Azuris

Azuris

Made by:
Marton Szilárd, Informatics III
Mózes Botond, Informatics III
Molnár Balázs, Informatics III
Leading teacher:
Szántó Zoltán

Table Of Content

1. Introduction

Nowadays people are tending to play a lot of games as relaxation. Theese games can help users to relieve stress and meantime they can improve their functional abilities such as concentration, reflexes, perceptibility, fast-thinking.

Thanks to the presence of oodles game developing environments every year tons of new indie(independent video game) games appears. Despite of this huge amount new games the host community is far greater which means a lot of potential for new game developers.

There are plenty environment to build up a game but we decided to use Unity because it has a lot of prebuilt features which can facilitate the developing experience. Unity gives users the ability to create games and experiences in both 2D and 3D, and the engine offers a primary scripting API in C#, for both the Unity editor in the form of plugins, and games themselves, as well as drag and drop functionality.

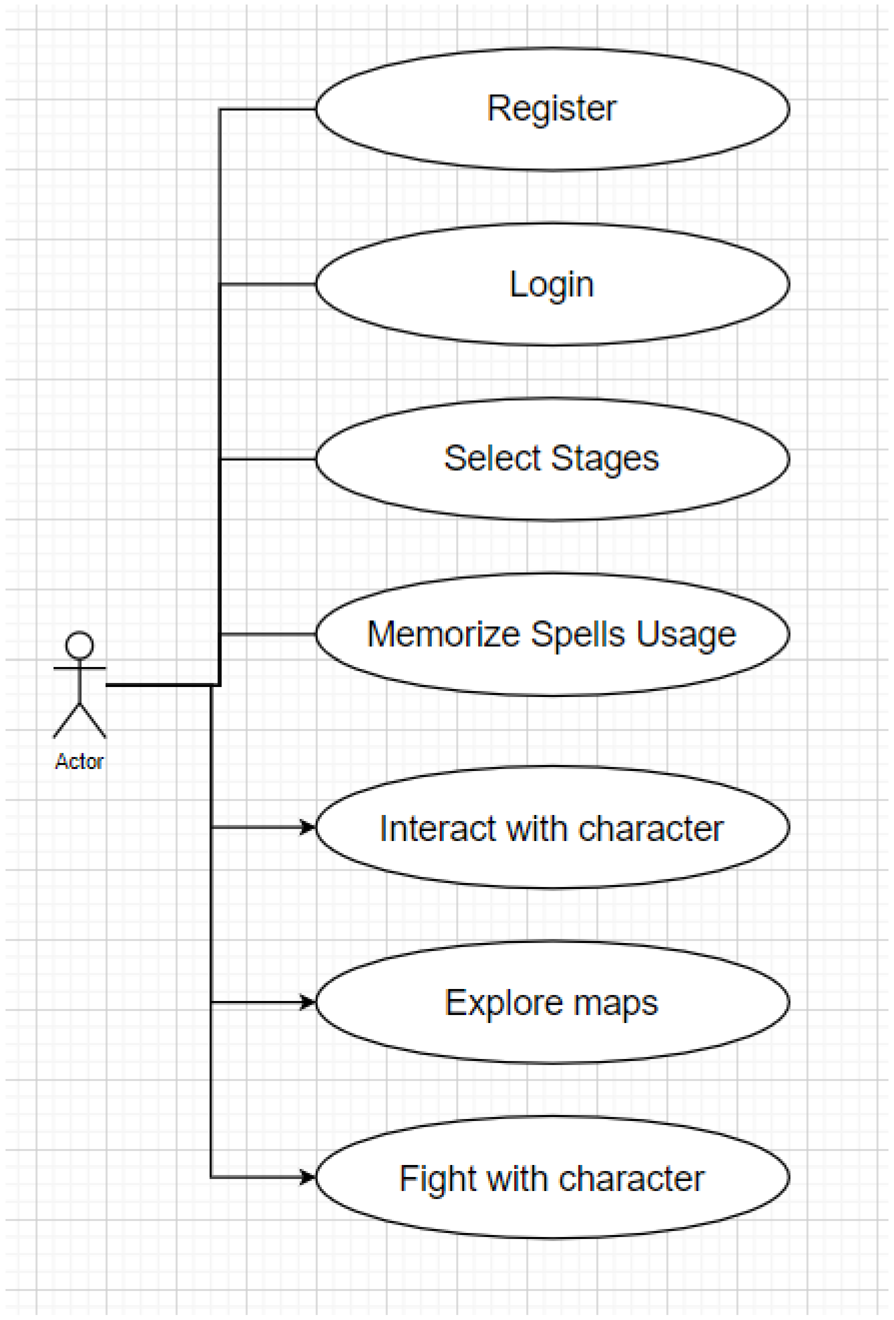
2. Targets

We would like to develope a game which deviates from usual game controls and gives a better user experience. The most important matter is to place the user in a magical world like he would be the character at Hogwarts in Harry Potter. The target would be to give an experience where the user could create the spell’s conjuring motion.

Furthermore the basic goals are the similar as the usual games, such as exciting, improving brain functionalities and memorie.

