Ashkan Khademian
SUT student number: 98105738
Sepehr Kianian
SUT student number: 98102154

GymGo

5th Feb. 2023

1. OVERVIEW

This is an app for Gym reservations. You can find Gyms and book a time if you want. You can also make your current reservations and cancel them if you don't want that time. Our project consists of too many codes and took about 150 person-hour work. We tried to give a birds-eye-view in this documentation however an insight can't be achieved unless the whole code is studied. please feel free to ask any questions you have from the project supervisors,

Ashk Khademian: github, email, linkedin

Sepehr Kianian: github, email, linkedin

2. Requirements

Availability

- The users should have access to the app, from 8 a.m. to 24 p.m., and without any interruption.

Security

- The login process should be secure. In this case, confidentiality is an important problem.
- The reservation payment should be secure. System integrity is an important problem in this case.
- The reservation cancellation should be secure. System integrity is an important problem in this case.

3. Functionalities

Gym and Reservation

- Users should be able to find any Gym Complex, view its details, and select any Gymnasium they want from it.
- The user also should be able to bookmark any gym complex he/she wants. Also, he/she should be able to view his/her bookmarked gym complexes.
- Users should be able to view any Gymnasium's details, see its reservation table, and book a time if he/she wants any. For the booking process, the user should pay for that time and have a receipt that he/she can prosecute for law violations. The user should be able to view his/her active and overdue reservations.
- Users should be able to cancel any reserved time before the time occurrence if they want to; part of the money should be brought back to the user based on the business policies.

User information

- If the user has just entered the app, he/she should sign in or create an account. Both of those processes should be based on the user's unique phone number, and get validated by sending sms codes.
- The user should be able to view and edit his/her user information, including full name, sex, birth date, and home address.

- Users should be able to view and edit their favorite sports. Their finding Gym Complexes' process should be also based on their favorite sports.

Others

- The user should get notified of important events relating to his/her account.
- The user should be able to contact support if there is any problem occurred relating to the app.
- The user should be presented with the app's general rules and policies.

4. Quick Installation

System Requirements

- Docker (version 20 or newer)
- Docker-compose plugin (preferable version version 1.28 or newer)
- Node (preferably version 16.15.0)

Setup

There is a `run.sh` in each repository (gymgo_backend and gymgo_frontend). You only need to execute those sh files (simply `./run.sh`).

The one in the frontend project will set up a node server on some local ip address and port 3000. Fig. 4.1 shows an example from the terminal that runs the node server.

Fig. 4.1. **Running frontend server**. A node server will be up and running if you use the `run.sh` file. In this example it is running on http://192.168.232.131:3000/ and we can see it through every preferred web browser.

The one in the backend project will set up all docker containers (thus it may take much time) consisting of the django app with uWSGI, PostgreSQL, Redis, RabbitMQ, and NGINX web proxy. These containers are connected with each other through a docker network (automatically built by docker-compose) and what matters in the end that the NGINX web proxy is binded with http://localhost :8000/ and that's where the frontend node server send its HTTP requests to. For a refined explanation of backend services please read "Backend View | Services" section.

Fig. 4.2 shows an example result from the `$ docker ps` command when all docker containers are built and running correctly in your system.

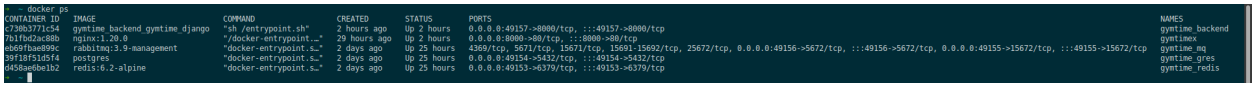

Fig 4.2. **Docker containers for the backend system**. The `$ docker ps` shows that all five backend services are running correctly.

5. Frontend View

Routes

Login Process

Login and Register

a page for the login and registration process. As it is mentioned, these two processes are done with sms verification. Thus the only action the user should do is to enter his/her phone number. The rest is done on the Verify page.

