

**Final Project Report**

**Cloud-Powered Event Management System**

Done by:

Amen Azat, 21B030774

23.11.2024

Almaty, 2024

**1. Executive Summary**

The goal of this project is to develop a secure, scalable and efficient event management system using various Google Сloud services. The app will allow users to create, manage and attend events by providing features such as registration, ticket purchase, notifications and real-time updates.

**2. Table of Contents:**

* Introduction
* System Architecture
* Table Entities
* Development Process
* API Design and Implementation
* Database Design and Optimization
* Event-Driven Architecture
* Machine Learning Integration
* Security Measures
* Scalability and Performance
* Challenges and Solutions
* Conclusion
* References
* Appendices

**3. Introduction**

**Background:**

* Google Cloud Platform is one of the best cloud platforms on a par with platforms such as Amazon Web Services (AWS), Microsoft Azure. The main advantage of such platforms is scalability, flexibility and fault tolerance. Also, an important advantage of such cloud platforms is a wide range of ready-made services and tools, which significantly saves development time and resources.

The Google Cloud Platform offers a user-friendly console with a user interface and analytics tools. GCP includes technologies such as the Compute Engine, App Engine, Cloud Kubernetes Engine, Cloud Function and Cloud Endpoints. Various cloud storages such as Cloud SQL, Cloud Storage, Firestore for data storage. And a wide range of different services, technologies and APIs.

**Project Goals:**

1. Creating a database to store users, events, tickets, etc.
2. Implement API endpoints for user registration, event creation, and ticket purchase.
3. Containerization of the application using Docker and Kubernetes.
4. Creating an event-driven architecture using Pub/Sub and Cloud functions.
5. Protecting the application with authentication, data authentication, and HTTPS.
6. Scaling the application using GKE features.

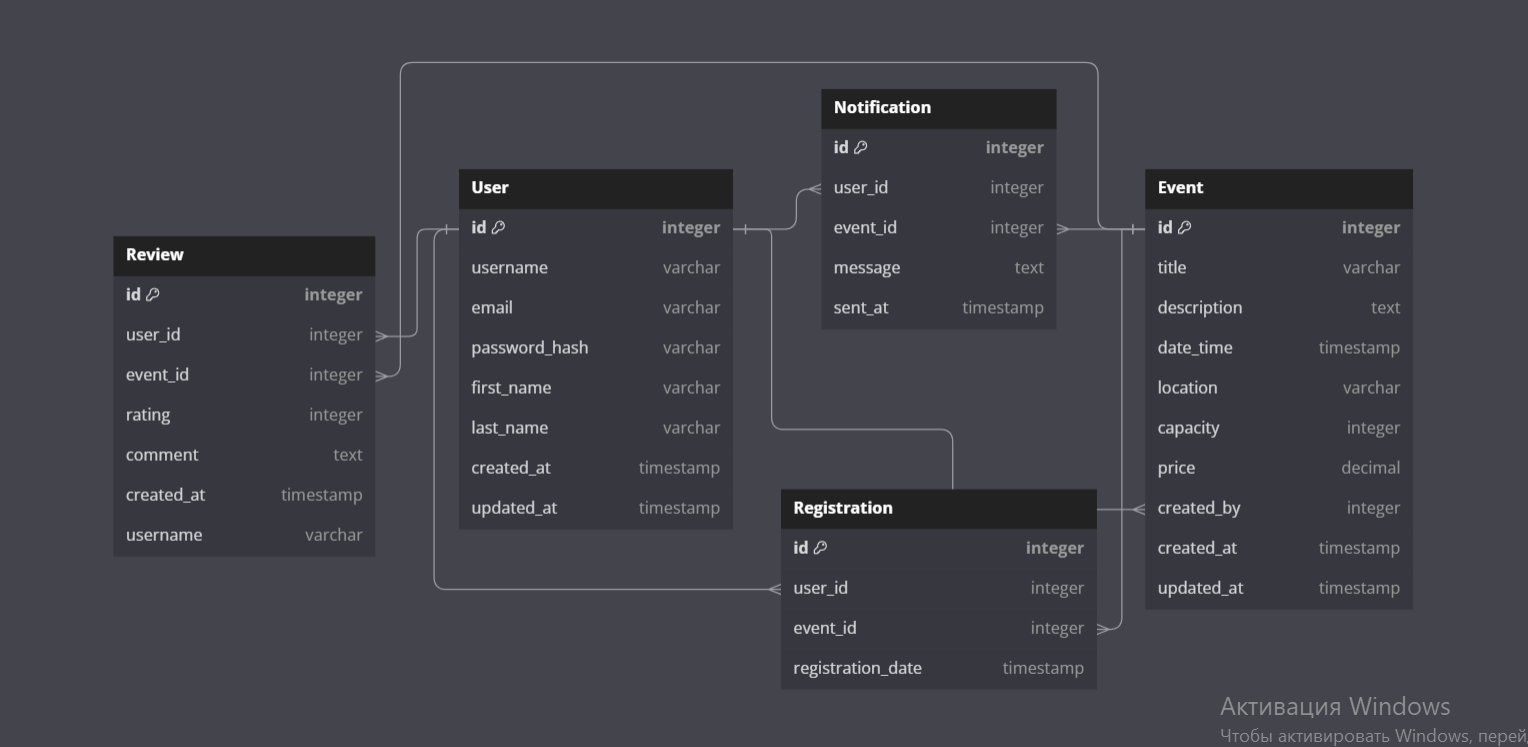
**Scope:**

* In my project included various technologies such as Flask (for backend), React (for frontend), Docker (for containerization) and GCP services that I listed above. This all helped me to develop event-driven, scalable and flexible web application. The limitations are: time frame set by the teacher, some requirements for the implementation of the project, as well as limited financial opportunities.

**4. System Architecture**

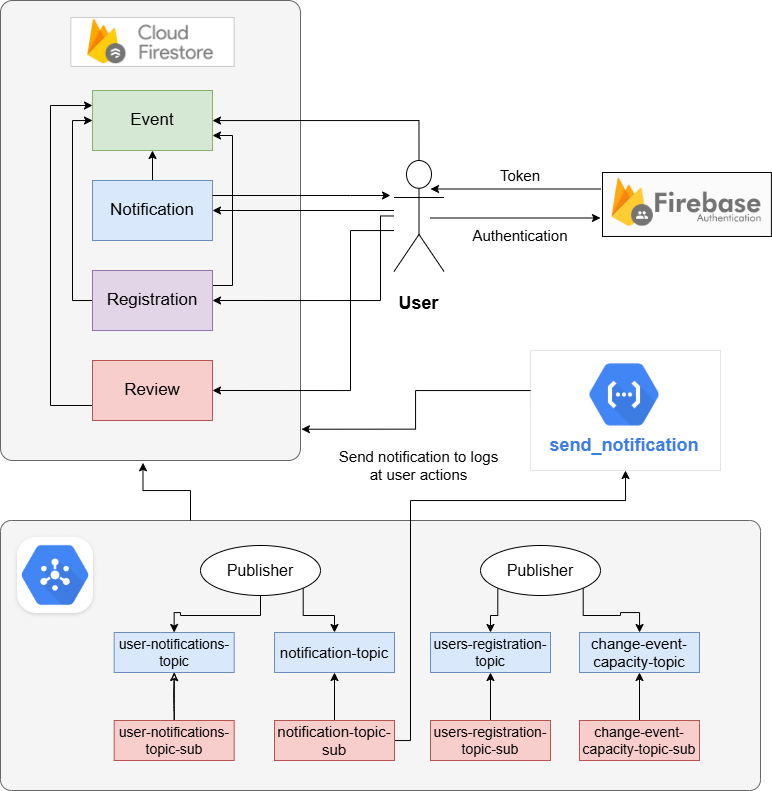
The architecture of the system is event-driven. Below I leaved diagrams with key components of the event management system:

* **Database Diagram** with entities and their relationships:



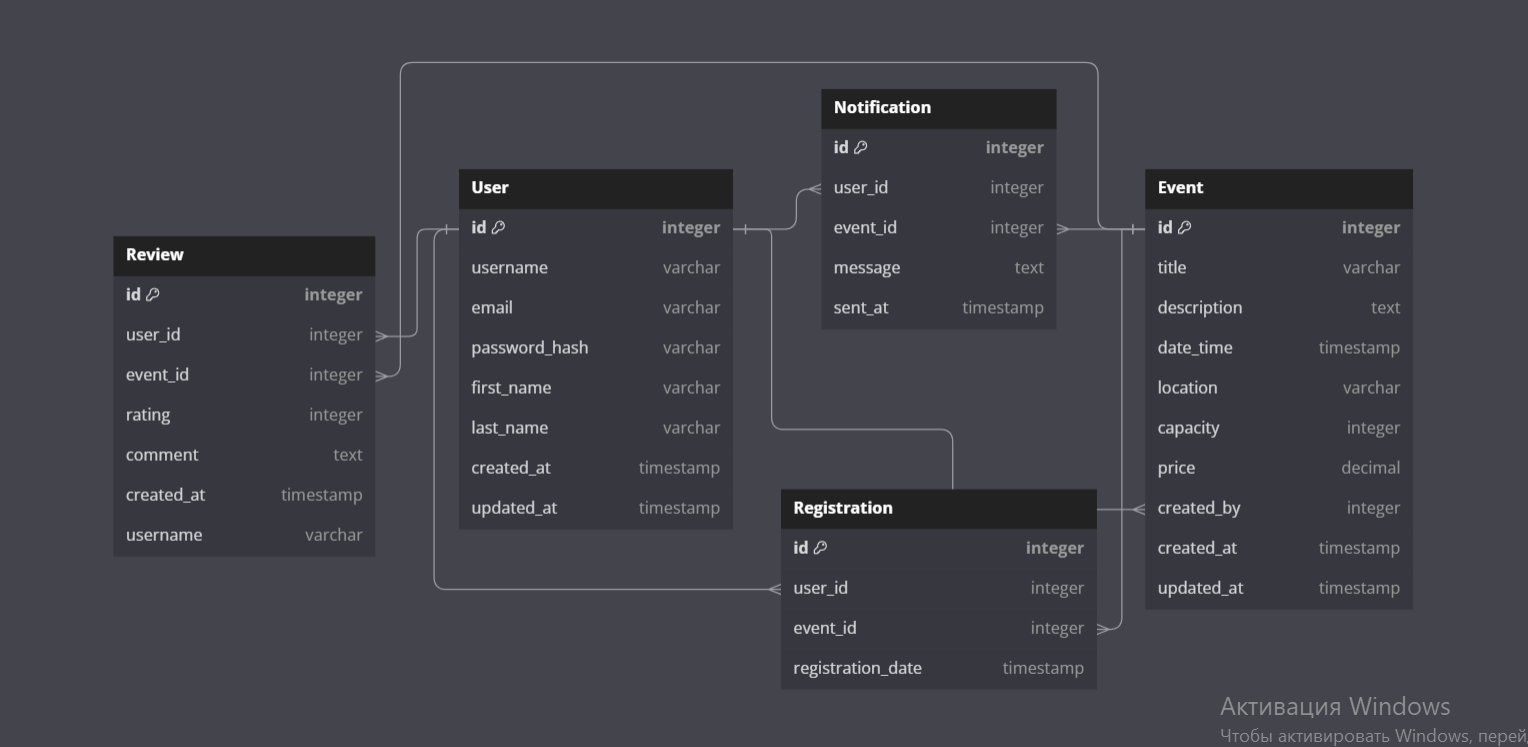
**Link:** <https://dbdiagram.io/d/Event-Management-System-67568cbbe9daa85aca14f4ce>

* **System Design Diagram** with Google Cloud services:

****

**5. Table Entities**

Instead of manually writing down all entities and their relationships, I used the DbDiagram platform *(Free online tool for drawing entity relationship diagrams by writing code)* to create them. In the picture below, you can see each entity and their connections between each other.



The main one is **User**; nobody can exist without User. Because only user can create events and manage them, leave reviews and register to events. Also, if there is no user, then notifications will have nowhere to go.

**Event** also very important, because all system built around events. User can create and manage events, leave review to event and register to event.

**Review** and **Registration** is related to event parts because they are really part of event. Users leave review to event and register to event.

And **Notifications**, its also important part of my app, because notifications sending when user do something in the system. They are sent to the logs and to the users, for example when User events got reviews and when someone register to User events.

