
2. A 'Retry' button is displayed on the screen - upon clicking it the app attempts to fetch data again
3. Upon successful data retrieval(If ever) the button and error message are removed and then the data is displayed

UC8 - View the list of all the Specializations(Add Specialization view)

- **Main Stakeholders:** Worker
- **Goal:** View the list of all the Specializations
- **Stakeholders:** Database, Leaders, Organisers
- **Conditions:** Must be logged in
- **Event flow description:**

1. The User opens the Screen for viewing the list of Specializations
2. The data is fetched from the Database
3. The list is displayed to the Worker
4. Optional: Filtering the Specializations according to their name

1.a Data not successfully retrieved

1. An error message is displayed informing the User data couldn't be retrieved
2. A 'Retry' button is displayed on the screen - upon clicking it the app attempts to fetch data again
3. Upon successful data retrieval(If ever) the button and error message are removed and then the data is displayed

3.a Worker enters a name of the Specialization

1. The Specializations are filtered according to the given name
2. Only the selected Specializations are displayed to the Worker

Worker taps the entry

1. The specified Specialization becomes bound/added to the Worker

UC9 - Create a Specialization

- **Main Stakeholders:** Worker
- **Goal:** Add a new Specialization to the list of Specializations
- **Stakeholders:** Database, Leaders, Organisers
- **Conditions:** Must be logged in
- **Event flow description:**

1. The User opens the Screen for viewing the list of Specializations
2. The data is fetched from the Database
3. The list is displayed to the User
4. Optional: Filtering the Specializations according to their name
5. The Worker enters the new name of the Specialization
6. Upon tapping the 'Add Specialization' button, the Specialization is created and added to the list, but not to the Worker himself
7. New data is sent to the Server and parsed there

1.a Data not successfully retrieved

1. An error message is displayed informing the User data couldn't be retrieved
2. A 'Retry' button is displayed on the screen - upon clicking it the app attempts to fetch data again
3. Upon successful data retrieval(If ever) the button and error message are removed and then the data is displayed

6.a Data not successfully sent and/or parsed on the Server

1. An error message is displayed informing the User of this bad request

Worker taps the entry

1. The specified Specialization becomes bound/added to the Worker

UC10 - View Job details(Worker view)

- **Main Stakeholders:** Worker, Administrators
- **Goal:** View Job details and specifics - such as Time, Location, Name, Description, ...
- **Stakeholders:** Database, Leader, Organiser
- **Conditions:** Must be logged in, must have selected a Job
- **Event flow description:**

1. Data is retrieved from the server
2. The User can read Job specifics
3. The Worker can view the list of Specializations

0.a Data not successfully retrieved

1. An error message is displayed informing the User data couldn't be retrieved
2. A 'Retry' button is displayed on the screen - upon clicking it the app attempts to fetch data again
3. Upon successful data retrieval(If ever) the button and error message are removed and then the data is displayed

UC11 - Apply to the Job(Worker)

- **Main Stakeholders:** Worker
- **Goal:** Send his application to the Job's supervising Organiser/Leader
- **Stakeholders:** Database, Organiser, Leader
- **Conditions:** Must be logged in, the corresponding Screen is opened
- **Event flow description:**

1. The Worker taps the 'Apply for a Job' entry in the hamburger menu
2. The data is fetched from the Server and the list displayed to the User
3. The Worker can now select the Job which he wants to apply for
4. Job details are fetched and displayed to the Worker

1.a, 3.a Data not successfully retrieved

1. An error message is displayed informing the User data couldn't be retrieved

2. A 'Retry' button is displayed on the screen - upon clicking it the app attempts to fetch data again
3. Upon successful data retrieval(If ever) the button and error message are removed and then the data is displayed

3.a The worker taps on the Apply button

1. A form is opened that Worker fills out
2. Data is sent to the server

0.a Illegal input

i. An error message is displayed informing the User data couldn't be retrieved

ii. A 'Retry' button is displayed on the screen - upon clicking it the app attempts to fetch data again

iii. Upon successful data retrieval(If ever) the button and error message are removed and then the data is displayed

1.a Data not successfully sent to the Server

i. An error message is displayed informing the User data wasn't successfully sent to the Server

ii. A 'Retry' button is displayed on the screen - upon clicking it the app attempts to send the data again

iii. Upon success(if ever) the Screen is closed. Otherwise the aforementioned procedure is again executed

UC12.a - View the list of all the Jobs and their details - Leader

- **Main Stakeholders:** Leader
- **Goal:** View the list of all the Jobs and their details
- **Stakeholders:** Database, Worker
- **Conditions:** Must be logged in
- **Event flow description:**

1. The user selects the option 'Job applications'
2. The data is fetched from the Database
3. The list is displayed to the User
4. Each list entry is a Job and contains its details

1.a Data not successfully retrieved

1. An error message is displayed informing the User data couldn't be retrieved

2. A 'Retry' button is displayed on the screen - upon clicking it the app attempts to fetch data again
3. Upon successful data retrieval(If ever) the button and error message are removed and then the data is displayed

UC12.b - View the list of all the Jobs - Organizer

- **Main Stakeholders:** Organizer
- **Goal:** View the list of all the Jobs
- **Stakeholders:** Database, Worker
- **Conditions:** Must be logged in
- **Event flow description:**

1. The data is fetched from the Database
2. The list is displayed to the User
3. Each list entry is a Job
4. There are 4 tabs that are used for Job classification
   (a) Auction - the Jobs that are currently competing
   (b) Active - the Jobs that have got assigned Workers
   (c) Pending - Jobs that do not have Workers
   (d) Completed - Jobs that are done - Organiser has confirmed that they are completed

0.a Data not successfully retrieved
1. An error message is displayed informing the User data couldn't be retrieved
2. A 'Retry' button is displayed on the screen - upon clicking it the app attempts to fetch data again
3. Upon successful data retrieval(If ever) the button and error message are removed and then the data is displayed

UC13 - View the list of own Festivals

- **Main Stakeholders:** Leader
- **Goal:** See the list of all the Festivals that the Leader created
- **Stakeholders:** Database, Workers, Organisers
- **Conditions:** Must be logged in
- **Event flow description:**

1. The User presses the sandwich button and then taps the 'My Festivals' entry

2. Data is fetched from the Server
3. A new Screen is shown, featuring the list of Festivals - they are divided into 3 categories:
   (a) Active
   (b) Pending
   (c) Completed
4. The Leader can tap on the Festival to open up its details

1.a, 4.a Data not successfully retrieved

1. An error message is displayed informing the User data couldn't be retrieved
2. A 'Retry' button is displayed on the screen - upon clicking it the app attempts to fetch data again
3. Upon successful data retrieval(If ever) the button and error message are removed and then the data is displayed

UC14 - View the Festival details

- **Main Stakeholders:** Leader
- **Goal:** Inspect Festival details and specifics
- **Stakeholders:** Database, Workers
- **Conditions:** Must be logged in, must have selected a Festival
- **Event flow description:**

1. Data is fetched from the Server
2. Festival details are displayed to the User
3. There are 3 buttons the Leader can tap that will take them to a new Activity/Screen
   (a) Events
   (b) Add new events
   (c) Approve organizers

1.a Data not successfully retrieved

1. An error message is displayed informing the User data couldn't be retrieved
2. A 'Retry' button is displayed on the screen - upon clicking it the app attempts to fetch data again
3. Upon successful data retrieval(If ever) the button and error message are removed and then the data is displayed

UC15 - View the list of Festival Events

- **Main Stakeholders:** Leader
- **Goal:** See the list of all the Events that belong to this Festival
- **Stakeholders:** Database, Workers, Organisers
- **Conditions:** Must be logged in, and must have selected a Festival
- **Event flow description:**

1. The User taps the 'Events' button
2. Data is fetched from the Server
3. A new Activity/Screen is shown, featuring the list of Events - they are divided into 3 categories:
   (a) Active
   (b) Pending
   (c) Completed
4. The Leader can tap on the Event to open up its details

1.a, 4.a Data not successfully retrieved

1. An error message is displayed informing the User data couldn't be retrieved
2. A 'Retry' button is displayed on the screen - upon clicking it the app attempts to fetch data again
3. Upon successful data retrieval(If ever) the button and error message are removed and then the data is displayed

UC16 - Create an Event

- **Main Stakeholders:** Leader
- **Goal:** Add a new Event to this
- **Stakeholders:** Database, Leader, Organizers
- **Conditions:** Must be logged in and must have selected a Festival
- **Event flow description:**

1. The User opens the Screen for viewing Festival details, and then taps the 'Add new event' button
2. The data is fetched from the Database
3. The form for creating an Event is shown to the Leader
4. Upon tapping the 'Create' button, the Event is created and added to this Festival

5. New data is sent to the Server and parsed there

1.a Data not successfully retrieved
   1. An error message is displayed informing the User data couldn't be retrieved
   2. A 'Retry' button is displayed on the screen - upon clicking it the app attempts to fetch data again
   3. Upon successful data retrieval(If ever) the button and error message are removed and then the data is displayed

2.a Illegal input
   1. An error message is displayed informing the User that they have entered illegal input
   2. The notification asks the User to re-format and re-enter the input. The expected format and rules are displayed
   3. The steps above are repeated if new input isn't accepted either

4.a Data not successfully sent and/or parsed on the Server
   1. An error message is displayed informing the User of this bad request

UC17 - Bind an Event to an Organizer

- **Main Stakeholders:** Leader
- **Goal:** Bind an Event to an Organizer
- **Stakeholders:** Database, Leader, Organizers, Workers
- **Conditions:** Must be logged in and must have selected a Festival and one of its Events
- **Event flow description:**
   1. The User opens the Screen for viewing Festival details, and then taps the 'Add new event' button
   2. The data is fetched from the Database
   3. The form for creating an Event is shown to the Leader
   4. Upon tapping the 'Create' button, the Event is created and added to this Festival
   5. New data is sent to the Server and parsed there

1.a Data not successfully retrieved
   1. An error message is displayed informing the User data couldn't be retrieved

2. A 'Retry' button is displayed on the screen - upon clicking it the app attempts to fetch data again
3. Upon successful data retrieval(If ever) the button and error message are removed and then the data is displayed

2.a Illegal input

1. An error message is displayed informing the User that they have entered illegal input
2. The notification asks the User to re-format and re-enter the input. The expected format and rules are displayed
3. The steps above are repeated if new input isn't accepted either

4.a Data not successfully sent and/or parsed on the Server

1. An error message is displayed informing the User of this bad request

UC18 - Open a Job to Auctioning

- **Main Stakeholders:** Organizer
- **Goal:** Open this Job up for auctioning
- **Stakeholders:** Database, Leader, Organizers
- **Conditions:** Must be logged in and must have selected a Job
- **Event flow description:**

1. The Organizer selects one of the pending Jobs
2. Data is fetched from the server
3. The Jobs details screen is opened to the Organizer
4. The Organizer presses the 'Auction' button
5. Data is delivered to the Server

1.a Data not successfully retrieved

1. An error message is displayed informing the User data couldn't be retrieved
2. A 'Retry' button is displayed on the screen - upon clicking it the app attempts to fetch data again
3. Upon successful data retrieval(If ever) the button and error message are removed and then the data is displayed

4.a Data not successfully sent and/or parsed on the Server

1. An error message is displayed informing the User of this bad request

UC19 - View the list of own Events

- **Main Stakeholders:** Organizer

- **Goal:** See the list of all the Events that this Organizer is responsible for
- **Stakeholders:** Database, Workers, Leader
- **Conditions:** Must be logged in
- **Event flow description:**

  1. The User taps the 'My Events' button
  2. Data is fetched from the Server
  3. A new Activity/Screen is shown, featuring the list of Events - they are divided into 2 categories:
     (a) Active
     (b) Completed
  4. The Organizer can tap on the Event to open up its details

1.a, 3.a Data not successfully retrieved

  1. An error message is displayed informing the User data couldn't be retrieved
  2. A 'Retry' button is displayed on the screen - upon clicking it the app attempts to fetch data again
  3. Upon successful data retrieval(If ever) the button and error message are removed and then the data is displayed

UC20 - Create Job

- **Main Stakeholders:** Organizer
- **Goal:** Create a new Job
- **Stakeholders:** Database, Leader, Workers
- **Conditions:** Must be logged in, must have selected one of Events
- **Event flow description:**

  1. The User taps the 'New Job' button
  2. The User fills in the form
  3. Upon completion, they tap the 'Create Job Activity'
  4. The updated data is sent to the Server and saved

1.a Illegal input

  1. An error message is displayed informing the User that they have entered illegal input
  2. The notification asks the User to re-format and re-enter the input. The expected format and rules are displayed

3. The steps above are repeated if new input isn't accepted either

3.a Data not successfully sent to the Server

1. An error message is displayed informing the User data wasn't successfully sent to the Server
2. A 'Retry' button is displayed on the screen - upon clicking it the app attempts to send the data again
3. Upon success(if ever) the Screen is closed. Otherwise the aforementioned procedure is again executed

UC21 - Apply for a Festival

- **Main Stakeholders:** Organizer
- **Goal:** Apply for the selected Festival
- **Stakeholders:** Database, Organiser, Leader
- **Conditions:** Must be logged in, the corresponding Screen is opened
- **Event flow description:**

1. The list of Festivals is displayed to the Organizer
2. The Organizer selects one of the Festivals
3. Data is sent to the Server

0.a Data not successfully retrieved

1. An error message is displayed informing the User data couldn't be retrieved
2. A 'Retry' button is displayed on the screen - upon clicking it the app attempts to fetch data again
3. Upon successful data retrieval(If ever) the button and error message are removed and then the data is displayed

2.a Data not successfully sent to the Server

1. An error message is displayed informing the User data wasn't successfully sent to the Server
2. A 'Retry' button is displayed on the screen - upon clicking it the app attempts to send the data again
3. Upon success(if ever) the Screen is closed. Otherwise the aforementioned procedure is again executed

UC22 - View Job Applications

- **Main Stakeholders:** Worker

- **Goal:** View the list of all own Job Applications - and their details and info
- **Stakeholders:** Database, other Workers
- **Conditions:** Must be logged in and have tapped the 'My Applications' button
- **Event flow description:**

1. The list is fetched from the Server
2. The User is taken to a new Screen where all the entries are displayed

0.a Data not successfully retrieved

1. An error message is displayed informing the User data couldn't be retrieved
2. A 'Retry' button is displayed on the screen - upon clicking it the app attempts to fetch data again
3. Upon successful data retrieval(If ever) the button and error message are removed and then the data is displayed

1.a Worker taps on an entry

1. The Worker is taken to a new Screen/Activity
2. Job details and specifics are displayed to him

UC23 - Verify Worker Job Application

- **Main Stakeholders:** Organizer
- **Goal:** Upon inspection, if the Job Application is deemed quality, verify this Application so that it can compete with other Application in the Job Auction
- **Stakeholders:** Database, Worker(s)
- **Conditions:** Must be logged in, must have selected a Job to verify
- **Event flow description:**

1. Upon tapping the 'Auction' button, the Job is added to the Job Auction process
2. Data is sent to the Server

1.a Data not successfully sent to the Server

1. An error message is displayed informing the User data wasn't successfully sent to the Server
2. A 'Retry' button is displayed on the screen - upon clicking it the app attempts to send the data again
3. Upon success(if ever) the Screen is closed. Otherwise the aforementioned procedure is again executed

UC24 - Print Festival Pas

- **Main Stakeholders:** All Users
- **Goal:** Festival Stakeholder can print Cards granting them an entrance to the Festival itself
- **Stakeholders:** Database
- **Conditions:** Must be logged in
- **Event flow description:**

1. The User selects one of the Festivals or Jobs that they're responsible for
2. They download the .pdf of the Pass from the Server

2.a Data not successfully retrieved

1. An error message is displayed informing the User data couldn't be retrieved
2. A 'Retry' button is displayed on the screen - upon clicking it the app attempts to fetch data again
3. Upon successful data retrieval(If ever) the button and error message are removed and then the data is displayed

UC25 - Job Entries Auction

- **Main Stakeholders:** Workers
- **Goal:** Select the Workers according to their Job Entries who will be given the task of performing the Job
- **Stakeholders:** Database, Leader, and Organizer
- **Conditions:** Must be logged in
- **Event flow description:**

1. After a specified period, according to the Job requirements, one or more Workers' entries are selected
2. The Jobs are added to the Worker's list of Jobs to be performed
3. Job is moved from pending to the active category
4. Data is uploaded to the Server

3.a Data not successfully sent to the Server

1. An error message is displayed informing the User data wasn't successfully sent to the Server
2. A 'Retry' button is displayed on the screen - upon clicking it the app attempts to send the data again

3. Upon success(if ever) the Screen is closed. Otherwise the aforementioned procedure is again executed

UC26 - Search Users

- **Main Stakeholders:** Everyone
- **Goal:** Find the desired Users
- **Stakeholders:** All
- **Conditions:** Must be logged in, and have tapped the 'Search' button in the hamburger menu
- **Event flow description:**

1. The User taps the hamburger menu, and then selects the 'Search' option
2. Upon entering a name in the search text field, the Users are filtered
3. The list of Users is displayed to the User

2.a Data not successfully retrieved

1. An error message is displayed informing the User data couldn't be retrieved
2. A 'Retry' button is displayed on the screen - upon clicking it the app attempts to fetch data again
3. Upon successful data retrieval(If ever) the button and error message are removed and then the data is displayed

2.a The User taps an entry in the list

1. The generic profile of that User is opened and displayed to the User

UC27 - Extend Job Auction by 1 day

- **Main Stakeholders:** Organizer
- **Goal:** Extend the Job Auction by 1 day due to various reasons - up to
- **Stakeholders:** Database, Workers
- **Conditions:** Must be logged in
- **Event flow description:**

1. The User taps the button to extend the Auction by 1 day
2. The Auction is extended by 1 day
3. Changes are sent to the Server to be parsed and saved there

3.a Data not successfully sent to the Server

1. An error message is displayed informing the User data wasn't successfully sent to the Server

2. A 'Retry' button is displayed on the screen - upon clicking it the app attempts to send the data again
3. Upon success(if ever) the Screen is closed. Otherwise the aforementioned procedure is again executed

Use Case Diagrams

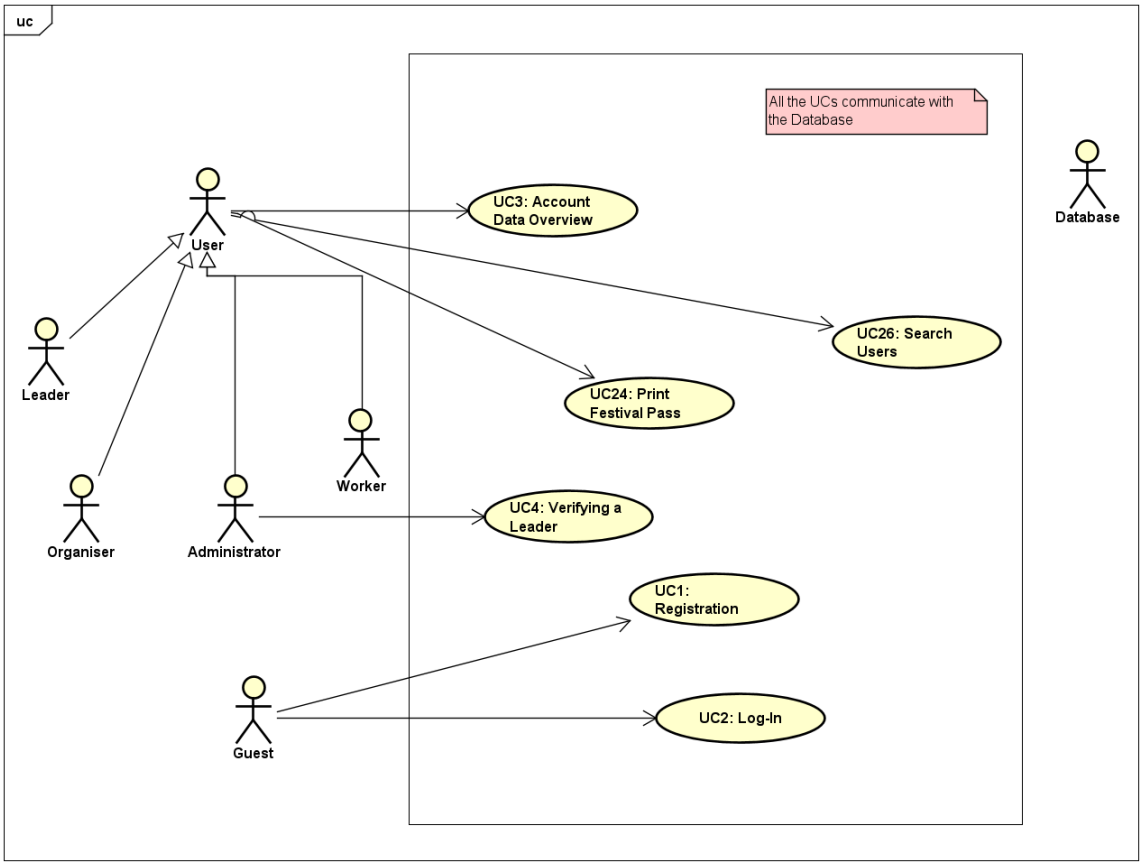


Figure 3.1: Use Case diagram - General Overview

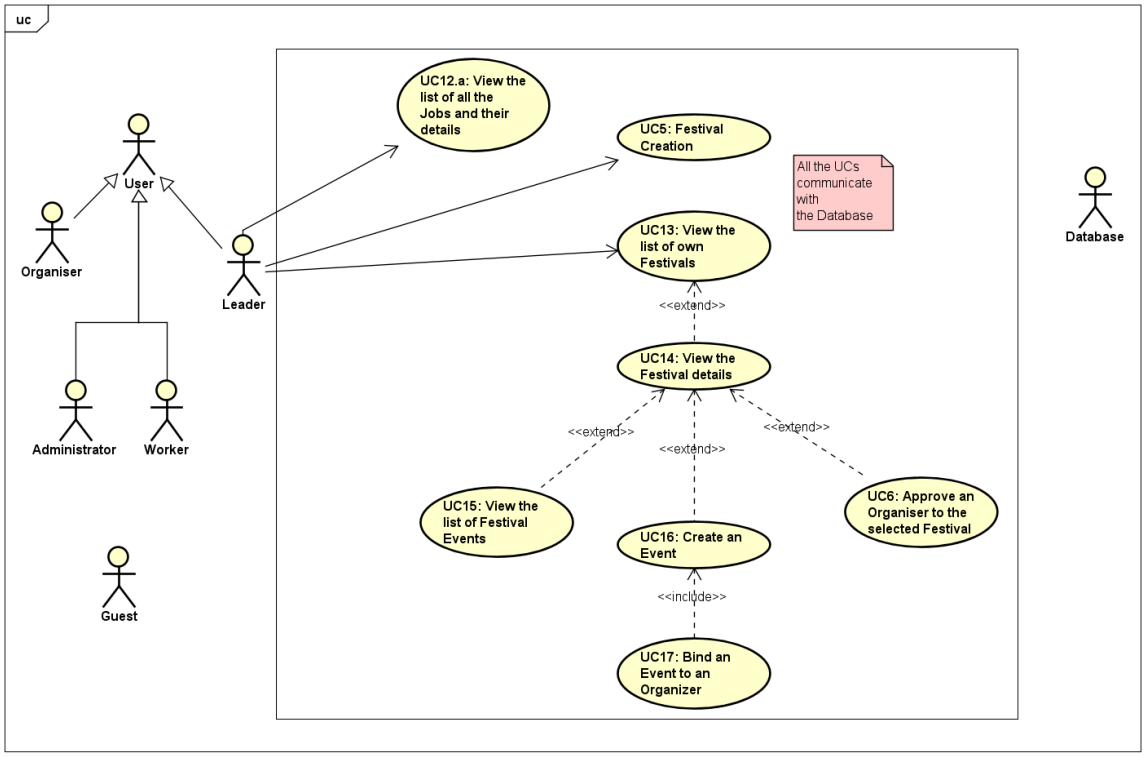


Figure 3.2: Use Case diagram - Leader

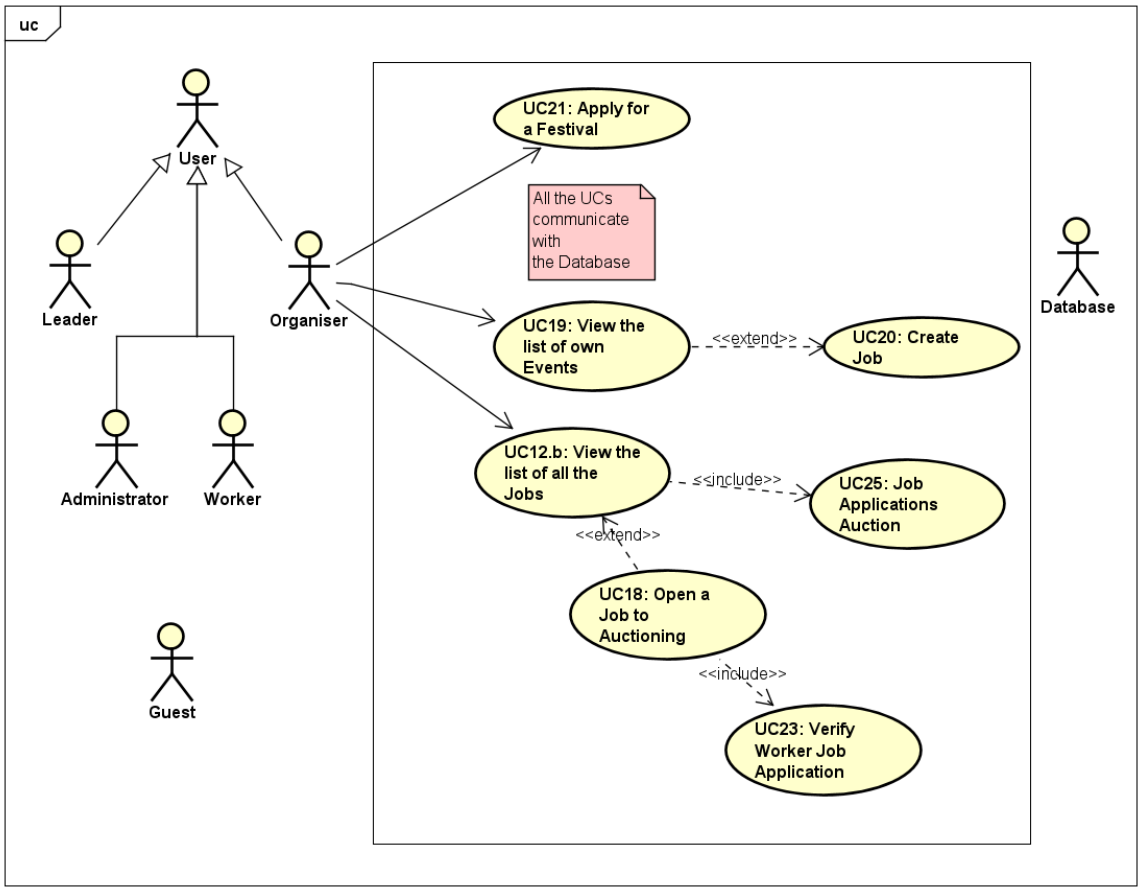


Figure 3.3: Use Case diagram - Organizer

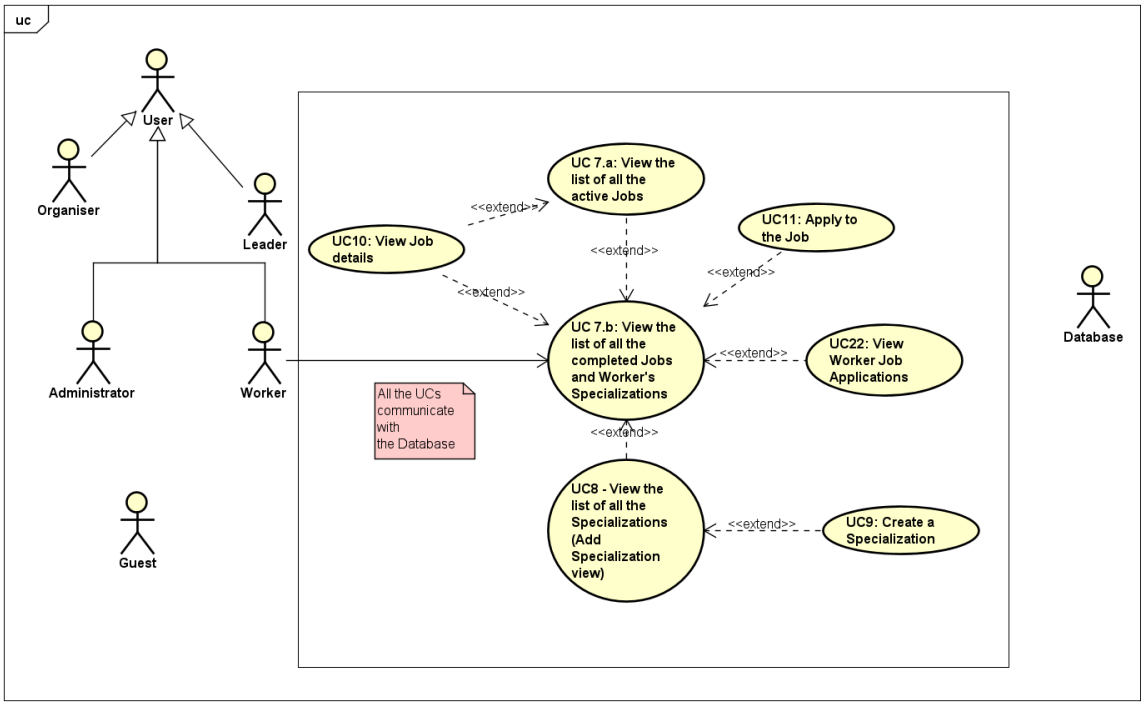


Figure 3.4: Use Case diagram - Worker

3.1.2 Sequence Diagrams

Use Case 8 View the list of all the Specializations This Use Case depicts the Worker's ability to view, search, add, and bing Specializations to his profile. Specializations are skills and professsions that the Worker is proficient at. There is a global list of Specializations. The Worker can either pick from them, or add a new Specialization.