3. Requirement specification

3.1. Functional requirements



1. diagram - Use-case diagram

- **Register/Login:** The user must register in order to log in and start playing the game
- **Select stages:** The user has the ability to choose whatever stage he would like to play on the current map
- **Memorize spell usage:** The user can learn new spell motion before selecting stage
- **Interact with character:** The user can do whatever he want on the current stage with the player, the user can explore, fight, search and find items
- **Explore maps:** The user can play different maps and locations
- **Fight with character:** The user can fight and kill enemies in order to progress on the map

3.2. Non-functional requirements

Developing the game we decided to determine the following criteria:

- **Diversity**: the game should give the opportunity to users to do whatever they want in the playground
- **Efficiency:** we shouldn't make unnecessary time series comparisons
- **Scalability:** the game can be developed further just like the server which must be extended along with player base growth

3.3. Software requirements specifications

3.3.1. For server application:

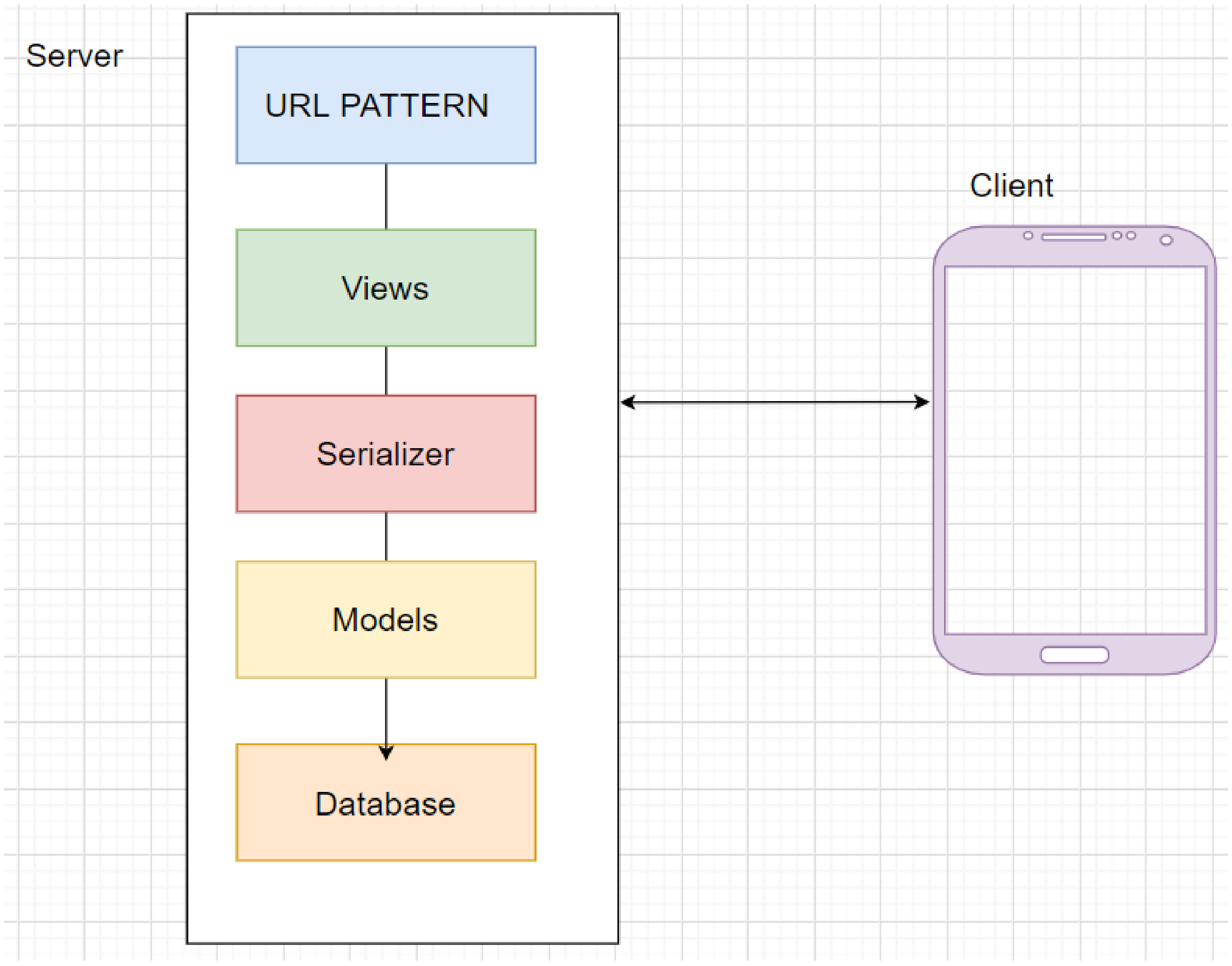
- Minimal requirements:
    - ~2.6 GHz, 4 core processor
    - 4 GB RAM
    - Any operating system
    - Fast (apr. 500 Mbps) internet access
- Optimal requirements:
    - ~2.6 GHz, 4 core processor
    - 16 GB RAM
    - Any operating system
    - 1 Gbps internet access

3.3.2. For client application:

- Android operating system at least 6.0 Marshmallow
- Internet access
- 100MB storage space

- 2GB RAM
- Quad core 1.2 GHz

4. System architecture



2. diagram – System architecture

We used this architecture in order to decrease the size of the app and maintain portability. The client application communicates with HTTP requests. These request URLs will be linked to specific views which processes the all sort of data and sends it to models which refresh the database and the database sends it back to models, then models to views. After all processes view sends back a HTTP response to client.

5. Project Management

Development phases were performed on using GitLab version control system because we were familiar with it. We had to struggle on front-end because the GitLab doesn't support big project files so we cannot push our project to this because the project was bigger than the allowed size(25 MB). We had to use a lot of resources to make up our features(pictures, sprites, assets) but when we used gitignore to ignore these files the on who tried to pull it had problems because couldn't start the game without these assets. So we had to use other alternatives on front-end.