Verify

This is a page where further action of the login and registration process is done. After the user has entered his/her phone number on the previous page, an sms verification code will be sent to the user's phone number. The user then should enter his/her phone number on this page to enter his/her panel the following moment.

Fig. 5.1. **Login process pages.** These two pictures are snapshots of the "Login and Register" and "Verify" pages.

Gym and Reservation process

This is a page for the overall user panel structure. It covers the other nested routes mentioned further.

Home Search

This is a page for finding the appropriate Gym Complex based on the Gym Complex id, which is a 4-digit code.

Gym Complex

This is a page that shows general information about the specific Gym Complex. The user can bookmark his/her favorite Gym Complexes so that he/she can see them on his/her “Bookmarks” page later.

Gymnasium

This is a page of the specific Gymnasium the user has selected on the previous page. It shows general information about the gymnasium. The user can choose a service and see its Time table if he/she wants to book a time.

Time Table

This is a page that shows the reservation table of the specific gymnasium the user has entered before. Here the user can select a non-reserved time. Then he/she enters the payment page and if he/she paid for the time, that time will be reserved for him/her.

Fig. 5.2. **Gym and Reservation process pages.** These three pictures are snapshots of the “Gym Complex”, “Gymnasium” and “Time Table” pages.

Other processes

Profile

This is a page that shows the general information about the user. The user can also edit his/her general information. What general information consists of is shown in the figure below.
The user can also edit his/her favorite sports, which is done by clicking into the section and being redirected to the next page.

Favorite Sports

This is a page for editing favorite sports. Favorite sports are effective on Gym Complex search.

Fig. 5.3.1. **Other processes page.** This picture is a snapshot of the “Profile” page.

Receipt

This is a page that shows the times the user has booked, both open and overdue.

Bookmarks

This is a page for displaying the bookmarked Gym Complexes. The user can bookmark the Gym Complex he/she wants on its page.

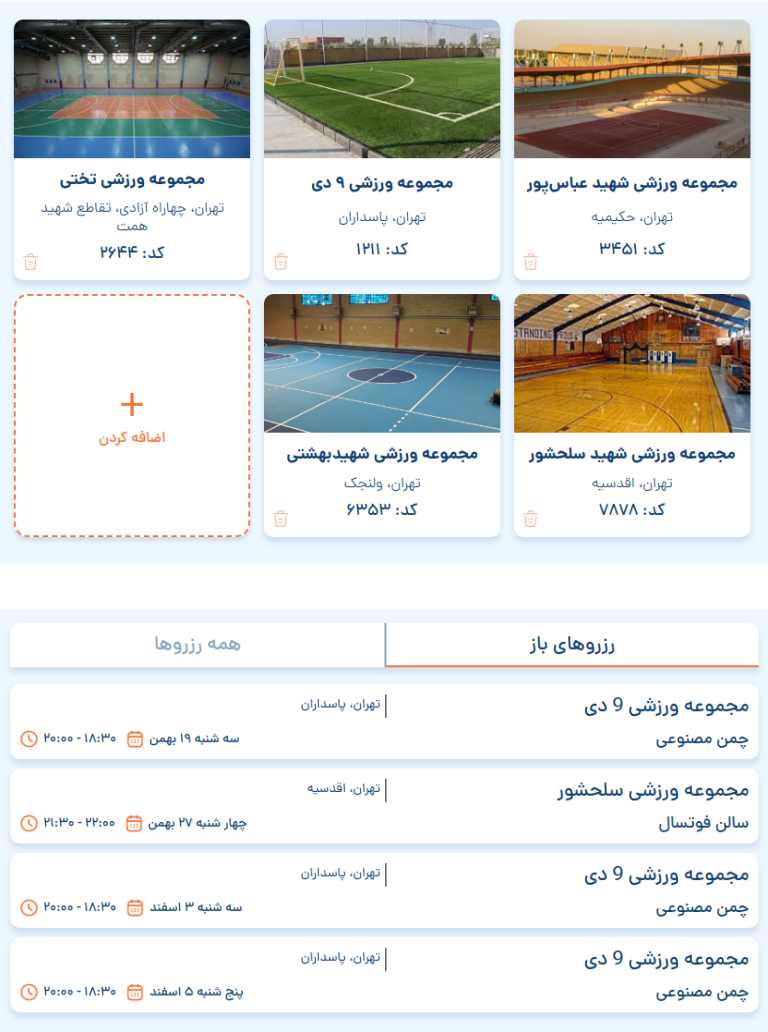

Fig. 5.3.2. **Other processes page.** These two pictures are snapshots of the “Bookmarks”, and “Receipt” pages.