Specializations that the Worker already possesses have a check mark next to them. Upon tapping on a Specialization it will become bound to him, if not already. The corresponding Toast will be displayed - alertign the Worker either of a successful adding, or that he already has that Specialization.

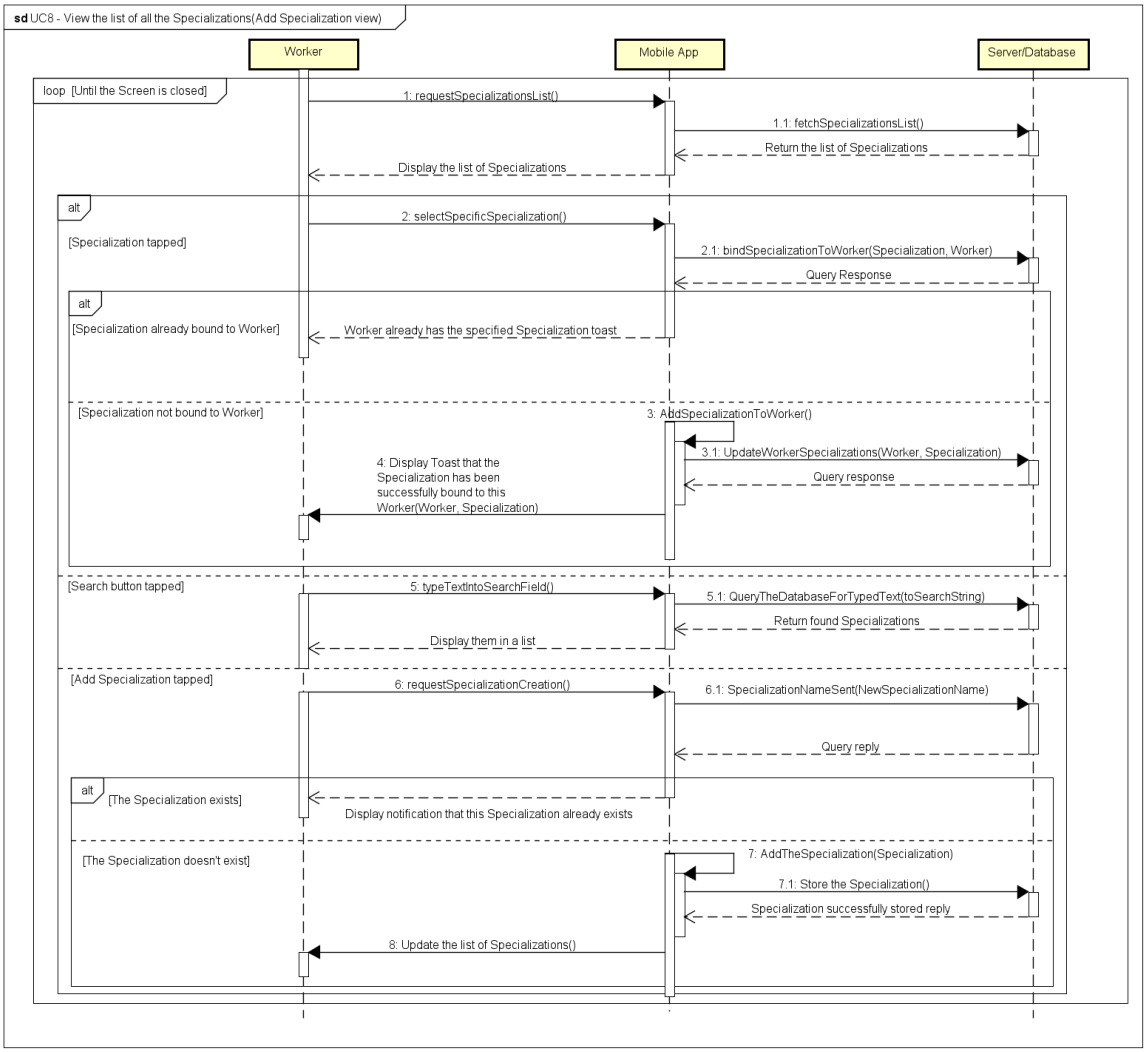


Figure 3.5: Specializations Sequence Diagram

Use Case 13 View the List of own Festivals

Leaders can create Festivals - and then view the list of the Festivals that they have created. They are split into 3 categories:

1. Active - festivals that are currently being carried out
2. Pending - festivals that are yet to start
3. Completed - festivals that have concluded

Upon tapping on these Festivals they can see further details about them. Here it is possible to view this Festival's Events, add new Events, and approve Organizers to this Festival. This UC basically guarantees easy Festival management and overview.

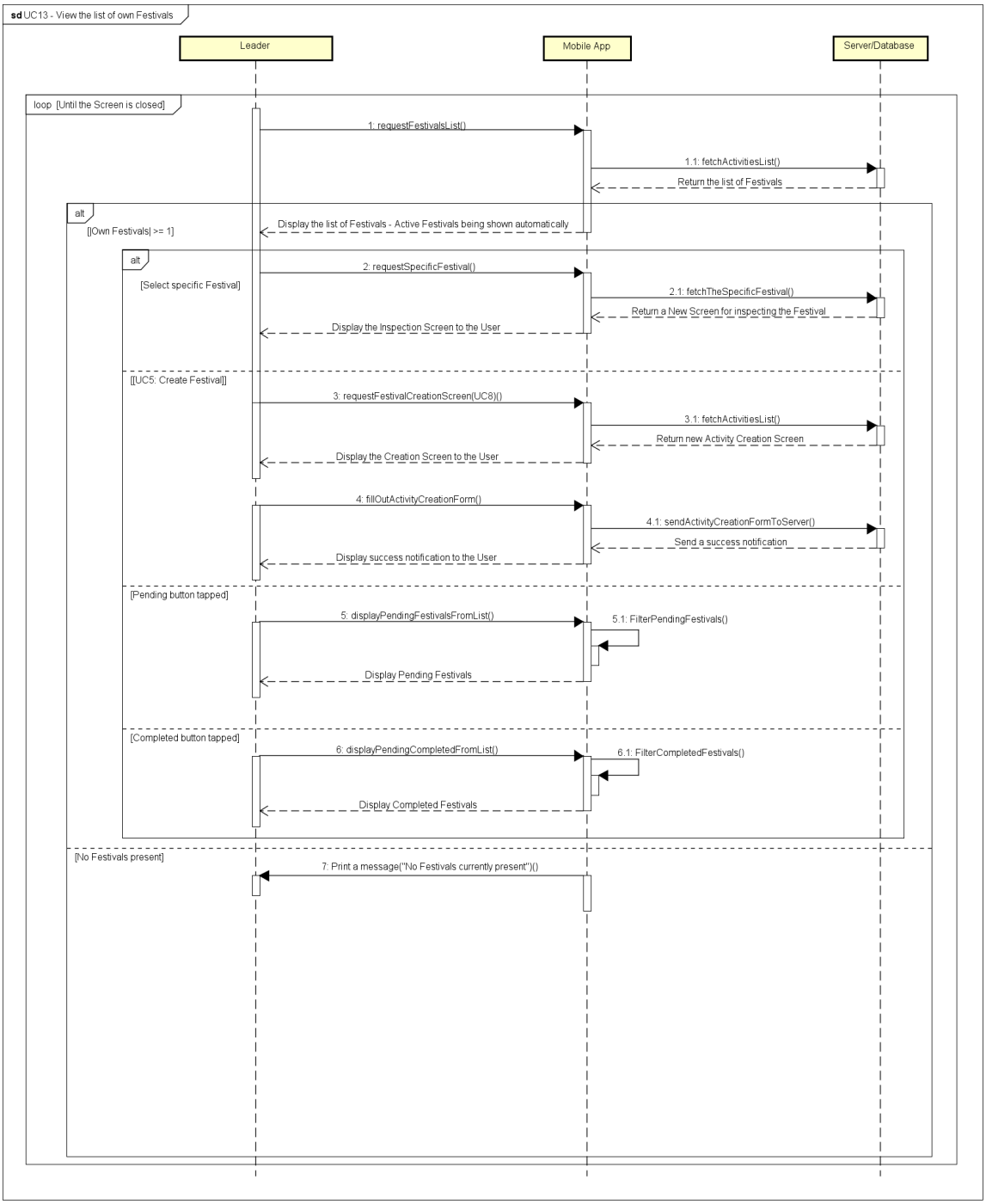


Figure 3.6: Organiser/Leader Festival List Sequence Diagram

3.2 Other requirements

- The System should allow concurrent usage by multiple Users
- The System should use UTF-8 - allow diacritical characters
- The System should use HRK and EUR as currency
- The System should be fast, reponsive and stable
- In case of some instability or error, the System should be able to recover gracefully and allow further usage
- User data should be stored safely and securely, as well as be accordingly encrypted
- The connection to the Server/Database should be reliable and secure
- The system should be easy and intuitive to use - not too complicated
- Implementation - using moder Object-Oriented Programming Languages such as Java or Kotlin
- The System should be as bug resistant as possible

4. Architecture and System Design

System architecture can be divided into three sub-systems :

- Web server
- Android application
- Database

Android application It allows user to view and interact with the specific components of the graphic interface. In Android, graphical components are written in the XML (Extensible Markup Language) which is the language that can be understood by human and by the machine. On interaction with the GUI elements certain parts of the Android code are executed, mostly the ones that send HTTP requests to the Web Server. To send HTTP requests we are using Android's Volley library.

Web Server Web Server is the one who responds to the HTTP requests that are sent by Android application. Primary task of the web server is communication with the client (Android application) and fetching data from the database. Required data is then sent back to the client that displays data on the GUI. Response can be sent in the JSON (JavaScript Object Notation) format or as the plain String. Web server is also the one responsible for authentication and authorization of the user, server checks if the user has permission to access the certain data. If user does not have the required permission, Server will send Error response back to the client whose task is to handle the error properly.

Programming language in which we have coded the Android application is Java, GUI components are written in the XML language and server-side code is written in Python. IDE (Integrated development environment) we are using is Android Studio for the client-side code and IDLE for the server-side code.

Design pattern we are using is MVC(Model-View-Controller). It is a common architectural pattern which is used to design and create interfaces and the structure of an application. This pattern divides the application into three parts that are dependent and connected to each other. These designs are used to distinguish the presentation of data from the way the data is accepted from the user to the data that is being shown. MVC design pattern consists of:

- Model
- View
- Controller

Model The Model component corresponds to all the data-related logic that the user works with. This can represent either the data that is being transferred between the View and Controller components or any other business logic-related data. For example, a Customer object will retrieve the customer information from the database, manipulate it and update it data back to the database or use it to render data. In Android application Model is usually represented as independent Java class that contains data about e.g. Customer. That data is the one that is kept in the data base and displayed in the View component.

View The View component is used for all the UI logic of the application. For example, the Customer view will include all the UI components such as text boxes, dropdowns, etc. that the final user interacts with. In our application an example of View would be Activity for register, user interacts with the View, enters the necessary information that are then transferred to the Model and finally to the database via Controller.

Controller Controllers act as an interface between Model and View components to process all the business logic and incoming requests, manipulate data using the Model component and interact with the Views to render the final output. For example, the Customer controller will handle all the interactions and inputs from the Customer View and update the database using the Customer Model. The same controller will be used to view the Customer data.

4.1 Database

Database used for this project is a relation based database made in SQLite. Relation is usually referred to as a table that has tuples. Tuple is an object that represents an information. Purpose of the database is easy and fast data manipulation, including saving, deleting, updating and sending data to the server. Database has relations:

- User
- Festival
- Event
- Specialization
- WorkerSpec
- Auction
- Application
- Job
- JobSpec
- FestivalOrganizers

4.1.1 Tables details

User has entities for every user of the app. It has all needed personal information about the user and his role.

User		
user_id	INT	User identification number
username	VARCHAR	Unique username
password	VARCHAR	User account password
firstname	VARCHAR	Users first name
lastname	VARCHAR	Users last name
picture	VARCHAR	Profile picture string
phone	VARCHAR	Users phone number
email	VARCHAR	Users email address
role	VARCHAR	Role user will persue in application

Festival contains all needed information about the festival.

Festival		
festival_id	INT	Festival identification number
creator_id	INT	ID of the user who created the festival
name	VARCHAR	Festival name
desc	VARCHAR	Short festival description, can be empty
logo	VARCHAR	Festivals logo string
duration	INTERVAL	Festival duration interval
active	BOOLEAN	True if festival is active, False if it's unactive

Event contains information about the event of the festival.

Event		
event_id	INT	Event identification number
festival_id	INT	ID of the festival that event belongs to
organizer_id	INT	ID of the user who created the event
name	VARCHAR	Event name
desc	VARCHAR	Short event description, can be empty
location	VARCHAR	Events location
start_Time	TIMESTAMP	Event start time

end_Time	TIMESTAMP	Event end time

Specialization contains all different specializations and their names.

Specialization		
specialization_id	INT	Specializations identification number
name	VARCHAR	Name of the specialization

WorkerSpec contains specifications about the user who wants to apply as a worker and his specializations.

WorkerSpec		
worker_id	INT	Workers identification number
specialization_id	INT	Specializations identification number

Auction contains information about auctions.

Auction		
auction_id	INT	Auction identification number
start_Time	TIMESTAMP	Auction start time
end_Time	TIMESTAMP	Auction end time

Application contains information about applications for auctions.

Application		
application_id	INT	Application identification number
auction_id	INT	Auction identification number that worker applies for
worker_id	INT	Workers identification number
price	FLOAT	Offered pay for the job
comment	VARCHAR	Additional comment for application, can be empty
approximate_time	INT	Time needed to complete the job, in days
number_of_people	INT	Number of people that will be doing the job

Job contains information about jobs that had an auction.

Job		
job_id	INT	Job identification number
event_id	INT	Events identification number that job is for
worker_id	INT	Workers identification number that does the job
auction_id	INT	Auctions identification number that auctioned the job
start_Time	DATETIME	Jobs start time
is_Completed	BOOLEAN	True if job is finished, false if it's still active

JobSpec contains specializations that are needed for the job.

JobSpec		
job_id	INT	Job identification number
specialization_id	INT	Specializations identification number

FestivalOrganizers contains information which user, that is also an organizer, applied for which festival.

FestivalOrganizers		
festival_id	INT	Festival identification number
organizer_id	INT	Organizers identification number
status	INT	1 when organizer is waiting on leader, 0 when rejected and 1 when accepted

4.1.2 Database diagrams

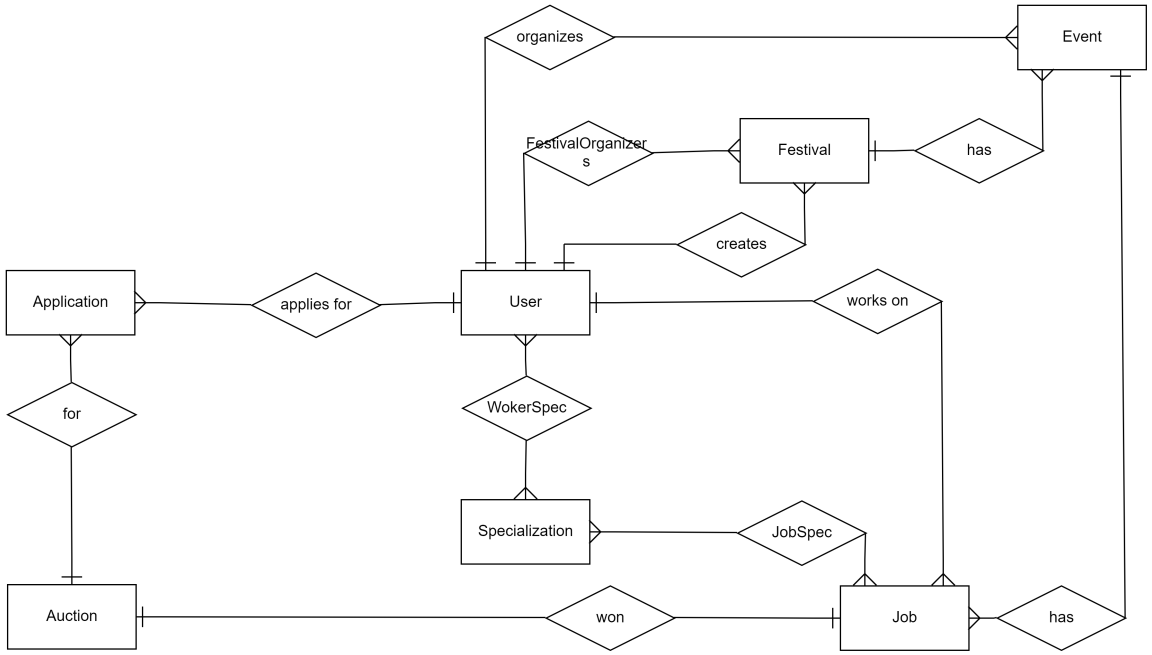


Figure 4.1: E-R Diagram

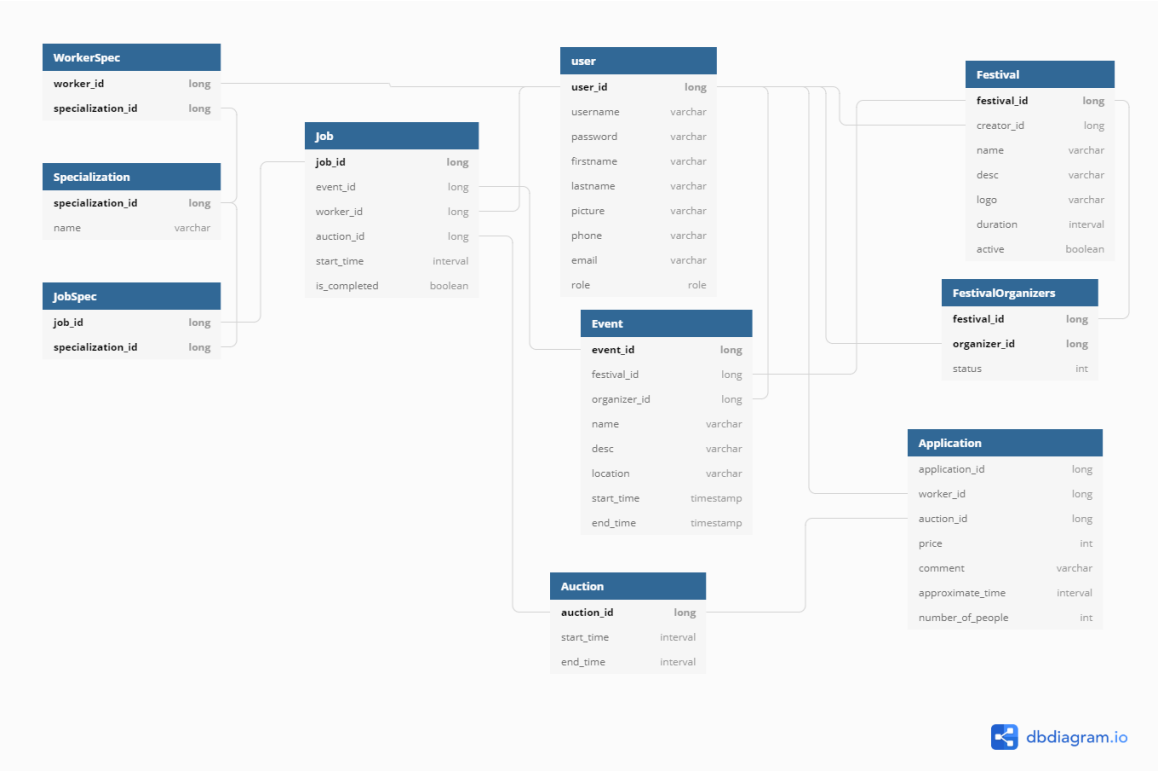


Figure 4.2: Database Diagram

4.2 Class Diagram

Generally, classes can be split into 6 parts:

1. Activity classes
2. Controller classes
3. API Classes
4. Fragment Classes
5. Model Classes
6. Adapter Classes

Controller classes are used for communication between the server and the mobile ”front-end” application. They handle all the logic and data exchange. They are the ones who actually use all the other Classes.