Also, I want to share link to this diagram. So, you can explore it by yourself:

* <https://dbdiagram.io/d/Event-Management-System-67568cbbe9daa85aca14f4ce>

**6. Development Process**

**Technologies Used**:

In my project I used following technologies:

1. Flask – python framework, used to develop server side of my project
2. React – javascript framework, used to develop client side of my project
3. Docker – used for containerization my application
4. Firebase (Firebase Authentication and Firestore) – used for users authentication and data storage
5. Google App Engine – for deploying my server side
6. Google Cloud Functions – serverless functions used to logging user actions in system
7. Google Cloud Pub/Sub – used to build even-driven system
8. Google Cloud Endpoints – used to identify my app endpoints

**Implementation**:

* **Google Cloud SDK and Cloud Shell:**

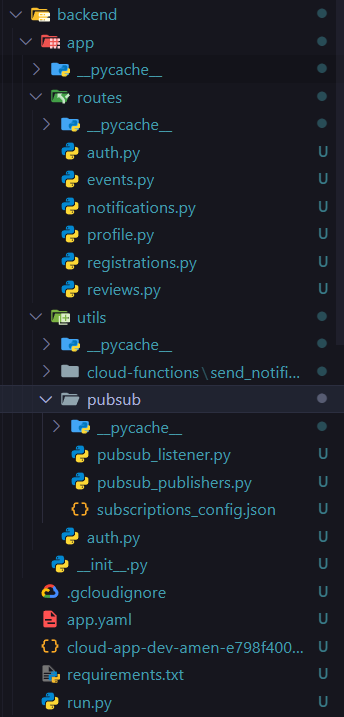
The Google Cloud SDK is a powerful tool for working with Google Cloud and its services directly from the terminal of a local device. Therefore, I installed the Google Cloud SDK on my device to interact directly with the Google Cloud Platform.

Cloud Shell is a direct console, a development environment from Google Cloud, which is located right in the console itself. The first option was more convenient for me, so I used it only.

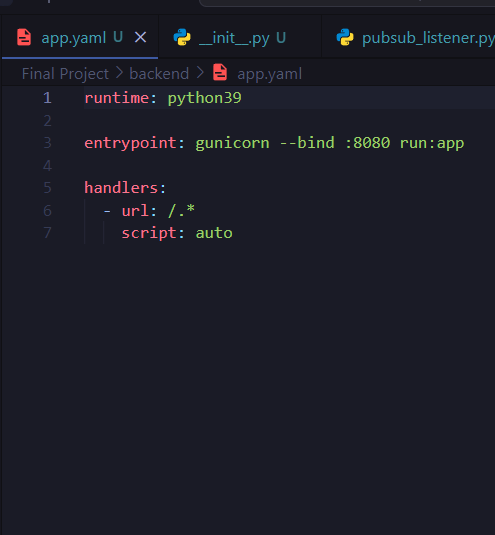
My choice fell on the Google Cloud SDK, because it is much faster and more convenient for me, so I can access some resources or GCP data directly from the IDE terminal in which I write code.

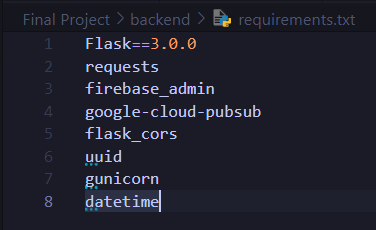
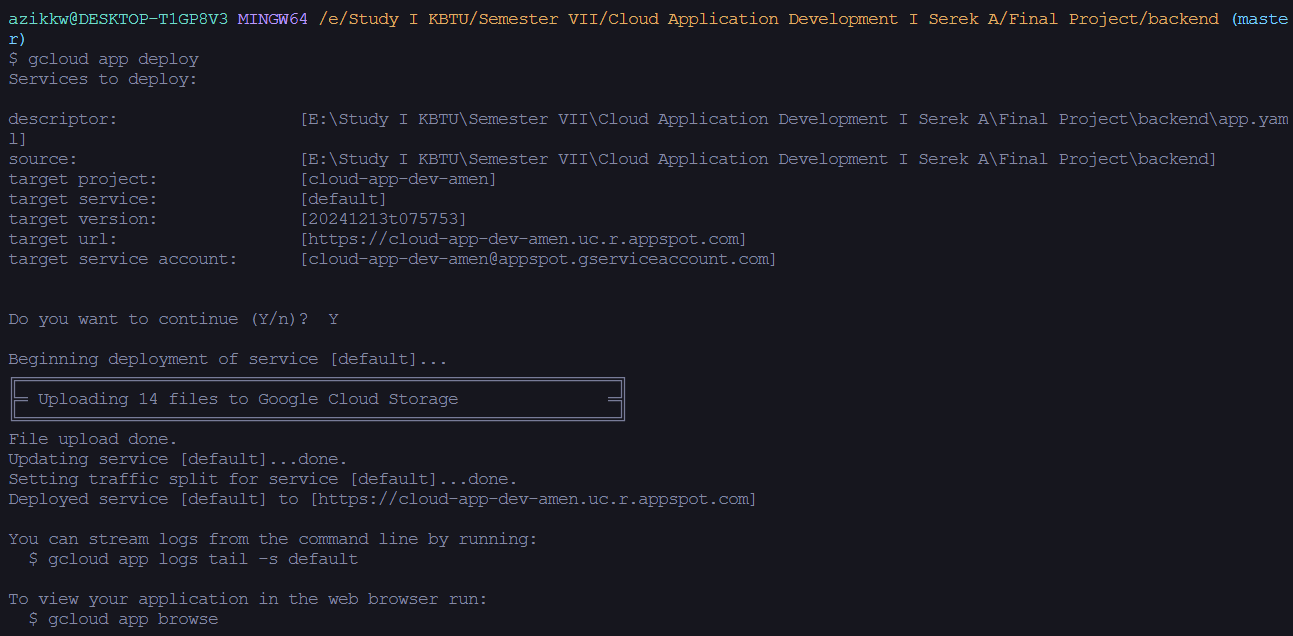
* **Google App Engine:**

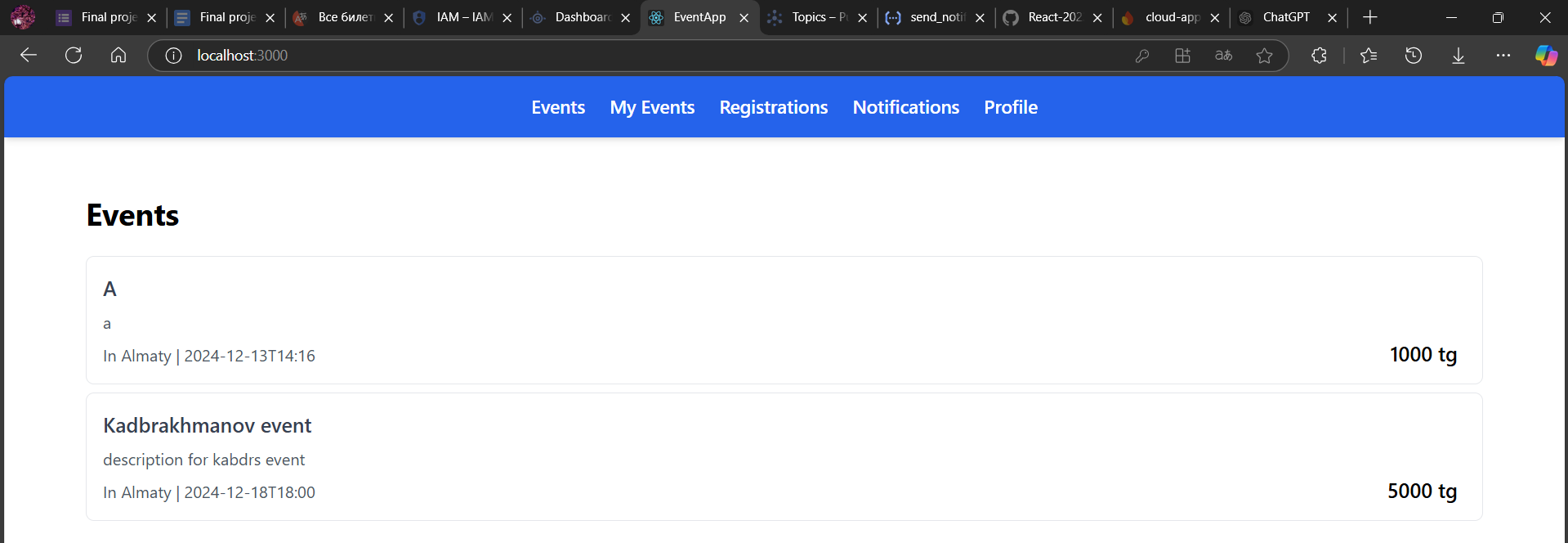
1. First thing I did, it of course writing code my project. I used flask framework and firebase for that.

****

1. Secondly, after finishing developing, I created app.yaml configuration for deployment to App Engine:



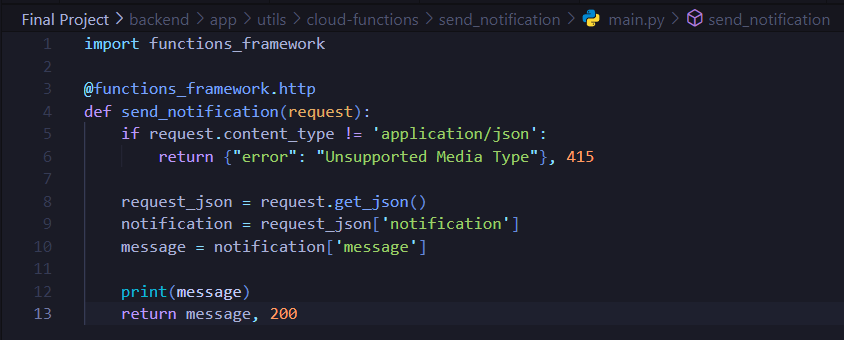
1. Next, I added requirements.txt with all libraries and dependencies for project.****
2. And finally, deploy application to app engine using gcloud app deploy:****
3. And finally, front side of the project works properly, and it means that back side also works perfect:



* **Google Cloud Functions:**

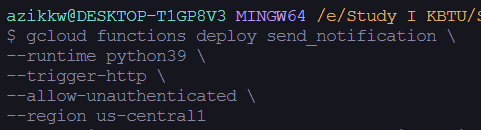
I implemented **“send\_notification”** cloud function to my system**:**

1. Firstly, I created a function code using python:

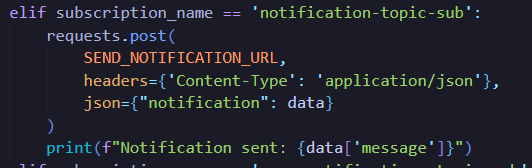


This function sends notification (in logs) related to all actions in the system, such as authentication, event managing, registration to events and reviewing.

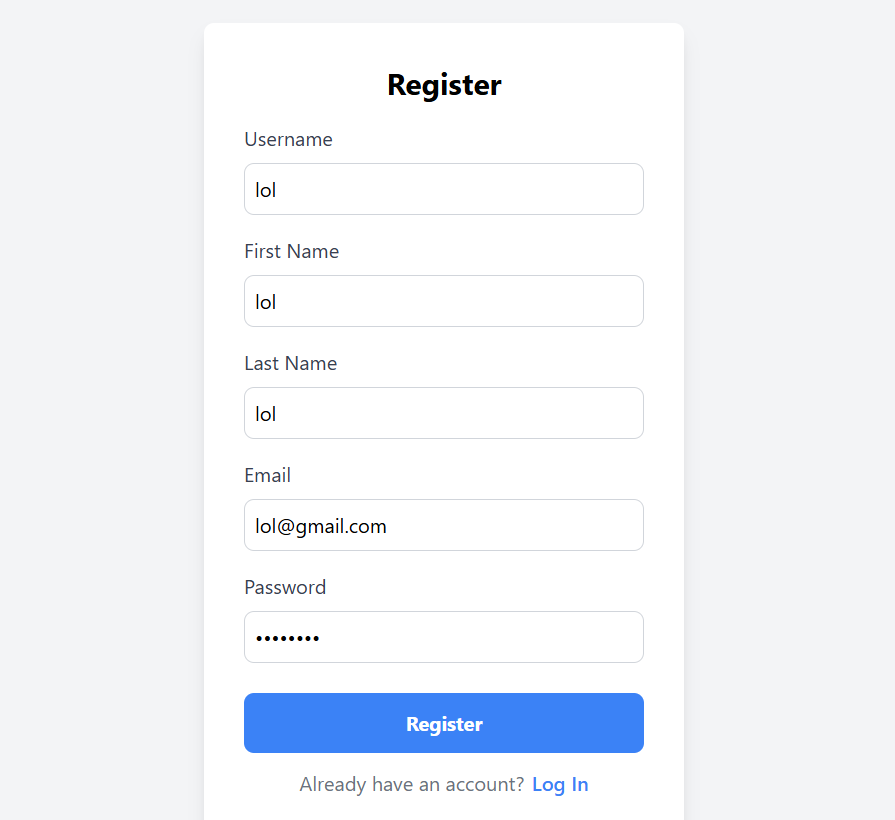
1. Add requirements.txt file with dependencies: 
2. Deployed cloud function using following configuration