We used Kanban board which supports issues and other features which helped us to keep tracking the project such as labels, milestones. Every team person had their own assigned issues to solve, than we reviewed each other's code to make sure everything is working fine.

6. Project Planning

Developing our game started with a planning phase which included brainstorming, ideas searching, checking similar projects and games. We planned the main functionalities, project structure, wireframes, UML diagrams and assigned every team member his to do list.

We had to investigate the possible ways to build up a game. After several choices we chose Unity. The main idea was an android game and thanks to Unity we could change platforms inside the environment. At first we had problems with switching platforms and after several weeks we had to rebuild our game from the bottom.
The Unity came with C#, so we had to learn it and operate with it.

On server-side we used Django because after short learning phase it proved to be easy and it was created for mobile and web services.

7. Detail structure

7.1. Server

7.1.1. Introduction

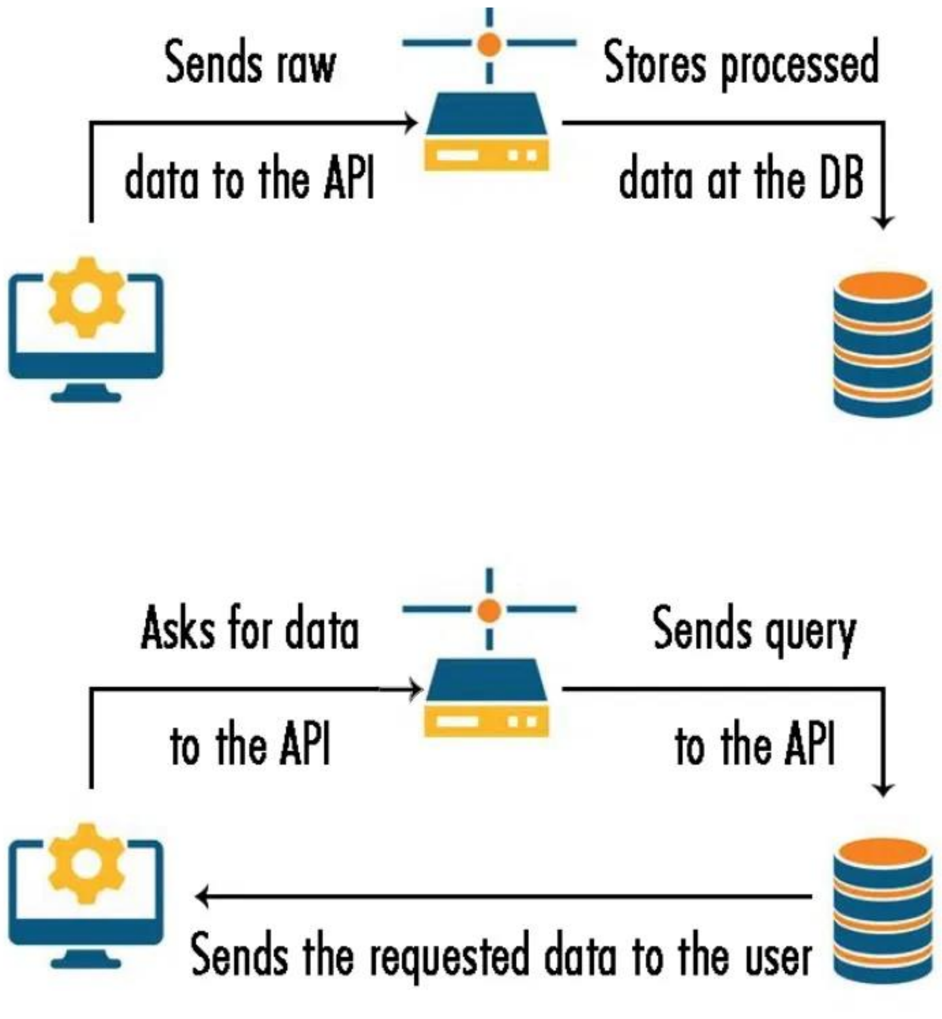
The server was implemented with the help of Django Rest Framework(DRF). The biggest reason to use Django REST Framework is because it makes serialization easy!

Django's serialization framework provides a mechanism for “translating” Django models into other formats. Usually these other formats will be text-based and used for sending Django data over a wire, but it's possible for a serializer to handle any format (text-based or not).

In Django, you define your models for your database using Python. While you can write raw SQL, for the most part the Django ORM handles all the database migrations and queries.

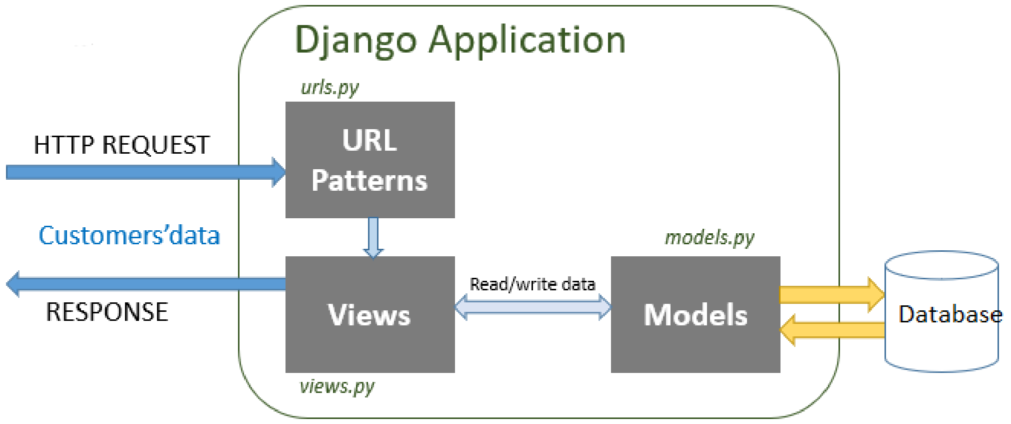
REST APIs let us send information back and forth between an interface and a database. We can use one REST API to feed any platform we want. This removes duplicity of code and helps to escalate it easily.