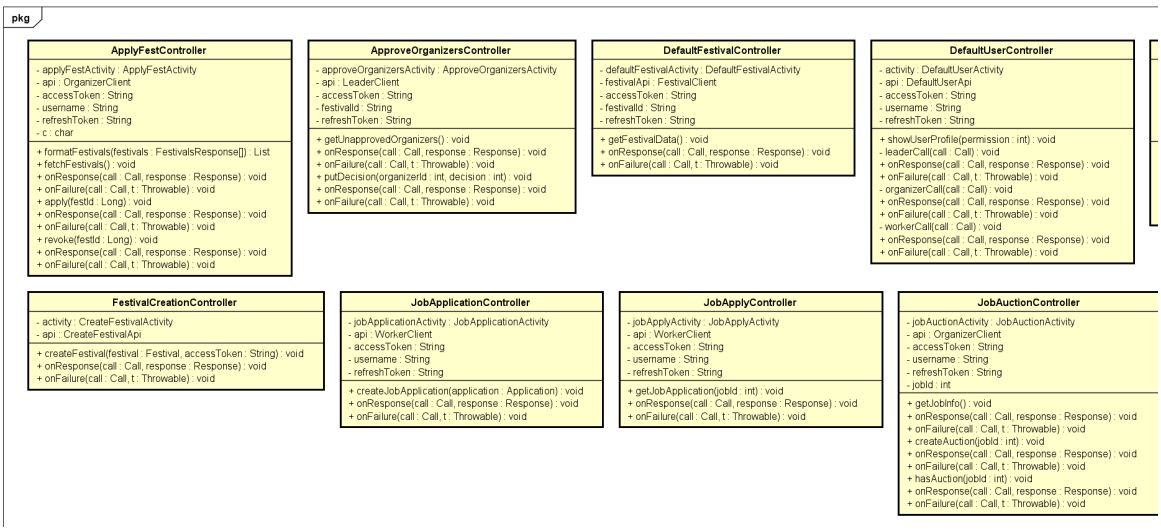


Figure 4.3: Controller Class Diagram 1

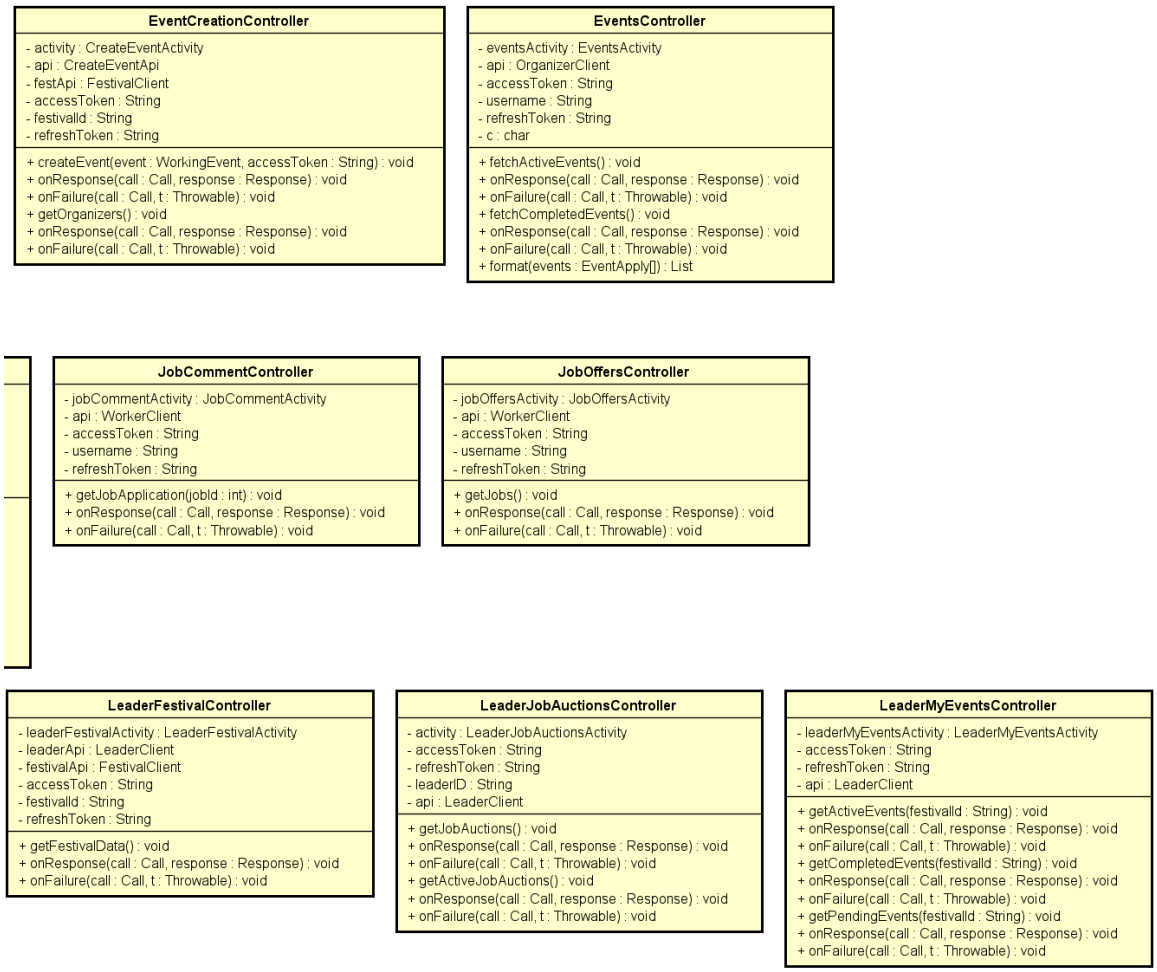


Figure 4.4: Controller Class Diagram 2

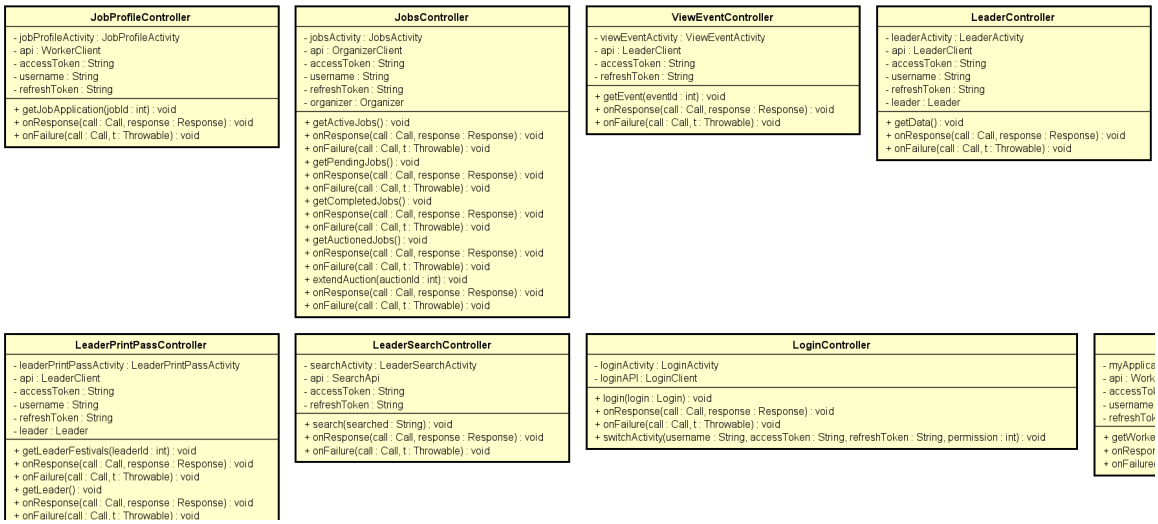


Figure 4.5: Controller Class Diagram 3

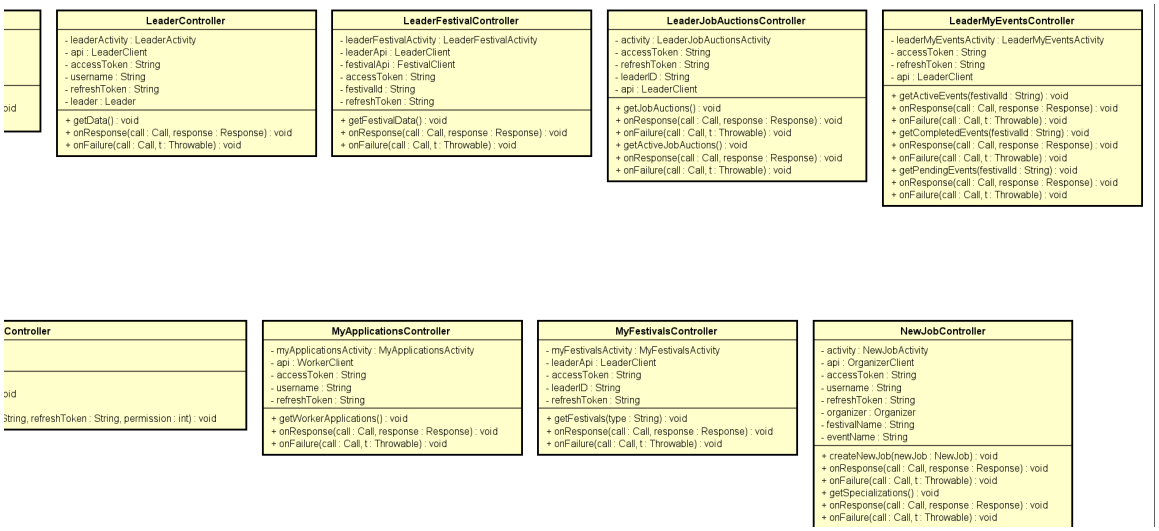


Figure 4.6: Controller Class Diagram 4

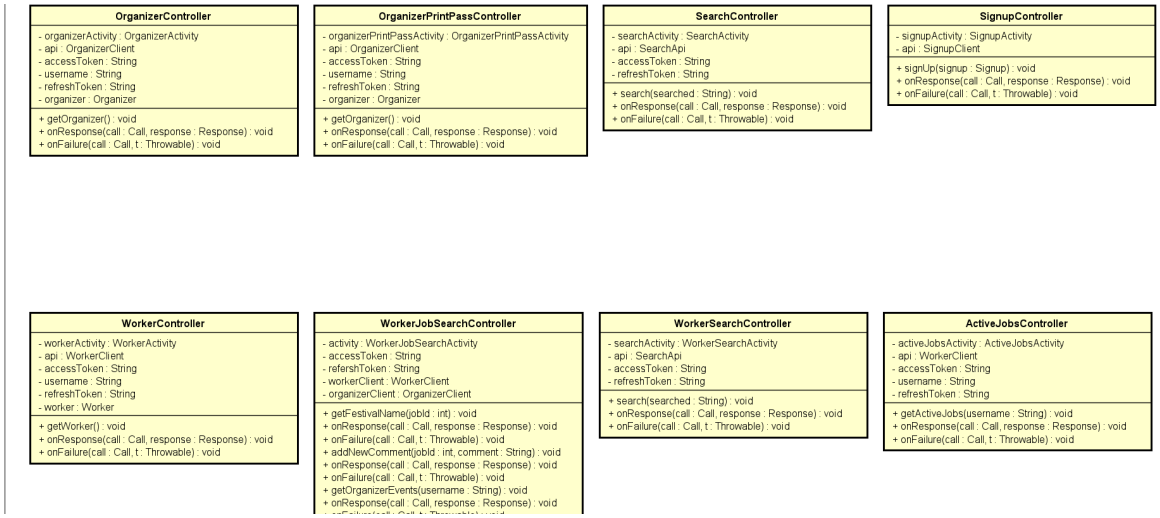


Figure 4.7: Controller Class Diagram 5

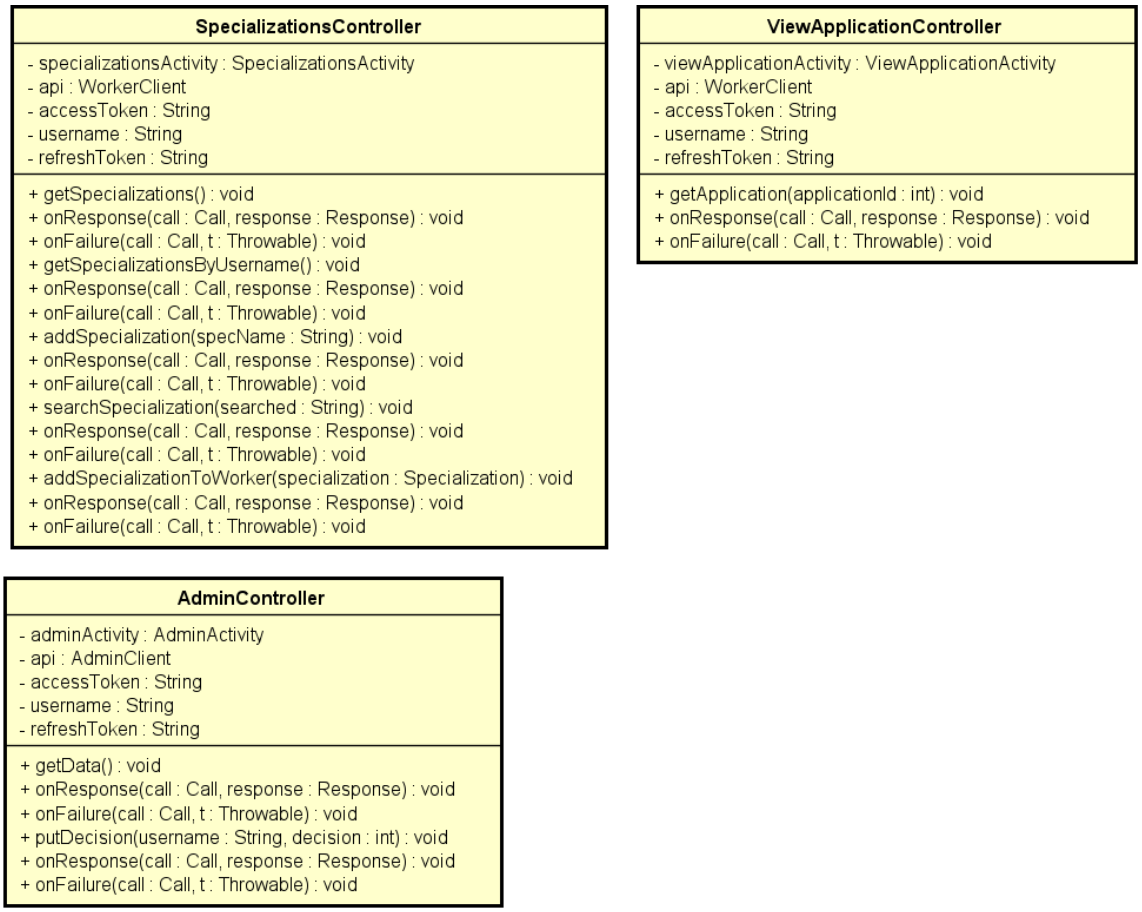


Figure 4.8: Controller Class Diagram 6

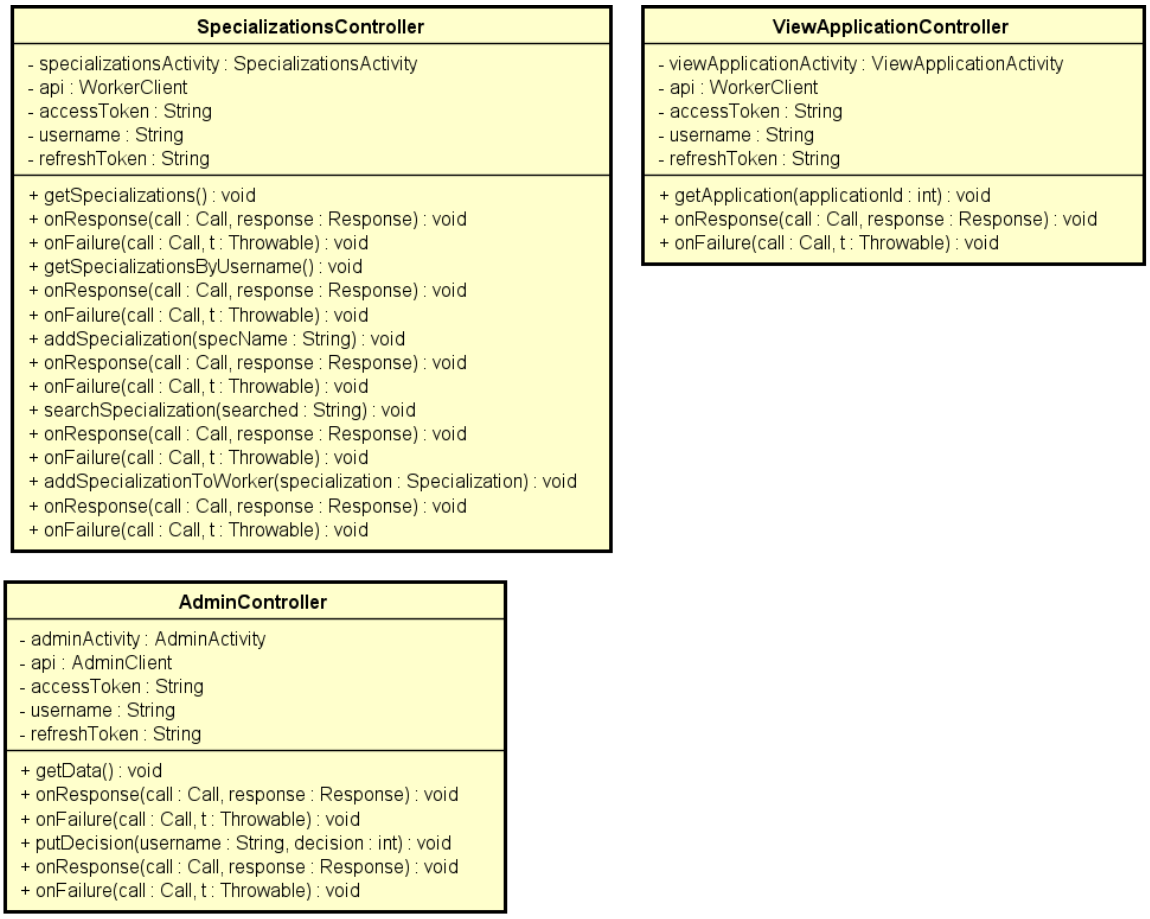


Figure 4.9: Controller Class Diagram 7

The Model Classes are used to hold all the data. Therefore, theya re the intermediary classes used for storing data during data exchanges with the server adn the database. Data is pulled into them, as well as pushed from them into the database. Basically, they represent data storage.

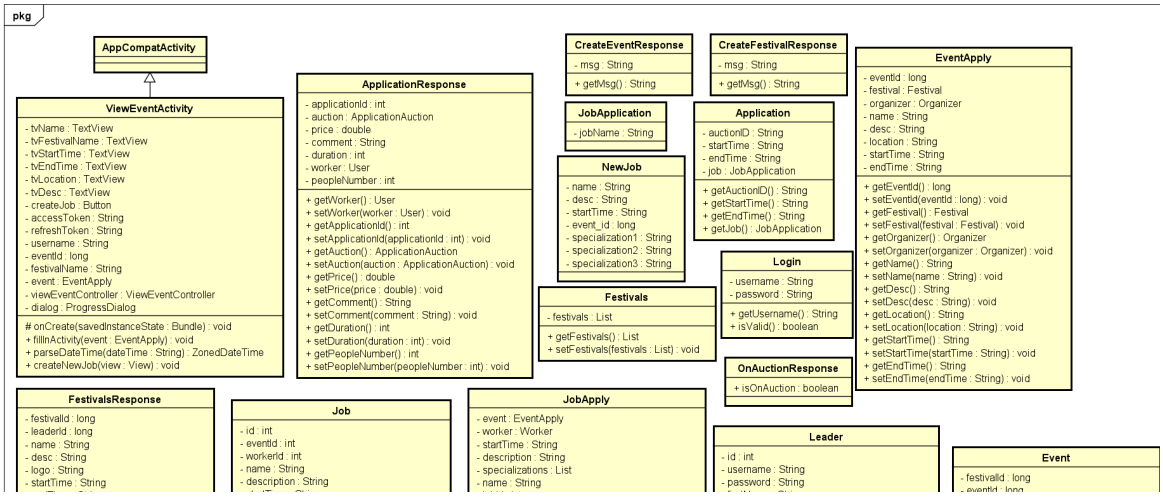


Figure 4.10: Models Class Diagram 1

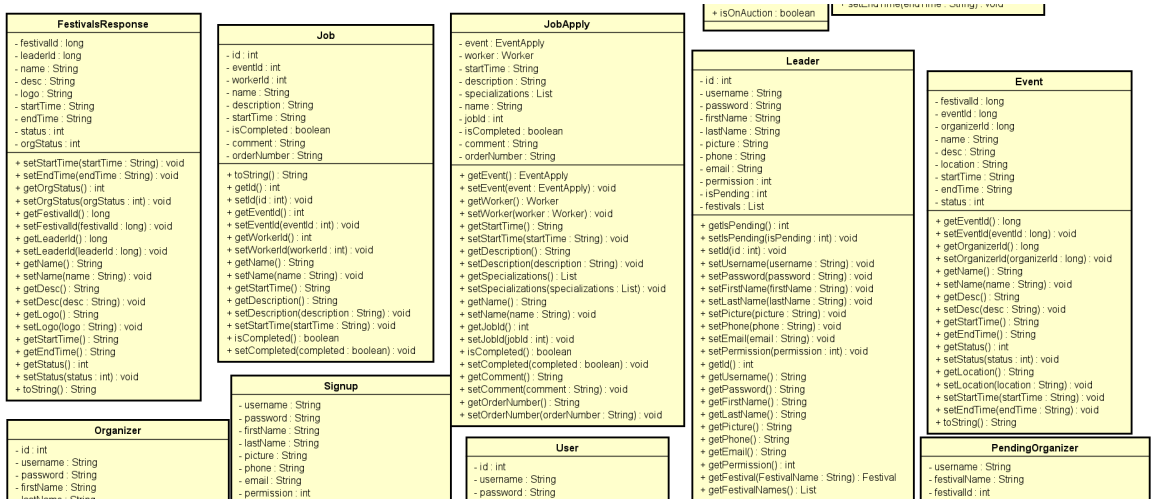


Figure 4.11: Models Class Diagram 2

Figure 4.12: Models Class Diagram 3

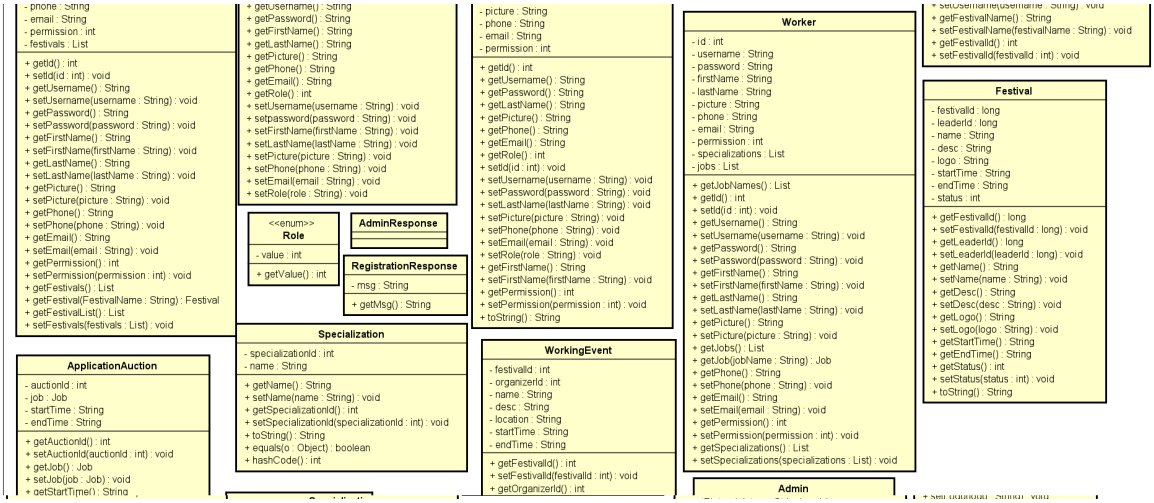


Figure 4.13: Models Class Diagram 4

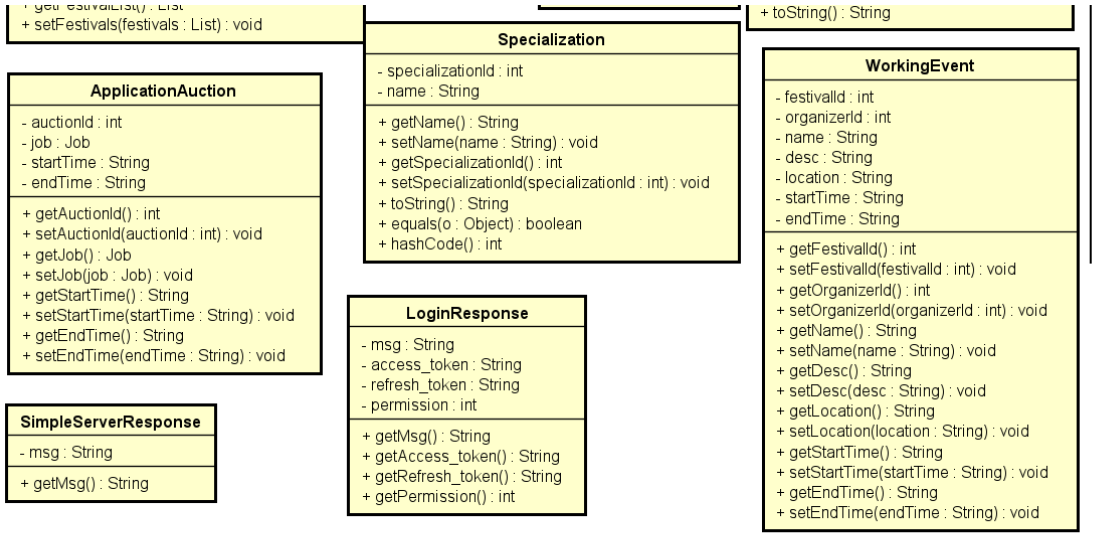
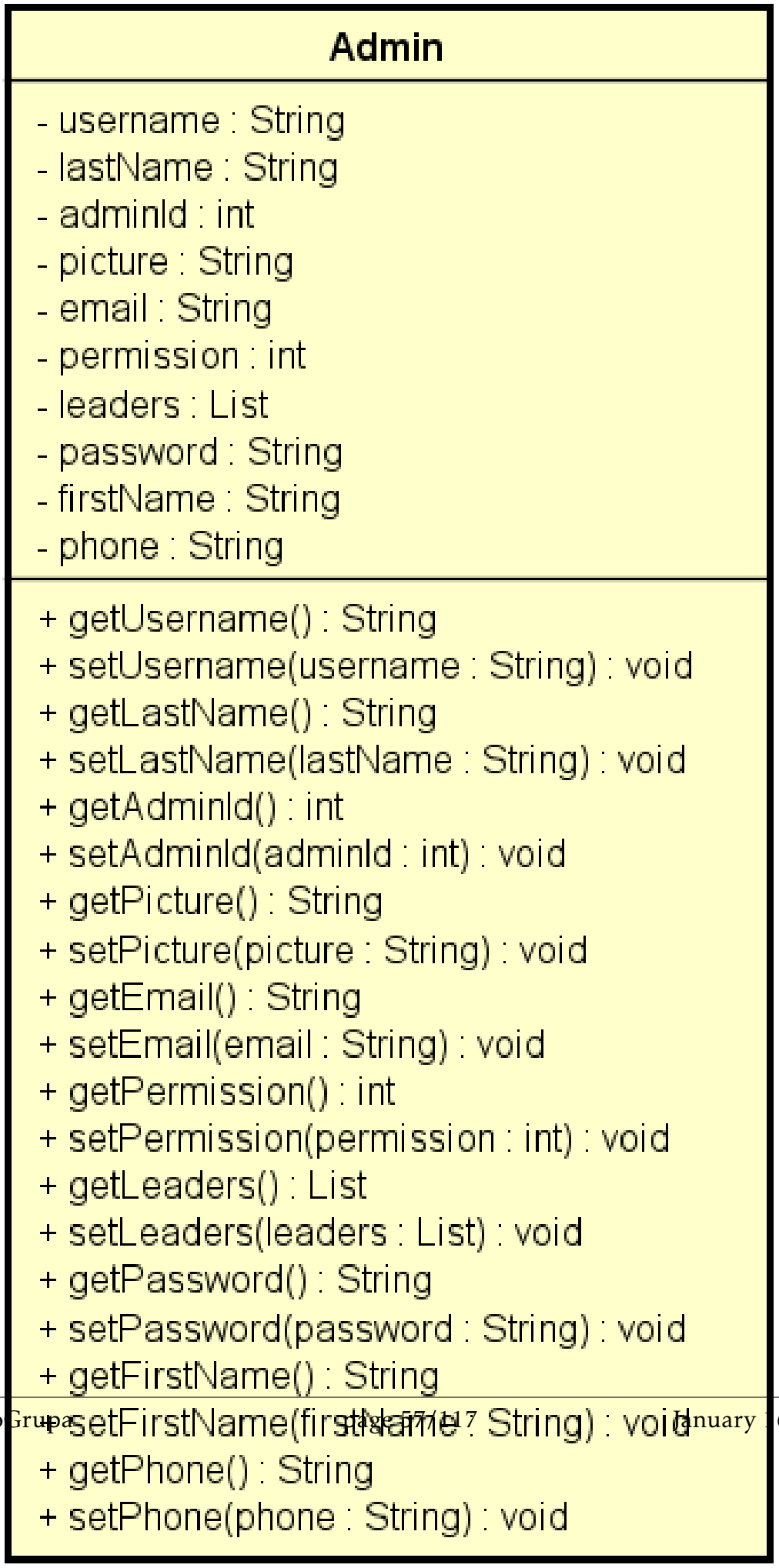


Figure 4.14: Models Class Diagram 5


Admin
- username : String
- lastName : String
- adminId : int
- picture : String
- email : String
- permission : int
- leaders : List
- password : String
- firstName : String
- phone : String
+ getUsername() : String
+ setUsername(username : String) : void
+ getLastName() : String
+ setLastName(lastName : String) : void
+ getAdminId() : int
+ setAdminId(adminId : int) : void
+ getPicture() : String
+ setPicture(picture : String) : void
+ getEmail() : String
+ setEmail(email : String) : void
+ getPermission() : int
+ setPermission(permission : int) : void
+ getLeaders() : List
+ setLeaders(leaders : List) : void
+ getPassword() : String
+ setPassword(password : String) : void
+ getFirstName() : String
+ setFirstName(firstName : String) : void
+ getPhone() : String
+ setPhone(phone : String) : void

API Classes are used for defining the mobile application - server(-¿ database)communication interface, and their diagrams will not be displayed here due to them being just interfaces.

Activity classes are basically the ”front-end” classes of the application. They in cooperation with .XML files represent the static and the dynamic display of pages to the User.

XML classes provide static look, while these Activity classes provide dynamic look, as well as various UI functionalities, transitions, ...