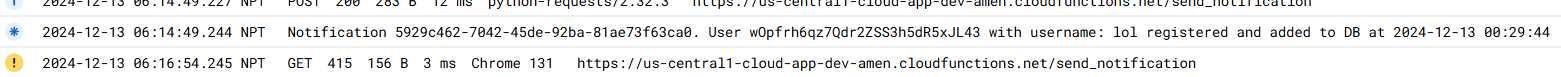


1. And finally integrated cloud function to the application using pub/sub



1. Function works real example:

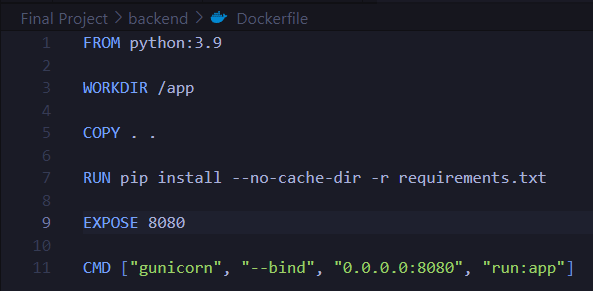


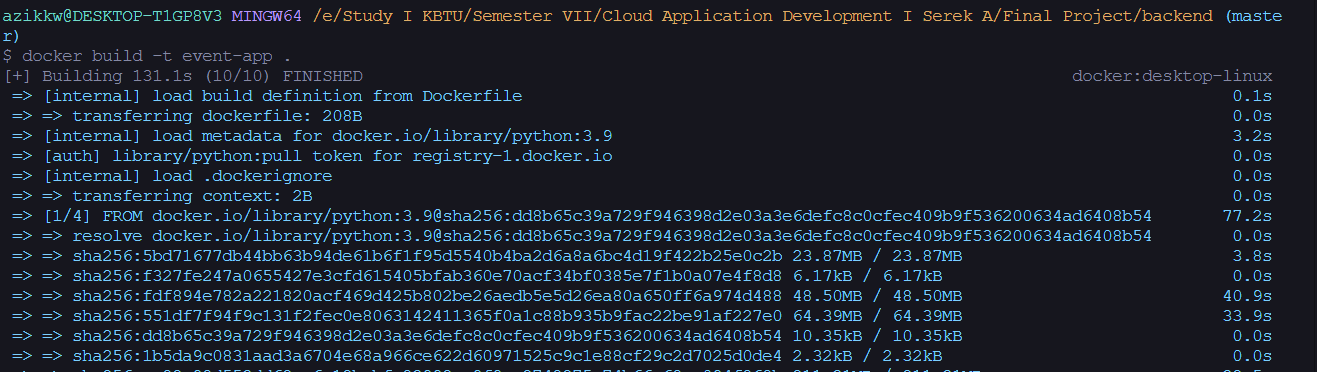


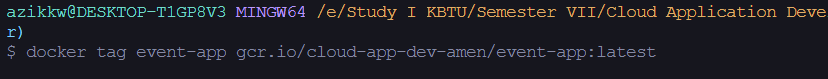
* **Containerization:**

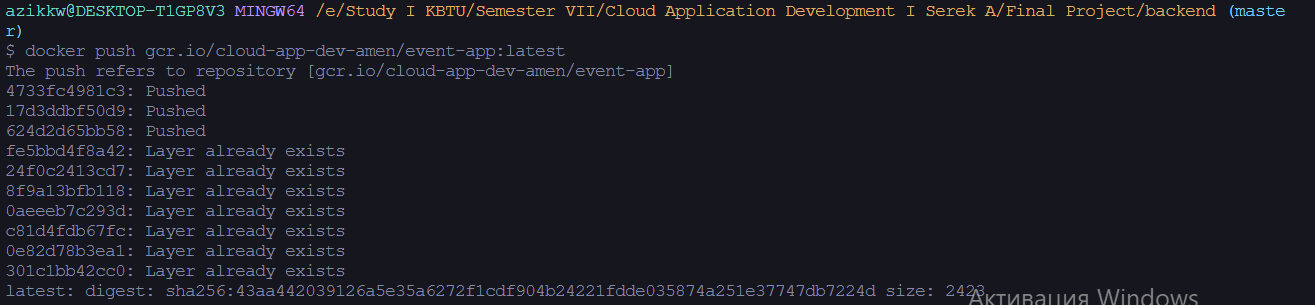
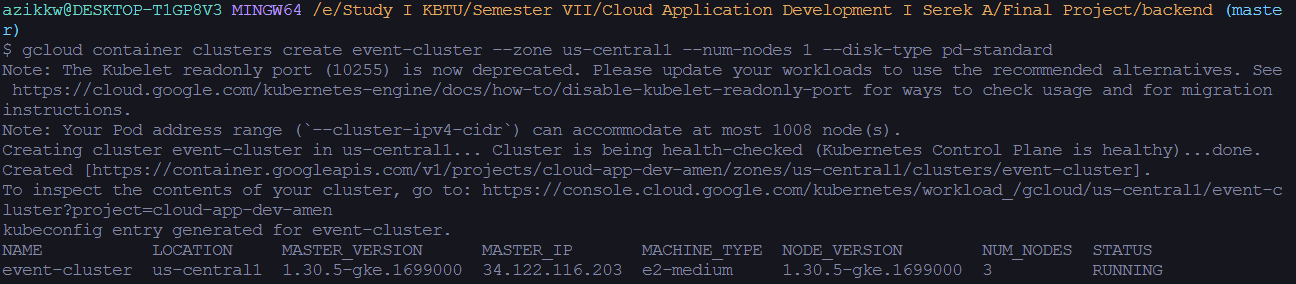
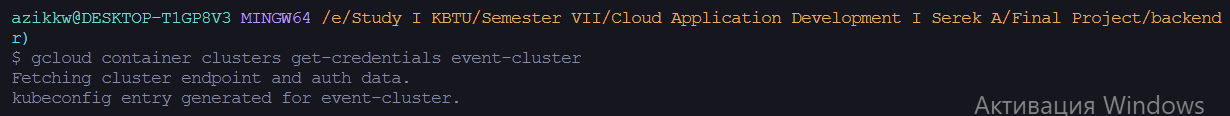
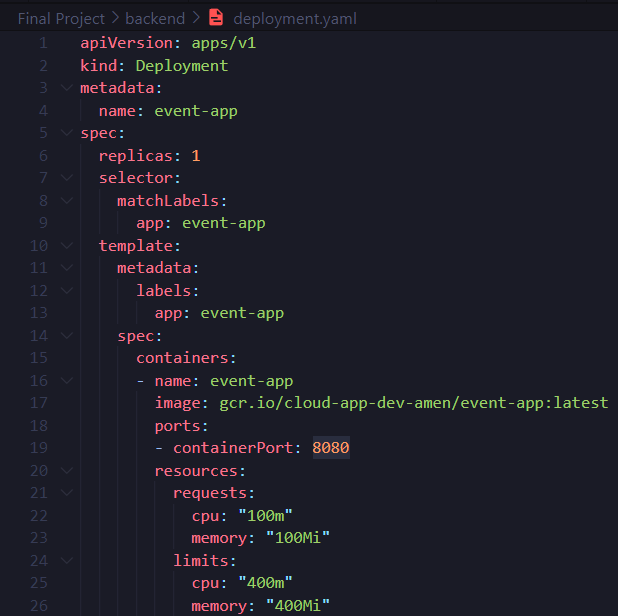
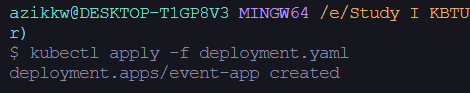
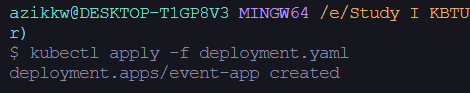
Steps taken to deploy the containerized application on GKE.

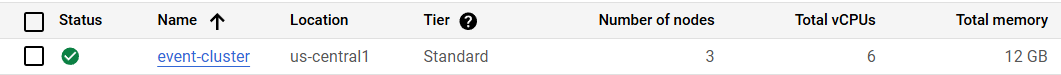
1. Create a Dockerfile with instrucations for creating Docker Image:



1. Create a Image using docker build:
2. Tag Docker Image to send it to Google Container Registry (GCR):

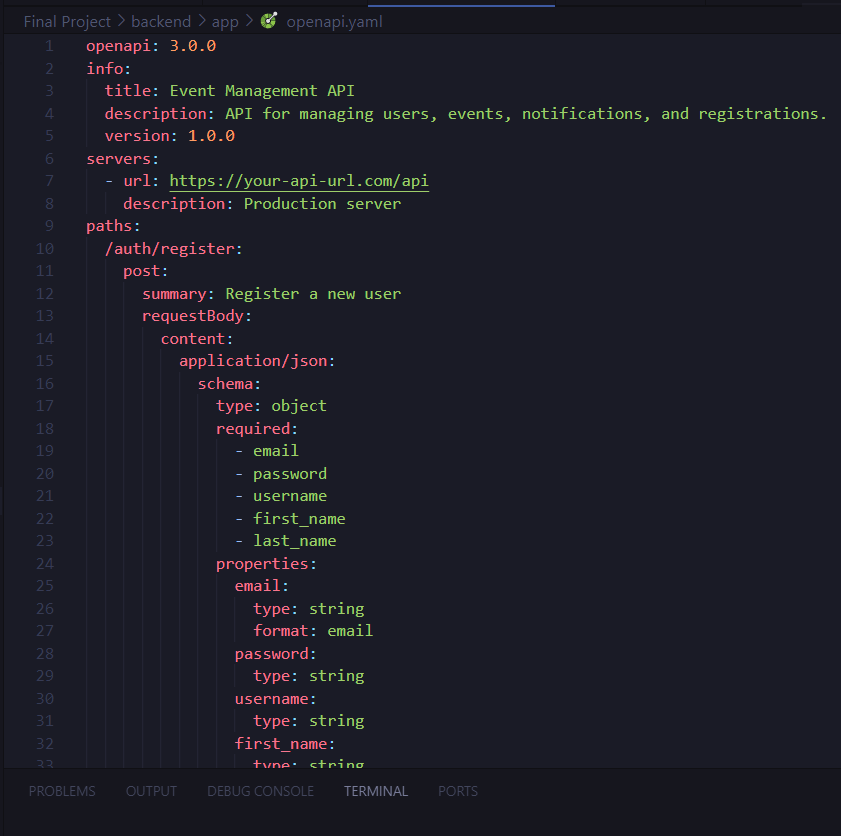
****

1. Push Image to GCR:
2. Next step is to create kluster in Google Kluster Engine (GKE):
3. Connecting to event-cluster cluster:
4. Next, create deployment.yaml with deployment instructions for Kubernetes:
5. Deploying the application using deployment.yaml:
6. And finally, check deployed app:



* **Google Cloud Endpoints:**

Step by step implementation of Google Cloud Endpoints:

1. The first step is create openapi.yaml configuration file in the project directory:

This image does not contain all openapi.yaml. As you can see on image, in this file we need to write version of swagger, info about your API, host and schemes, after that you write Endpoints and describe them including method type, parameters, responses and security.

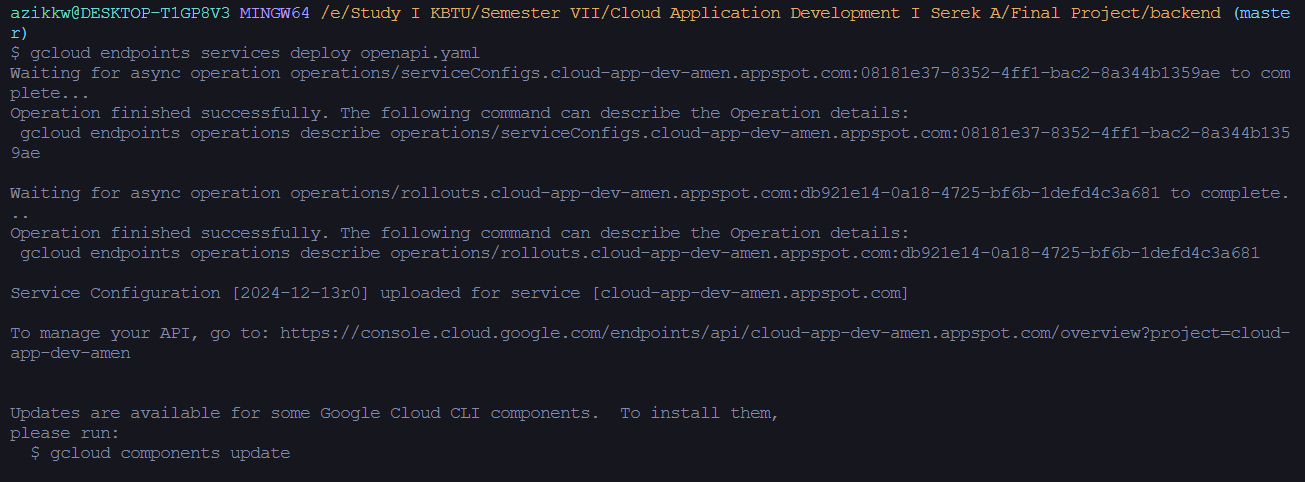
1. **Important to now.** The host name must in this format: “[YOUR\_PROJECT\_ID].appspot.com”, otherwise it doesn’t work. Below official documentation:

