Software architecture document for project Citylogia

0. Authors

- Ilya Volf
- Igor Chiesov
- Andrew Weiss

1. Goals and limitations

1.1. Key functional requirements

1) The application must have searching by application content.
2) The application must display places on the map.
3) The application must support adding new cities in future.
4) The application must have an ability to receive updates which may be installed in Google Play.
5) The application must support background map updates.
6) The application must support rating places and leaving feedback to them by users, so users may see an average rating and a list of reviews of some place.

1.2. Non-functional requirements

1) The application must have a decent response to users actions.
2) The application must be supported by, literally, every Android device.
3) The application must support several simultaneous users.

1.3. Architectural goals

1) Support of at least 1000 concurrent connections of users with the further possibility to increase the number of concurrent connections.
2) Support as many users as it is only possible not at the expense of app's functionality.
3) Provision of possibility to extend the list of supported cities over time.
4) Since the implementation of the application is split into several stages, then it is necessary to support an integral implementation of the application functional stage by stage.
5) Storing information about places, their rating and reviews on the database. So a user will always be kept in touch receiving from the server fresh data about places.
6) Modifiability of the application structure for implementing new features in future.

1.4. Additional goals, restrictions and preferences

1) A tendency towards high compatibility of current Android-implementation and possible iOS-implementation in future.
2) Since, concurrently, we are studying Java, the application will be written on Java in order to simplify the process of implementation as well as improve skill of programming on Java.
3) The server will be written on C# since a person who is responsible for the development of the server knows this language well.

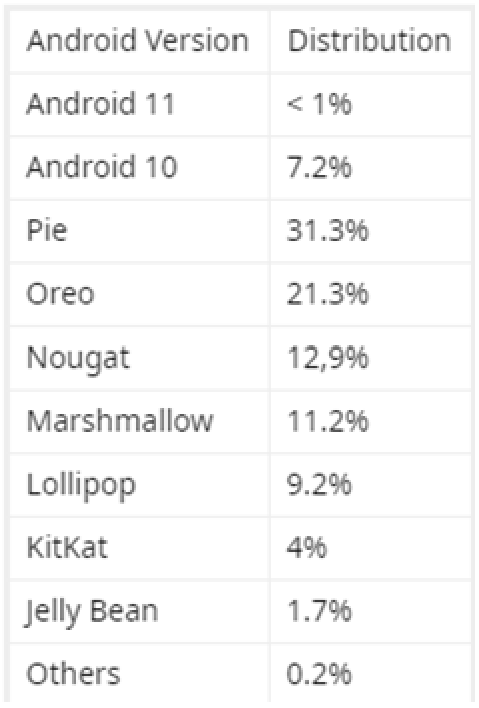
2. Goals analysis

2.1. Support as many users as it is only possible not at the expense of app's functionality

It is important to maintain a decent ratio between the support of older versions of Android for better compatibility and availability for users and the support of important functionalities which was implemented only on recent versions of Android. So, we are to analyze which functional from new versions of Android we need to use in our application and whether there is a way to refuse from using this functional.

So, we end up with the minimal android version that will be supported by our application. The share of android devices with minimal android version and newer should be approximately 75% of all android devices of all android versions.

Information on the android version distribution is available in Android Studio. On February 2021, the state of version distribution is as follows



Android Version	Distribution
Android 11	< 1%
Android 10	7.2%
Pie	31.3%
Oreo	21.3%
Nougat	12,9%
Marshmallow	11.2%
Lollipop	9.2%
KitKat	4%
Jelly Bean	1.7%
Others	0.2%

The table shows that all versions earlier than Lollipop are almost unused, so it makes sense to consider only the versions starting with Lollipop and above.

In terms of functionality, Android 7.0 has some major changes in API. These include JIT compilation, which allows to greatly improve the app performance, memory usage enhancement, background processing optimization, new APIs for the notification system. All these features are greatly helpful in terms of our first application and make working with the system much easier and more efficient.

So, we decided to select Android 7.0+ as the minimal supported Android version. This is also in line with our goal of supporting about 75% of devices. To support Android 7.0+ devices, we use Android API of 24 level.

2.2. List of supported cities extensibility

We need to have information about all cities around the world for maximum extensibility, so we are going to use an integrated map platform. It is the easiest way since all we need to do is to organize marking interesting places in cities (nonetheless, it is not simple as it might seem).

There are 3 leading and reliable map platforms for us to choose

- Google Maps
- Mapbox
- Openstreet Maps

Among all platforms Google Maps has the most amount of free map views, supports turn by turn navigation, has the most efficient automatic location tracking, convenient customization tools and detailed documentation.

At the first stage we prepare an initial list of interesting places in Novosibirsk. In future, the database will be extended by users (they will be able to write articles about interesting places). For each new supported city an initial list of interesting places will be created and loaded into the database.

2.3. Modifiability of the application structure and integral development

It is obvious that all desirable functionalities cannot be implemented at once so we need to split our implementation into several stages. We need to plan the development of the application considering our strategy of gradual implementation in order to make sure that we will avoid so-called bottlenecks in future.