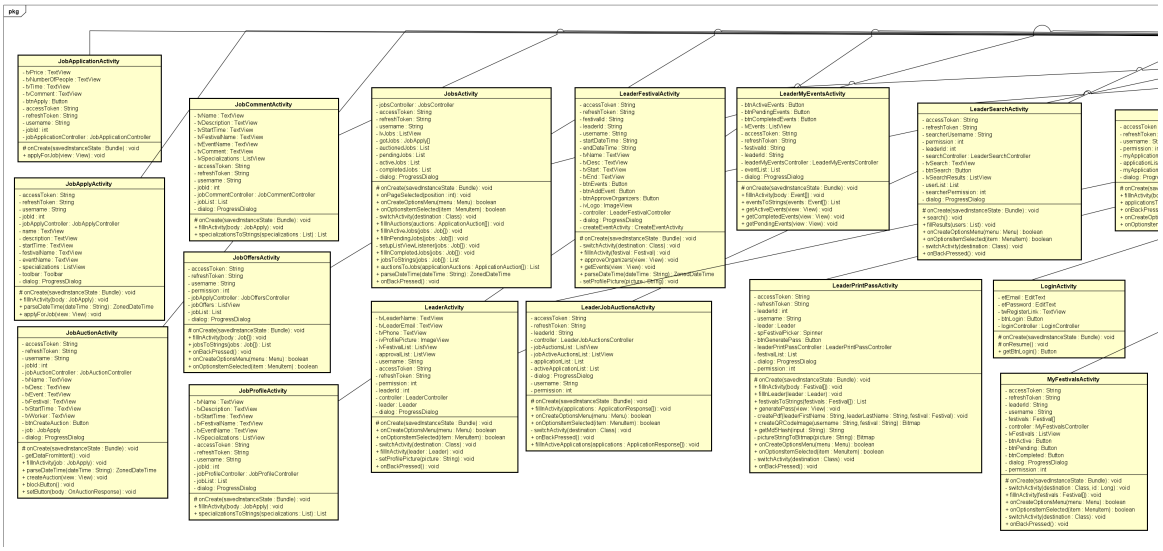


Figure 4.16: Activities Class Diagram 1

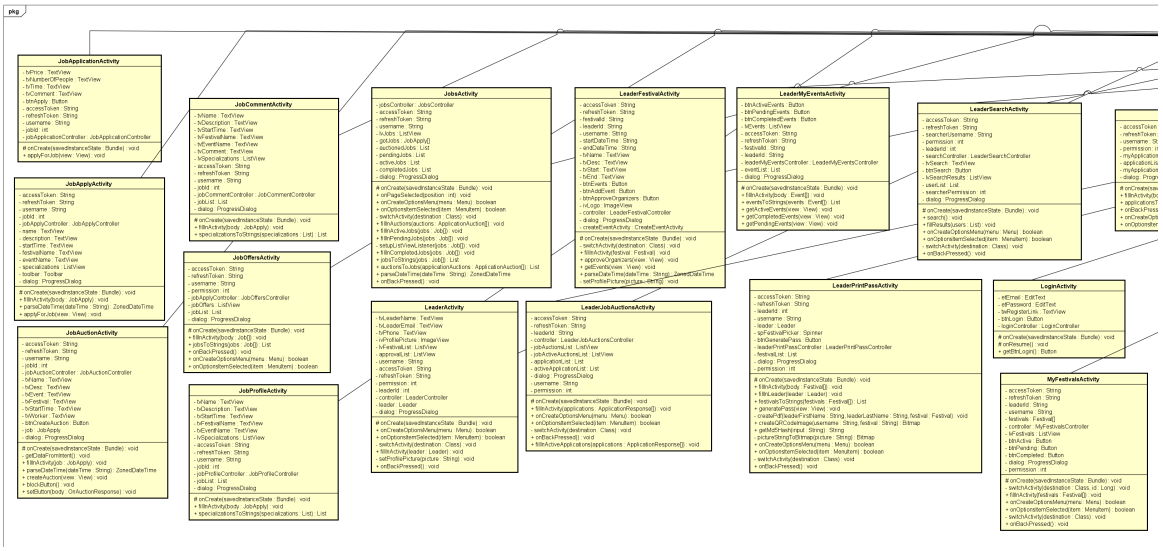


Figure 4.17: Activities Class Diagram 2

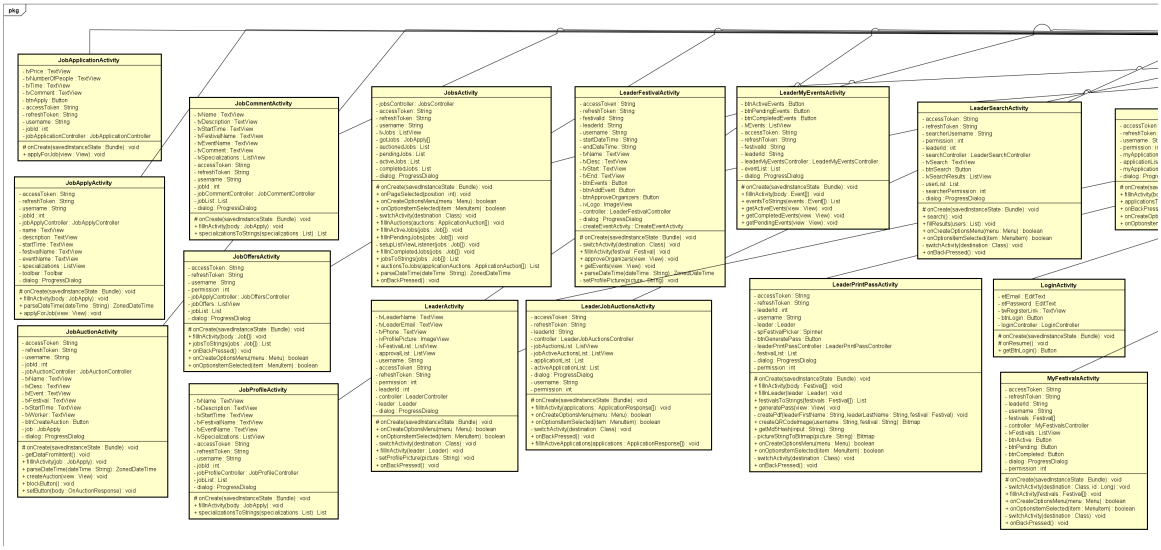


Figure 4.18: Activities Class Diagram 3

Adapter are basically used to hold Controllers, lists, and arrays of data. They are merely a utility class that is necessary for Android Studio to handle these multi-object constructs properly.

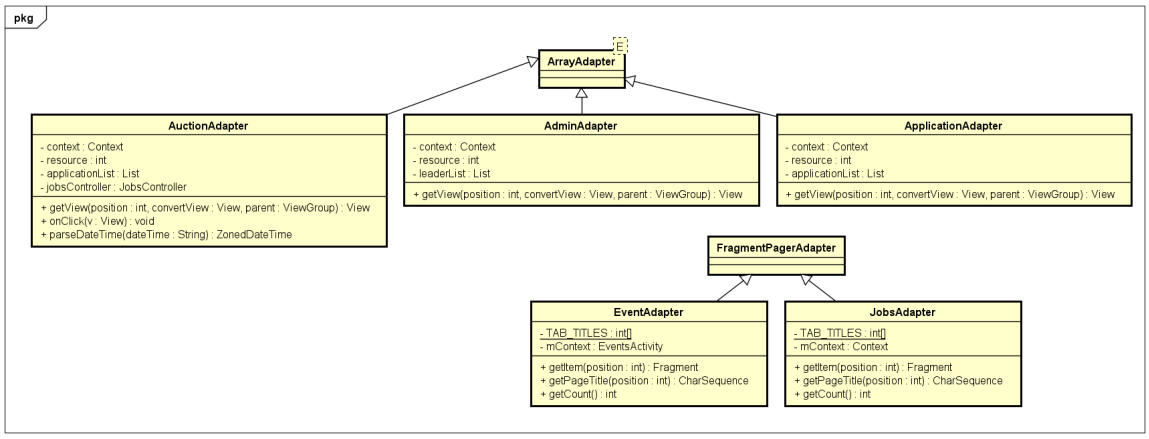


Figure 4.19: Adapters Class Diagram

Finally, Fragments are used for tabbed views inside a single Activity -¿ something like a Sub-Activity, but with an easy and seamless transition.

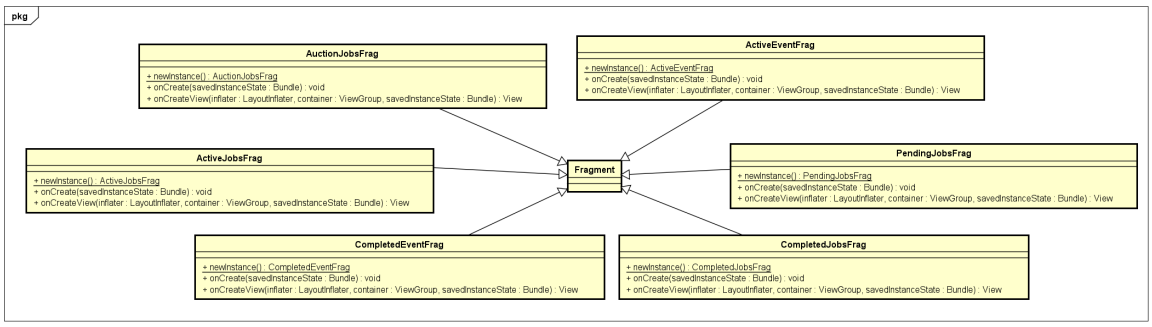


Figure 4.20: Fragments Class Diagram

4.3 State diagram

This diagram displays the abilities and functions of a Leader shown in a state-machine form. The Leader's functionalities and states revolve around Festivals.

To begin with, Leaders can create Festivals. They need to fill out the corresponding form and info. Once a Festival is made, it is added to the list of Festivals.

The Leaders can then view this list of all the Festivals. They are divided into 3 categories:

1. Active - festivals that are currently being carried out
2. Pending - festivals that are yet to start
3. Completed - festivals that have concluded

Second of all, on this Screen they can tap on a Festival to view its details. Here further Festival management is possible:

1. Events - View the list of Events
2. Add new Event - Add new Events to this Festival
3. Approve Organizers - Approve an Organizer to the selected Festival

Furthermore, Leaders can view a list of Job Applications. There all the Job Applications and their details are displayed in a list.

Finally, Leaders have also got the generic functionalities of logging out, printing a Festival Pass and searching Users.

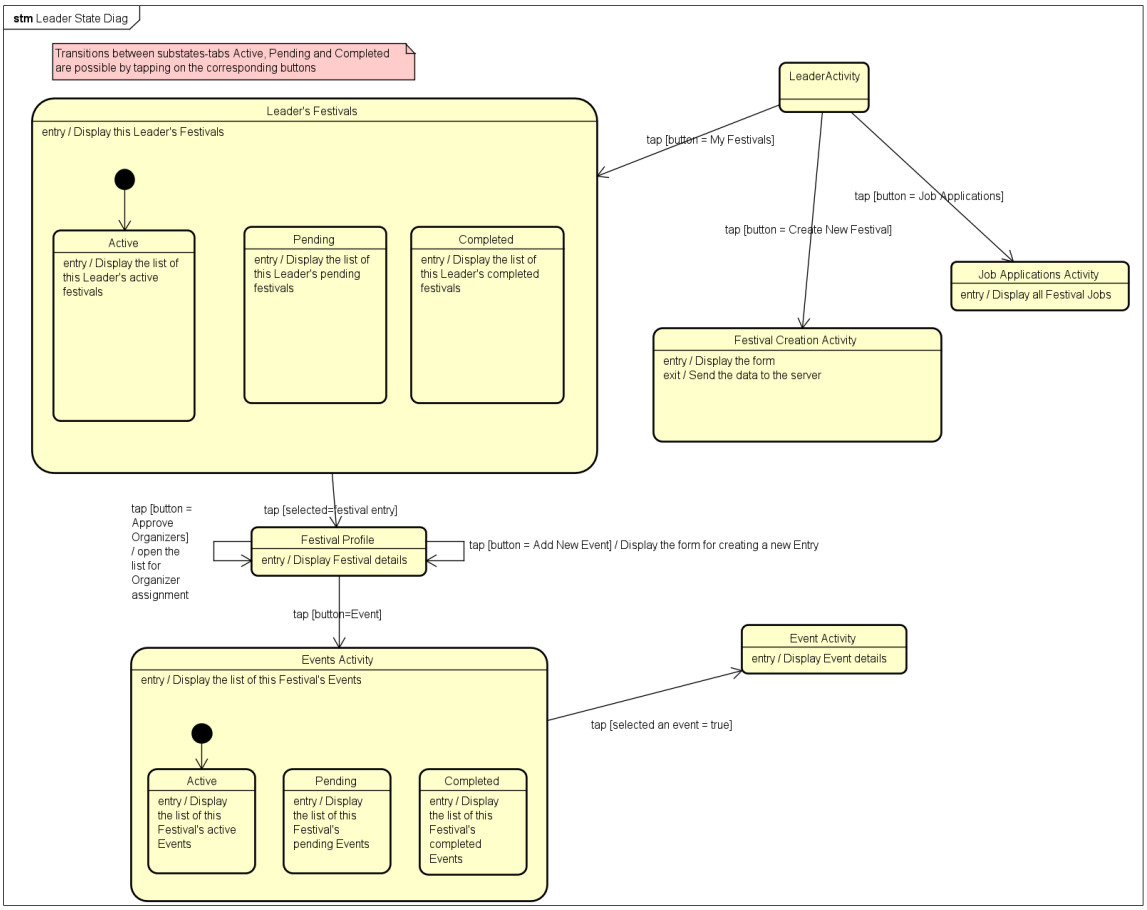


Figure 4.21: Leader State Diagram

4.4 Activity diagram

Two activity diagrams follow. The first one depicts the registration process, while the second one depicts a more complex system interaction.

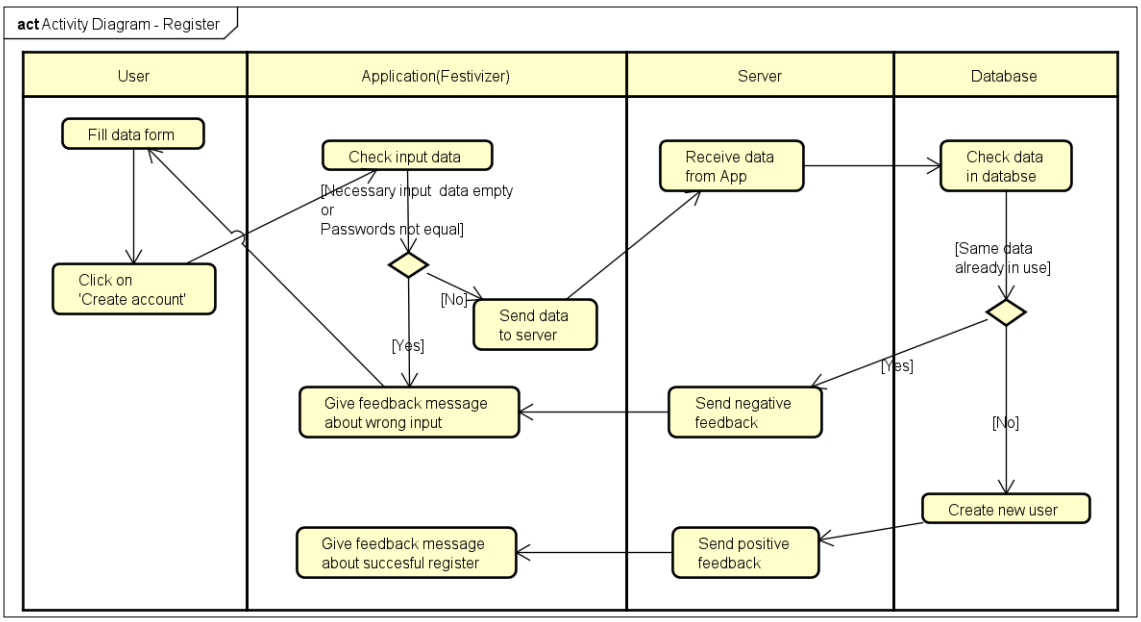


Figure 4.22: Register Activity Diagram

In the second diagram we have a multiple-User interaction. In the beginning, the Organizer logs in and applies to one of the Festivals. Therefore he must select a Festival.

Upon Festival selection he needs to be verified by that Festival's Organizer. When that is done, Organizer can select an Event from a Festival.

The Leader then creates a Job and opens it up to Workers, and finally, one Worker applies to it.

This diagram shows the interaction between logging in, Leaders, Organizers, and Workers. This interaction is shown in a relative time-domain.

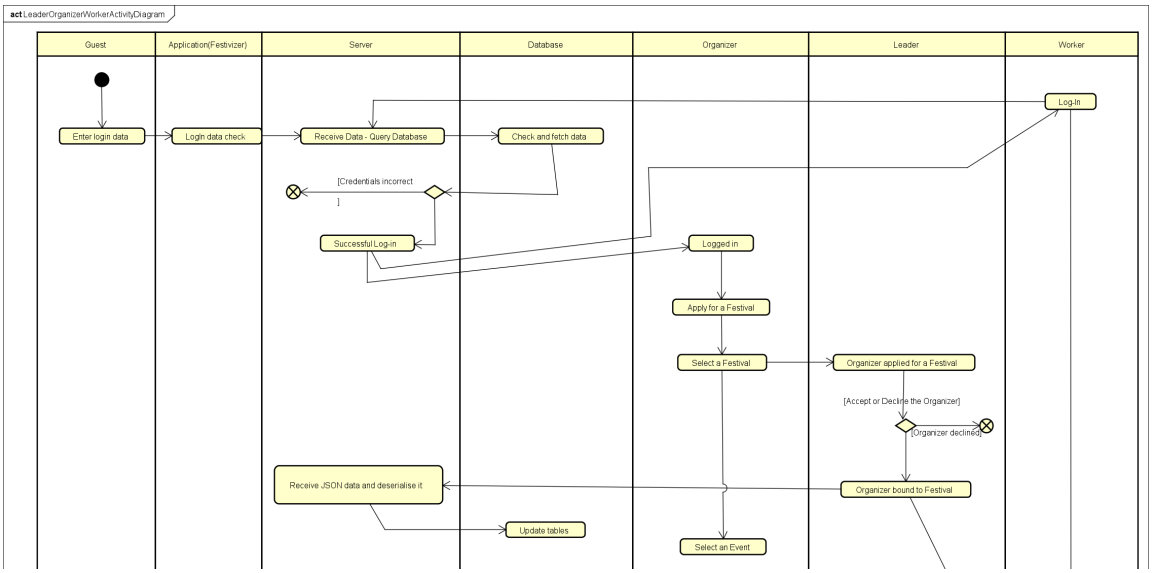


Figure 4.23: Interaction Activity Diagram 1

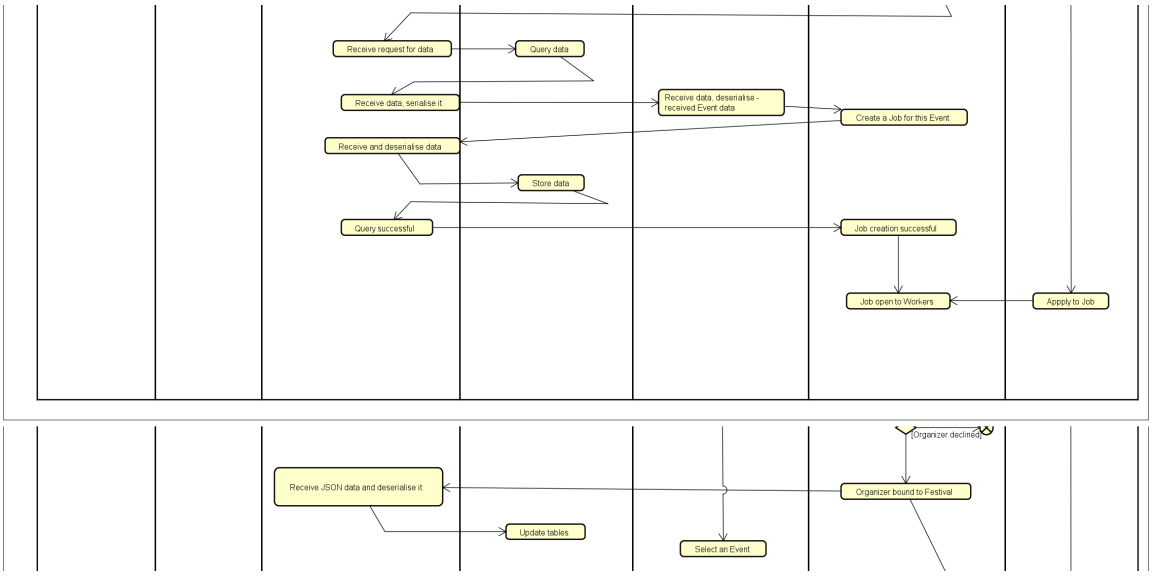


Figure 4.24: Interaction Activity Diagram 2

4.5 Component diagram

The application can so be divided into 2 separate parts - the front-end android application part, and the back-end server part. Even though a mobile application, this division much resembles the usual architecture of web applications.

To begin with, the back-end is the part hosted on the pythonanywhere cloud. It features the python back-end code and the SQLite database. The python script called 'Run.py' is the one which starts the Marshmallow, JWT and SQLAlchemy Python components. From there, other components - 'Models.py' and 'Resources.py' can continue to work. The 'Models.py' scripts basically contains Database models in Python class form, while the 'Resources.py' script handles server requests and responses.

As is visible, the Android application and the server need 2 interfaces - one for requests, and the other one for responses. The Android application is actually split into multiple parts: Controllers, API classes, Activities, and Models. This is due to the implementation of the MVC architecture, where Activities are the Android equivalent of Views. API Classes are the ones that provide the JSON GET, POST, and PUT interface methods for communication with the Server. All the logic is handled in the Controllers. They are the ones that call API methods, fetch, and send data, as well as update the Models and the Activities.

Activities control and manifest the design - they provide dynamic view experience to the user. Design layouts are actually .xml files that were made in Android Studio. Finally, Models are made by controllers, and they resemble the back-end Python models. They are class manifestation of database tables - and are used for holding dynamic data. Data is fetched into them, as well as pulled from them when updating server info.

The adapter class is used for holding multiple instances(lists) of some Models, and the utility class contains many subclasses that were hard to categorise into one of the other classes. Search is used for searching users –¿ found users are displayed using the defaultUser activity/view class.

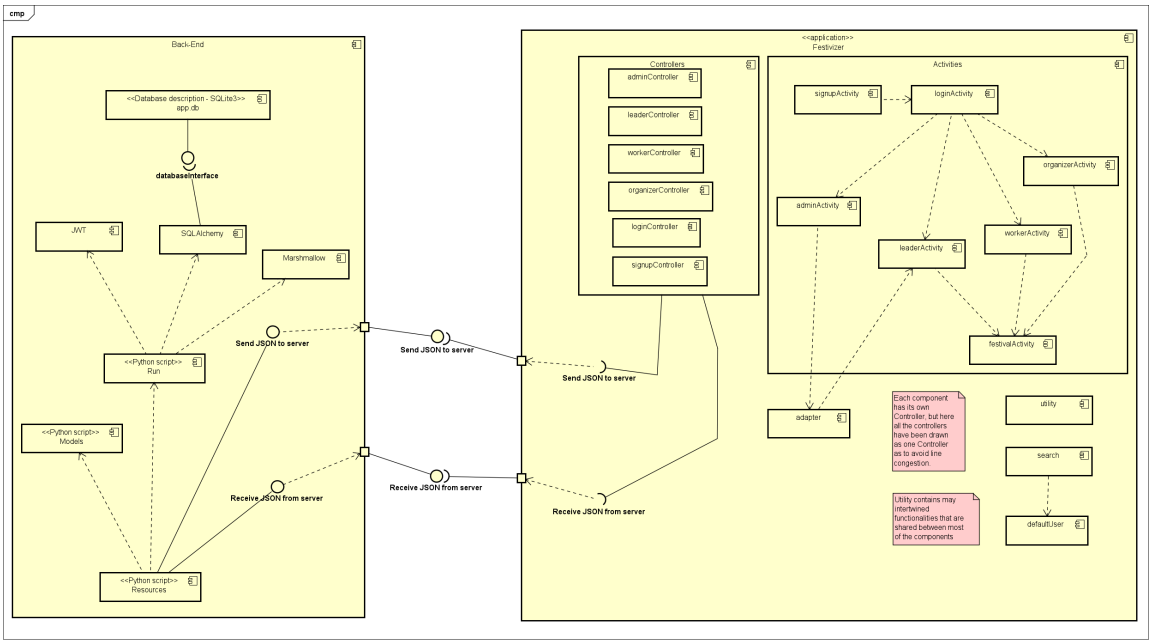


Figure 4.25: Component Diagram

5. Implementation and User Interface

5.1 Used technologies and tools

We have developed a mobile application for the Android operating system. The IDE of choice was Android Studio[1]. We have chosen it because it was developed by Google who are considered to be a sort of authority on Android. On the back-end we used Python[2] hosted on the cloud at the site Pythonanywhere[3].

The database was also hosted on the aforementioned cloud. The database of choice was SQLite3[4]. The pythonanywhere cloud offered the ability to start shell consoles and from there manage the database, as well as edit the Python back-end files. The back-end was made in Python Flask[5] - a micro web framework. The cloud made the deployment and integration seamless and easy.

Flask cooperates with further sub-components(as visible in the component diagram) - SQLAlchemy[6], Flask-JWT[7], and Marshmallow[8]. SqlAlchemy is the component used for communicating with the database, JWT is used for the OAuth 2.0 implementation[9], and Marshmallow is used for JSON serialisation and deserialisation. JWT = JSON Web Token.

Front-end design and logic were all developed in Android Studio which provided a very convenient IDE –¿ code-generation and assistance was of great help, as well as the ability to edit XML files through a GUI instead of only the classical text editing. The application can be deployed and/or debugged on the mobile device by connecting it to the PC/Mac that has Android Studio running with the application code loaded into it. This has provided the basis for smooth application development and testing.

[1] https://developer.android.com/
[2] https://www.python.org/
[3] https://www.pythonanywhere.com/user/kaogrupa/
[4] https://www.sqlite.org/index.html
[5] https://palletsprojects.com/p/flask/
[6] https://www.sqlalchemy.org/
[7] https://pythonhosted.org/Flask-JWT/
[8] https://marshmallow.readthedocs.io/en/stable/
[9] https://oauth.net/2/

Team communication was at first achieved using the Whatsapp application.[10]. This worked at first, but has since proven itself to not provide satisfactory level of organisation and communication features. Therefore it was replaced with Slack[11] - where multiple threads and channels have made organisation and project overview much easier. What's more, a Slack bot and tasks have further increased the productivity and organisation.

The version control system used was Git[12]. The most important branches were: master, dev, and devdoc. The application code was put on the dev branch, while the documentation resided on the devdoc branch. Before turning the application in, branched would be merged to the main branch –¿ the master branch.

The software used for documentation were: LaTeX[13] and Astah UML[14]. For LaTeX, we were provided with a template which greatly helped with LaTeX understanding and documentation writing. Astah UML was used for drawing diagrams and ha proven to be very accessible, intuitive, and satisfying to use.

[10] https://www.whatsapp.com//
[11] https://slack.com/
[12] https://git-scm.com/
[13] https://www.latex-project.org/
[14] http://astah.net/

5.2 Testing

5.2.1 Component Testing - JUnit

We have tested the utility and some core logic classes. One test is made to deliberately fail. The code and the test results follow.