Изображение выглядит как текст, снимок экрана, Шрифт

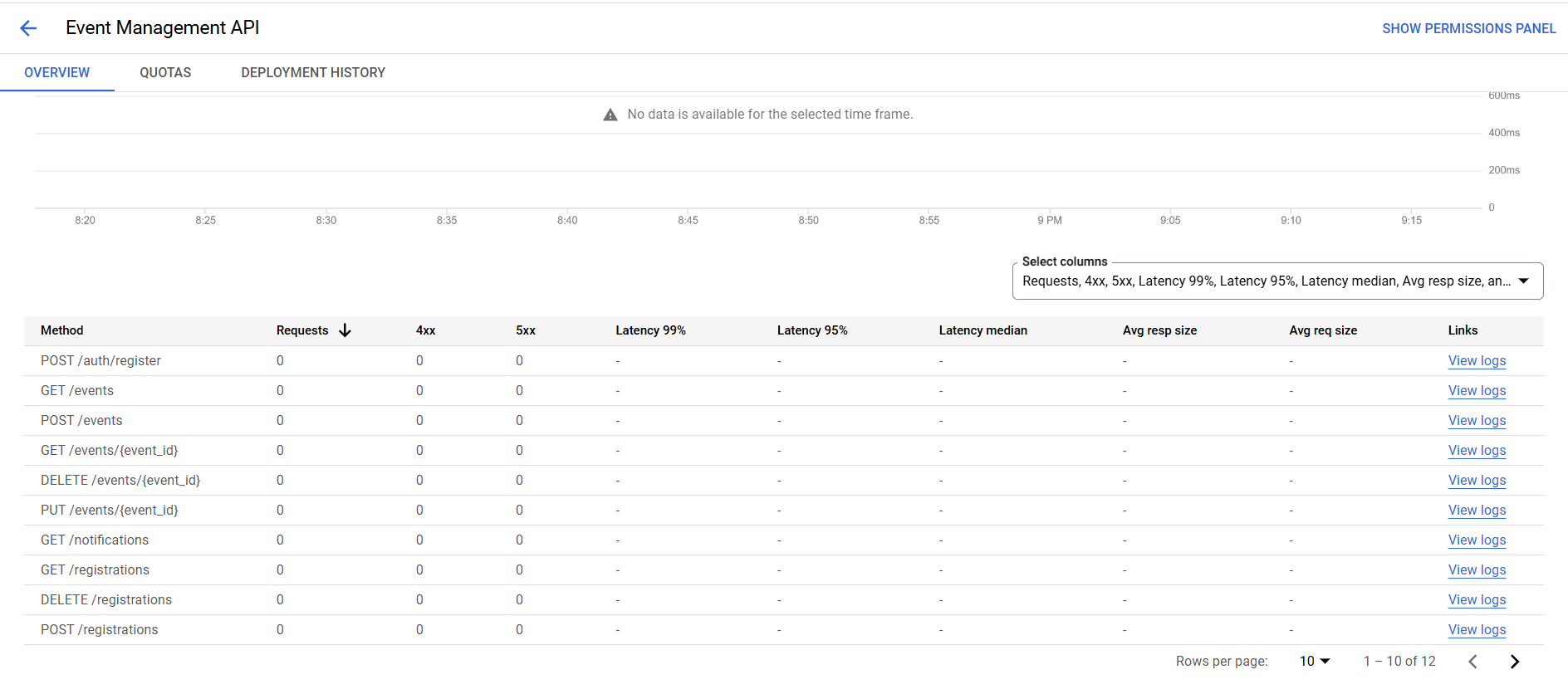
Автоматически созданное описание

1. Once openapi.yaml is complete, deploy your API to Cloud Endpoints using following command:

gcloud endpoints services deploy openapi.yaml



1. And finally, as a result Event Management API service created. There you can monitor your endpoinst, APIs:



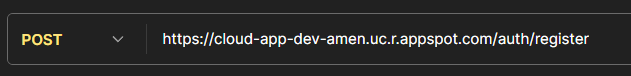
**7. API Design and Implementation**

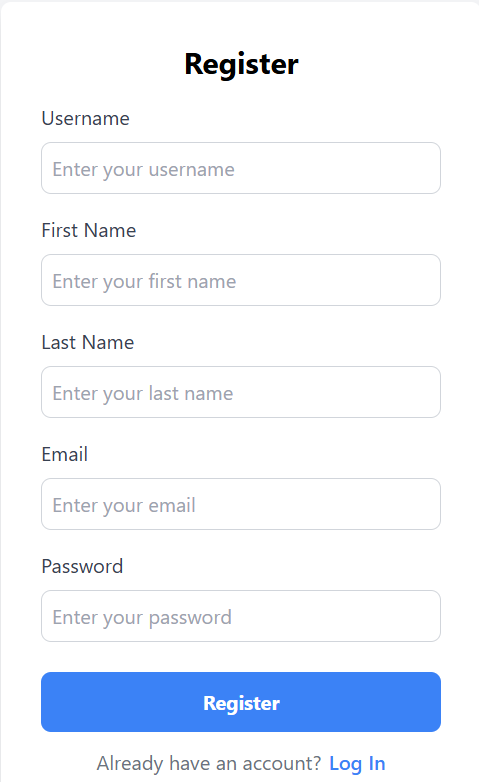
1. Methods list:

* **POST:** Create user (Register), Create event, Create registration to event, Create review
* **GET:** Get user, Get event/events, Get my-event/my-events, Get registrations, Get reviews, Get notifications
* **PUT:** Update event
* **DELETE:** Delete events, Delete registrations

2. Endpoints URL:

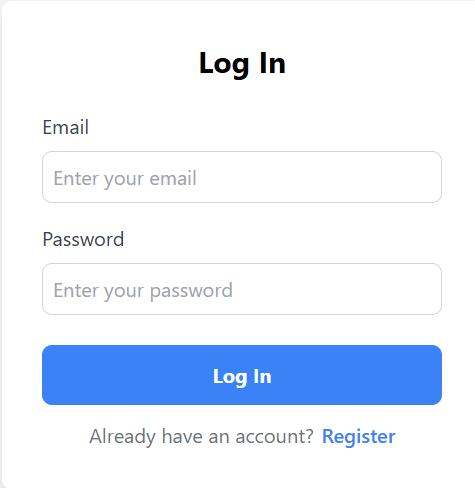
* **REGISTER endpoint:**





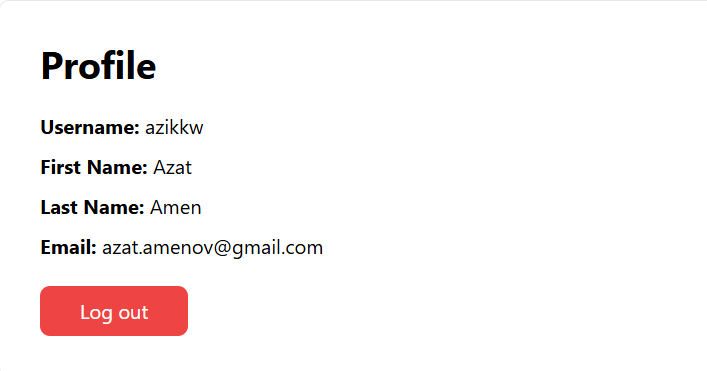
Functionality: This endpoint creates User and add it to database.

* **LOGIN:**



* **GET Profile endpoint:**

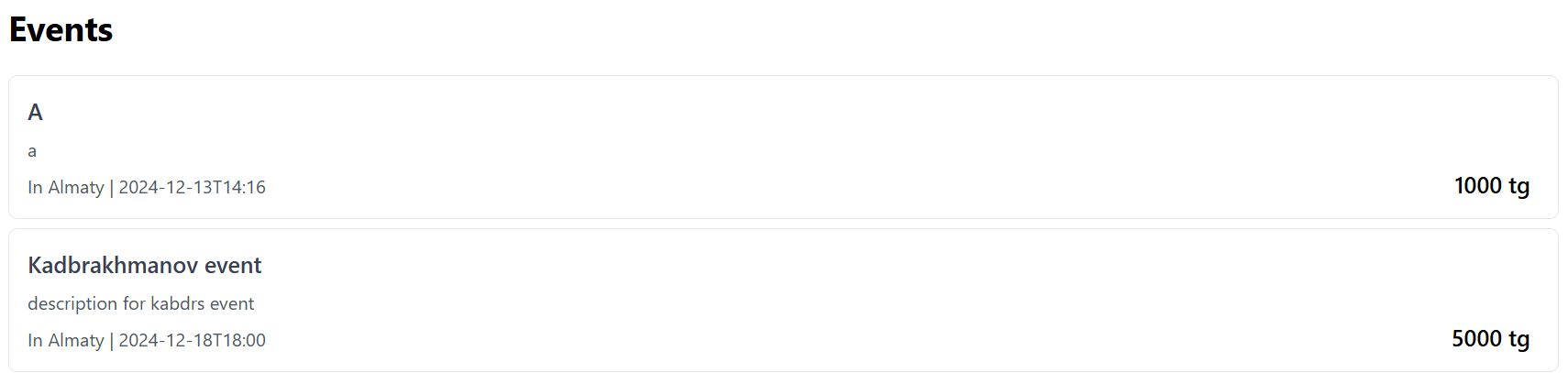




Functionality: Returns current user data by user\_id

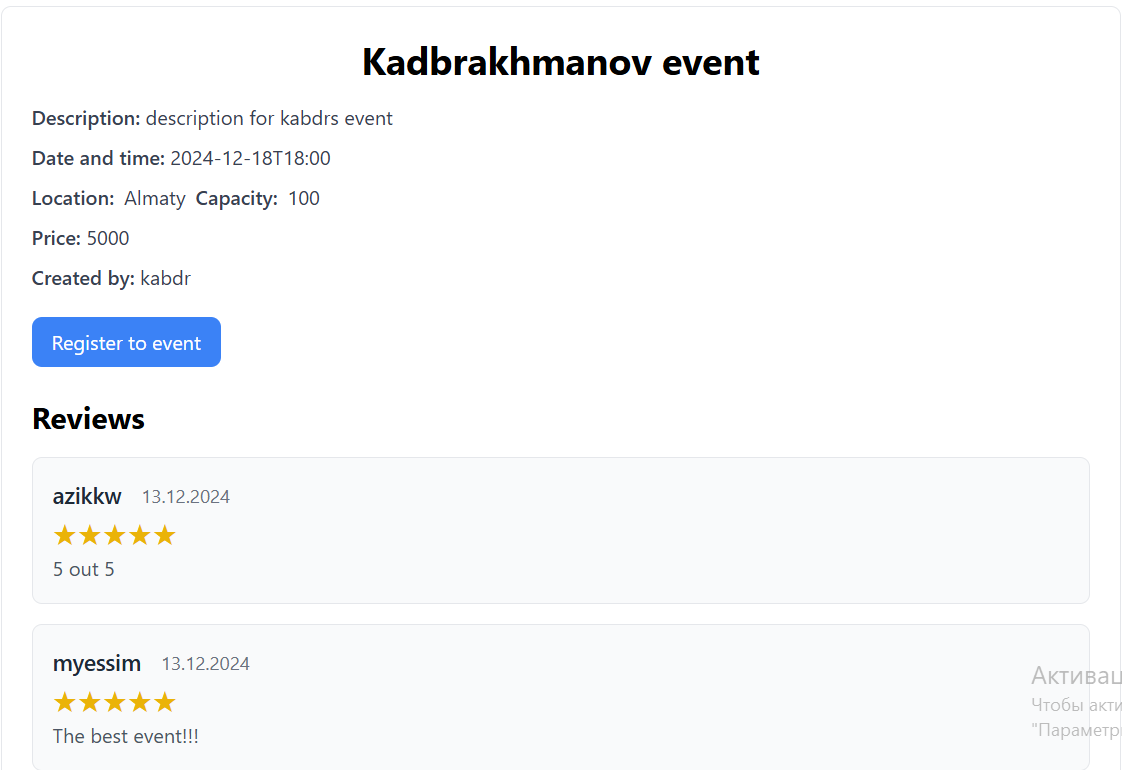
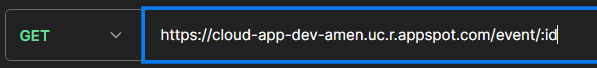
* **GET Events endpoint:**





Functionality: Returns all events with creator\_id that does not match with current user\_id

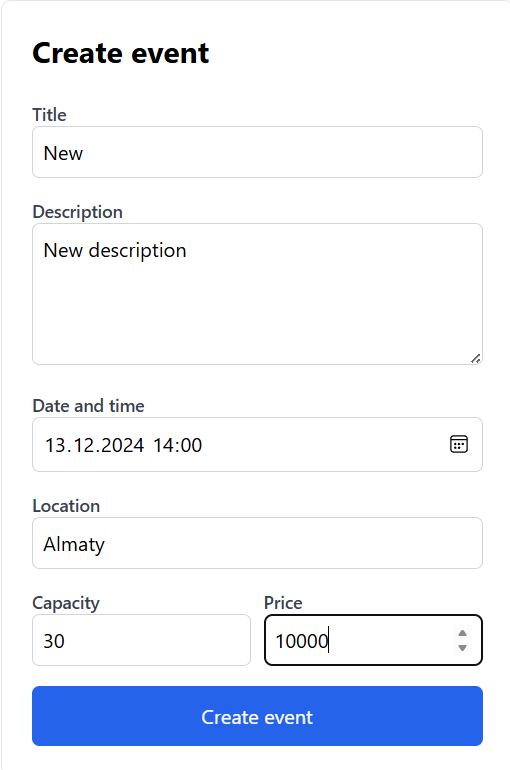
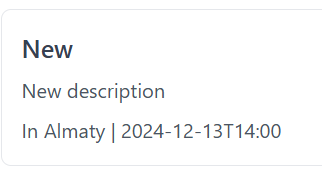
* **GET Event by event\_id endpoint:**