Overall Route structure

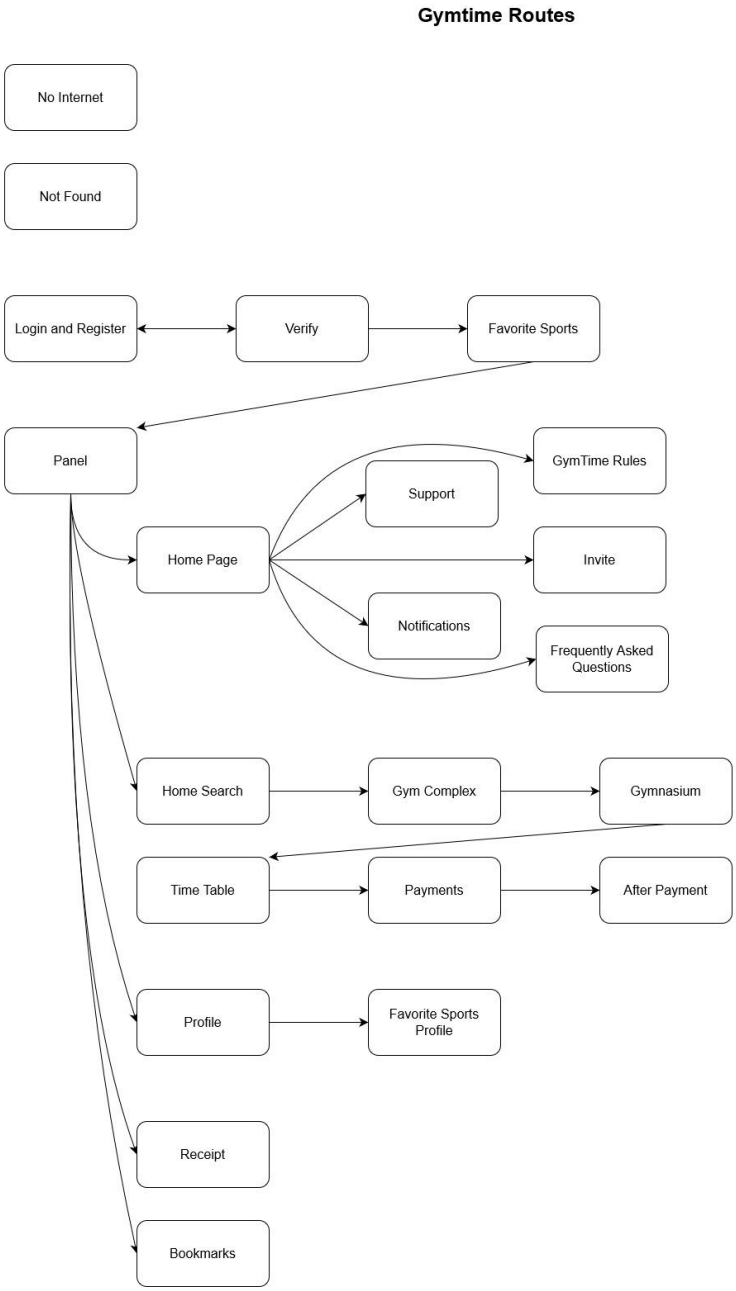

Fig. 5.4. **Overall Route structure.** This figure shows all the routes and their transition states.

Services

Login and registration

Since our login and registration functionality is with sms verification, we combined login and registration into one page. We also present an sms verification page, which captures sms codes and verifies them.

security

Confidentiality in the login and registration process is an important requirement. we don’t let our login info be accessed by other users.