We have split our functionality into essential and non-essential (actually, we have a more complex gradation of implementation stages). The essential part is mvp - it is the first thing to be implemented since the map with places, reviews and ratings is a basic and vital part of our application, after proper implementation of which non-essential functionality may be implemented (authorization, profiles, etc.).

To reduce possible refurbishments of the application structure in future, we produced an idea/understanding how functionality of each stage needs to be implemented; also developed our roadmap and realized what is the way of our further development.

So, when we implement an early stage (stage with higher priority) we already consider realization of later stages (stages with less priority). As a result, when it is time for moving to the next stage of the app's implementation, we are only left to implement these features without facing problems with editing functionality of early implemented stages.

3. Solution description

In accordance with goal analysis conducted earlier, we have ended up with the following solution description:

3.1. Modules and subsystems

Server

- API controller
- DataBase interaction service
- Authorization module

DataBase

- DataBase

Application

- Page-view controller
- Map controller
- Search system

Server

- The overall description of the server part: we use a distributed architecture. The controller that controls all processes is located separately from the data. It's worth mentioning that all of these are physically different points hence we split our connection stream into groups and reduce the simultaneous workload at each object of the architecture at any one time.

  For reducing workload on the server during a large amount of concurrent connections, caching of the static data on the client may be realized.

  - API Controller receives and sends queries according to the REST standard. It decides what module will be used for the processing of received Data. API controller manages all inner servies/modules.
  - DataBase interaction service operates with DataBase. All queries for receiving and updating data are passed through the service. It may be called by API Controller.
  - Authorization module is an intermediate link which manages user's registration and authorization in the system. Decides in what form passwords will be stored in the database and hashes them. Performs validation of user's login and password.

Database

- We store data about places (their reviews and ratings) in the postgreSQL database. Receiving and processing from the database are produced on the server's side with the use of Entity Framework Core. Delivery of data to clients is implemented according to the REST standard. In this way, we ensure uniformity of location data for all clients, synchronization of changes in this information and timely delivery of fresh information to users.

  At the first stage we prepare an initial list of interesting places in Novosibirsk. In future, the database will be extended by users (they will be able to write articles about interesting places). For each new supported city an initial list of interesting places will be created and loaded into the database.

Application

- Page-view controller decides what a page will be seen at a particular moment.
- Map controller is a service which manages drawing of the map, tracking current location of a user

and displaying places on the map. Map controller is also responsible for setting routes between a user and selected places.

- Search system is responsible for filtering all interesting places by categories (parks, theaters, etc). Besides this, it provides the search of interesting places itself: user types the name of some place in the search tab, and the typed place is depicted on the map.

3.2. Deployment

- The overall description of the deployment: we have split our functionality into essential and non-essential (actually, we have a more complex gradation of implementation stages). The essential part is mvp - it is the first thing to be implemented since the map with places, reviews and ratings is a basic and vital part of our application, after proper implementation of which non-essential functionality may be implemented (authorization, profiles, etc.).

  To reduce possible refurbishments of the application structure in future, we produced an idea/understanding how functionality of each stage needs to be implemented; also developed our roadmap and realized what is the way of our further development.

  So, when we implement an early stage (stage with higher priority) we already consider realization of later stages (stages with less priority). As a result, when it is time for moving to the next stage of the app's implementation, we are only left to implement these features without facing problems with editing functionality of early implemented stages.

    - DataBase integration service interacts with it through the EntityFramework Core. The service implements the functional of deleting, adding and updating of the data in the dataBase. For the encapsulated interaction with the dataBase Repository Pattern concept is used.
    - The base of the server is presented by API Controller and held in the AspNetCore framework. It is able to receive and send queries. Implements the functionality of managing places and users.
    - To access the full application's functionality it is required for a user to give access to track the user's location and to the internet location.
    - Android 7.0+ is the minimal supported Android version. To support Android 7.0+ devices, we use Android API of 24 level.

4. Key architectural elements

4.1. Modules API

The API controller takes full responsibility for managing the API of the application. For external interaction with clients, a RESTful API is implemented. Interaction with internal layers occurs through the standard API in terms of interfaces.

4.2. Authorization

During authorization, the presence of a user is checked and his password is validated on the server side. All authorization logic rests on the authorization module. If successful, the client will be returned a special user identifier (token), which he will use to further interact with the application ecosystem. With the help of this token, we will be able to distinguish between users and delineate their data.

4.3. Search

All search logic rests on the search system. A user selects types of places they are interested in, opens the search page that consist of a search bar and a list of places. This list may be sorted by 1) interest rate of the place 2) distance from the current position of this user. The client inputs the name of a place “Парк...”, and the list of places is updated immediately and contains only places, names of which contain this input part by the User. A unit of the list of places consists of a picture of a place, its name and description.

5. Platform

Server platform: cross platform, deployed on Intel Cascade Lake x86, ubuntu 20.04 via docker container

Client platform: Android devices

Language: C# on server and Java & Kotlin on android client

Server frameworks and libraries:

Asp Net Core as main framework,
PostgreSQL as DBMS,
Autofac as dependency resolver,
EF Core for interactions with DB,
Newtonsoft json for JSON(de-)serialization,
Docker to keep app running independently

Client frameworks and libraries:

Retrofit as a REST client,
OkHttp simplifying web intersection,
RxJava for async tasks