Functionality: Returns selected event page by event\_id

* **CREATE Event endpoint:**

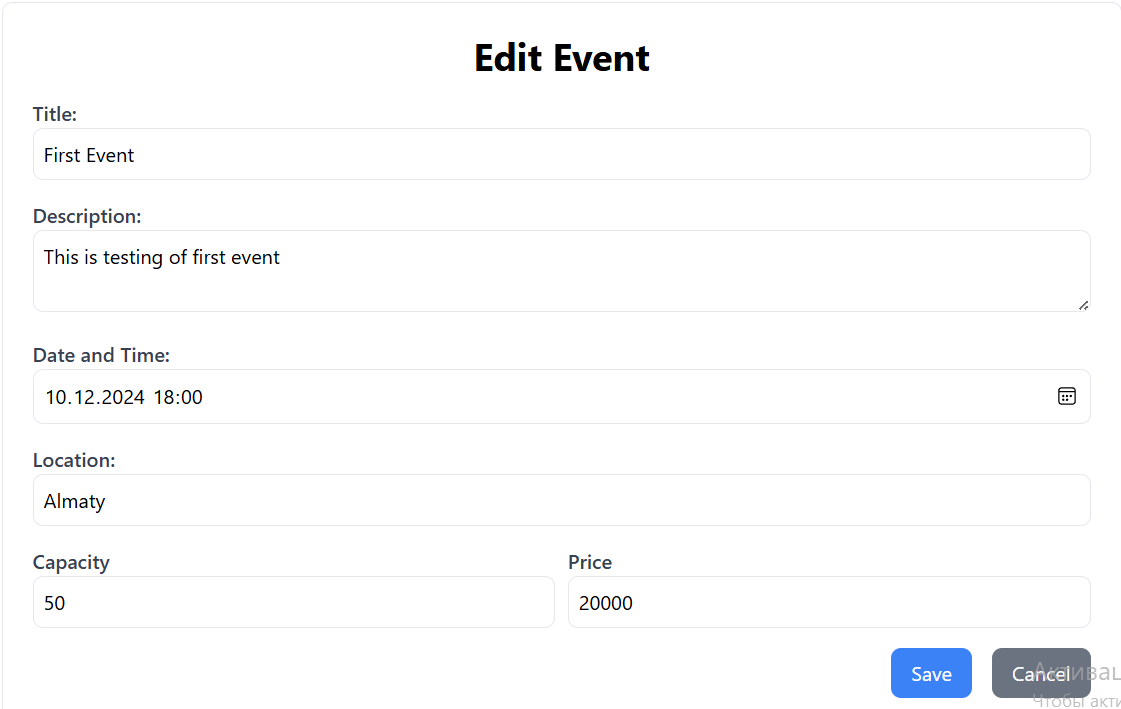
****

** > **

Functionality: This endpoint creates new event

* **UPDATE Event by event\_id endpoint:**

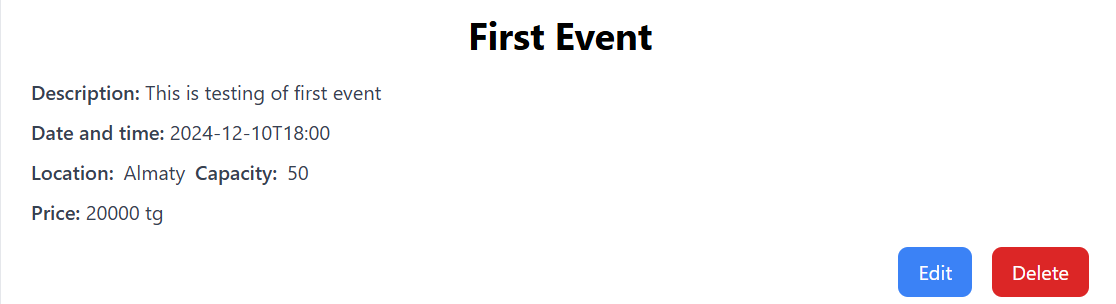
****

****

Functionality: This endpoint updates an event data by event\_id

* **DELETE Event by event\_id endpoint:**

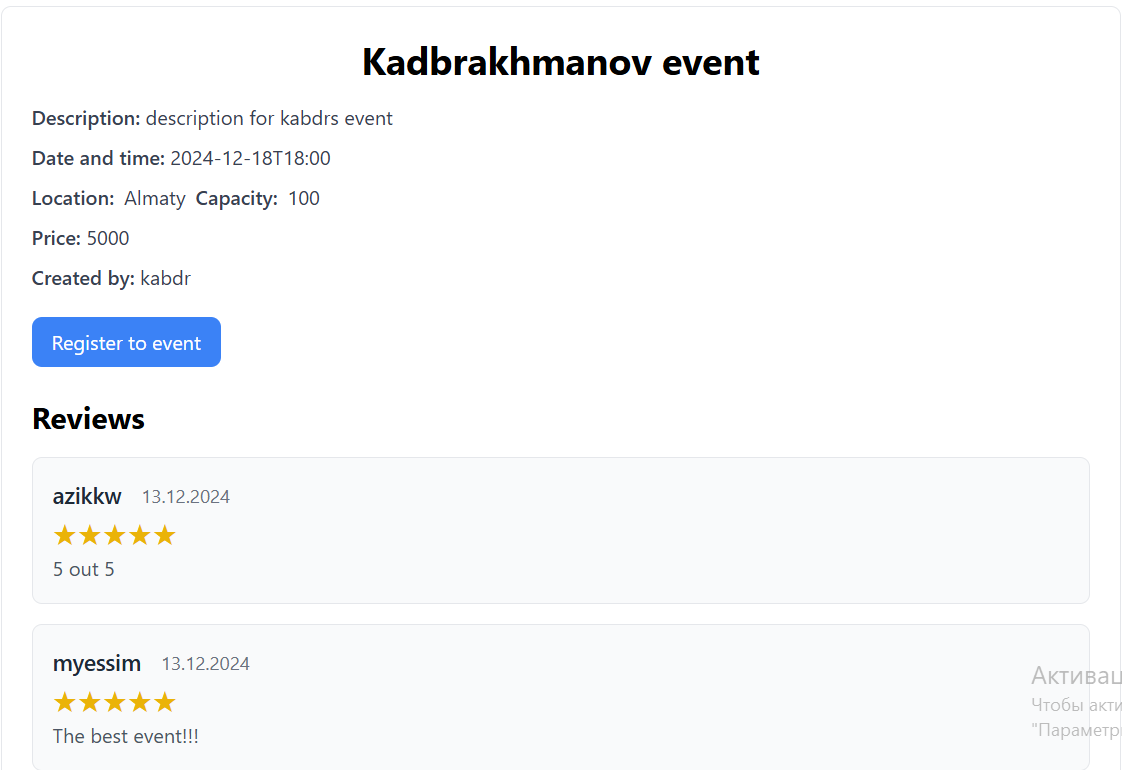




Functionality: This endpoint deletes event by event\_id

* **GET Reviews endpoint:**

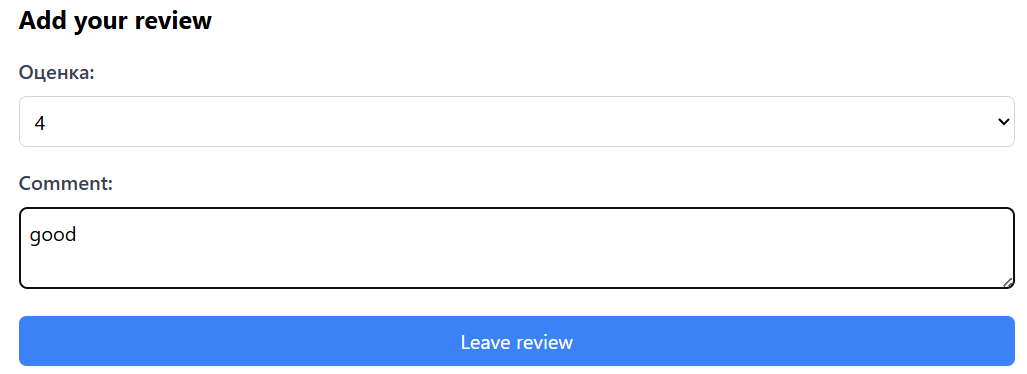




Functionality: Just returns all reviews for an event by event\_id

* **ADD Review to event endpoint:**

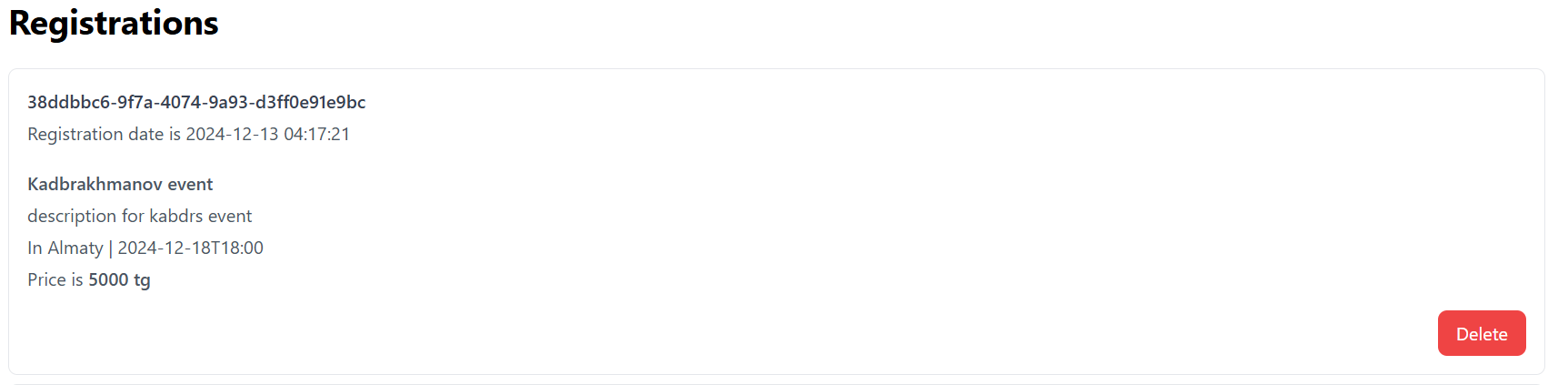




Functionality: Adds review to event by event\_id

* **GET Registrations endpoint:**





Functionality: Returns all user registrations user\_id

* **DELETE Registration endpoint:**

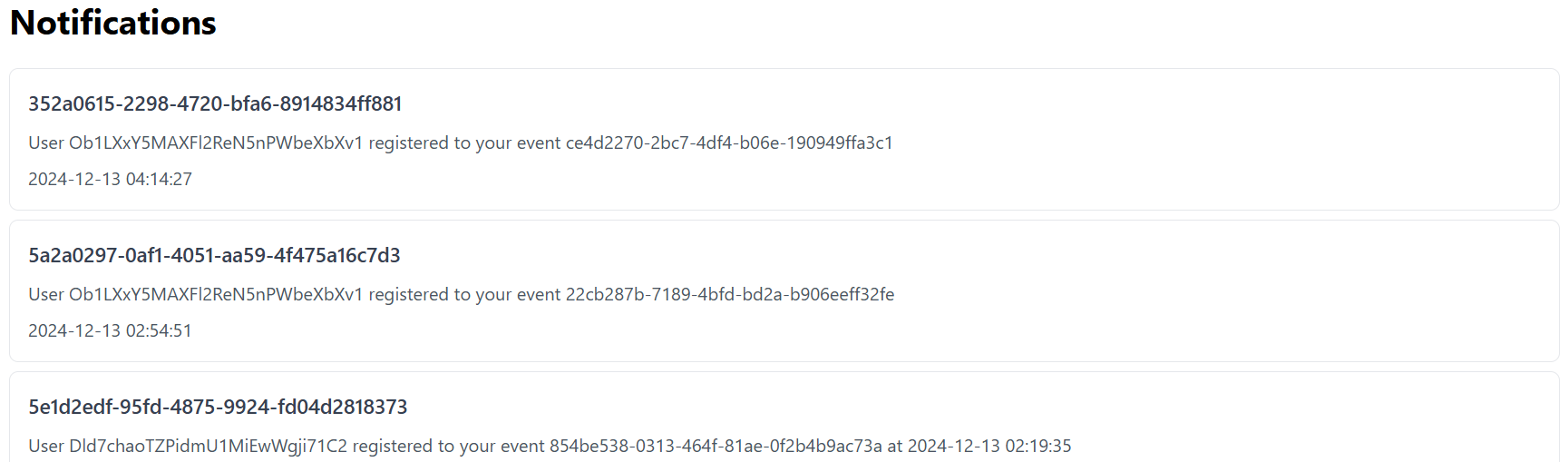




Functionality: Deletes registration to event by registration\_id

* **GET Notifications endpoint:**





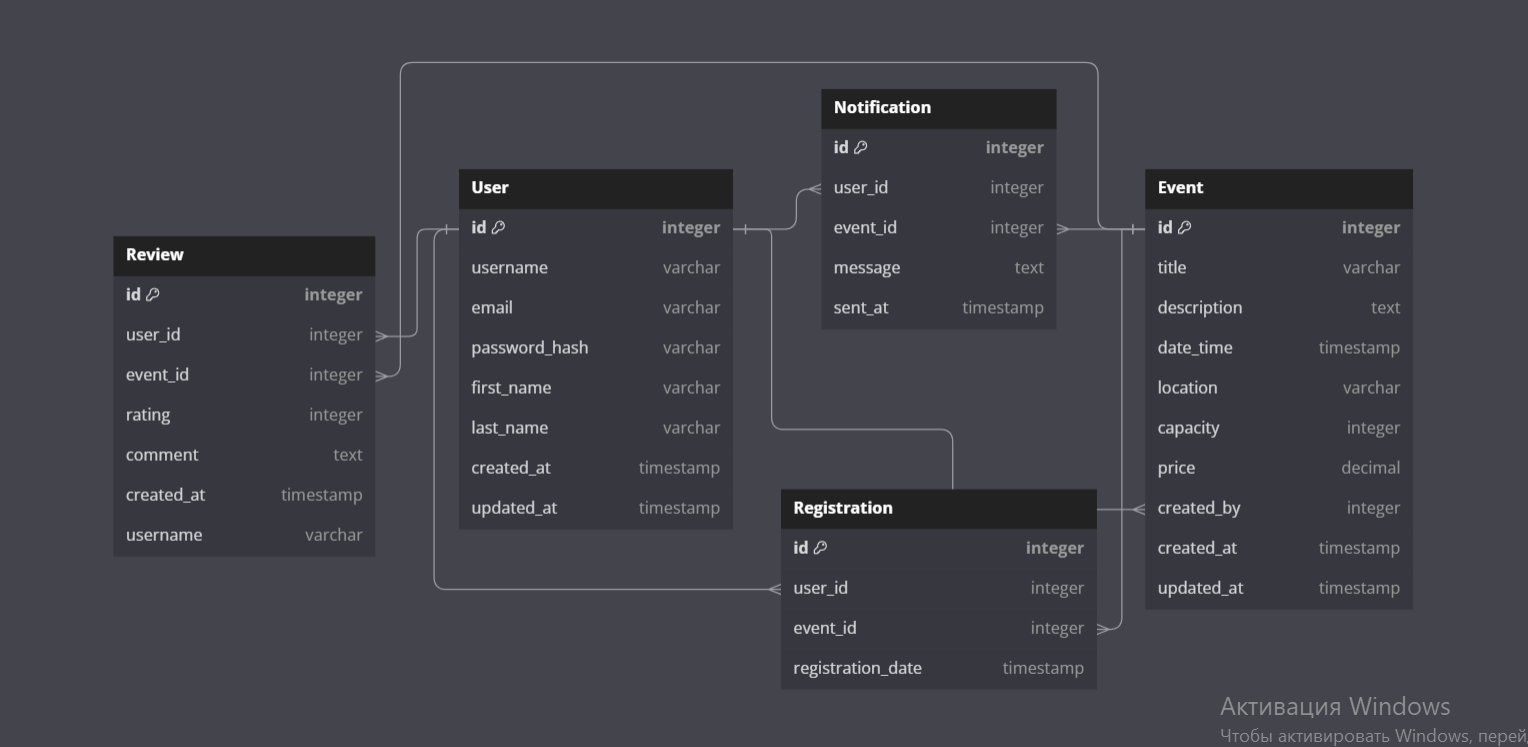
Functionality: Returns all notifications of user by user\_id field

This was full overview of my API with methods, endpoints, and UI visualizations.

**8. Database Design and Optimization**

For my project, I selected Firebase. It is NoSQL database developed by Google. I chose Firebase for several reasons:

1. Easy implementation and integration (for example, it already has predefined methods for searching for matches, filtering, sorting).
2. Monitoring tools for tracking user actions.
3. Also, access to data from Firebase can be obtained very quickly.
4. Very nice and user-friendly interface.

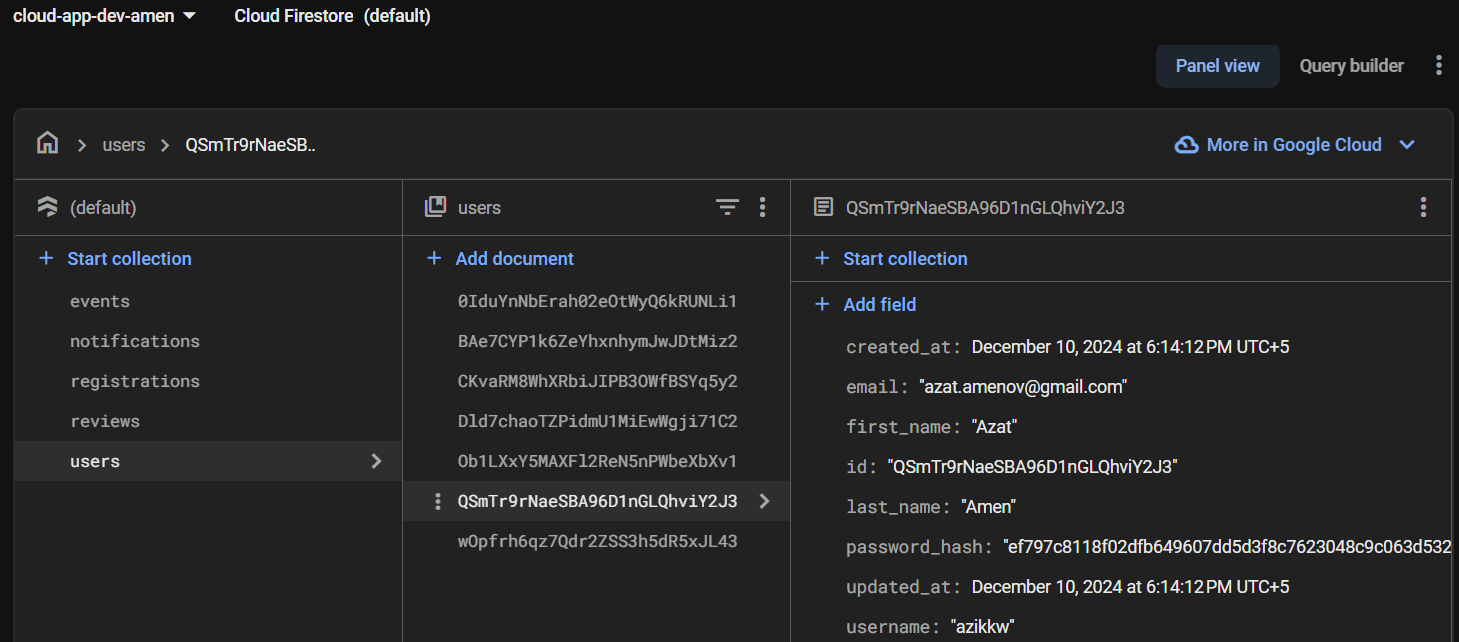
This is my database schema with entities and their relationships between each other:

The main one is **User**; nobody can exist without User. Because only user can create events and manage them, leave reviews and register to events. Also, if there is no user, then notifications will have nowhere to go.

**Event** also very important, because all system built around events. User can create and manage events, leave review to event and register to event.

**Review** and **Registration** is related to event parts because they are really part of event. Users leave review to event and register to event.

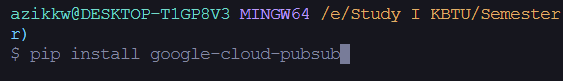
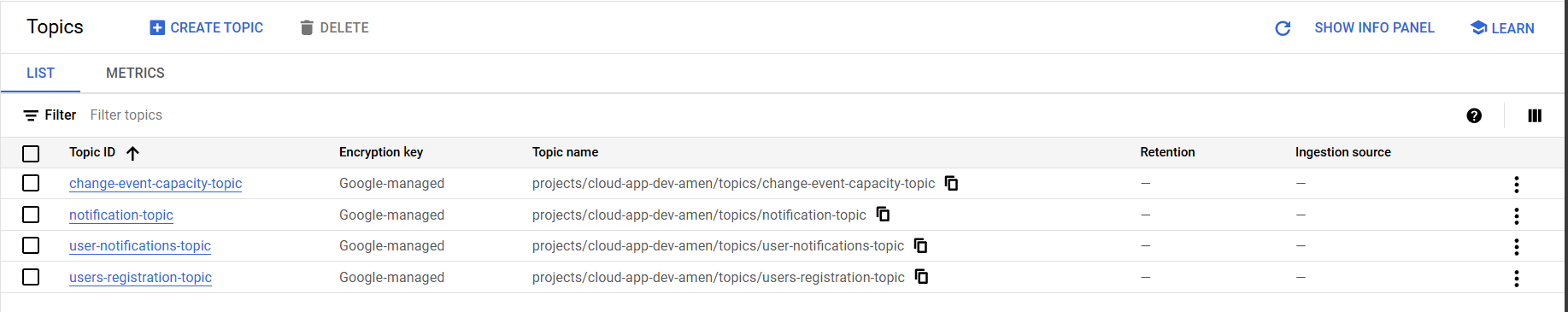
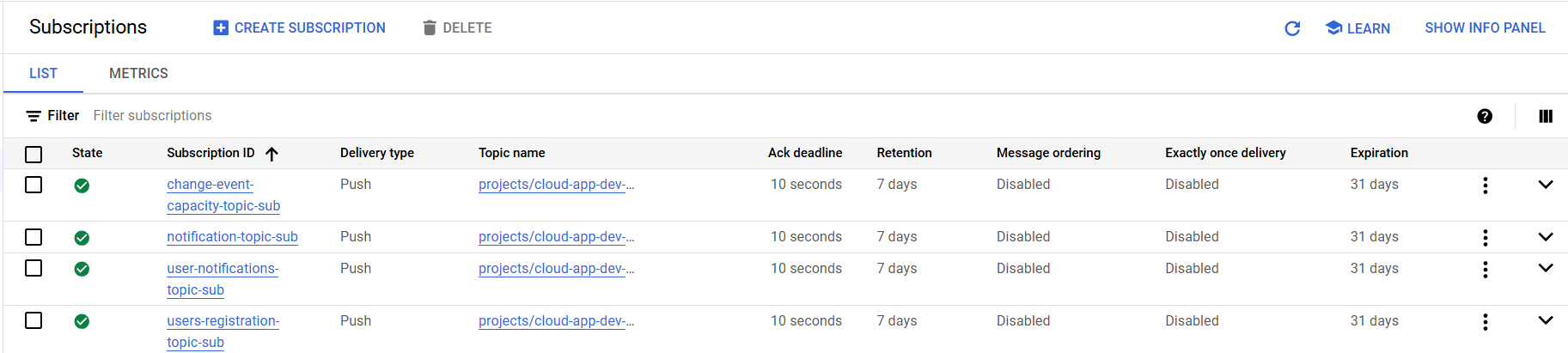
And **Notifications**, its also important part of my app, because notifications sending when user do something in the system. They are sent to the logs and to the users, for example when User events got reviews and when someone register to User events.

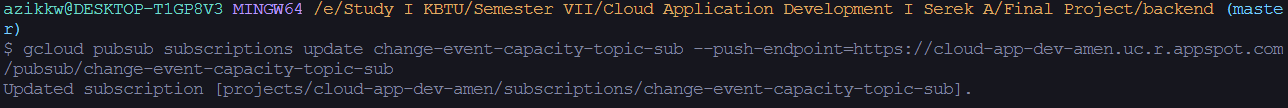


This is how looks Firebase Firestore database where I store all my data.