- Well documented
- There is a lot of free resources on the internet
- It has a great in-built interface



3. diagram – Django Architect

7.1.2. Architecture



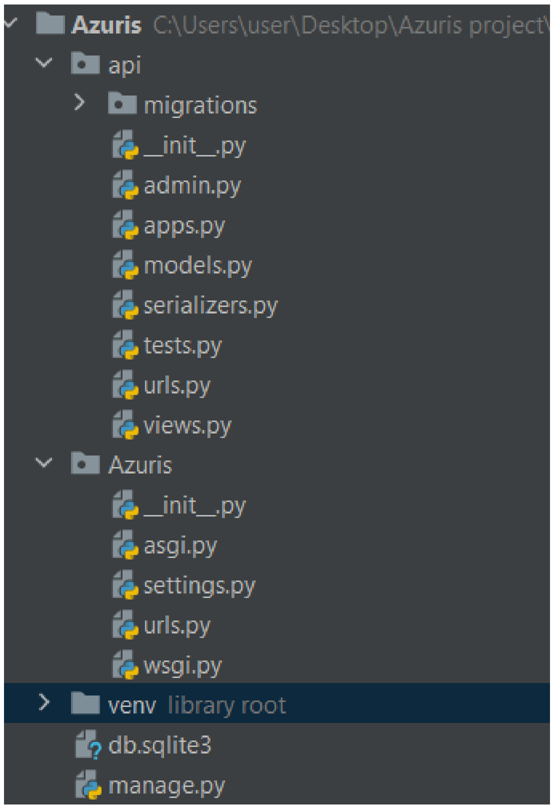
4. diagram – Django Rest Framework Architect

- HTTP requests will be matched by Url Patterns and passed to the Views
- Views processes the HTTP requests and returns HTTP responses (with the help of Serializer)
- Serializer serializes/deserializes data model objects
- Models contains essential fields and behaviors for CRUD(create, read, update and delete) Operations with Database

7.1.3. Technology

- Python 3.7
- Django 2.1.15
- Django Rest Framework 3.11.0
- Cryptography 1.2

7.1.4. Project structure



```
Azuris  C:\Users\user\Desktop\Azuris project\
  api
    migrations
    __init__.py
    admin.py
    apps.py
    models.py
    serializers.py
    tests.py
    urls.py
    views.py
  Azuris
    __init__.py
    asgi.py
    settings.py
    urls.py
    wsgi.py
  venv  library root
  db.sqlite3
  manage.py
```

5. diagram – File System

- api/apps.py: declares ApiConfig class (subclass of django.apps.AppConfig) that represents Rest CRUD Apis app and its configuration.
- Azuris/settings.py: contains settings for our Django project: Database engine, INSTALLED_APPS list with Django REST framework, Api Application
- api/models.py: defines api data model class (subclass of django.db.models.Model).
- migrations/0001_initial.py: is created when we make migrations for the data model, and will be used for generating database table.
- api/serializers.py: manages serialization and deserialization with RegistrationSerializer, AccountPropertiesSerializer, SpellPropertiesSerializer, SpellFileSerializer class (subclass of rest_framework.serializers.ModelSerializer).
- api/views.py: contains functions to process HTTP requests and produce HTTP responses

- api/urls.py: defines URL patterns along with request functions in the Views.
- Azuris/urls.py: also has URL patterns that includes api.urls, it is the root URL configurations.

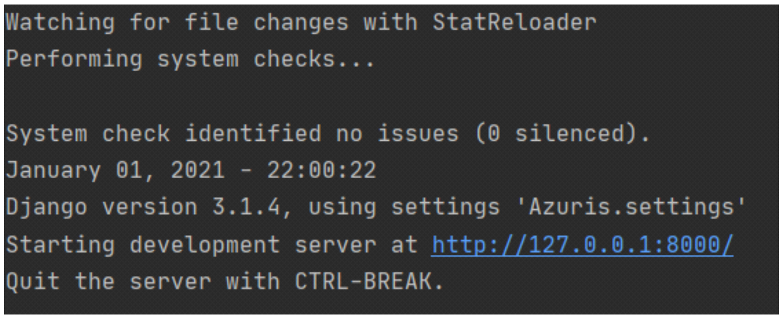
7.1.5. Usage

You need to start the server first.

Enter the file directory and write next code in cmd. Before this you have to download the dependecy.