First, the successful test:

```
package com.hfad.organizationofthefestival;

import
    com.hfad.organizationofthefestival.festival.Event.CreateEventActivity;
import com.hfad.organizationofthefestival.festival.Festival;
import com.hfad.organizationofthefestival.leader.Leader;
import com.hfad.organizationofthefestival.login.Login;
import com.hfad.organizationofthefestival.worker.JobApplyActivity;
import com.hfad.organizationofthefestival.worker.JobProfileActivity;
import com.hfad.organizationofthefestival.worker.Specialization;

import org.junit.Test;

import java.lang.reflect.Array;
import java.time.DateTimeException;
import java.time.ZoneId;
import java.time.ZonedDateTime;
import java.time.format.DateTimeParseException;
import java.util.ArrayList;
import java.util.List;

import static org.junit.Assert.*;

/**
* Example local unit test, which will execute on the development machine
    (host).
*
* @see <a href="http://d.android.com/tools/testing">Testing
    documentation</a>
*/
```

```
public class SuccessfulTests {

   Login testLogin = new Login("marko22", "");
   Login goodTestLogin = new Login("validMarko", "goodPass");

   @Test
   public void correctLogin() {

      assertEquals("marko22", testLogin.getUsername());
   }

   @Test
   public void isValidTest() {
      assertFalse(testLogin.isValid());
      assertTrue(goodTestLogin.isValid());
   }

   @Test
   public void specializationsToStringsTest(){
      List<Specialization> specializationsList = new ArrayList<>();

      Specialization toAdd1 = new Specialization("Writing JUnit tests");
      Specialization toAdd2 = new Specialization("Debugging");
      Specialization toAdd3 = new Specialization("Drinking alcoholic
          beverages");

      specializationsList.add(toAdd1);
      specializationsList.add(toAdd2);
      specializationsList.add(toAdd3);

      JobProfileActivity testJPA = new JobProfileActivity();
      List<String> actualStringList =
          testJPA.specializationsToStrings(specializationsList);

      List<String> expectedStringList = new ArrayList<>();
      expectedStringList.add("Writing JUnit tests");
      expectedStringList.add("Debugging");
      expectedStringList.add("Drinking alcoholic beverages");
```

```
        assertEquals(expectedStringList, actualStringList);

        // Let's test a mismatch
        specializationsList.clear();
        toAdd1 = new Specialization("Vacuuming");
        toAdd2 = new Specialization("Floor cleaning");
        toAdd3 = new Specialization("Desk cleaning");
        Specialization toAdd4 = new Specialization("Stage lights
            management");
        Specialization toAdd5 = new Specialization("Music");

        specializationsList.add(toAdd1);
        specializationsList.add(toAdd2);
        specializationsList.add(toAdd3);
        specializationsList.add(toAdd4);
        specializationsList.add(toAdd5);

        actualStringList =
            testJPA.specializationsToStrings(specializationsList);
        assertNotEquals(expectedStringList, actualStringList);
    }

    @Test
    public void parseDateTimeTest(){
        JobApplyActivity JAATest = new JobApplyActivity();

        String testDateTime = "2019.07.07 03:14:22";
        int year = 2019;
        int month = 7;
        int day = 7;
        int hour = 3;
        int minute = 14;
        int second = 22;

        ZonedDateTime actualZonedDateTime = ZonedDateTime.of(year, month,
            day, hour, minute, second, 0, ZoneId.systemDefault());
        ZonedDateTime expectedZonedDateTime =
```

```
        JAATest.parseDateTime(testDateTime);
    assertEquals(expectedZonedDateTime, actualZonedDateTime);

    // Let's try something that wouldn't work
    assertThrows(NumberFormatException.class, () ->
        {JAATest.parseDateTime("WRONG");});

    // Another good example
    testDateTime = "1000.01.01 03:14:22";
    year = 1000;
    month = 1;
    day = 1;
    hour = 3;
    minute = 14;
    second = 22;

    actualZonedDateTime = ZonedDateTime.of(year, month, day, hour,
        minute, second, 0, ZoneId.systemDefault());
    expectedZonedDateTime = JAATest.parseDateTime(testDateTime);
    assertEquals(expectedZonedDateTime, actualZonedDateTime);

    // 30.2 shouldn't exist
    assertThrows(DateTimeException.class, () ->
        {JAATest.parseDateTime("2019.30.02 03:14:22");});

    // Mismatch dates
    testDateTime = "1000.01.02 03:14:22";
    year = 1000;
    month = 1;
    day = 1;
    hour = 3;
    minute = 14;
    second = 22;

    actualZonedDateTime = ZonedDateTime.of(year, month, day, hour,
        minute, second, 0, ZoneId.systemDefault());
    expectedZonedDateTime = JAATest.parseDateTime(testDateTime);
    assertNotEquals(expectedZonedDateTime, actualZonedDateTime);
```

```
}

@Test
public void convertTimeTest(){
   int year = 2019;
   int month = 7;
   int day = 7;
   int hour = 3;
   int minute = 14;
   int second = 22;
   ZonedDateTime expectedZDT = ZonedDateTime.of(
   year, month, day, hour, minute, second, 0, ZoneId.systemDefault()
   );
   // First a proper one

   CreateEventActivity CEATest = new CreateEventActivity();

   String expectedZDTString = "2019-07-07T03:14:00.000+0000";
   String actualZDTString = CEATest.convertTime("03:14", "07.07.2019.");
   assertEquals(expectedZDTString, actualZDTString);

   // Another example - a bit older but still checks out
   expectedZDTString = "0768-01-01T00:00:00.000+0000";
   actualZDTString = CEATest.convertTime("00:00", "01.01.0768.");
   assertEquals(expectedZDTString, actualZDTString);

   // Now let's do a bad example - MARGINAL CASE
   // 3 digit year?
   assertThrows(DateTimeParseException.class, () ->
       {CEATest.convertTime("00:00", "01.01.768");});

   // What about normal years, but single-digit minutes?
   assertThrows(DateTimeParseException.class, () ->
       {CEATest.convertTime("1:00", "01.01.2020");});

   // What about single-digit months or days?
   assertThrows(DateTimeParseException.class, () ->
       {CEATest.convertTime("00:00", "1.01.2021");});
```

```
}

@Test
public void getFestivalTest() {
    Leader testLeader = new Leader(0, "testLeader",
    "123", "Testo", "Testen"
    , "SAMPLE-IMAGE", "09819819"
    , "test@test.com", 1, 0);

    // Let's first test an empty list
    List<Festival> festivalsList = new ArrayList<>();
    testLeader.setFestivals(festivalsList);

    assertEquals(null, testLeader.getFestival("CoolFest"));
    // Now let's add the CoolFest
    Festival coolFest = new Festival("coolFest", "very cool"
    , "SAMPLE-LOGO", "6 oclock", "never"
    );
    festivalsList.add(coolFest);

    assertEquals(null, testLeader.getFestival("wut"));
    assertEquals(coolFest, testLeader.getFestival("coolFest"));

    // Now let's test getFestivalNames method
    // Will need more festivals
    Festival superFest = new Festival("superFest", "very cool"
    , "SAMPLE-LOGO", "6 oclock", "never"
    );
    Festival marinFest = new Festival("marinFest", "very cool"
    , "SAMPLE-LOGO", "6 oclock", "never"
    );
    Festival boatFest = new Festival("boatFest", "very cool"
    , "SAMPLE-LOGO", "6 oclock", "never"
    );

    festivalsList.add(superFest);
    festivalsList.add(marinFest);
    festivalsList.add(boatFest);
```

```
    List<String> expectedFestNames = new ArrayList<>();
    expectedFestNames.add("coolFest");
    expectedFestNames.add("superFest");
    expectedFestNames.add("marinFest");
    expectedFestNames.add("boatFest");
    assertEquals(expectedFestNames, testLeader.getFestivalNames());
  }
}
```

And now the test that deliberately fails!

```
package com.hfad.organizationofthefestival;

import com.hfad.organizationofthefestival.festival.Festival;
import com.hfad.organizationofthefestival.worker.Specialization;

import org.junit.Test;

import static org.junit.Assert.assertThrows;

public class FailingTest {

@Test
public void emptyTests(){
  // Can we make a no-name specialization?
    assertThrows(IllegalArgumentException.class, () -> {
      Specialization emptySpec = new Specialization("");
    });

  // Can we make empty Festival?
    assertThrows(IllegalArgumentException.class, () -> {
      Festival boatFest = new Festival("", "very cool"
        , "SAMPLE-LOGO", "6 oclock", "never");
    });
  }
}
```

Their results:

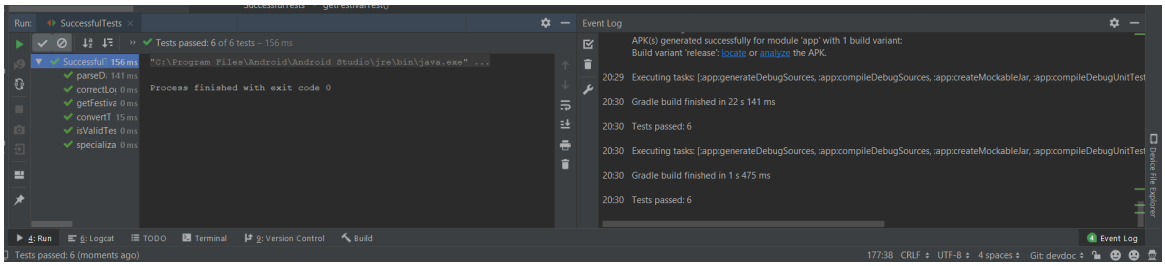


Figure 5.1: Successful tests

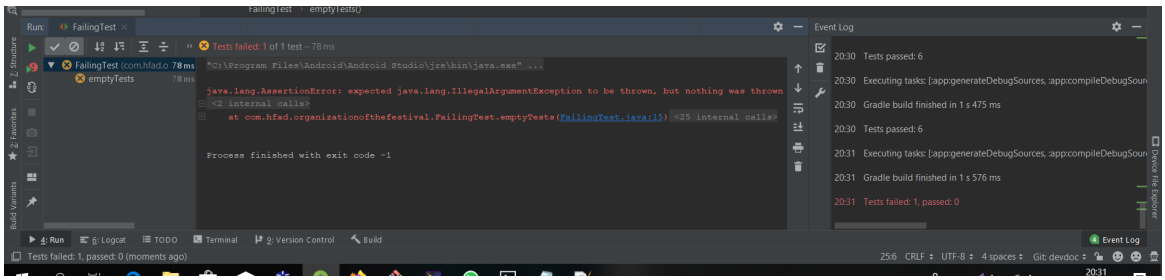


Figure 5.2: Failing tests

5.2.2 System Testing

Test Scenario 1: Leader register and festival creation test Input:

1. Tap 'No account yet? Create one'.
2. Input user data and chose 'Leader' from drop-down list, and Tap 'Create account'.
3. Tap 'Already a member? Login'.
4. Input admin user data, tap 'Login', and tap 'Accept' next to created leader, go back on login screen.
5. Input created leader user data, tap 'Login', tap 'Create new festival', input festival data, tap 'Create'.

Expected output:

1. Opened register screen
2. Get message about successful account creation.
3. Opened login screen
4. Get message about successfully accepted leader.
5. Get message about successfully created festival.

Actual result:

1. Opened register screen.
2. Get message about successful account creation.
3. Opened login screen.
4. Get message about successfully accepted leader.
5. Get message about successfully created festival.

Test Scenario 2 - Leader search test Input:

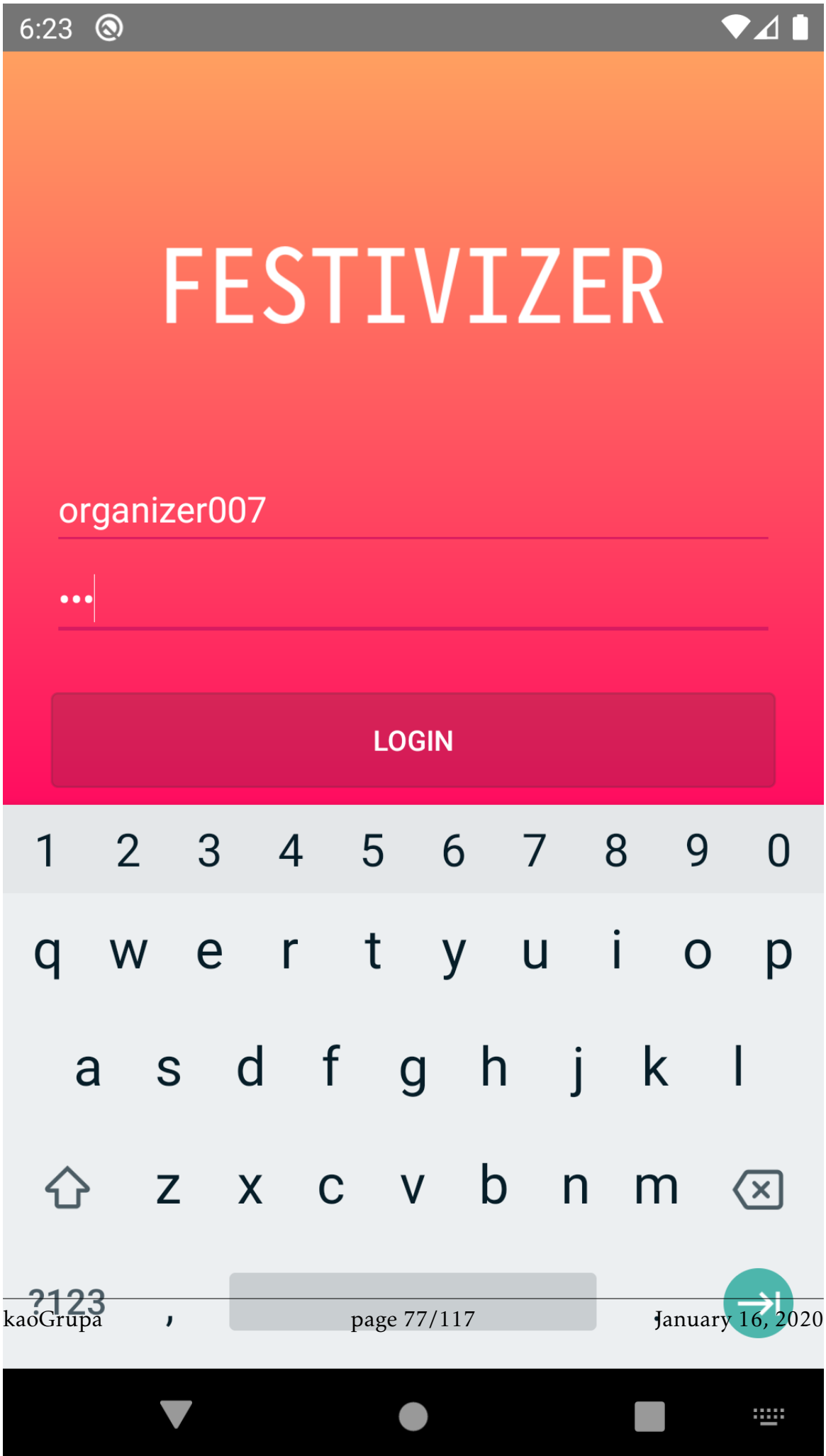
1. Input organizer user data, tap 'Login'
2. Tap 'Search'.
3. Leave empty field, tap 'Search'.
4. Input 'leader007', tap 'Search'.

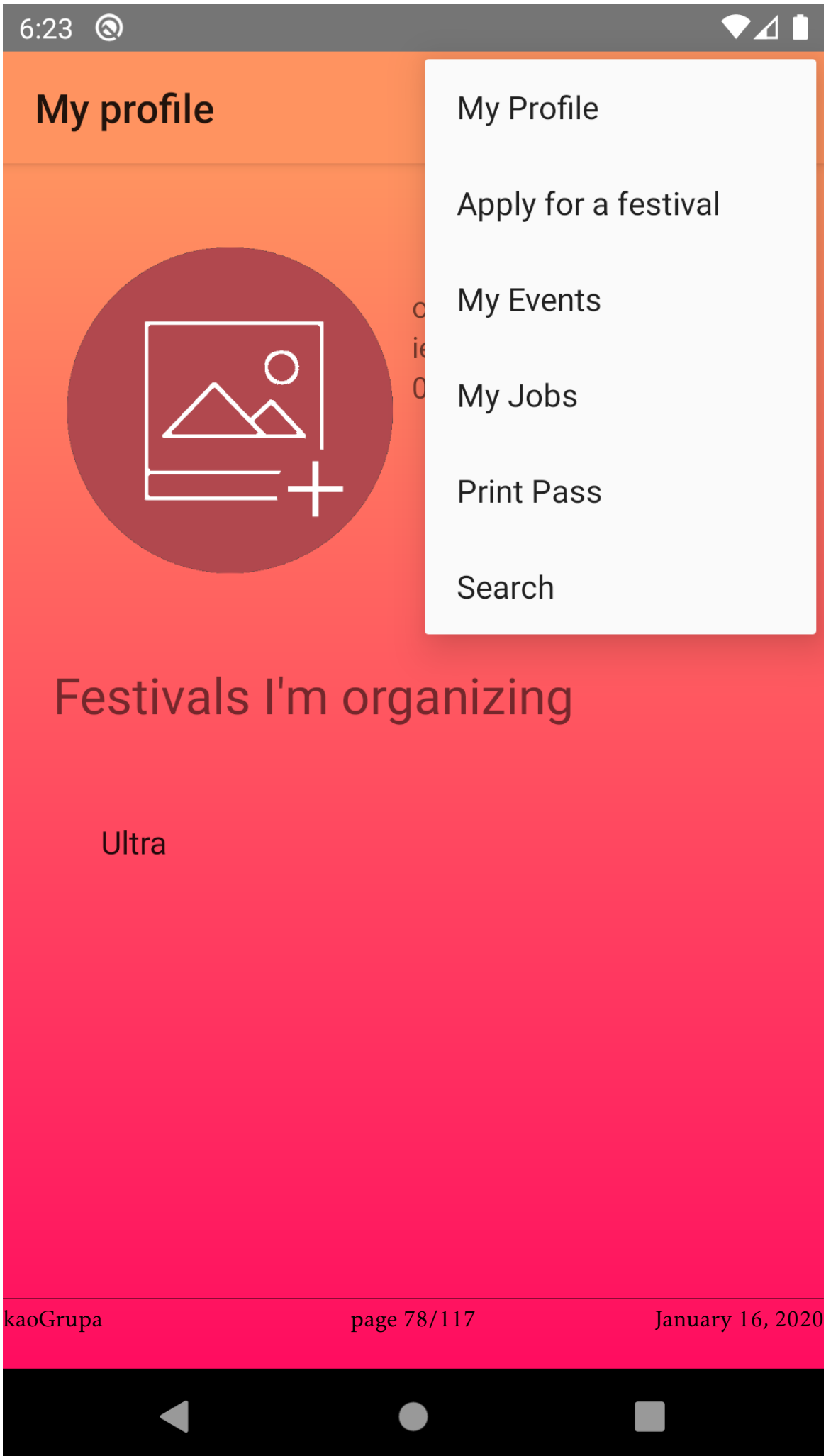
Expected output:

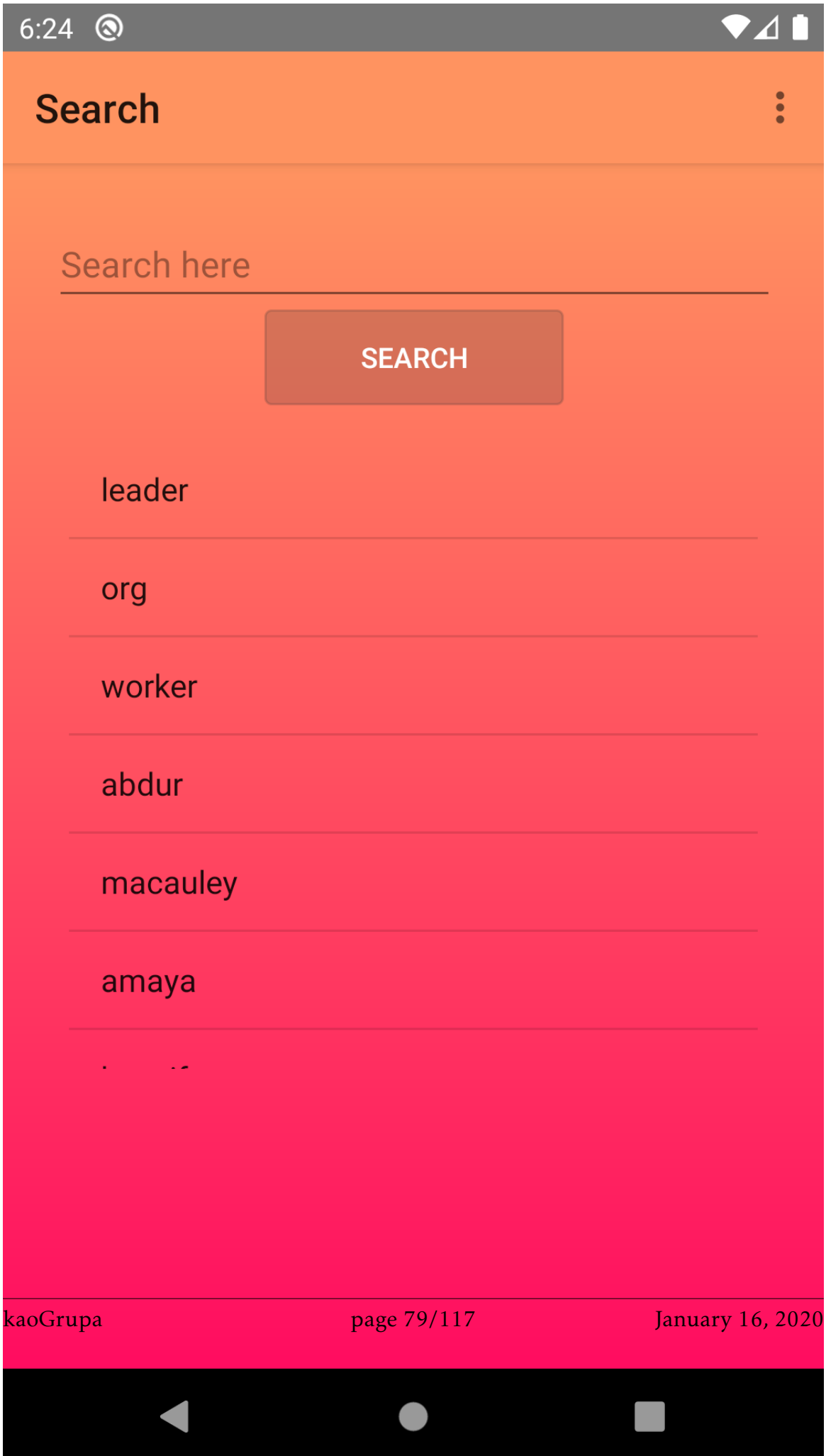
1. Login as organizer.
2. Open search screen.
3. List all users.
4. List only user 'leader007'.

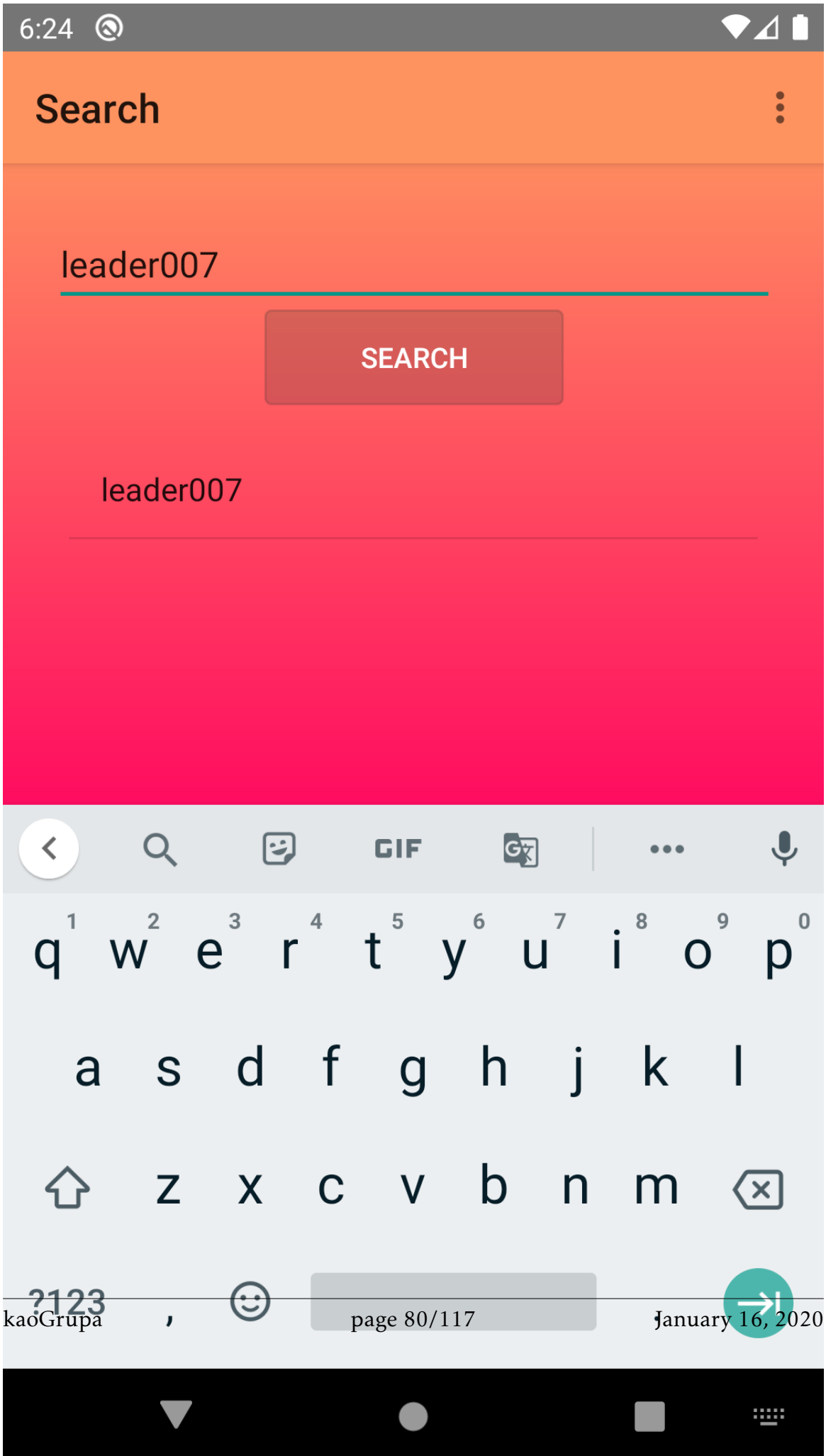
Actual result:

1. Login as organizer.
2. Open search screen.
3. List all users.
4. List only user 'leader007'.


6:23
FESTIVIZER
organizer007
LOGIN


6:23
My profile
My Profile
Apply for a festival
My Events
My Jobs
Print Pass
Search
Festivals I'm organizing
Ultra


6:24
Search
Search here
SEARCH
leader
org
worker
abdur
macauley
amaya


6:24
Search
leader007
SEARCH
leader007
GIF

Test Scenario 3 - Organizer job creation test Input:

1. Input organizer user data, tap 'Login'.
2. Tap 'My events', tap some active event, tap 'New Job'.
3. Input required data, tap 'Create job'.