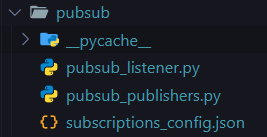
**9. Event-Driven Architecture**

Event-Driven architecture implementation using Google Cloud Pub/Sub:

1. Firstly, to start to work with Pub/Sub I installed **google-cloud-pubsub** library.
2. After that, in the next step, I created topics that will publish Publishers.****
3. And the subscriptions for all topics was generated automatically:
4. To ensure that subscribers will work in future I changed their **pushConfigs**. For example, for all did the following, but only names of Subs changed:

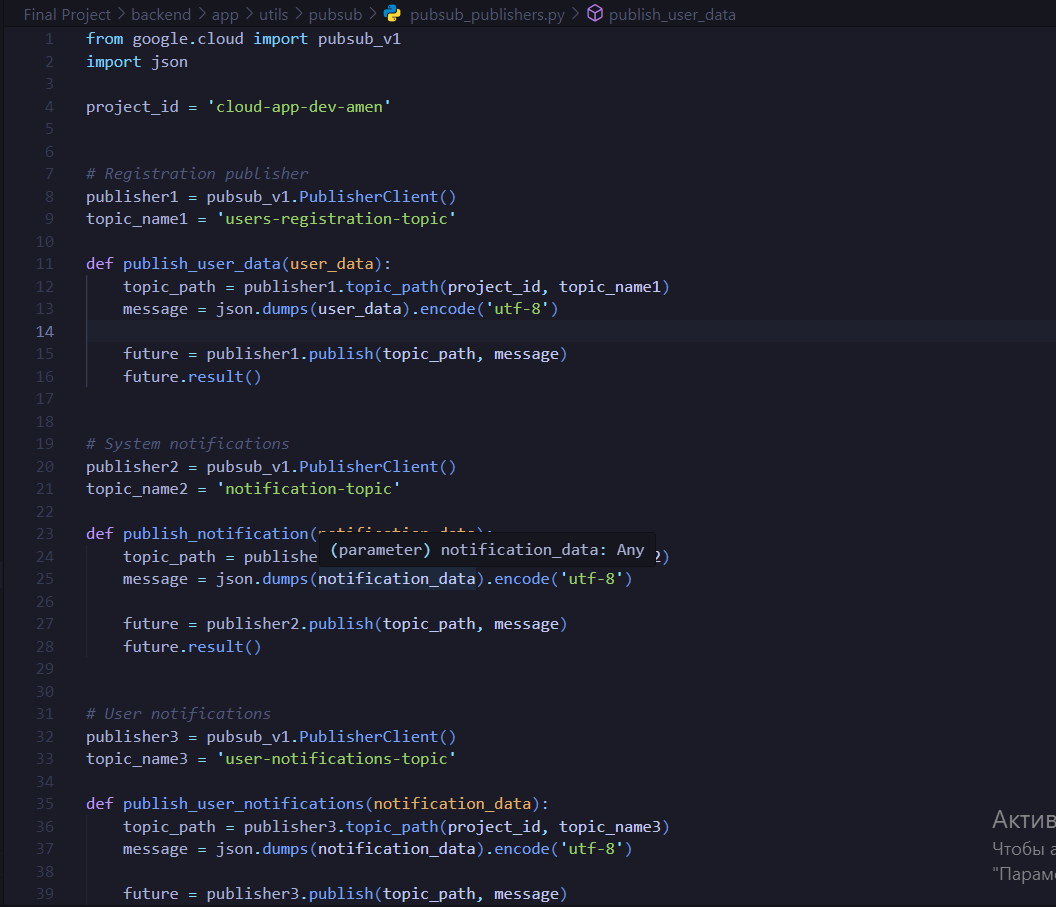
****

1. In the next step, I started to creating functions for publishers and listeners (Subscribers), I created folder for them:

****

1. And the next I wrote my publishers:
   * **publish\_user\_data:** When user register, it publishes topic with User data
   * **publish\_notifications:** At each user actions it publishes topic with Notification
   * **publish\_user\_notifications:** When some users leave review or register to event, it sends Notification to the event owner
   * **publish\_changing\_event\_capacity:** When user register to the event, the capacity value decreases to 1, and when user delete his/her registration its increases to 1

That is all about the Pubs in my system.

****

1. In this step I wrote listeners (Subscribers) that will handle published topics:****

Here I wrote one function that process all topics and do some actions based on the name of the published topic.



Here function handle all published topics and give them to the messages processing function.

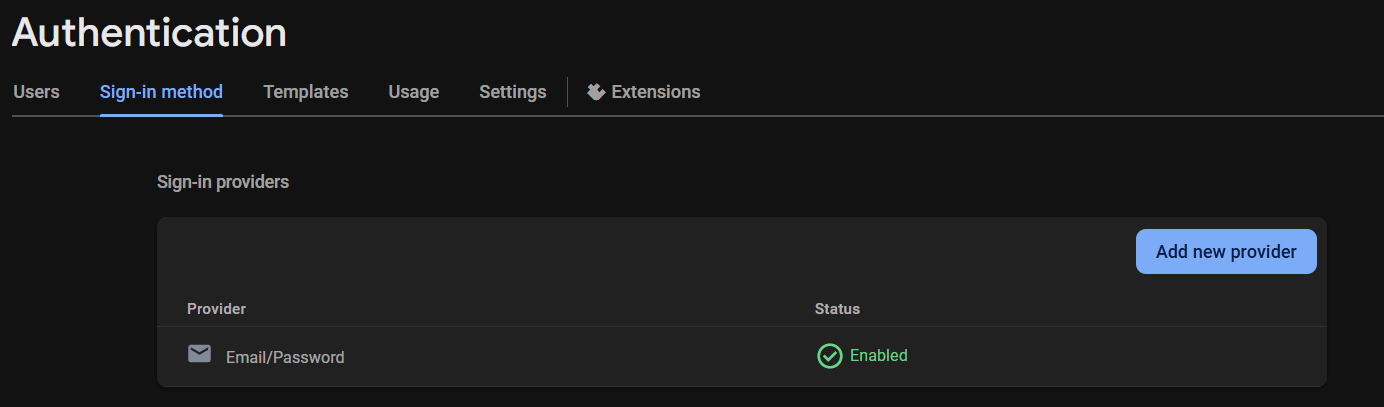
Cloud functions for events handling is a good idea that can be realized. Cloud functions execute in answer to some actions. They give an opportunity to write and deploy small event driven functions that will executed as response to some actions. But I think that Pub/Sub will be better solution for this situation. Because them was developed to solve problems like that. And when we have ready solution with documentation, it will be faster and easier to solve some kind of problems.

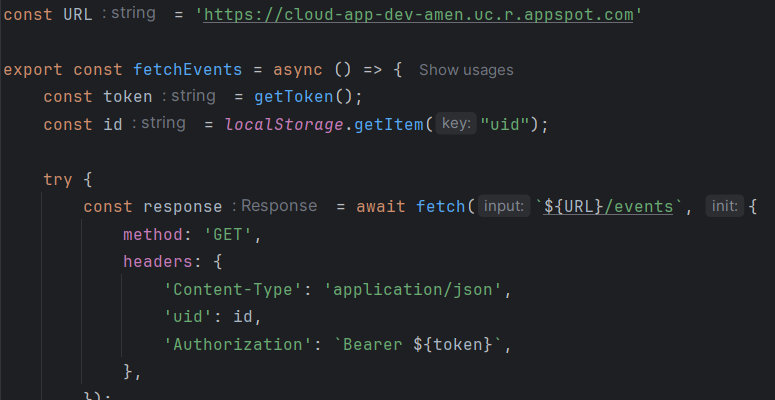
**10. Machine Learning Integration**

I would use AI to, for example, analyze user feedback on their opinions, or, for example, to make personalized recommendations of events based on their interests, or to represent the number of participants or some purpose of the chatbot for consultation. And I could improve the ideas that I have and make them effective. For input and output to my system, I would use Google Cloud AI Platform or Vertex AI.

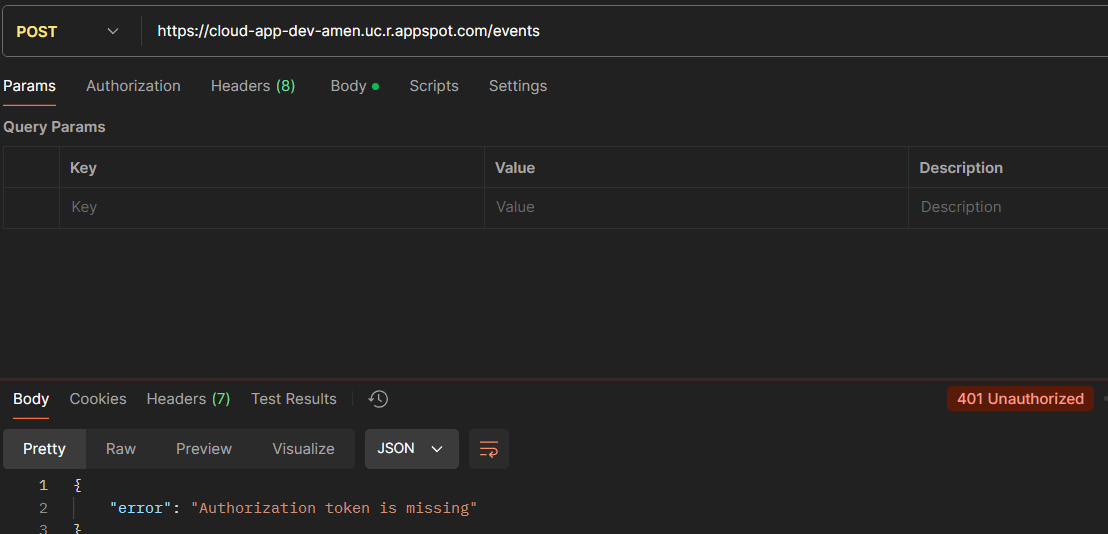
**11. Security Measures**

I implemented security practices to my application. Here is explanation of them below:

1. Authentication provided by Firebase. It offers various methods of users authentication such as email/password, phone/password, google account and etc. For my app I selected email/password authentication:
2. After login for user generates JWT token that is need in all server operations to enusere user authentication. And at each request to server in client side sends JWT token to approving:

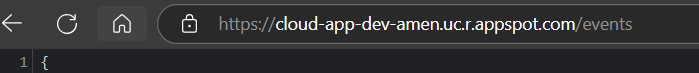


As an example, I will show example to enusre that my application secure. If you try access server endpoints withour token it returns following:

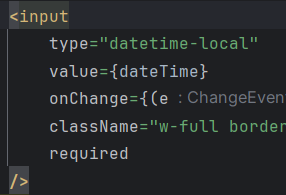


So, how you can it returns response with status 401 Unauthorized. It means that it works well.

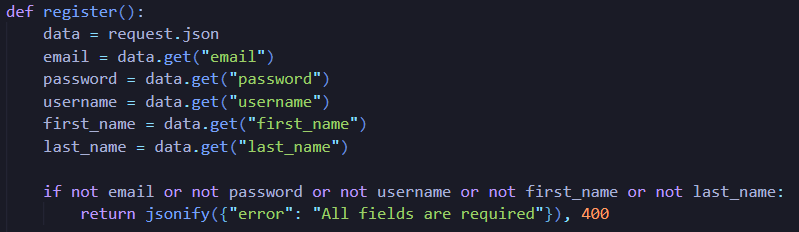
1. HTTPS:



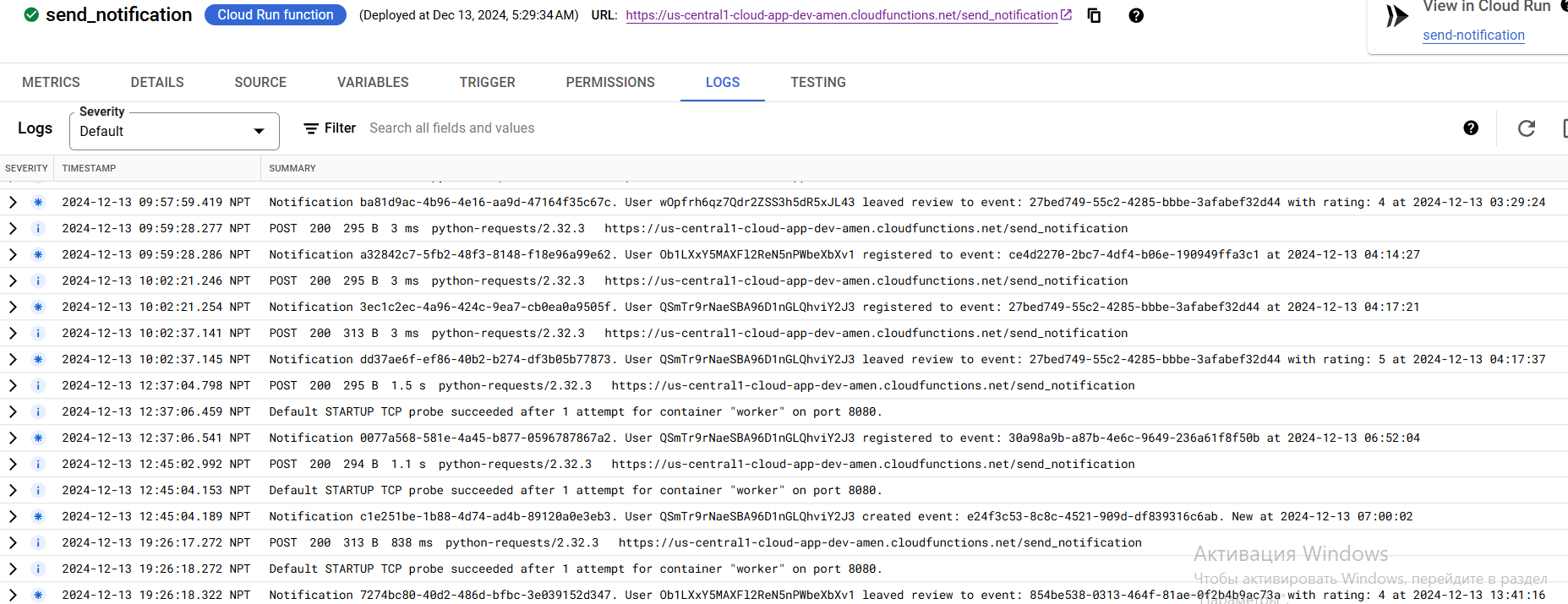
1. Also I added user data validation on client side. So, that enusre that to server never will come some empty data or data with errors.