...\Azuris> python manage.py runserver

If all is well you can see below.



```
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
January 01, 2021 - 22:00:22
Django version 3.1.4, using settings 'Azuris.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

6. diagram – Running Server

You can then retrieve the data. The following commands work without tokenization:

Only method \"POST\" allowed:

- http://127.0.0.1:8000/api/register/
- http://127.0.0.1:8000/api/login/

Only method \"GET\" allowed:

- http://127.0.0.1:8000/api/spell/
- http://127.0.0.1:8000/api/spell/%3Cint:id%3E/

You must log in an user and use a token to access the following options.

Only method \"GET\" allowed:

- http://127.0.0.1:8000/api/properties/

Only method \"PATCH\" allowed:

- http://127.0.0.1:8000/api/properties/update/

In each case, there is a field named 'type' that returns a 'successful' or 'error' message after the queries.

7.1.6. Authentication

Authentication is the mechanism of associating an incoming request with a set of identifying credentials, such as the user the request came from, or the token that it was signed with. The permission and throttling policies can then use those credentials to determine if the request should be permitted.

The Token authentication scheme uses a simple token-based HTTP Authentication scheme. Token authentication is appropriate for client-server setups, such as native desktop and mobile clients.

7.1.7. Password hash

By default, Django uses the PBKDF2 algorithm with a SHA256 hash, a password stretching mechanism recommended by NIST. This should be sufficient for most users: it’s quite secure, requiring massive amounts of computing time to break.

7.1.8. Database

SQLite

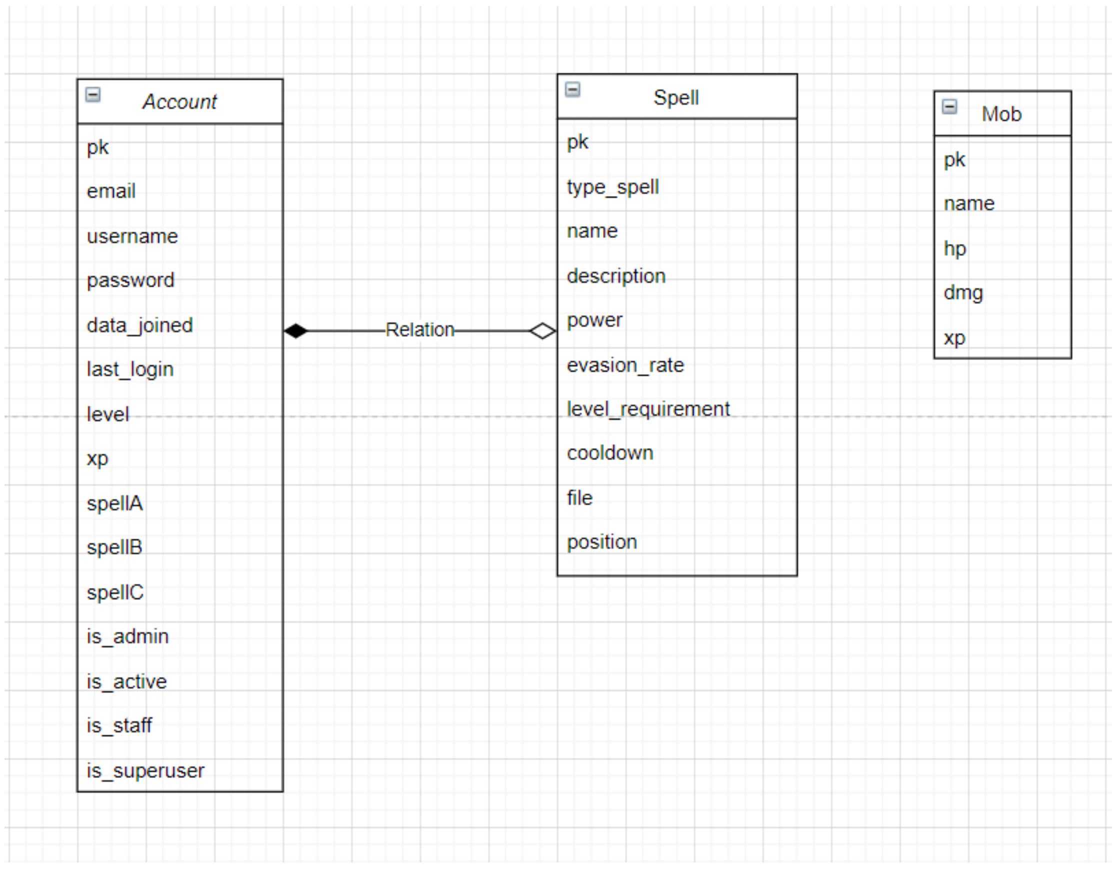
SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. It is a database, which is zero-configured, which means like other databases you do not need to configure it in your system. The code for SQLite is in the public domain and is thus free for use for any purpose, commercial or private. SQLite is built into all mobile phones and most computers and comes bundled inside countless other applications that people use every day.

Our database has three tables:

- Account: The account contains the registration of the player's data, along with the respective abilities. Two types of players can be registered, Admin type, and a Simple player. The Admin type of user has privilige against the simple type player, yet this type of user can only be created by the server's page. Every Account has the following value: pk, email, username, password, data_joined, last_login, level, xp, spellA, spellB, spellC, is_admin, is_active, is_staff, is_superuser

- Spell: This table contains the spells which the player can use. Every spell has the following value: pk, type_spell,name, description, power, evasion_rate, level_requirement, cooldown, file, position.

- Mob: This table contains the information about the Mobs. For now, only the admin has access to this table. Further future developing will be part of the connection with the frontend. Every Account has the following value: pk, name, hp, dmg, xp.



7. diagram – Database Class Model

7.2. Shape generation