Expected output:

1. Login as organizer.
2. Open job creation screen.
3. Get message about successfully created job.

Actual result:

1. Login as organizer.
2. Open job creation screen.
3. Get message about successfully created job.

Test Scenario 4 - Organizer print pass test Input:

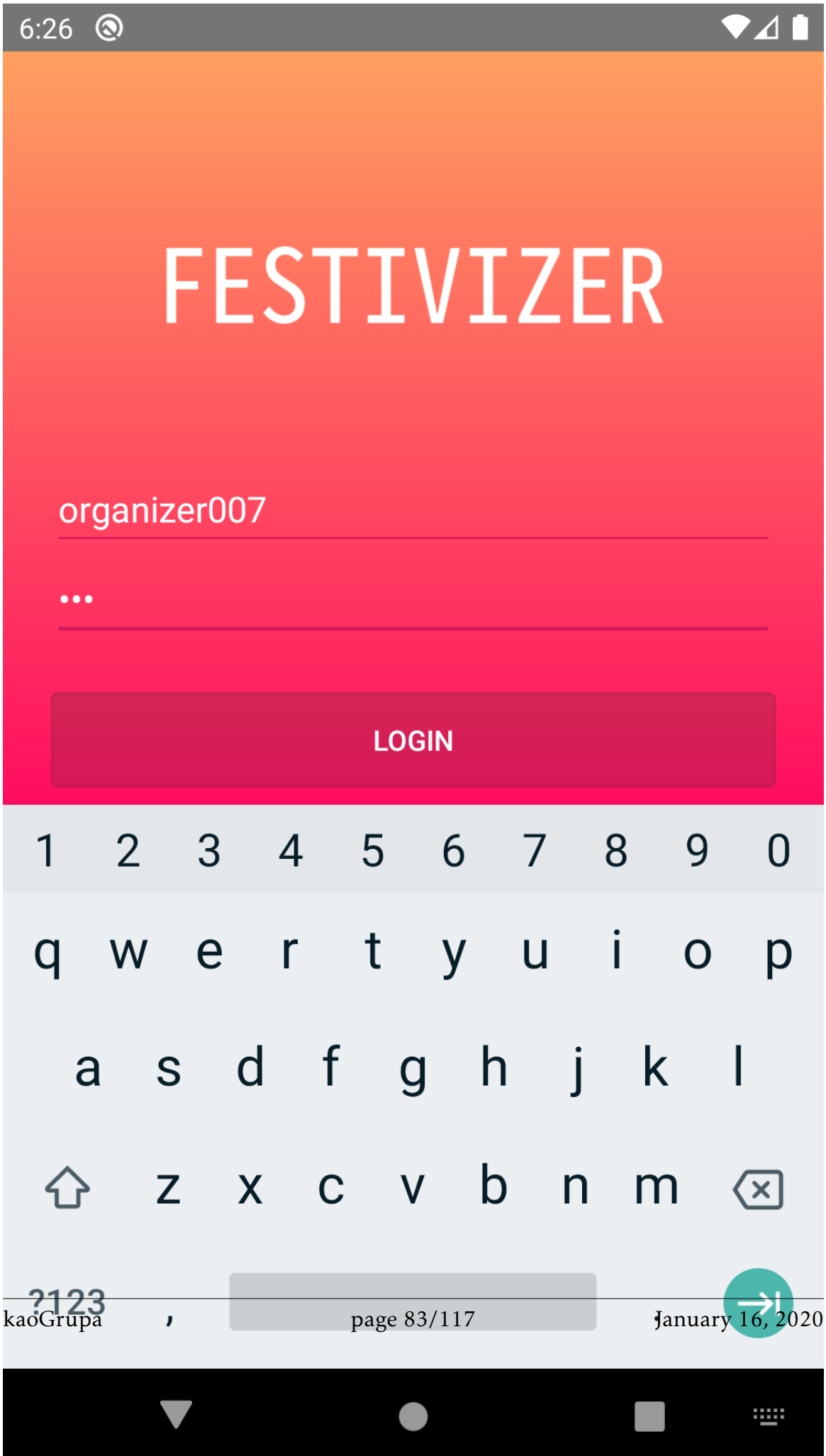
1. Input organizer user data, tap 'Login'.
2. Tap 'Search'.

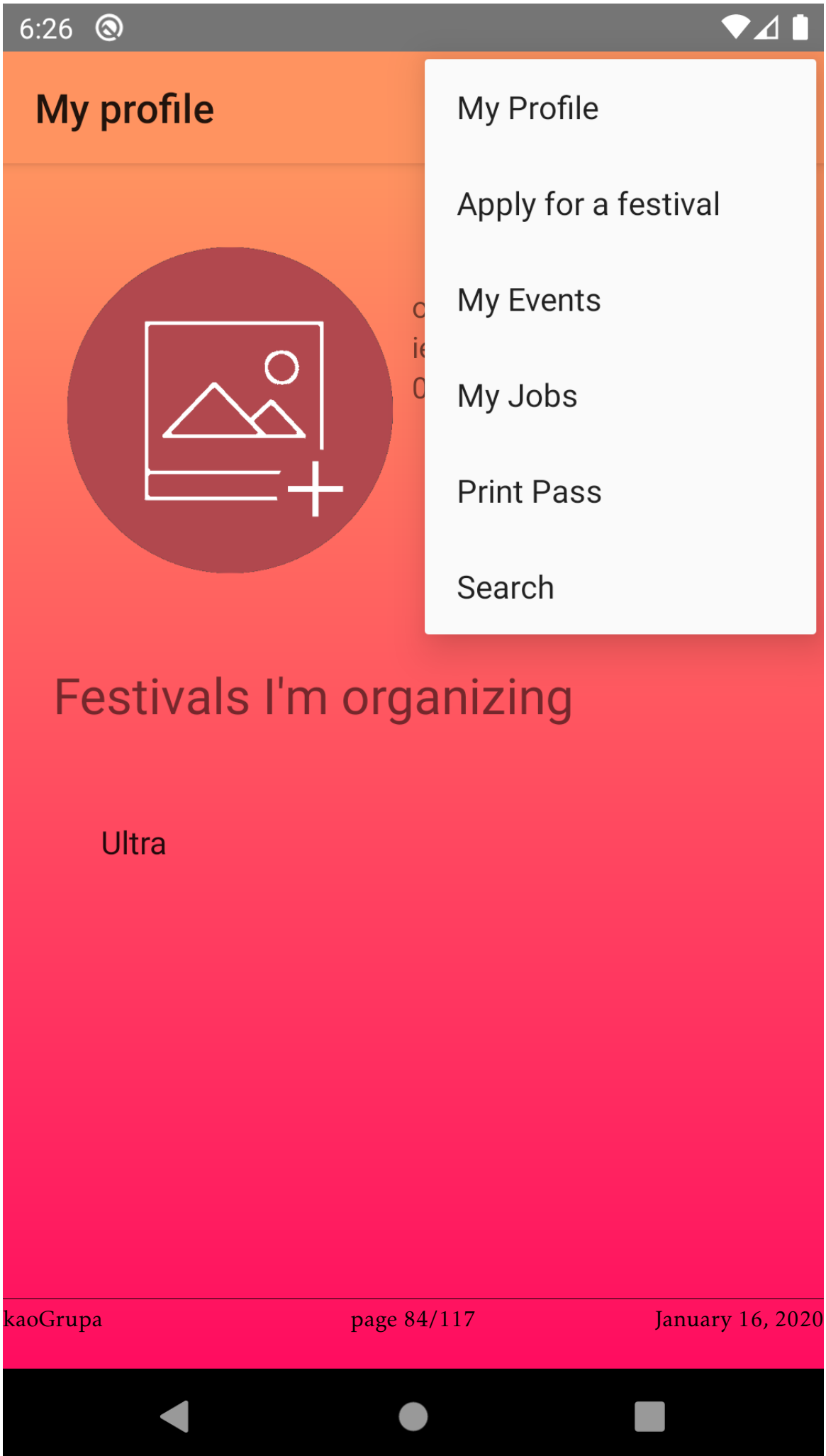
Expected output:

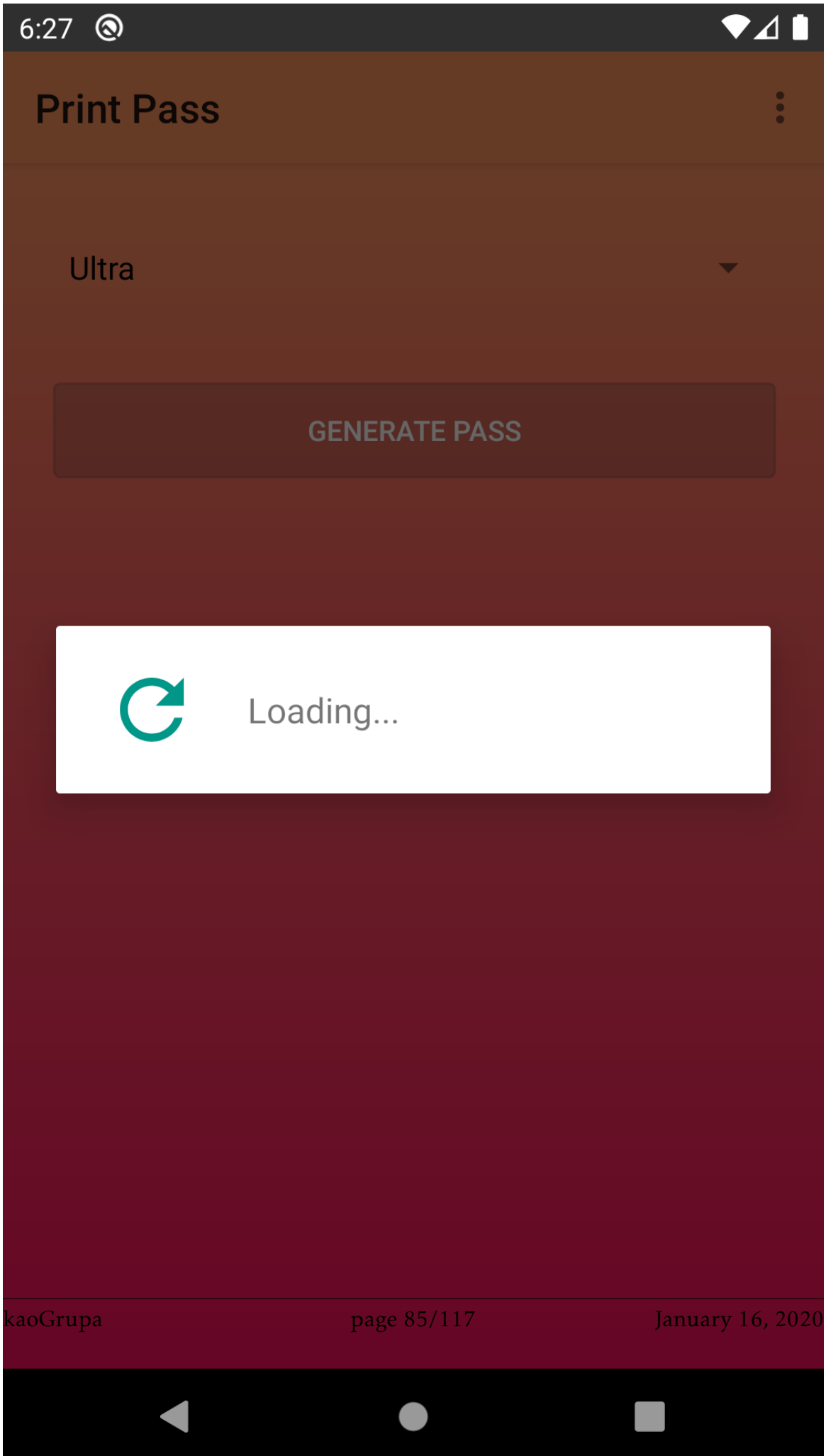
1. Login as organizer.
2. Create required PDF file with pass, get message about successfully created file.

Actual result:

1. Login as organizer.
2. Loading in infinite loop. (BUG shown)


6:26
FESTIVIZER
organizer007
LOGIN


6:26
My profile
My Profile
Apply for a festival
My Events
My Jobs
Print Pass
Search
Festivals I'm organizing
Ultra


6:27
Print Pass
Ultra
GENERATE PASS
Loading...

Test Scenario 5 - Organizer print pass test Input:

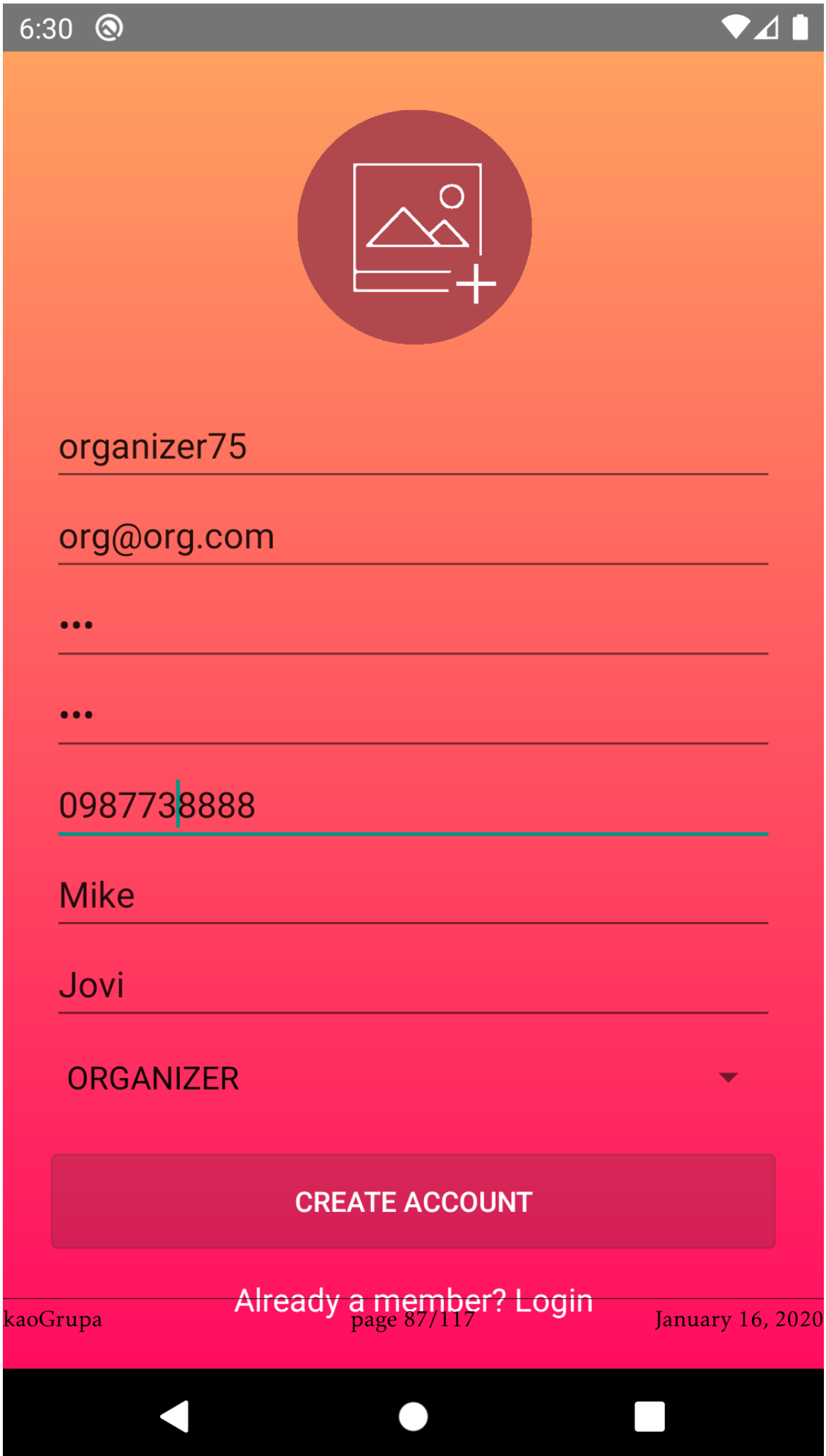
1. Tap 'No account yet? Create one'.
2. Input user data and chose 'Organizer' from drop-down list, and Tap 'Create account'.
3. Tap 'Already a member? Login'.
4. Input organizer user data, tap 'Login'.
5. Tap 'Apply for a festival', tap on some festival or festivals.

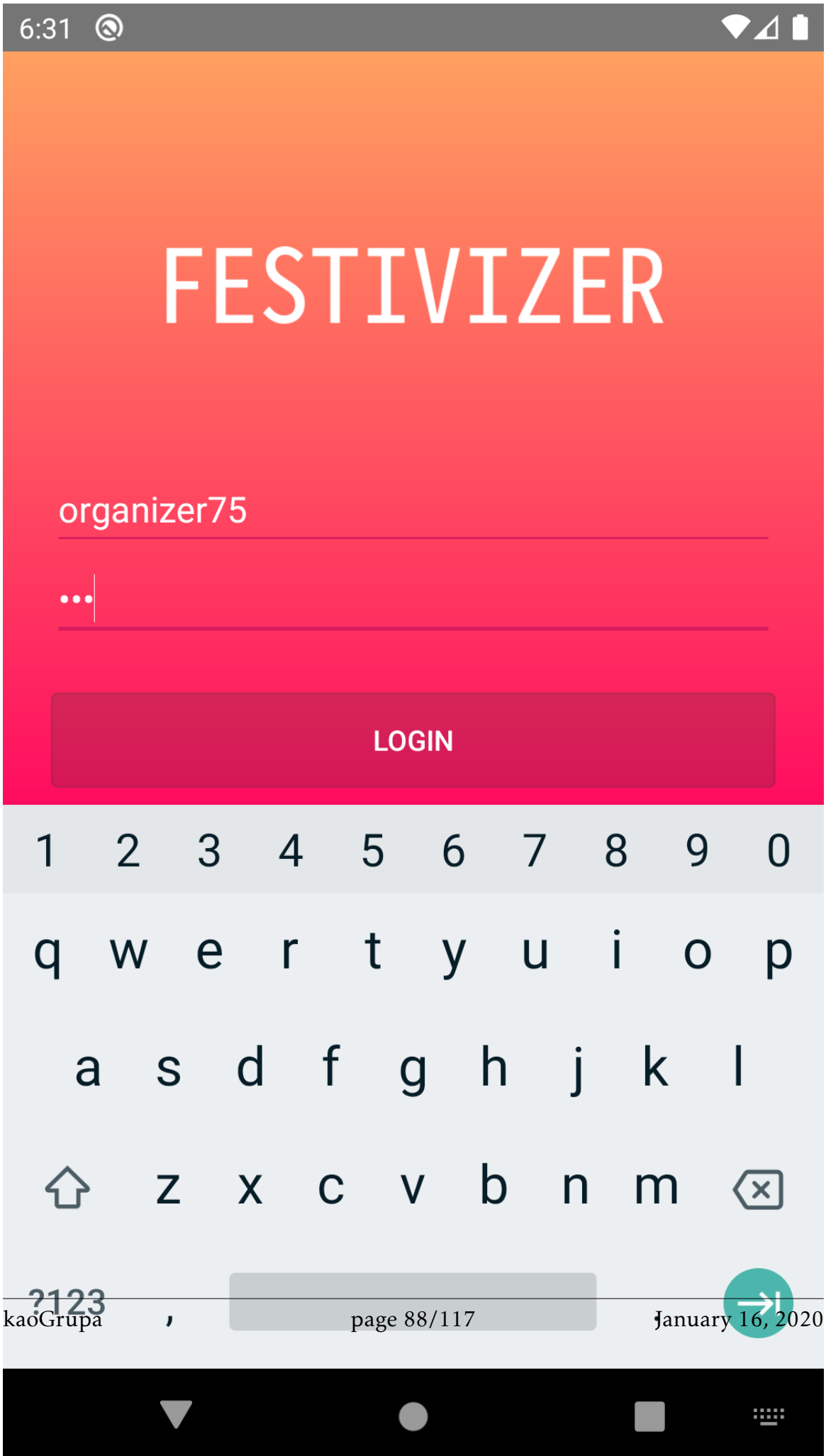
Expected output:

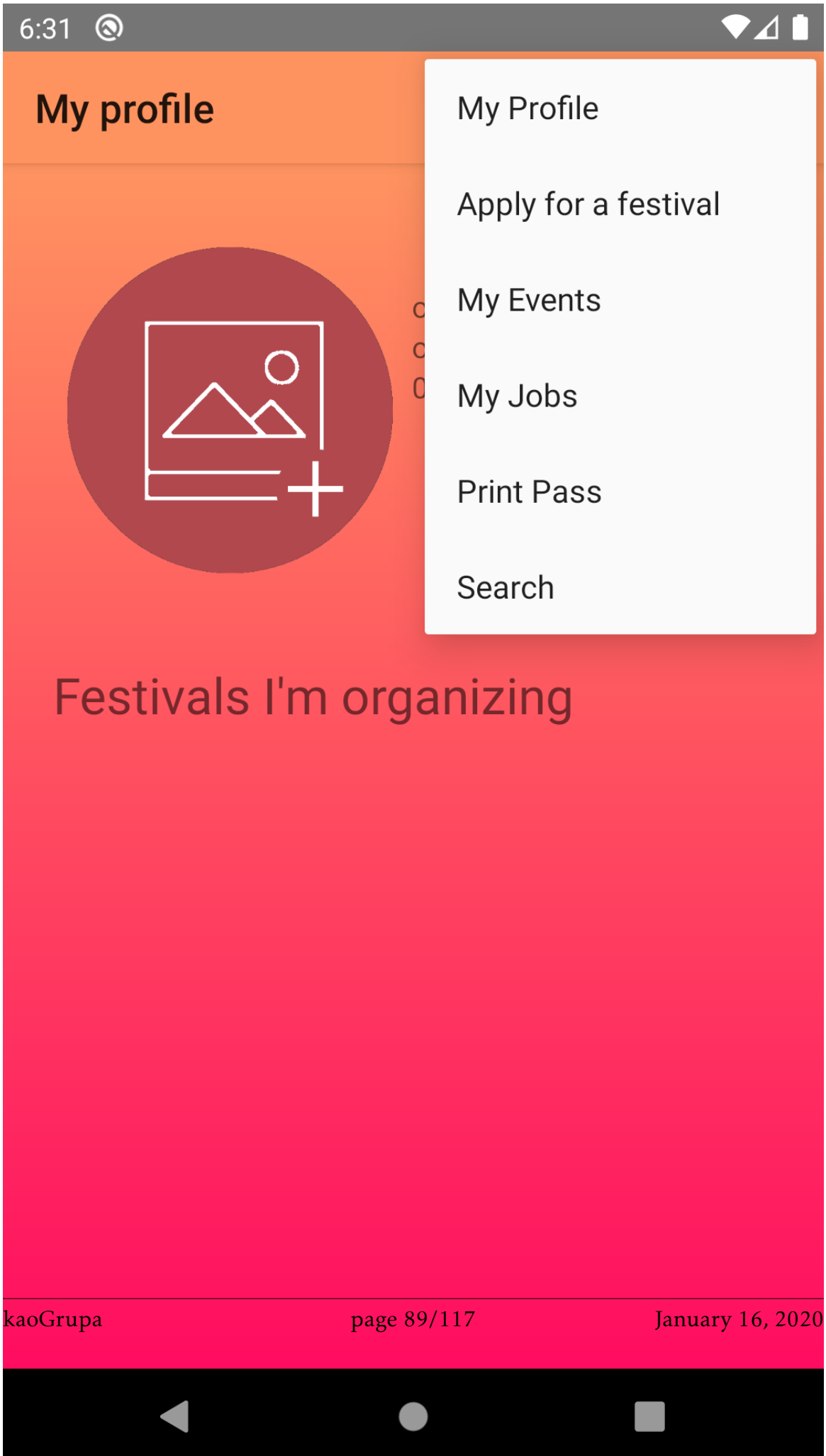
1. Opened register screen.
2. Get message about successful account creation.
3. Opened login screen.
4. Login as organizer.
5. Get message about successfully applied for a festival, and see tick next to selected festivals.

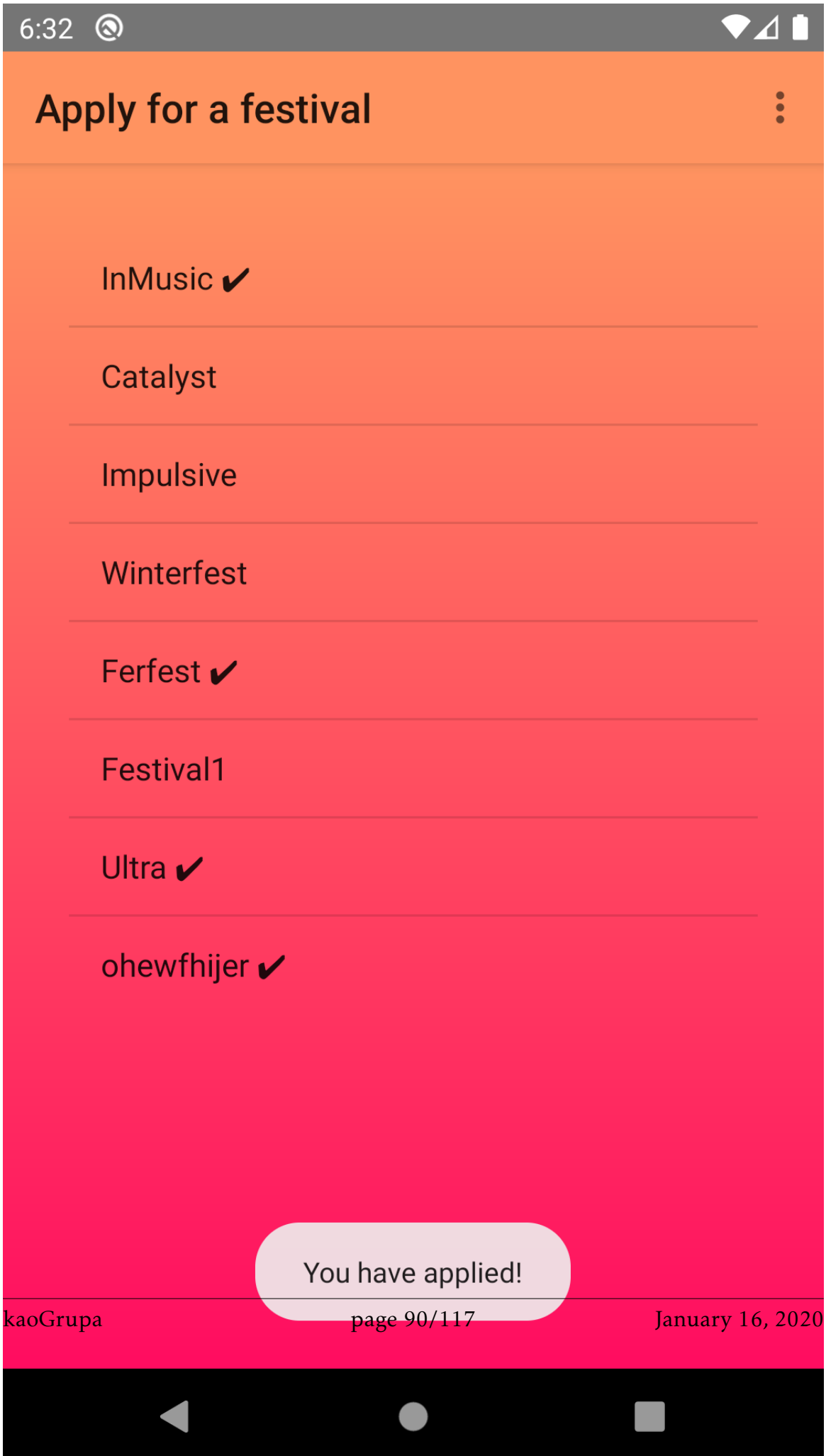
Actual result:

1. Opened register screen.
2. Get message about successful account creation.
3. Opened login screen.
4. Login as organizer.
5. Get message about successfully applied for a festival, and see tick next to selected festivals.


6:30
organizer75
org@org.com
•••
•••
0987738888
Mike
Jovi
ORGANIZER
CREATE ACCOUNT
Already a member? Login


6:31
FESTIVIZER
organizer75
LOGIN


6:31
My profile
My Profile
Apply for a festival
My Events
My Jobs
Print Pass
Search
Festivals I'm organizing


6:32
Apply for a festival
InMusic ✔
Catalyst
Impulsive
Winterfest
Ferfest ✔
Festival1
Ultra ✔
ohewfhijer ✔
You have applied!

