**Software Architecture Report**

1. **Purpose of the project**

SportMatch is an application for sports enthusiasts that helps them connect with other players and organize matches on sports fields in Bucharest. We have noticed that the frontend design can be improved for the user experience, along with notifications for the chat feature. Also, in order to simplify the user registration process in the application, we have integrated the Google authentication feature. Additionally, the users require more information when choosing a sport venue for the event, therefore the app includes more details and a review feature for each of them.

Users can filter events by date and number of players, and send requests to join the desired events. Each event is organized by a user who becomes an administrator, contacts the sports field for rental and accepts participants to form a team. The application has a chat system to improve the experience of organizing events and has access to the map to locate the fields. SportMatch aims to facilitate the organization of sports events and promote a community of sports enthusiasts with common interests in Bucharest.

1. **How to run the project locally**

* Clone the repository from [GitHub: https://github.com/inginerie-software-2023-2024/proiect-inginerie-software-sportmatch].
* Open the project in Android Studio.
* Run the application on an Android emulator or physical device. The settings on the physical device must be configured based on this Android Studio documentation: <https://developer.android.com/studio/run/device> or the emulator must be set up and downloaded: https://developer.android.com/design-for-safety/privacy-sandbox/download#emulator.

1. **How to build the project**

Click on "Build" Menu:

In the toolbar or the top menu, click on the "Build" menu.

Choose "Make Project" or "Rebuild Project":

Select "Make Project" to build only the modified files and dependencies.

Select "Rebuild Project" to clean and rebuild the entire project, which can be useful if you're experiencing build issues or want to start fresh.

Monitor the Build Process:

Observe the "Build" tool window for any errors or warnings during the build process. If there are issues, the output will provide information about what went wrong.

Check the Output:

Once the build is successful, you can find the generated APK in the "build" directory of your project.

Building the project involves compiling the source code and generating an executable Android application (APK).

* Open the SportMatch project in Android Studio.
* Navigate to the "Build" menu.
* Select "Build Project" to compile the source code.
* The APK file will be generated in the specified output directory.

But when pressing "Run," Android Studio performs the following steps anyways:

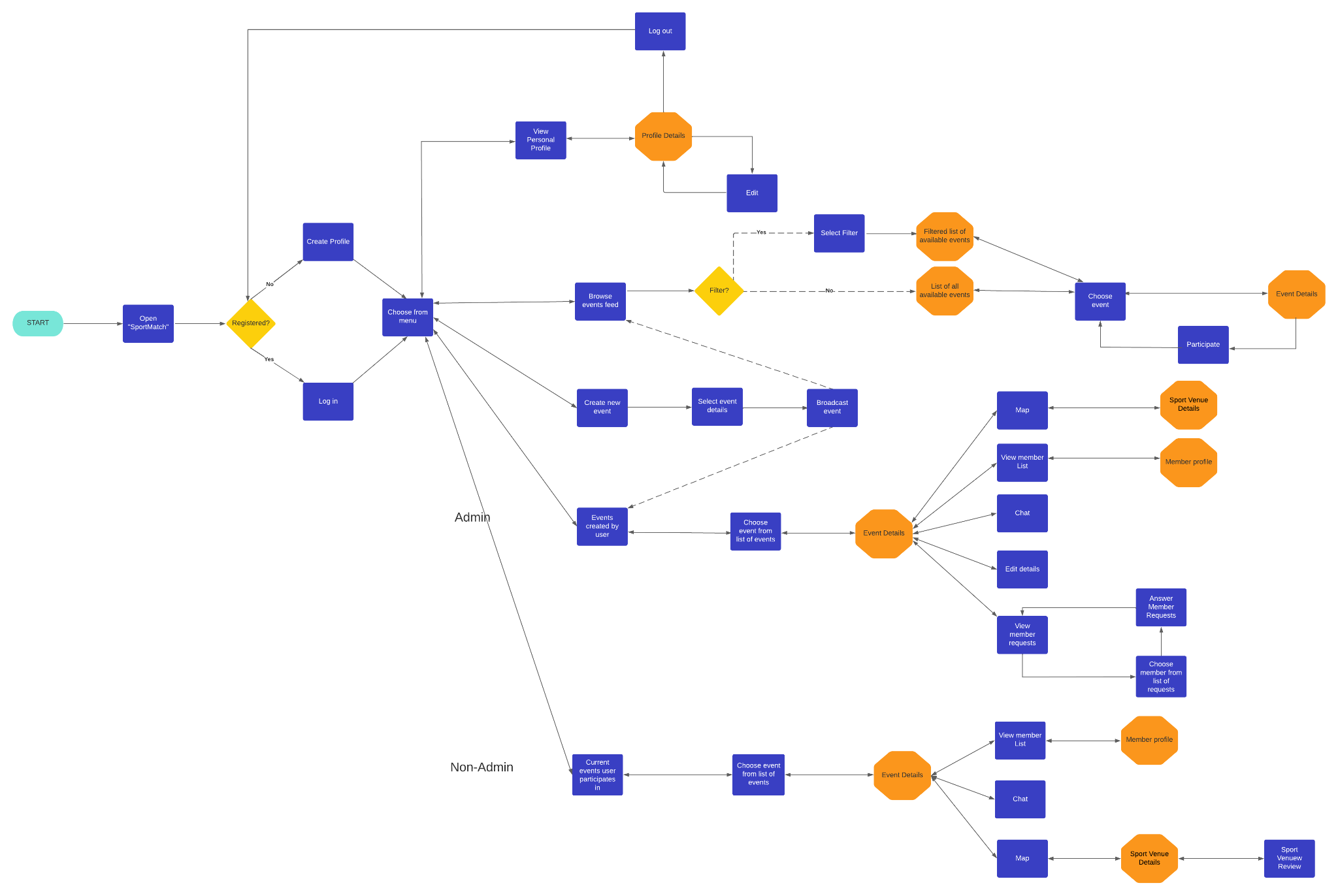
* Build Project: The source code is compiled, and resources are processed to create the APK.
* Install APK: The generated APK is then installed on the target device (emulator or physical device).
* Launch Application: The installed application is launched on the device for testing or debugging.