We used a Python script to generate the shapes for each spell. Each shape is a time series, which contains x and y coordinates in the given time. First we created a sample from approx. 20 time points. This step is followed by interpolating these time series to get a new series with 128 coordinate pairs. These final time series will be compared in the client side with the time series determined(drawed) by the user.

7.3. Client

The game consist of 6 views:

1. Start menu
    - The user has three options to choose: Login, Register, Quit
    - At start the game shows this page
    - If the user is not registered can not be logged in
2. Login menu
    - Here the user can log in if there is any previously registered account
    - The login validation happens both on client side and server side
    - On client side the validation consist only from a null check, if there is no data given in input field
    - The server make the bigger effort, there is the full validation
    - If the user’s input matches with database data then the log in is successful
3. Register menu
    - Consist of 4 input fields: Name, email, password, confirm password
    - Same validation technique as in Login menu
    - Only valid emails can be registered
    - The registrable name can not be identical with a previously registered one
    - The email can not be registered if already in database
4. Home menu
    - The user can choose between stages
    - The skills can be viewed from here by tapping on the skills book
    - In the skills book the player can choose 3 skills to use in the upcoming game
    - In skills book can be read the description of the skills

5. Ingame
    - This part of the game consist more scenes
    - The starting point of the stage is in a house

- Leaving the house we can find our character on a tilemap where he can fight mobs, collect loots from chests, explore the map
- There are more parts of the stage, the player can freely cross their borders
- There is a cave, a dungeon where the player can enter in several rooms
- In the top room he can get into by standing on the top of switch button, this will open the door
- Once the player is in that room he cannot leave until he kills all the mob

The main game was part of a sequence of testing the Unity environment which was dedicated to learning the basics of unity. Another brick in the wall was the C#, another new language to learn, however it was new it was nothing but joy to learn it. C# is the main scripting language of the Unity. Once a script is written and assigned to an object Unity gives life to the code by creating a UI where we can set our parameters or call functions by occurring events.

The technique used was step-by-step, meaning that we worked it out from nothing and created every piece of the cake by ourselves.

The map was created by using Tiled. This is a software which is specified for map creating. We had to gather resource for making maps. Creating maps is like painting pixels, every pixel should be painted properly to get a bigger picture, this part took the most time and energy.

After creating tiles the second part was importing in unity, in this step we used Tiled2Unity to import it. This software imported automatically everything to ready to use tilemap.