Test scenario 6 - Worker multiple tests Input:

1. Input worker user data, tap 'Login'.
2. Tap 'Add specialization'.
3. Input new specialization and tap 'Add specialization'.
4. Tap one specialization from list.
5. Tap 'Apply for a job', tap one job from list, tap 'Apply', input data, tap 'Apply'.
6. Tap back button, tap 'My applications', and tap one from list.
7. Tap 'Search', input 'Marko', tap 'Search'.
8. Tap 'Print Pass'.
9. Tap 'Active jobs', tap 'My applications'
10. Tap 'Log Out'.

Expected output:

1. Login as worker.
2. Opened specialization screen.
3. Added new specialization on a list.
4. Get message about successfully added specialization, see tick next to chosen specialization.
5. Get message about successfully applied for a job.
6. See data about chosen application.
7. See results from search for 'Marko'.
8. Get message about successfully created PDF file with pass, and get file created in storage.
9. See 'Active jobs' screen, see 'My application' screen.
10. Log out, and see login screen.

Actual result:

1. Login as worker.
2. Opened specialization screen.
3. Added new specialization on a list.
4. 'Randomly' chosen specialization and tick next to that randomly chosen specialization. (BUG shown)
5. Get message about successfully applied for a job.
6. See data about chosen application.
7. See results from search for 'Marko'.

8. Get message about successfully created PDF file with pass, and get file created in storage.
9. See 'Active jobs' screen, see 'My application' screen.
10. Log out, and see login screen.

Test Scenario 7 - Admin accept leader test Input:

1. Input admin user data, tap 'Login'.
2. Accept one leader from list of pending leaders.

Expected output:

1. Login as 'admin'.
2. Get message about successfully accepted leader.

Actual result:

1. Login as 'admin'.
2. Get message about successfully accepted leader.

5.3 UML Deployment Diagram

Due to the application including a lot of users, we have opted for the specification deployment diagram. The application consists of two parts: the mobile application(kind of a front-end) and the back-end hosted on the Pythonanywhere cloud. The application uses protocols GET(for information retrieval from the server), and POST in order to send/communicate information to the server. Furthermore, the server consists of 4 parts:

- App.db - definition of the database for SQLite 3
- Models.py - database tables modelled as Python classes
- Run.py - endpoint initialisation and control
- Resource.py - Used for handling server requests and responses

As far as hardware goes - basically 2 devices are required - the mobile phone and the cloud hosting computer. Network is also required as it is used to convey information between these 2 endpoints. While there is only one server instance, it is expected that multiple concurrent mobile phones will be using the application. Therefore, in the instance deployment diagram more than one mobile phone could be present.

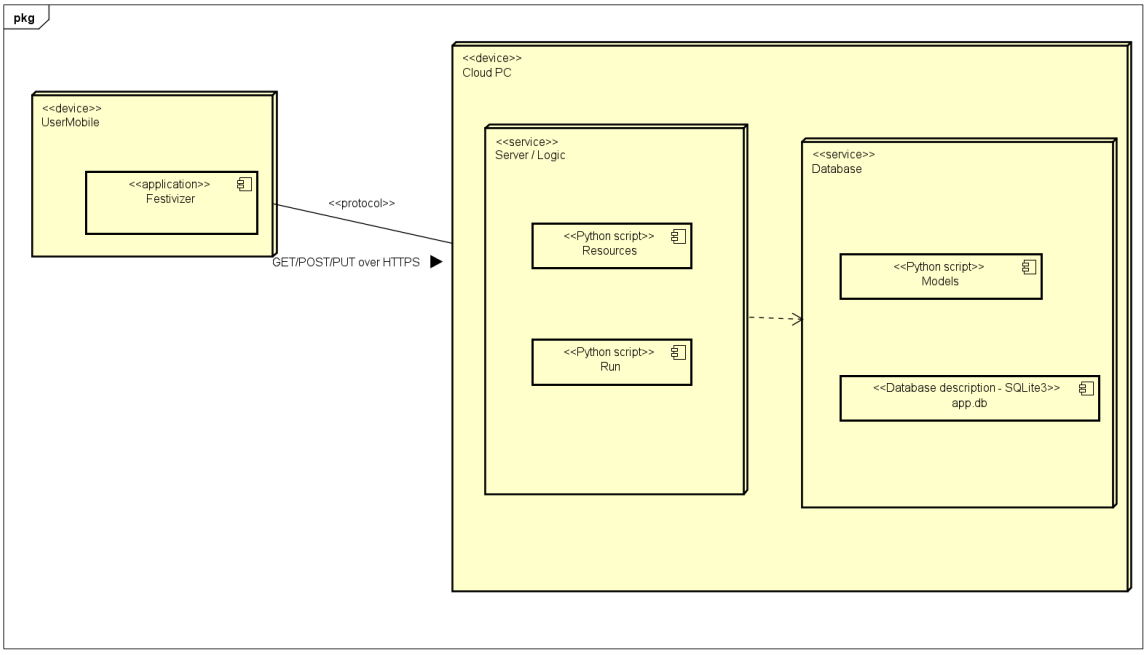


Figure 5.14: Deployment Diagram

5.4 Deployment instructions

5.4.1 Building the APK

The APK can be built in the following way: First, you need to git clone the application's repository: The git repository

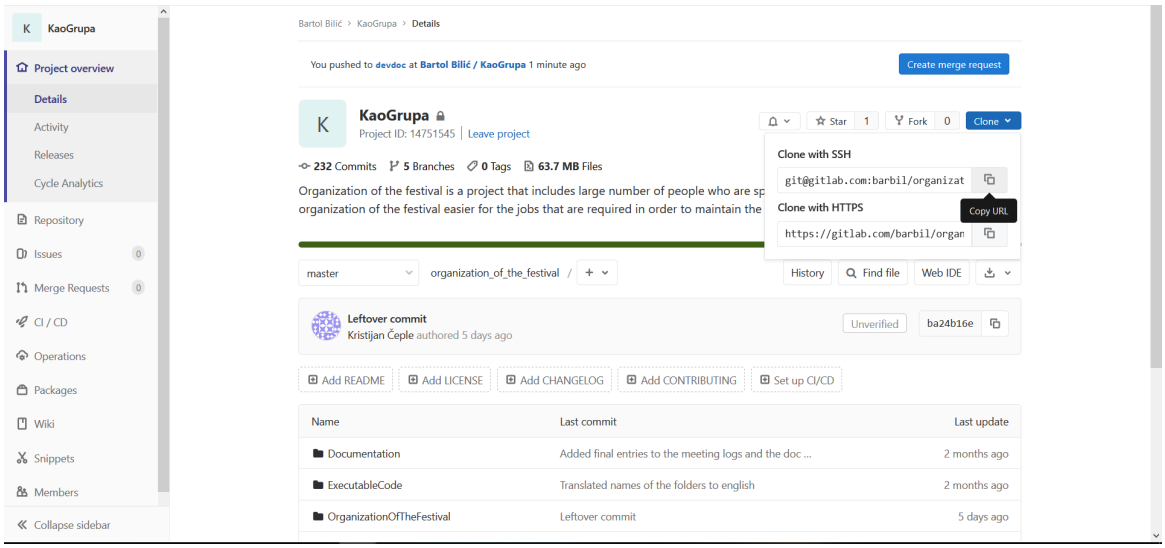


Figure 5.15: Cloning from git

After pulling, the project needs to be opened in Android Studio. Merely go **File -¿ Open**, and then navigate to the cloned git repository. There you will find the project file that can be opened. Now, you would want to build an APK to run it on your device.

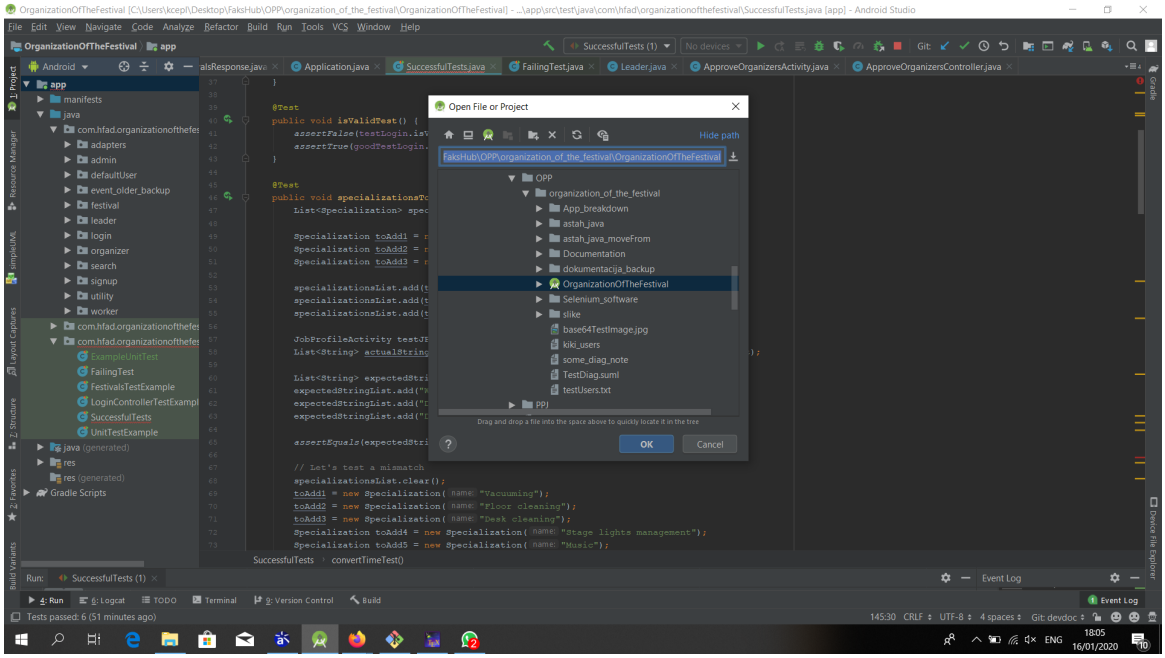


Figure 5.16: Opening the Project

There is also an alternative. It is to go to our store, and install the app from there. Also, it is possible to build a signed build in Android Studio that is ready for deployment on any App store. Link of our application: App Store entry

Here, we will go into the process of generating your APK. Go to **Build -¿ Build Bundle(s)/APK(s)** or **Build -¿ Build Signed Bundle/APK**. If opting for signed bundles and APKs, you will need to generate a secure key that will be used for the signing process.

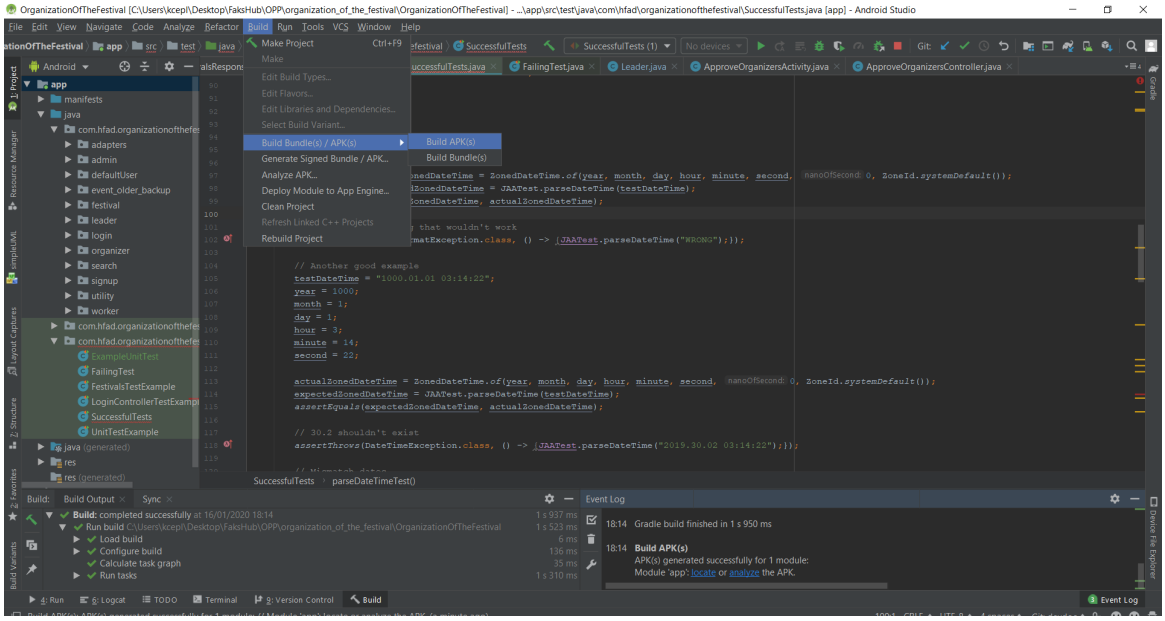


Figure 5.17: Building process

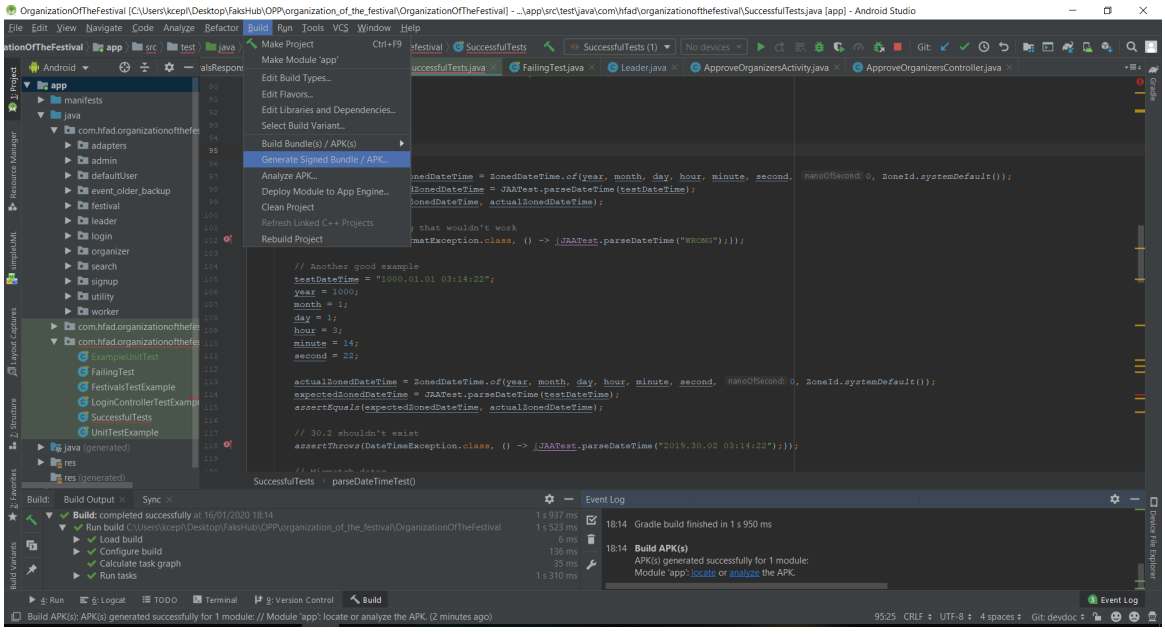


Figure 5.18: Building process

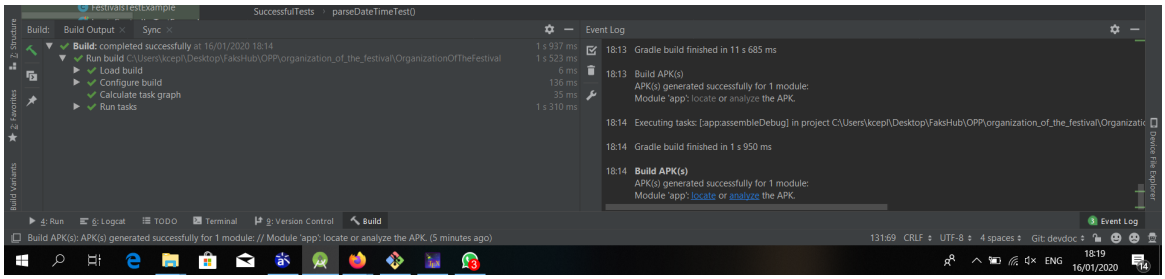


Figure 5.19: Building process

Once you have the APK, it can be uploaded to any app store. **The APKs that will be uploaded to the app store SHOULD be signed.** It is also possible to put this .apk on the mobile device, and install it it there.

Upon running the application, it will automatically connect to the kaogrupa pythonanywhere site, and communicate with the server. You are now ready to use the application. 2

5.4.2 Setting up the server side

The files are already present on the git URI, branch server: The server branch. First of all this git should be cloned down somewhere. The files inside will be put onto the server. For deployment we used pythonanywhere as a cloud site, since it allows easy and seamless deployment of Flask applications.

Second of all, you should either register or log in into pythonanywhere site.

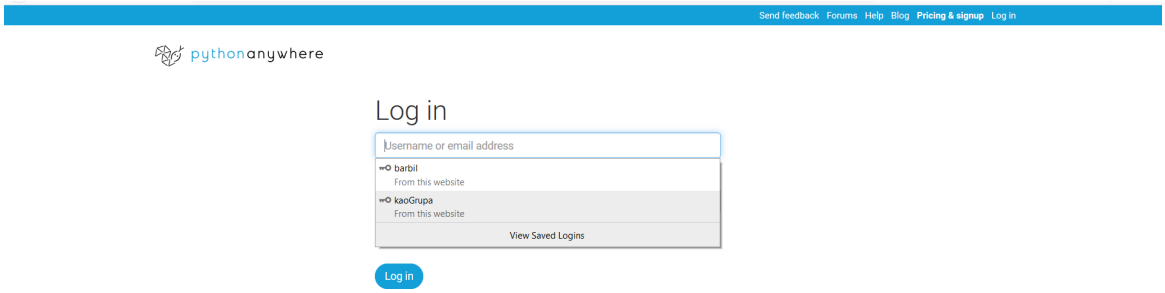


Figure 5.20: Logging-In

Afterwards you should be taken to the dashboard screen. It should look something like this:

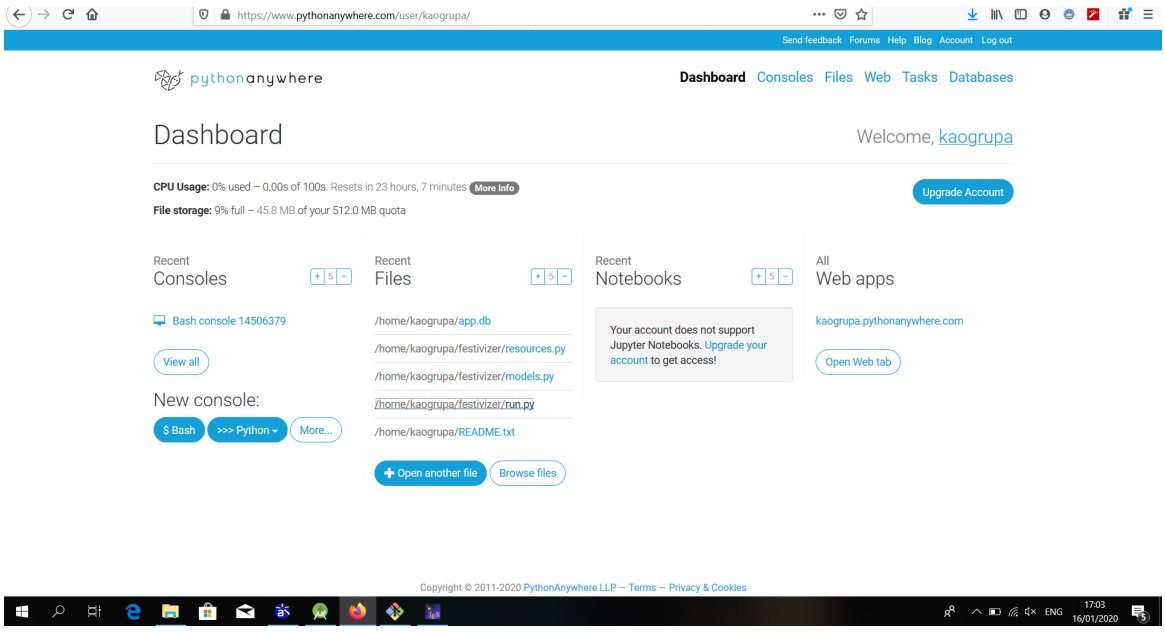


Figure 5.21: Dashboard

Then click on the 'Open Web tab' button under the 'Web apps' title. Once there, click on the 'Add a new Web App' button.

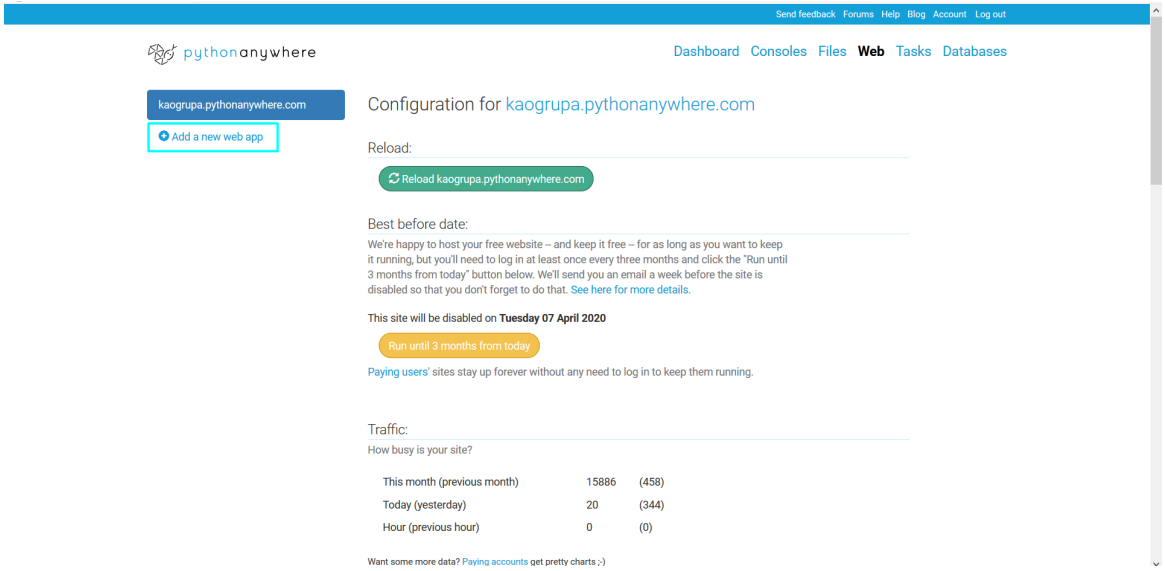
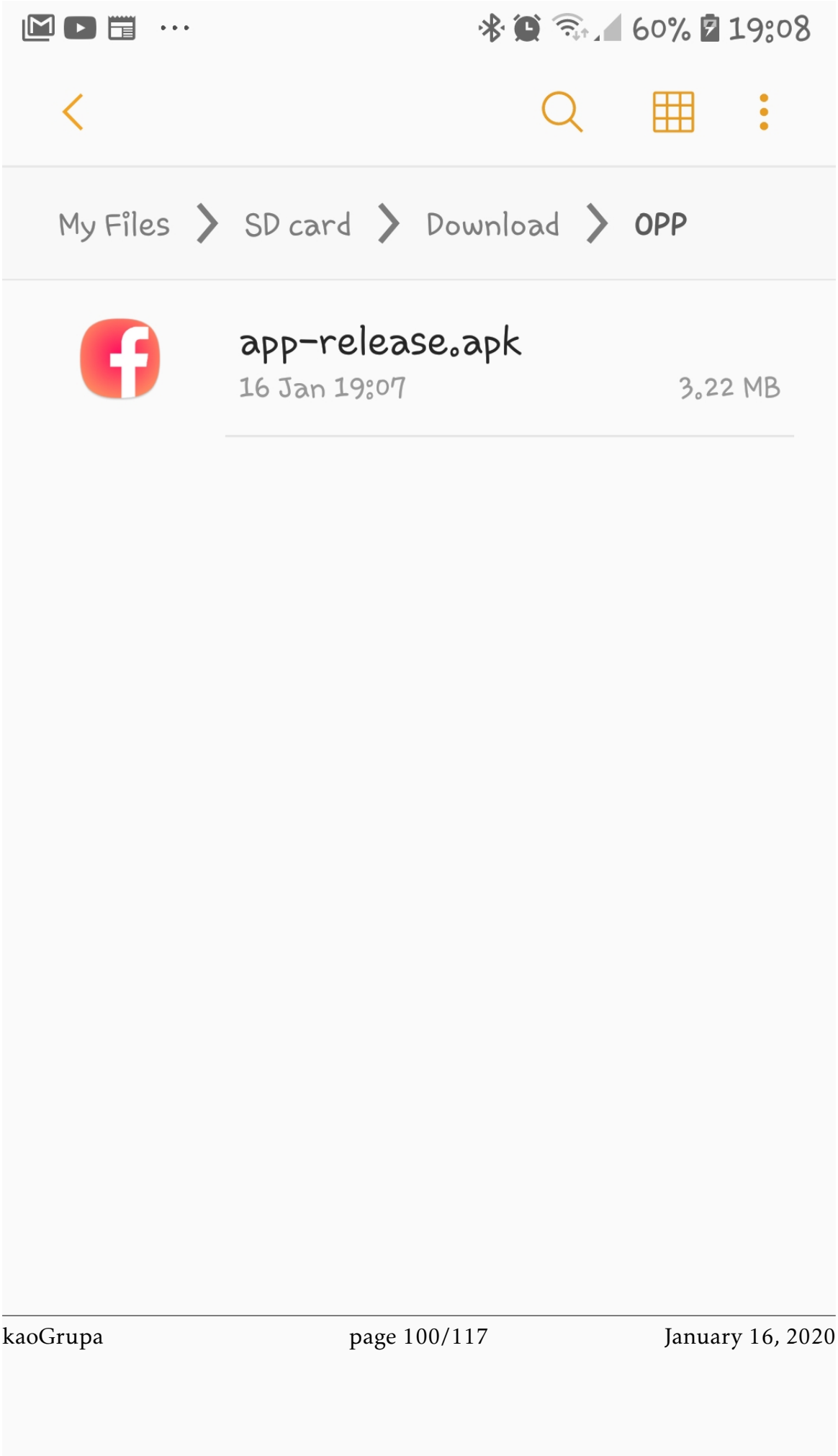