****

But there is also some validations on the server. It servers as a last check to verify that data can be added to database for example:



1. And finally, I guess that without monitoring tools any security procedures will be weak. So, I created Cloud Function that send notification to logs at each user action. It gives an opportunity to track user actions in the system and respond to any incidents in a timely manner.



**12. Scalability and Perfomance**

**Scalability** is the ability of a system to handle more work or support more users when needed. It is important because it helps applications run smoothly even when there are more requests or higher demand.

What is **Horizontal Autoscalling**?It means adding more machines or instances to share the workload. It is good for applications that do not keep data on the instance (stateless), so any instance can handle any request.

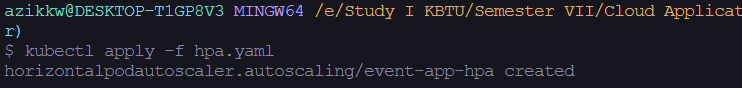
**Horizontal Autoscalling Implementation:**

To do this, we create an hpa.yaml file that contains the HorizontalPodAutoscaler manifest. HPA will receive the metrics of the processor used from the "metrics server", and if our condition is met, kubernetes will start adding the number of replicas of our module.

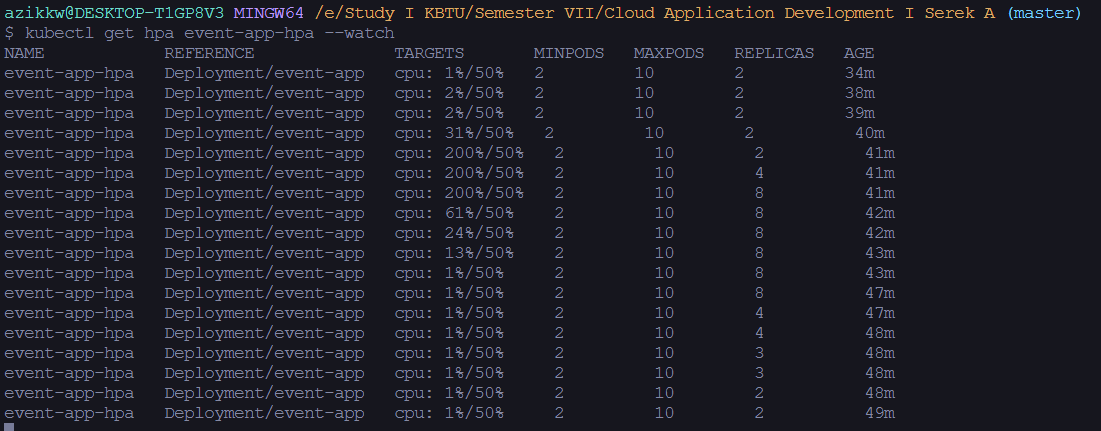
1. Create hpa.yaml file:



1. Next we apply our hpa.yaml to implement HPA.



To test the work of the HPA, I will increase the artificial load on my To-do application, and we will observe the changes of horizontalPodAutoScaler live using kubectl get hpa –[YOUR HPA] --watch command:



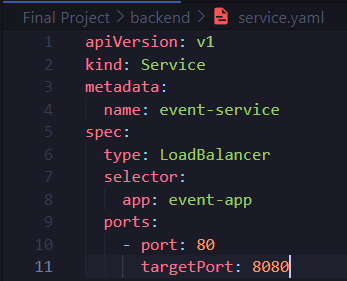
So, this is HPA work image:

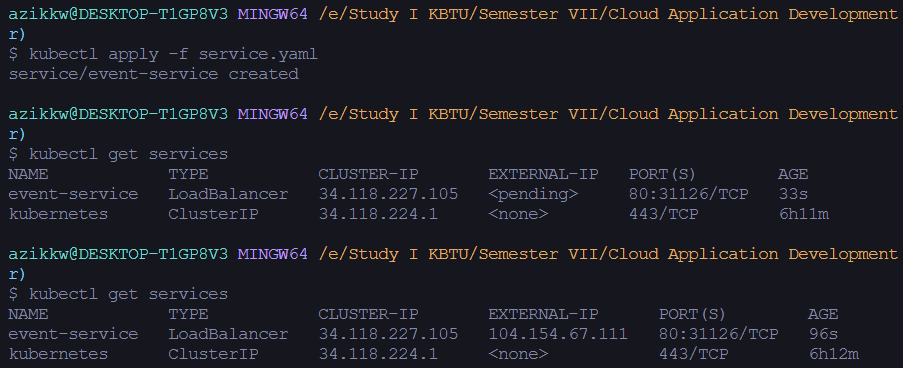
* When it just started to work and CPU is less than 50%, it uses minimum replicas number.
* When CPU is more than 50% and the load on application is increasing, the number of replicas is also starting to grow according to the load.
* When I lowered the load on application and it starts consuming less and less CPU, the number of replicas also starts to decrease accordingly.

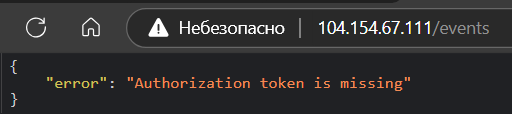
**Load balancer** distributes incoming requests between pods to ensure an even distribution of all resources and high availability of the application.

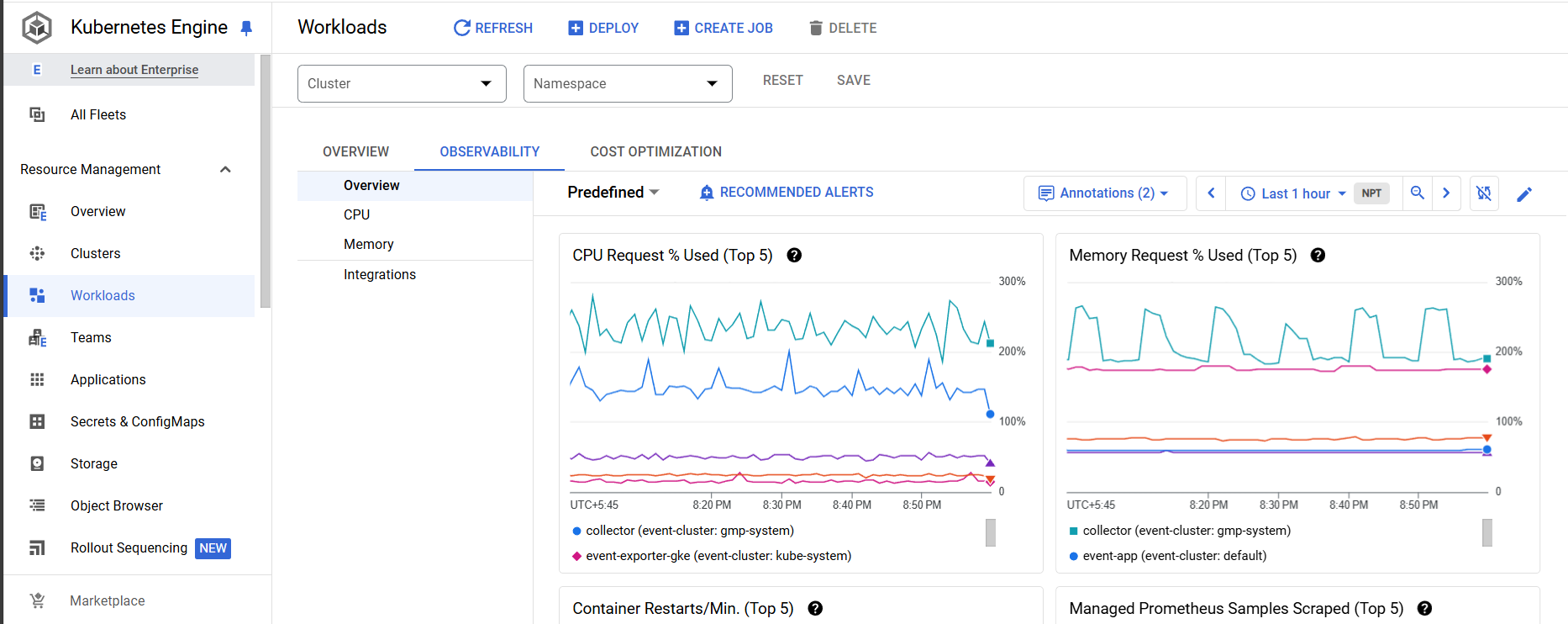
**Load balancer implementaion:**

1. First step to create Load Balancer is define service.yaml with service (LoadBalancer type):



1. And then, implement **service.yaml** using kubectl apply -f service.yaml command and check results using kubectl get services. And how you can see on image LoadBalancer successfully integrated to the kubernetes:
2. And finally how you can see the EXTERNAL-IP works, and can be access (On the image is unauthorized because my application requirese jwt token for authentication user).



About **perfomance** is that Kubernetes Engine enable Cloud Monitoring tools for containers. There you can check both CPU and memory utilization and implement changes if its needed: 

**13. Challenges and Solutions**

The problems I have encountered:

1. The first problem was to choose which database to use. Because each of them has its own benefits, it was hard to choose one. But finally I selected Fireabse, and think that it was good decision.
2. Also I thinked how to implement authentication, and when I found Firebase with its Authentication my problem solved momentally.
3. It was also difficult to figure out how to organize the system so that it would be readable and scalable in the future.
4. Another problem was that it took me a long time to deploy my API to Google Cloud Endpoints, but the solution was very simple. It was necessary to read the documentation. As soon as I tried to implement something from there, it immediately worked properly. The problem was in the name of the host
5. And besides, the very implementation of an event-driven architecture was not an easy task at first. It was a little confusing to understand how everything was going on. But as soon as I started writing and trying it out in real practice, things immediately improved.
6. Implementing the client and server side at once was a bit difficult, but I liked it. There were some difficulties in coming up with the process of their interaction, but in the end everything turned out fine.

**14. Conclusion**

In conclusion, first of all, I would like to say that it was a very interesting experience. I was developing an event-driven cloud system for the first time. As a result, I have an event-driven cloud based application, where Users can create an account, create and manage events, as well as register for other people's events and leave feedback. And all this is accompanied by a pleasant user interface.

I have integrated cloud functions to send notifications to logs, which ensures constant monitoring of user activity. I also put my application in a container and deployed it in GKE. In addition, I created an API in Cloud Endpoints and added authentication via JWT to my application. I used the Firestore cloud database to store user data, and also used tools to ensure smooth operation and reliability.

During the development of the project, I gained knowledge in event-driven architecture for myself. I also realized the importance of being able to read and understand documentation.

In the future, I will probably improve and increase the functionality of the application itself. I have also added more tools to improve monitoring. I would increase the number of cloud functions and expand the event-driven system.

**15. References**

[1] Google Cloud Documentation.(2024). *Python 3 Runtime Environment*. Retrieved from:<https://cloud.google.com/appengine/docs/standard/python3/runtime>

[2] Google Cloud Documentation.(2024). *Create a Cloud Run function by using the Google Cloud CLI*. Retrieved from: <https://cloud.google.com/functions/docs/create-deploy-gcloud#functions-clone-sample-repository-python>

[3] Google Cloud Documentation.(2024). *Google Kubernetes Engine (GKE) Documentation*. Retrieved from: <https://cloud.google.com/kubernetes-engine/docs/concepts/kubernetes-engine-overview>

[4] Kubernetes Doumentation.(2024). *Autoscaling Worloads*. Retrieved from: <https://kubernetes.io/docs/concepts/workloads/autoscaling/>

[5] Kubernetes Doumentation.(2024). *HorizontalPodAutoscaler Walkthrough*. Retrieved from**:** <https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/>

[6] Google Cloud Documentation.(2024). *Getting started with Cloud Endpoints for the App Engine flexible environment with ESP*. Retrieved from: <https://cloud.google.com/endpoints/docs/openapi/get-started-app-engine>

[7] Google Cloud Documentation.(2024). *Pub/Sub Documentation*. Retrieved from: <https://cloud.google.com/pubsub/docs>

**16. Appendices**

I have described my every step in great detail for each of the sections, so there is no need for this.

But I want to provide link to project code with all recourses and also link to diagrams:

* **GitHub:** [https://github.com/azikkw/Cloud-AppDevelopment-2024/tree/master/Final %20Project](https://github.com/azikkw/Cloud-AppDevelopment-2024/tree/master/Final%20%20Project)
* **Database Diagram:** <https://dbdiagram.io/d/Event-Management-System-67568cbbe9daa85aca14f4ce>