Gym Complexes

We provide functionalities to first find Gym Complexes, then visit its Gymnasiums, see their reservable free times on the Timetable page, and reserve a time customers want. We also provide a functionality to bookmark your favorite GymComplexes.

Notifications

We provide some pseudo-pwa features like push notifications. Ticketing messages, reservation cancellations, and other events get notified to you with this ability.

Ticketing with Supports

We have a page for chatting with support, to help you through the problem that has occurred to you.

Deployment

We have dockerized our project for fast deployment.

Architecture

Commonly Used Design Patterns

In this section, we provide some usual design patterns that have been used in the project.

Memento

We have used this pattern to persist the login information in secured cookies whenever we change that information.

Composite

Because of the routing structure, some pages are inside other pages. Thus we have used a Composite design pattern for page structure.

Observer

This is a design pattern that is commonly used for event-driven structures. Thus It's used for processing user inputs, handling server responses, and other event-driven functionalities that are defined in the UI layer.

Software Infrastructure

We have used Typescript, HTML, and CSS for page functionalities, structures, and stylings. We have also used the Vue.js framework, which uses Those commonly used design patterns, in the development process.

6. Backend View

Services

Gymtime Backend is fully dockerized. It has some major and minor services listed as follows,

Major Services

1. **gymtime_backend:** This docker container (service) serves the actual python (Django) web server with the help of uWSGI.
2. **gymtimex:** This service is a server proxy in front of our Django app. All HTTP request comes to this NGINX service first and it passes the request to the actual Django server **(gymtime_backend)**.

Minor Services

1. **gymtime_gres:** A dockerized PostgreSQL working as the main DB of the project
2. **gymtime_redis:** A dockerized Redis server working as the main cache of our API.
3. **gymtime_mq:** A dockerized RabbitMQ server working as the Celery task queue of our Django app.

The main diagram of our project is represented in Fig. 3.1.

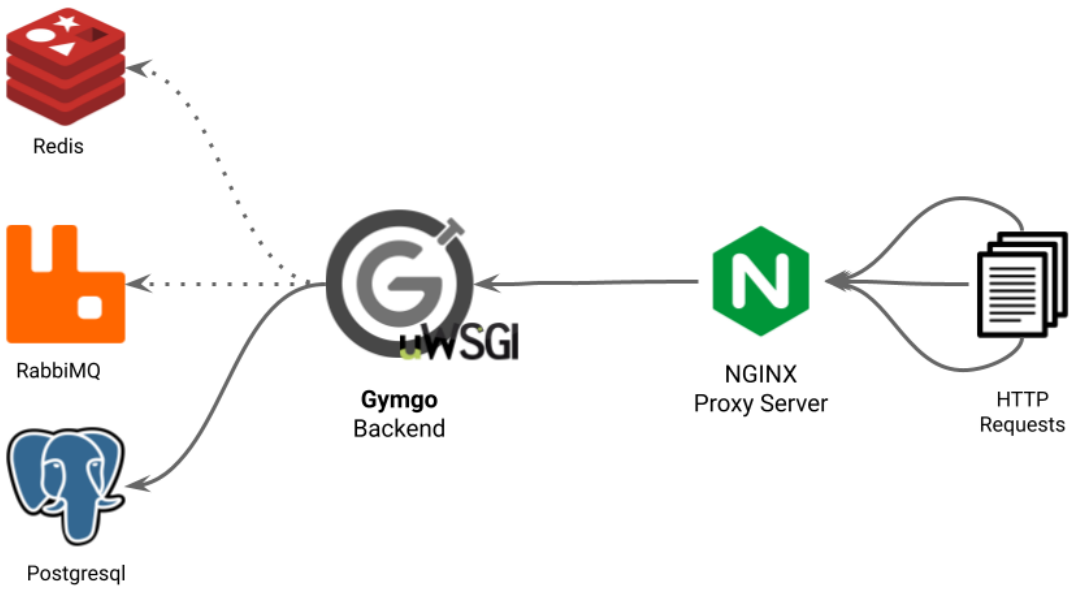

Fig. 3.1. **Flow of HTTP requests.** HTTP request first comes to the NGINX server; then the proxy server passes them to Gymgo backend (a Django app served by uWSGI); then the app calculates

the result and may use its database (PostgreSQL), cache database (Redis) and generates celery tasks (in the RabbitMQ queues) then returns the response.