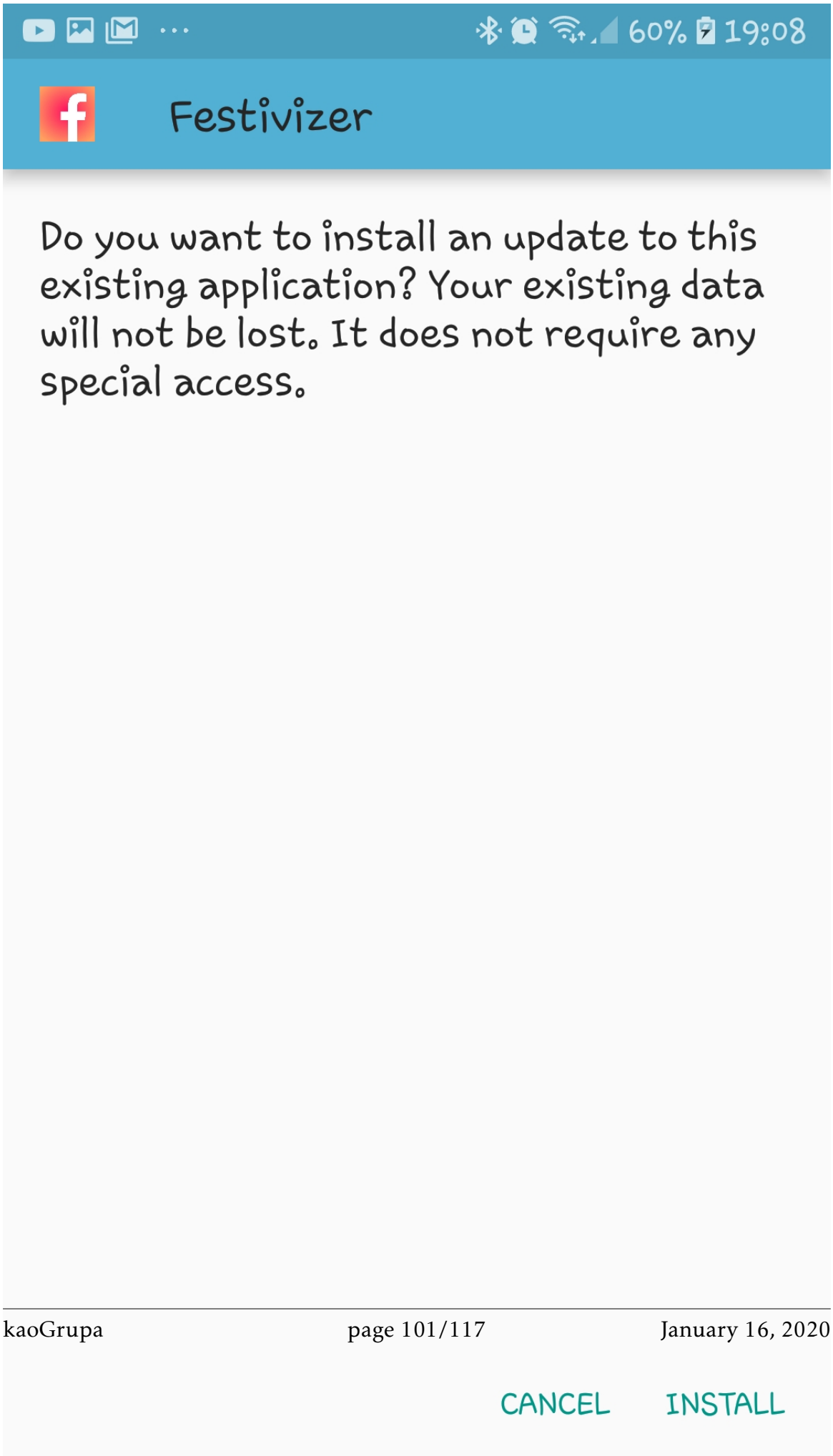
Figure 5.22: Adding a new Web App

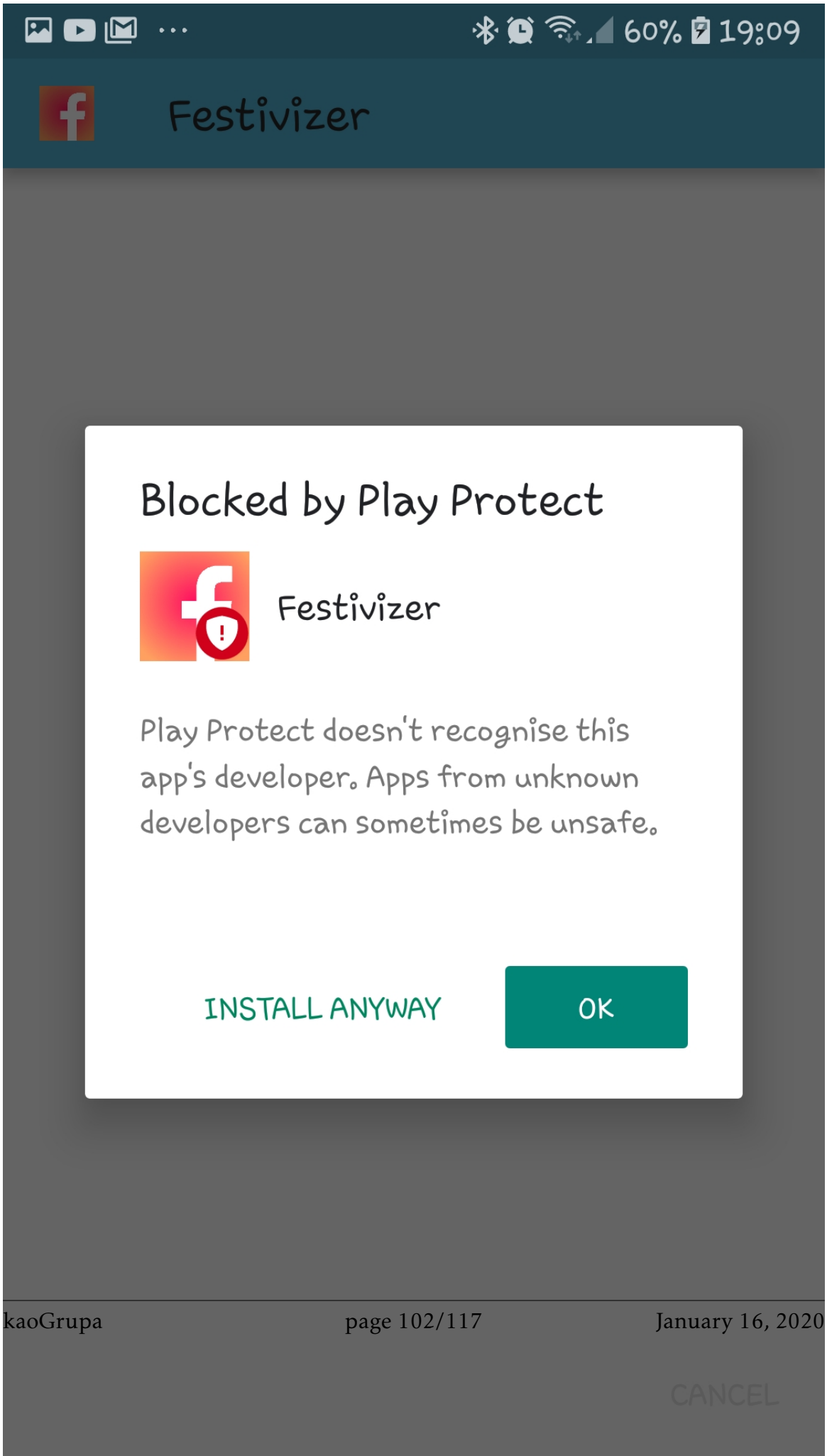
A wizard will appear on the right side, where you should select Flask. Python 3.6 is the minimum supported Python version. The default file flask_app.py will be generated.

Now the project is set up. All that is needed is to upload the files. Flask_app.py entry point file –¿ you should put into it the app.py code from git. Also upload the other files to the server. You should now be set up –¿ all that is left to do is to change the hardcoded kaogrupa URL in the application to point to your server. Once that is done, you have got your own back-end set up, and can comfortably change and experiment on the application.

That is depicted on thee pictures below:


60% 19:08
My Files
SD card
Download
OPP
app-release.apk
16 Jan 19:07
3,22 MB


60% 19:08
Festivizer
Do you want to install an update to this existing application? Your existing data will not be lost. It does not require any special access.
CANCEL
INSTALL


60% 19:09
Festivizer
Blocked by Play Protect
Festivizer
Play Protect doesn't recognise this app's developer. Apps from unknown developers can sometimes be unsafe.
INSTALL ANYWAY
OK
CANCEL

6. Conclusion and Outline of Planned Future Work

Task of our group was to develop mobile phone application that allows organization of the festivals. After working hard for 13 weeks, we managed to develop alpha version of the application that satisfies given task. Work on this group project can be partitioned into two parts: (1) Organization of development, (2) Bare development and documentation.

Organization of development includes gathering team members, assigning a task to our group by our teaching assistant, and work on documentation requirements. Documentation requirements that we set for ourselves were slightly changed during development, due to the progress and challenges in development.

Bare development and documentation includes, as it says, development of application that satisfies given task (that we called 'Festivizer' and made some really cool custom logo) and parallel to development we wrote documentation. Some of our team mates were new in field of programming Android applications so they had to gain new skills. Learning that skills was left on us/them who didn't know that. But as things get going, that was no longer a problem. We started a little bit slow, but we managed to gain high development speed thanks to our team work. At beginning experienced team mates (we had couple of them) helped rest of us to get into new filed. As we were developing application that includes front-end (Android application) and back-end (server and database) we wrote documentation. Although, mostly we firstly completed development of certain part of application and than we wrote documentation for that part. And that is one thing of all that we would changed if would develop some other application together. We would firstly well define application requirements in documentation and after that we would start to develop using some of iterative development options. With that approach we would be more productive, and even if we realize that we misjudged some part of development plan, we could slightly changed requirements and continue to move forward - quickly and without stopping.

Team communication was firstly done using WhatsApp, but soon we realized

that it is not very practical so we moved to better platform for teamwork - Slack. In Slack we had different chats for different sub-tasks - like design, front-end, back-end (server and database), and we also had tasks there and git push notifications - so we could see in any time what someone is doing with application.

Working on this project was really valuable experience that taught all of us how to better work as a team, how to concentrate on stuffs that we need to accomplish and not on someone of us as a one. We were many but we managed to work as a team and concentrate on on our objective - task we had to accomplish. That after all was and a good experience. We were really good team-mates and alongside developing application, we managed to have some fun and work in positive and constructive environment. We are satisfied with skills we acquired and are looking forward to work on some similar projects in same or other teams.

Literature

1. Oblikovanje programske potpore, FER ZEMRIS, http://www.fer.hr/predmet/opp
2. I. Sommerville, "Software engineering", 8th ed, Addison Wesley, 2007.
3. T.C.Lethbridge, R.Langaniere, "Object-Oriented Software Engineering", 2nd ed. McGraw-Hill, 2005.
4. I. Marsic, Software engineering book, Department of Electrical and Computer Engineering, Rutgers University, http://www.ece.rutgers.edu/~marsic/books/SE
5. The Unified Modeling Language, https://www.uml-diagrams.org/
6. Android documents, https://developer.android.com/docs
7. Head First Android Development , A Brain Friendly Guide, 2nd Edition
8. Astah Community, http://astah.net/editions/uml-new
9. https://stackoverflow.com
10. Framework introduction, https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm
11. Flask documentation, http://flask.palletsprojects.com/en/1.1.x
12. SQLAlchemy, https://docs.sqlalchemy.org/en/13

Image and diagram index

Appendix: Preview of group activity

Meeting log

Kontinuirano osvježavanje

1. Meeting
   - Date: October 4, 2019
   - Attended: P.Fribert, D.R.Sparemblek, B.Bilic, K.Ceple, D.Strbad, L.Lendel, K.Jezic
   - Meeting theme:
     - initial getting to know each other, individual skills
2. Meeting
   - Date: October 5, 2019
   - Attended: P.Fribert, D.R.Sparemblek, B.Bilic, K.Ceple, D.Strbad, L.Lendel, K.Jezic
   - Meeting theme:
     - creating back end, front end and server team
3. Meeting
   - Date: October 15, 2019
   - Attended:D.R.Sparemblek, B.Bilic
   - Meeting theme:
     - making first draft database
4. Meeting
   - Date: October 23, 2019
   - Attended:D.R.Sparemblek, B.Bilic, P.Fribert
   - Meeting theme:
     - making database in python
5. Meeting
   - Date: October 24, 2019
   - Attended:L.Lendel, B.Bilic, K.Jezic, D.R.Šparemblek, P.Fribert

- Meeting theme:
  - signup and login screen functions

6. Meeting
   - Date: November 6, 2019
   - Attended: P.Fribert, D.R.Sparemblek
   - Meeting theme:
     - foreign keys in database
7. Meeting
   - Date: November 7, 2019
   - Attended: P.Fribert, B.Bilic, K.Ceple, D.Strbad, L.Lendel, K.Jezic
   - Meeting theme:
     - server functions
     - front end: hashing passwords and checking emails
     - documentation update
8. Meeting
   - Date: November 10, 2019
   - Attended: D.R.Sparemblek, B.Bilic, L.Lendel, K.Jezic
   - Meeting theme:
     - debugging application, checking code
9. Meeting
   - Date: November 11, 2019
   - Attended: P.Fribert, D.R.Sparemblek, B.Bilic, K.Ceple, D.Strbad, L.Lendel, K.Jezic
   - Meeting theme:
     - final changes before commiting project
10. Meeting
    - Date: November 13, 2019
    - Attended: P.Fribert, K.Ceple
    - Meeting theme:
      - documentation specifics
11. On further meetings: There were a lot further meetings, whose details we did not note down, and have forgotten about. We(especially some group members) had very frequent meetings with intertwining and diverse themes, so it is hard to pinpoint and remember what and when was exactly done. It is possible to give an outline though:

12. Pre-winter break
13. Implemented:
    (a) OAuth 2.0
    (b) XML Designs
    (c) Project redesign - MVC applied
    (d) Skeleton project set-up
    (e) Log-in and register fixed
    (f) Basic functionalities implemented
14. Start of January - Implemented most of the functionalities
15. The first week of post-winter break: debugging and more additions
16. Second week: further debugging, completing, documentation

Activity table

Continuously refreshed

	Bartol Bilic	Daniel Rey Sparemblek	Karlo Jezic	Danijel Strbad	Luka Lendel	Kristijan Ceple	Petra Fribert
Project management	2	2					2
Project assignment description						3	
Functional demands						2	
Use Cases descriptions						10	
Use Case diagrams						2	
Sequential diagrams						2	
Other requirement descriptions						0.25	
Architecture and system design							3
Database							6
Class diagram				2			
State diagram							
Activity diagram							
Component diagram							
Used technologies and tools							
Software Testing							
Deployment Diagram							
Deployment manual							
Meeting diary							2
Conclusion							
Literature						1	1
Front end layout	1			6	9		
Front end logic	4		10	5	6		2

	Bartol Bilic	**Daniel Rey Sparemblek**	**Karlo Jezic**	**Danijel Strbad**	**Luka Lendel**	**Kristijan Ceple**	**Petra Fribert**
Creating database	8	11					5
Integrating with the database	5	5	5				
back end	20	3					
Architecture development	2	1					
2nd Cycle	2nd Cycle	2nd Cycle	2nd Cycle	2nd Cycle	2nd Cycle	2nd Cycle	2nd Cycle
Use Cases descriptions						4	
Use Case diagrams						2	
Sequential diagrams						1	
Project management 5							
Architecture and system design 10							
Database		5					5
Class diagram						1.5	
State diagram					1		
Activity diagram					1		
Component diagram					2		
Used technologies and tools					1		
Software Testing				7		3	
Deployment Diagram						1	
Deployment manual						1	
Meeting diary						0.25	
Conclusion				2			
Other Documentation				8		3	

During the semester, we've kept a separate Excel diary table. Here it is in its exported form.

DATUM	OSOBA	OPIS POSLA	BROJ SATI
4.12.2019.	Petra Fribert	changing database	1
4.12.2019.	Petra Fribert	screen designs on paper	3
4.12.2019.	Petra Fribert	documentation revision	2
5.12.	Petra Fribert Daniel R Sp	screen design	4
4.12	Karlo Jezic	signup logic	6
5.12	Karlo Jezic	signup logic	5
6.12.	Petra Fribert Daniel R Sp	screen design	2
7.12.2019.	D Strbad	layout design	8
9.-10. 12. 2019.	D Strbad	layout design	3
9.12.	Petra Fribert Daniel R Sp	layout design	2
10.12.	Petra Fribert Daniel R Sp	layout design	6
14.12.	Kristijan Ceple Daniel R Sp	menu and layout fix	4
4.12	Luka Lendel	signup logic	6
5.12	Luka Lendel	signup logic	5
19.12.	Daniel R Sp.	menu implementation	4
19.12.	Kristijan Ceple	menu implementation	1.5
2.1	Daniel R Sp., Kristijan Ceple	organizer activites	4.5
3.1.	Daniel R Sp.	organizer activites	3
6.1.	Daniel R Sp.	tabs	4
13.1.	Petra Fribert	Worker activity	4
13.1.	Petra Fribert	Bugs and fixes	6
12.1.	Petra Fribert	Activities and layout fixes	4
Prosli tjedan	Petra Fribert	Activities and layouts	10
prosli tjedan	Daniel R Sp.	Activites and server	20
4.1	Karlo Jezic	Applcation logic	6
5.1	Karlo Jezic	Applcation logic	6
6.1	Karlo Jezic	Applcation logic	6
7.1	Karlo Jezic	Applcation logic	8
8.1	Karlo Jezic	Applcation logic	9
9.1	Karlo Jezic	Applcation logic	7
10.1	Karlo Jezic	Applcation logic	7
11.1	Karlo Jezic	Applcation logic	7
12.1	Karlo Jezic	Applcation logic	12
16.1.	Petra Fribert	Opis baze i dijagram	1

DATUM	OSOBA	OPIS POSLA	BROJ SATI
4.1	Bartol Bilic	Application logic, server logic	10
5.1	Bartol Bilic	Application logic, server logic	10
6.1	Bartol Bilic	Application logic, server logic	10
7.1	Bartol Bilic	Application logic, server logic	10
8.1	Bartol Bilic	Application logic, server logic	10
9.1	Bartol Bilic	Application logic, server logic	10
10.1	Bartol Bilic	Application logic, server logic	10
11.1	Bartol Bilic	Application logic, server logic	10
12.1	Bartol Bilic	Application logic, server logic	16
4.1.	Kristijan Ceple	Application logic, server logic	2
5.1.	Kristijan Ceple	Application logic, server logic	1
6.1.	Kristijan Ceple	Application logic, server logic	4
7.1.	Kristijan Ceple	Application logic, server logic	5
10.1.	Kristijan Ceple	Application logic, server logic	1
13.1	Kristijan Ceple	Docs	6
14.1	Kristijan Ceple	Docs	6
15.1	Kristijan Ceple	Docs	6
16.1	Kristijan Ceple	Docs	6
12.1.	Kristijan Ceple	Docs	8
21.12.	D Strbad	layout design	2
14.1.	D Strbad	Docs	5
15.1.	D Strbad	Docs	6
16.1.	D Strbad	Docs	7
4.1	Luka Lendel	Application logic	6
5.1	Luka Lendel	Application logic	6
6.1	Luka Lendel	Application logic	6
7.1	Luka Lendel	Application logic	8
8.1	Luka Lendel	Application logic	9
9.1	Luka Lendel	Application logic	7
10.1	Luka Lendel	Application logic	7
12.1	Luka Lendel	Application logic	14

Note: There is some overlapping between the two tables. It's very hard to distinct what exactly was happening when.

Changes Diagram

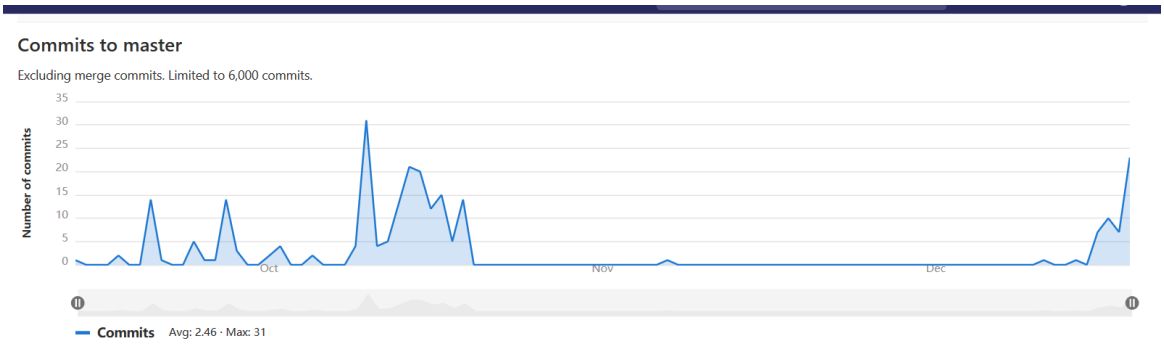


Figure 6.1: Contributors

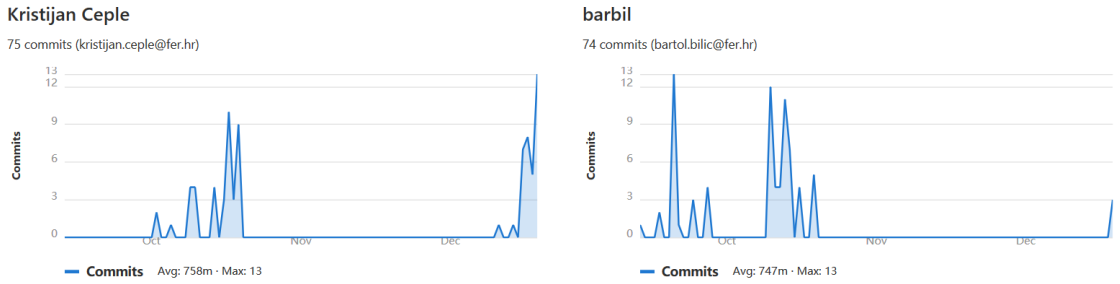


Figure 6.2: Contributors

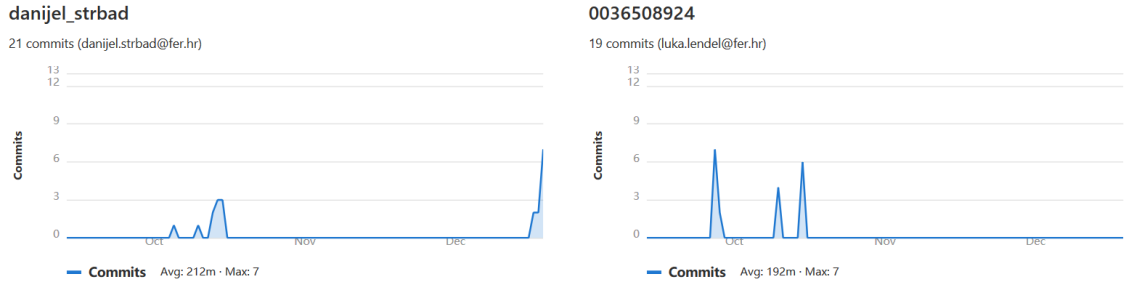


Figure 6.3: Contributors

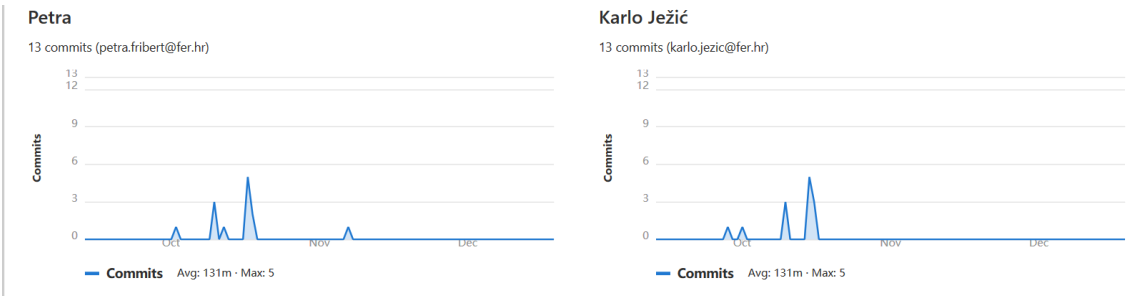


Figure 6.4: Contributors

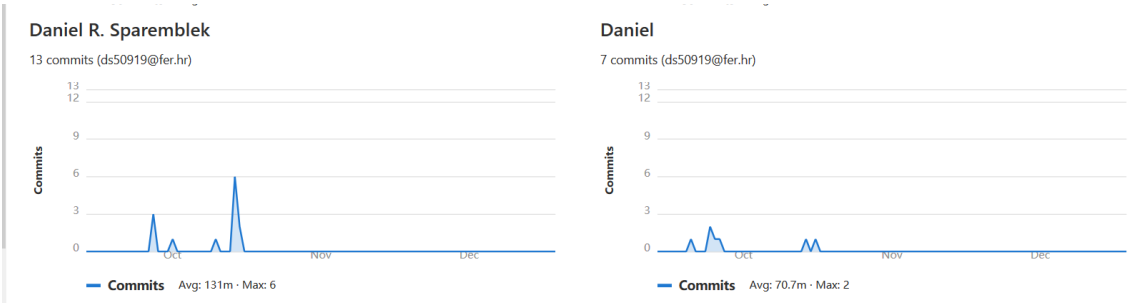


Figure 6.5: Contributors

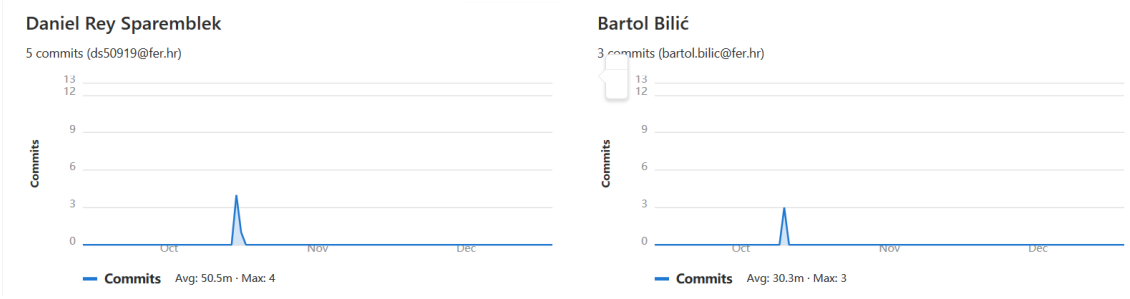


Figure 6.6: Contributors

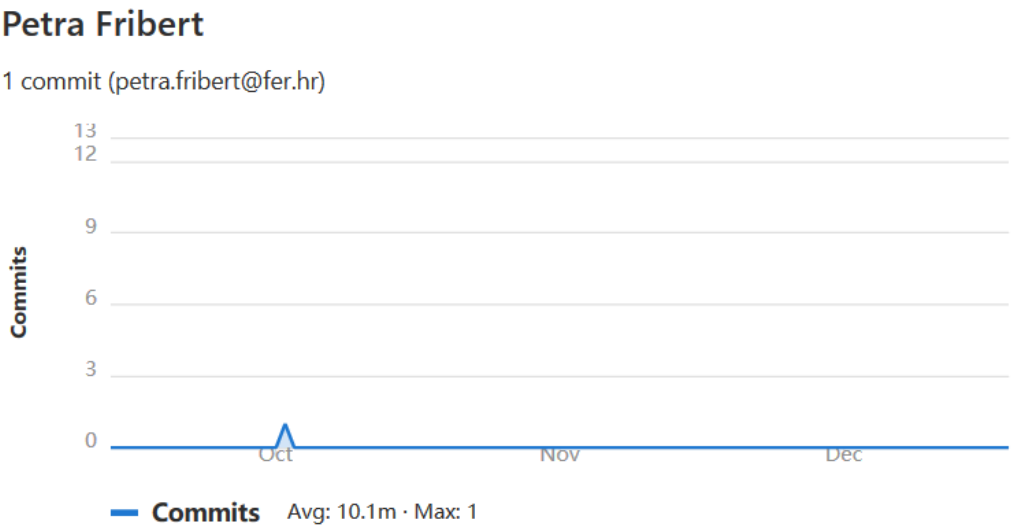


Figure 6.7: Contributors