Every piece of UI element was collected from the internet, some of them was modified for specific component. All movement related animation had to be created manually, for this Unity had an internal in-build animation system where we could manage to decide for every second which sprite(picture) should appear while running the animation. After this step in the Animator menu needed to create an animation tree where we could play an animation after an even occurrence.

The camera needed to be modified. At first it followed the character who was in the center of camera every time, after this we had to create camera boundaries in order to stay between map boundaries. Next thing was to create a cinematic style, which means we delayed the camera movement and the camera moved when the character was near to the edge of the camera and when the character reached the edge of the camera a delay started and after the delay the camera started following the character again.

The health system had two steps to be made, the first one was the creation of UI elements(hearts). In the left top corner the hearts are shown 3 of them. The second part was creating the functionalities, so whenever the main character got a hit the hearts will be half-emptied then emptied fully, and whenever the character lost all of his 3 hearth it dies. For hearth can be given any amount of health and the UI element reaction can be variety too.

The enemies was created statically which means they have a home position where they return after fight. Some enemy are area enemies, walking around, some are asleep. They start an action whenever the character get in their reaction zone, so they start following the player and when reaching the attack radius they start fighting, dealing damage to character. When the character got hit or the enemy they going to get knockbacked and they enter in stagger mode and they will restricted from any actions. The enemies AI is simple, when the character reaches the action zone the enemy will wake up and follow him and hit him. This animations(sleeping, waking up, walking, fighting) was all made by 2 hands, every part of the animation needed to create separately.

7.3.1. Communication with the server

The communication with the server side of the application was implemented with the help of the Unity Networking package. With the UnityWebRequest component we could send GET/POST/PATCH requests. The incoming message format from the server was Json, after receiving that we had to parse it, with the JsonUtility package.

Registration: HTTP POST request
- Data fields: username, email, password, password confirmation
- Response: Success or Error message

Login: HTTP POST request
- Data fields: email, password
- Response: if the given credentials were approved then we get back the information about the account along with the token, which is used to modify the player data, otherwise we get an Error

Updating player data: HTTP PATCH request
- Header: token
- Data fields: at least one of the followings: level, xp, spellA, spellB, spellC
- Response: Error or Success

Requesting all spells: HTTP GET request
- Response: the IDs of the spells

Requesting a specific spell: HTTP GET request
- Data fields: id
- Response: id, name, description, type, power, evasion rate, cooldown, position, shape

7.3.2. Shape comparison