Architecture

We describe our backend architecture concerning the Django apps. The two main apps are **gym** and **account** which have the main logics of accounting and gym management; then there are some side apps (**reservation**, **financial**, **sms**, **discount**, and **crm**), and all together serve all requirements of our project. Fig. 3.2 shows these apps, their main features, and their dependency relationships with each other as a whole.

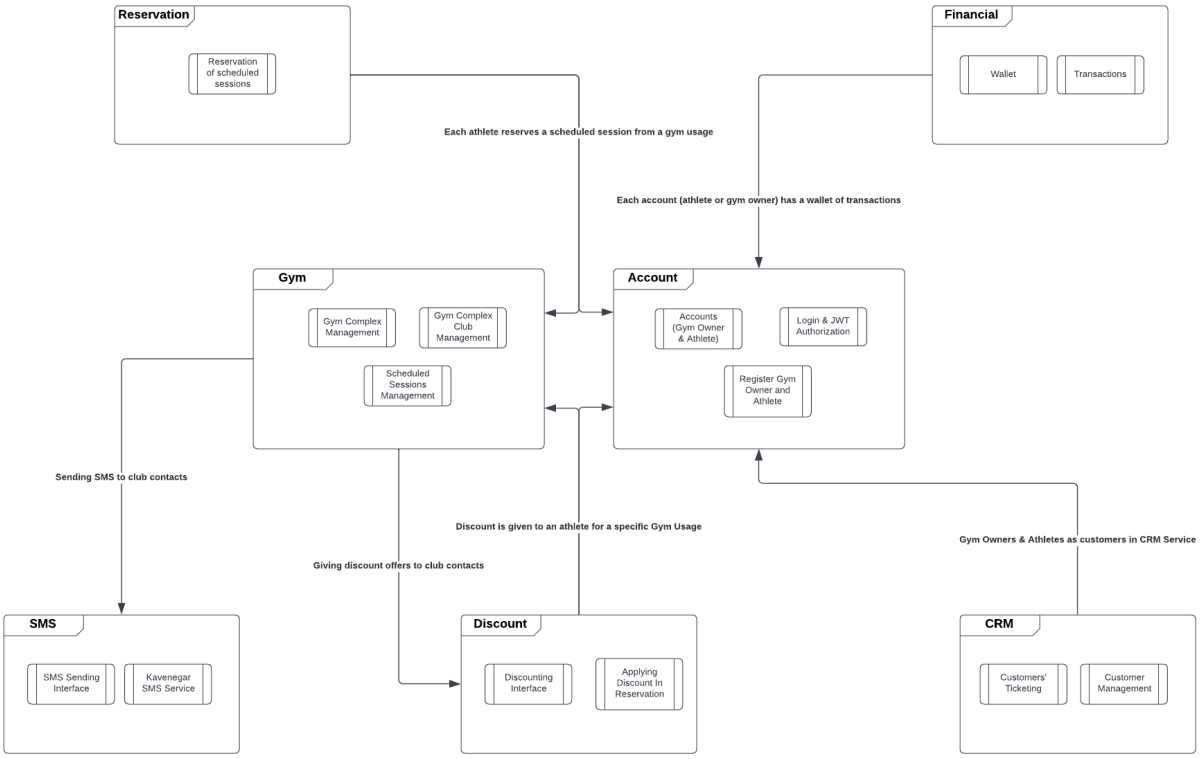

Fig. 3.2. **Django apps and their interactions with each other**. Two main apps, **Gym** and **Account**, provide the system's main features such as “Gym Complex Management” and “Accounting”; then their interaction with other apps completes the system’s feature list. Each edge shows a dependency (using) relation.

Terminology

1. **Gym Owner:** One of the two account types in our software system. Gym owners can create and manage their Gym Complexes in our app.