1. **Deploy the project**

Local deployment in the context of Android Studio and Firebase means running the app on a local development environment, on an emulator or on a physical device. All the Firebase configurations (such as API keys, authentication settings, etc.) were set up once and stored in the project configuration. These configurations are used no matter where the app is run.

1. **Contribution guide:** <https://github.com/inginerie-software-2023-2024/proiect-inginerie-software-sportmatch/blob/main/CONTRIBUTING.md>
2. **Application entry points**
   * Data Sources: SportMatch stores information about events, users, chat messages and sport venue details in the Firebase Realtime Database.
   * Data Inputs: creating events, sending chat messages, and updating user profiles generate data inputs that are stored in the database. For example, creating an event includes inputs such as date, location, event name, venue and number of players allowed.
   * Configuration Files: in the app’s build.gradle we have the Google Maps services plugin, dependencies for Firebase libraries (authentication, storage, messaging), Lottie for animations, Glide for image loading, Volley for network requests and others. The google-services.json file, which is a configuration file generated by Firebase, contains project-specific settings and credentials required for the app to connect to Firebase.
3. **High-level diagrams of the architecture**
   * User journeys:

Admin user for an event and for a user that's not admin:



* + Most valuable output:

The application has a chat system to make the experience of organizing events easy and has access to the map to locate all the sport fields when creating an event and to choose one of them by clicking on it. Google Cloud API is used for this feature. The app has separate feeds for the events the user only participates in, is admin in (only admins can accept or reject a participation request from other users for that event and make changes to the event data) and a feed with all the available events to choose from.

**8. Deployment plan**

* Where is the application deployed: Local Emulator or physical devices with Android operating system.
* How the CI/CD pipeline works:
* Code Changes:

Developers work on the code locally on their machines and push changes to the version control system (e.g., Git).

Manual Build Trigger:

When developers are ready to integrate their changes, they manually trigger a build process. This typically involves running Gradle commands locally or using Android Studio to build the app.

Build Process:

The build process compiles the code, resolves dependencies, and generates the APK manually.