We choose to compare the shapes with the DTW(Dynamic Time Warping) algorithm due it's simplicity. One of the advantages of this algorithm is that it can compare two time series with different lengths. First we normalized the drawed and the spell's shape. At first we used MinMax normalization, but after testing the comparison was evaluated with false result, because it is realized between two border values.

$$\frac{value - min}{max - min}$$

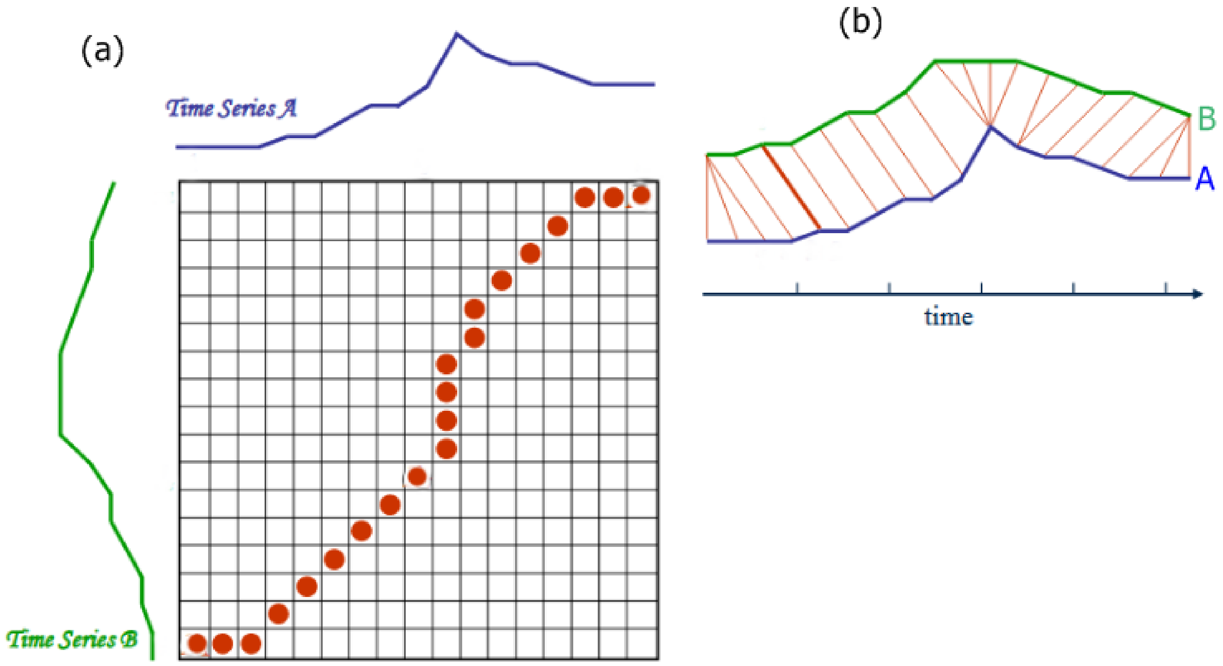
8. diagram – MinMax Normalization Formula

At second try we implemented the ZScore normalization, which gives better result because it uses the standard deviation and the mean of the x and y coordinates.

$$\frac{value - \mu}{\sigma}$$

9. diagram – ZSCORE Normalization Formula

After normalization the DTW algorithm compares the two time series. In the first step it builds a 2D matrix in which will be stored the distances. It compares each point and calculates the Eucliden distance between the two coordinates in two dimension. The next step is to compare the three neighboring costs and find the minimum. The addition of the sum and the minimum cost will give the current cost. In the last step the shortest distance is in the [n,m] indexes from the distance matrix, where n stands for the first shape's length and m stands for the second shape's length. This value is divided by the sum of n and m, this will give us the shortest distance between the two time series.



10. diagram – DTW Algorithm

7.3.3. Triggering the spell

If the distance between the two time series if lover than 0.25 than we activate the given spell. There are 3 types of spells by position:

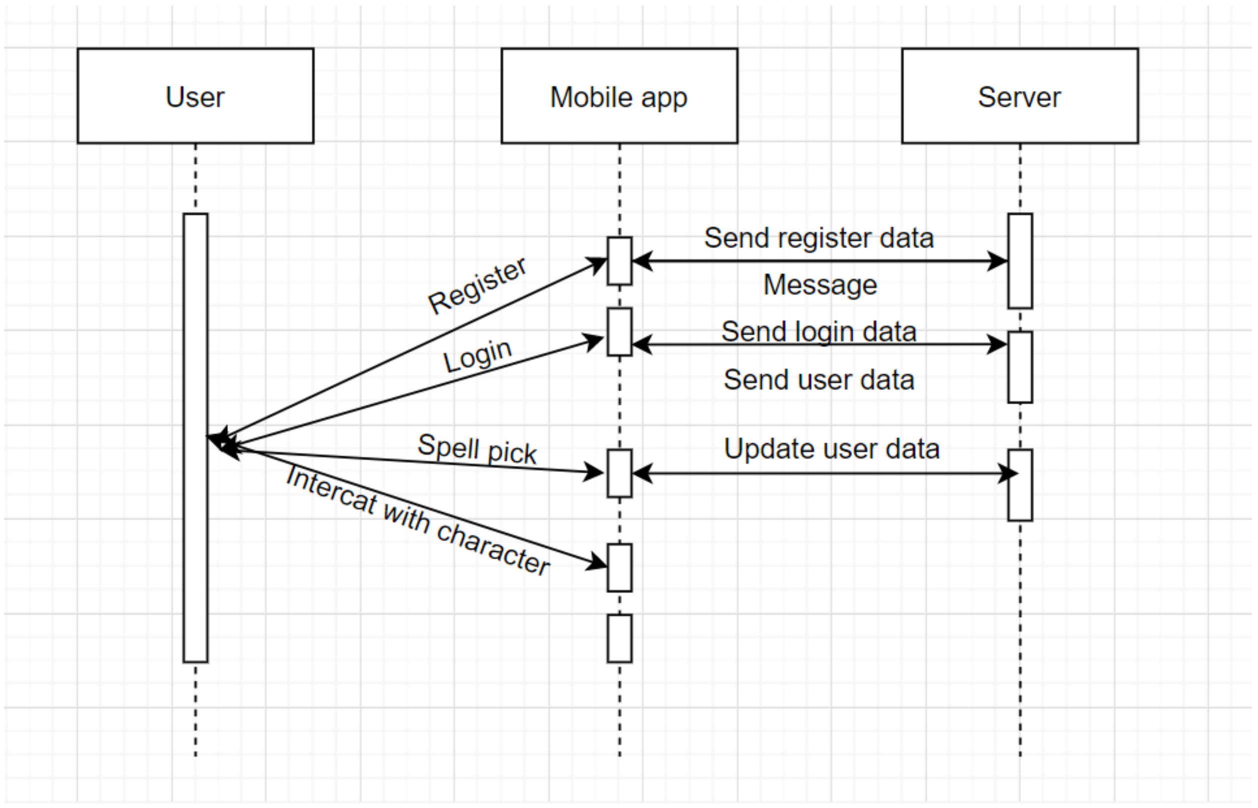
- moving spell (eg. fireball) which starts from the character and starts moving in the direction where the character is facing
- auto targeting spell (eg. freezing cloud) which searches for the closest enemy and appears on them
- self targeting spell (eg. hex shield) which appears on the main character

There are two types of spells which are offensive and defensive. The some of the offensive spells have a evasion rate which represents the chance of spell failure. The defensive spells only have effect on the main character and can give buffs such as shield or heal.

Each spell has a cooldown time given is seconds. When the spell is on cooldown it cannot be triggered.

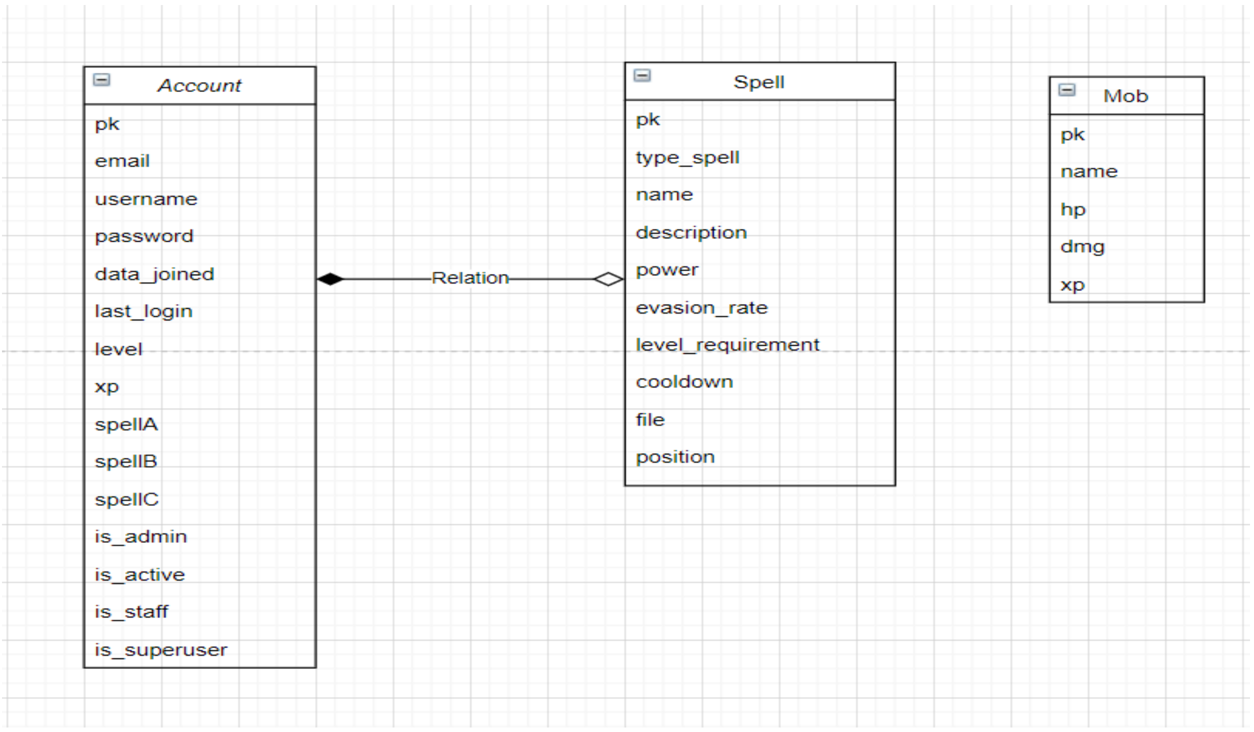
8. Diagrams

8.1. Sequence diagram



11. diagram – Sequence Diagram

8.2. Class diagram



12. diagram – Class Diagram

9. Further development potential

- Item system
- Optimized DTW algorithm
- Multiplayer
- Multiple type of weapons
- Diverse maps
- More Skills
- New mobs
- Quests
- Back-end refactoring
- Inventory expanding
- Storyline
- Progress saving

10. Summary

In conclusion, we can confirm, that we achieved most of our targets. This project was a real challange for us, which we managed with succes. Even for the smallest things, we had to collaborate, make discussions severel times. The most important lesson turned out to be that every project needs to be well planned in advanced.

11. Bibliography

https://www.django-rest-framework.org/tutorial/quickstart/

https://docs.djangoproject.com/en/3.1/topics/

https://medium.com/django-rest/logout-django-rest-framework-eb1b53ac6d35

https://www.sqlite.org/arch.html

https://bezkoder.com/django-rest-api/

https://seanba.com/tiled2unity

https://www.mapeditor.org/

https://www.codecademy.com/articles/normalization

https://app.diagrams.net/