2. **Gym Complexe:** Main component in the gym app. Each Gym complex has gymnasiums and their scheduled sessions, all managed by a gym owner.
3. **Gymnasium:** One subcomponent of the gym complex. A Gym Complex is broken into multiple gymnasiums each having its scheduled sessions (which can be reserved in our app by athletes).
4. **Gym Usage:** Once Again each gymnasium can break into multiple gym usages. It's because we observed it's usual to use the same gymnasium for all football, basketball, and volleyball causes.
5. **Athlete:** One of the two account types in our software system. Athletes can find and reserve scheduled sessions.
6. **Club:** Each gym complex has a club of contacts made by its athletes (or potential athletes) so that its owner can send bulk sms messages to them.

Miscellaneous

events

Although our Django app is focused on an object-oriented design, it has an **Events** module that can connect apps in an event-driven way. Fig. 3.3 shows a flow in which a reservation is made in our system and the **reservation** app notifies other apps that this event has occurred.

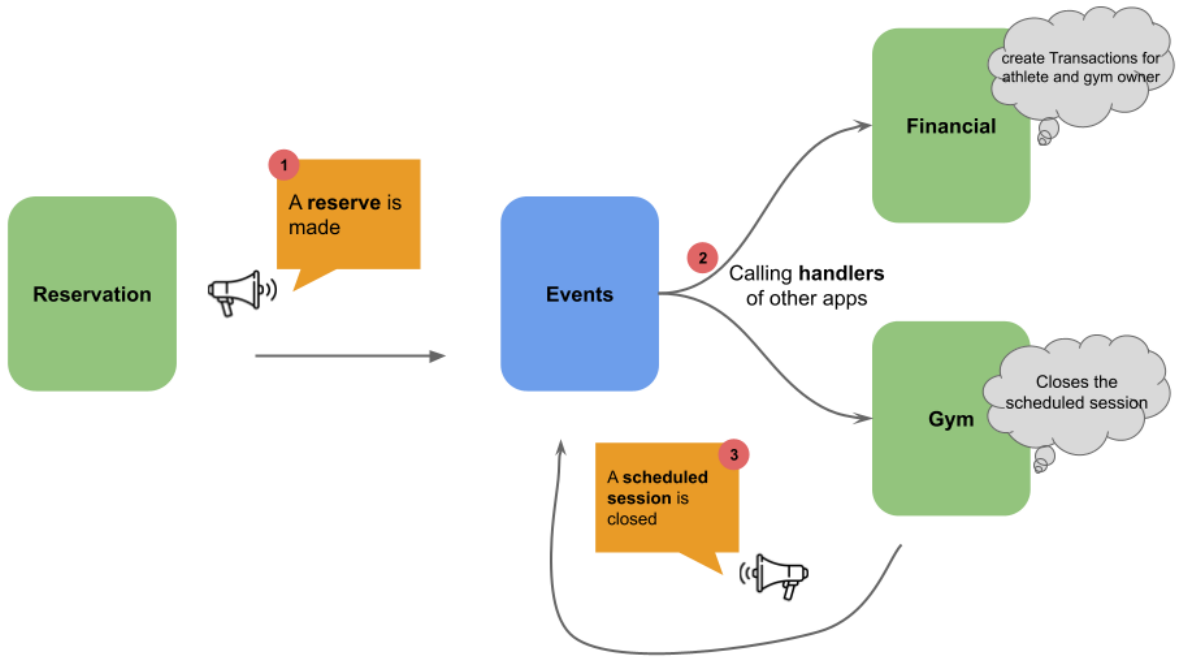

Fig 3.3. **Event-driven design**. When a reservation is made, the **reservation** app notifies this event and the **events** app lets the handlers (of other apps) know about this event. Through handling an event, some new events can be triggered (step 3).

gymtime_backend

This Django app consists of Django, celery, redis, and wsgi configs.

utility

Some useful and reusable utilities were used all across the code.

Functionalities

Gym app

1. Gym owners create gym complexes and their gymnasiums, Managing them and their scheduled sessions.
2. Athletes Retrieve different scheduled sessions available for their desired sport.
3. Gym owners manage contact clubs of each gym complex and send bulk sms to them (via sms app).