* Automated Testing:

After the build, automated tests (unit tests, integration tests, UI tests) are executed to ensure that the code changes haven't introduced any regressions or issues. The app has 4 tests, one that tests the Log entries, one that tests if the user is correctly redirected to register if the register button is clicked and to login if the login button is clicked, checking if the field of each Activity are displayed and can be filled with the user’s information.

Developers manually conduct testing, including unit tests and manual testing of the app's functionality.

* Continuous Deployment (CD)

When the testing phase is complete, developers manually create a release version of the APK

**9. Description of the QA process**

To verify the functionality of the app, we handled both manual and automatic testing, both during the implementation or at the finished stage of a specific feature.

An example of text suites we followed:

TEST SUITE 1- Login

| **test case** | **expected goal** |
| --- | --- |
| trying logging in using a valid username and password | the user is logged in successfully |
| trying logging in using a username that does not exist in the database | - the user is not logged in  - the user is informed of the issue |
| trying logging in using an existing username with a wrong password | - the user is not logged in  - the user is informed of the issue |
| trying logging in using Google authentication | the user is logged in successfully |
| trying logging out | the user is logged out successfully |
| checking if the user remains logged in after exiting the app | the user is still logged in after exiting the app |

TEST SUITE - Participating in an event

| **test case** | **expected goal** |
| --- | --- |
| (as a user, in an event he is not participating in ) sending a request to enter an event | actual user side:  - the user will be informed the request is sent  admin of event side:  - the admin will receive a notification  - the user will be added to the list of requests |
| (as an admin) confirming a request to enter an event | - the user will be added to the list of participants  actual user side:  - the user will now have access to the event  admin of event side:  - the user will disappear from the list of requests |
| (as a user, in an event he is not participating in) checking the participation list of an event | - the full list of participants will be displayed |

**10. External dependencies included in the project**

1. APIs used: Google Cloud API
2. Libraries

**dependencies {**

**implementation 'androidx.appcompat:appcompat:1.6.1'**

**implementation 'com.google.android.material:material:1.8.0'**

**implementation 'androidx.constraintlayout:constraintlayout:2.1.4'**

**implementation 'com.google.firebase:firebase-database-ktx:20.2.0'**

**implementation 'com.google.firebase:firebase-database:20.2.0'**

**implementation 'com.google.firebase:firebase-auth:21.3.0'**

**implementation 'androidx.navigation:navigation-fragment:2.5.3'**

**implementation 'androidx.navigation:navigation-ui:2.5.3'**

**implementation 'com.google.firebase:firebase-storage-ktx:20.1.0'**

**implementation 'androidx.cardview:cardview:1.0.0'**

**testImplementation 'junit:junit:4.13.2'**

**androidTestImplementation 'androidx.test.ext:junit:1.1.5'**

**androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'**

**implementation 'com.github.bumptech.glide:glide:4.14.2'**

**implementation 'com.google.android.gms:play-services-maps:18.1.0'**

**implementation 'com.airbnb.android:lottie:4.2.2'**

**implementation 'com.google.firebase:firebase-messaging:23.1.2'**

**implementation 'com.google.firebase:firebase-analytics:21.3.0'**

**implementation 'com.google.android.gms:play-services-gcm:17.0.0'**

**implementation 'com.android.volley:volley:1.2.1'**

**implementation "androidx.work:work-runtime:2.7.0"}**

Firebase Libraries (e.g., firebase-database, firebase-auth, firebase-storage, firebase-messaging, firebase-analytics):

Firebase is maintained by Google and generally follows good security practices.

AndroidX Libraries (e.g., appcompat, material, constraintlayout, navigation-fragment, navigation-ui, cardview):

AndroidX libraries are part of the Android Jetpack and are maintained by Google. They are considered reputable, and security updates are generally communicated through official Android channels.

**Google Play Services Libraries (e.g., play-services-maps, play-services-gcm):**

These are official Google libraries. Similar to Firebase

**Lottie:**

Lottie is an animation library by Airbnb. It's widely used and generally maintained well.

**Volley:**

Volley is a networking library for Android. While it's not as actively developed as some other libraries, it's still widely used and generally considered secure.

**JUnit and Espresso:**

These are testing libraries and are considered secure.

**Work Manager (androidx.work:work-runtime):**

Work Manager is part of Android Jetpack and is maintained by Google.