Account app

1. Register & Login of accounts (Both gym owner and athlete).
2. Changing profile information.
3. Logging out from the account.

Reservation app

1. Reserving available scheduled sessions (followed by using discount codes).

SMS app

1. Creating sms model objects and sending them periodically (using celery workers).
2. Sending sms with Kavenegar service.

Financial app

1. Automatic creation of financial transactions on specific events.

CRM app

1. Customers (accounts) create and send tickets to our system admins and product owners.
2. Product owners observing and replying to customers' tickets in Django admin.
3. Defining and managing crm tasks for crm admins (including calling and taking surveys from customers)

Discount app

1. Offering discount codes from business owners and gym owners.

Gymgo Admin

Backend project consists of an admin platform for our business owners. This admin is implemented via Django admin. The first page of gymgo admin is shown in Fig 3.4.

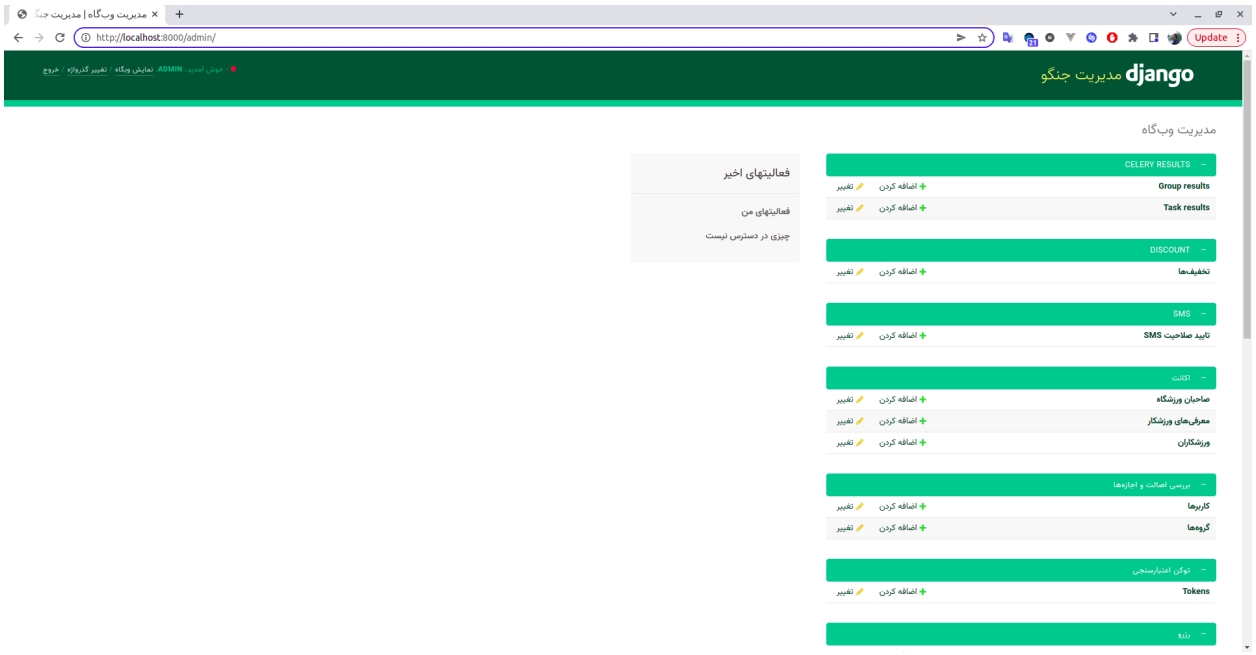

Fig. 3.4. **The main page of admin**. Each django app has a separate section in django admin. Django admin preserves a url route defined as “host-addr/admin/” for all admin pages.

You can create a super user account for entering the admin site with the following command which is also explained in here.

```
$ python manage.py createsuperuser
```

The initial language of the admin site is Farsi defined as a variable, `LANGUAGE_CODE = 'fa'`, inside the **gymtime_backend** app (gymtime_backend/settings/development.py).