**FireApp**

**Documentation of APIs**

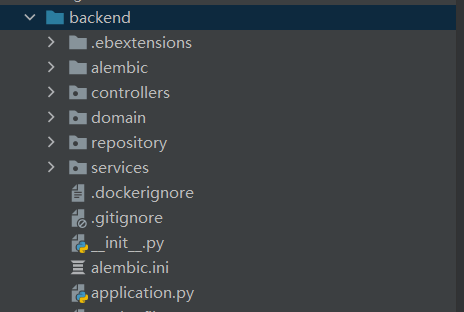
Hang Su

# Code Structure of APIs

An application programming interface (API) is a way for two or more computer programs to communicate with each other. It is a type of software interface, offering a service to other pieces of software.

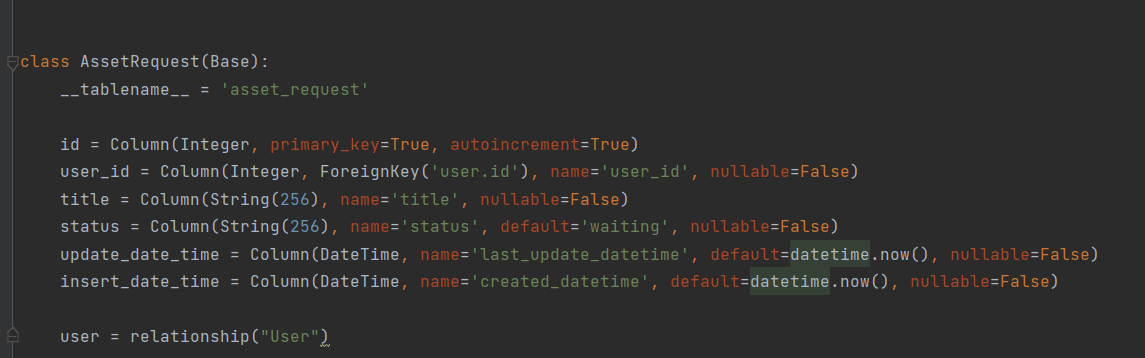
The API facilitates front-end and back-end interaction, enabling engineers to call various functions without knowing the back-end code. This chapter describes the process of constructing an api in fireapp.

All the backend code is in the **backend** folder. The files that need to be designed to construct an API include: **domain, repository,constollers**.



## Domain

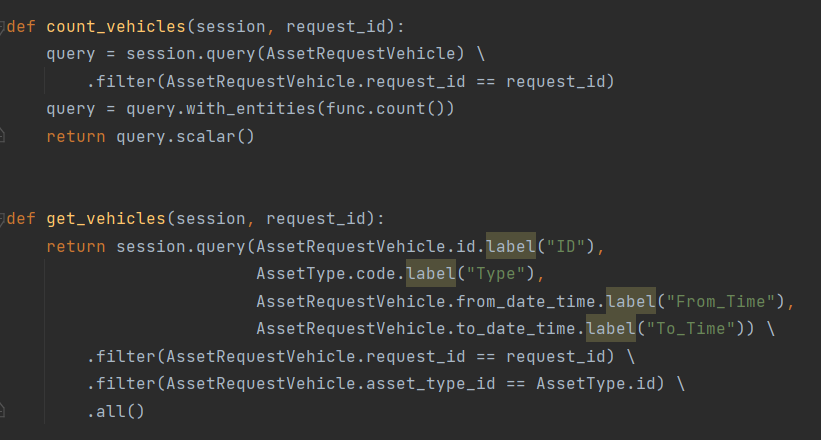
**Domain** describes the information in our database. Each .py file in the domain folder represents a table in the database. By converting the table into a class, we can use the query function to add, delete, modify, and query the database. Through this class, we can understand the format requirements of each attribute. For example: **asset\_request.py**



## Repository

Various functions are stored in the **repository**, including data filtering, data format conversion, and data processing. Write some commonly used methods as a function, which is convenient to be called in different tasks.

For example:**asset\_request\_vehicle\_repository.py**



## Constollers

**Controllers** contain code about the API itself. Fireapp uses the flask framework to design the API. Depending on the type of API, different functions are used. We mainly use the **get** and **post** request.

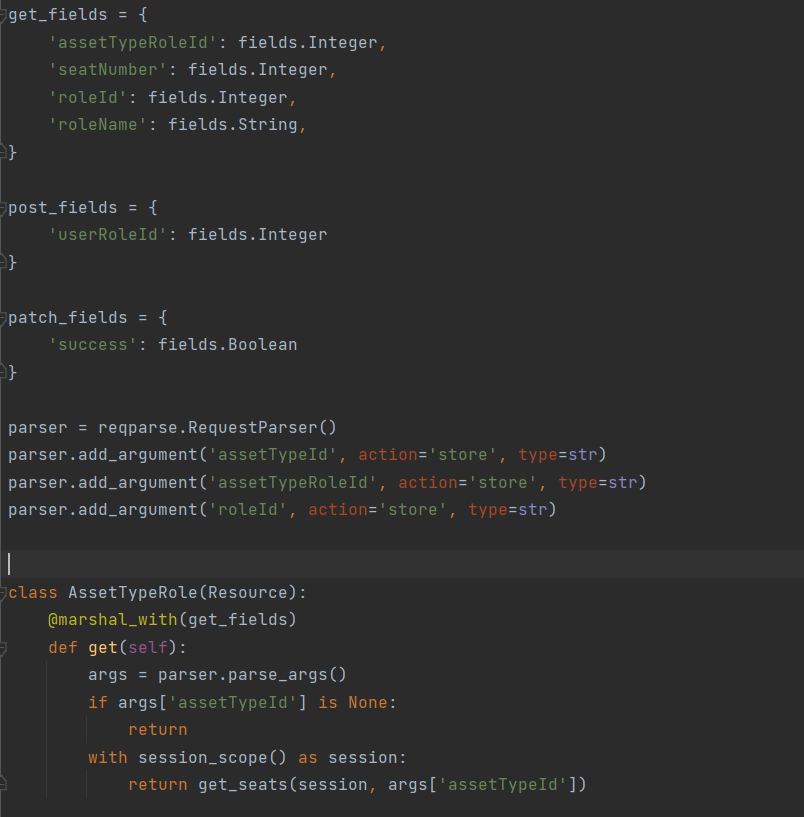
For a request, the three important parts are arguments, parser and result format. The arguments define what passed data is included in the url. The parser converts these data into parameters, which is convenient for backend backend function calls. result formats defines the format of the returned result and which parameters are output. In get or post, we can call functions in repository to process data.

For example:

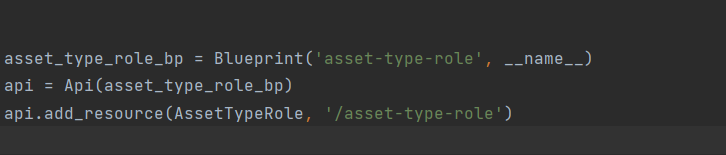
**get\_fields** define the result format.

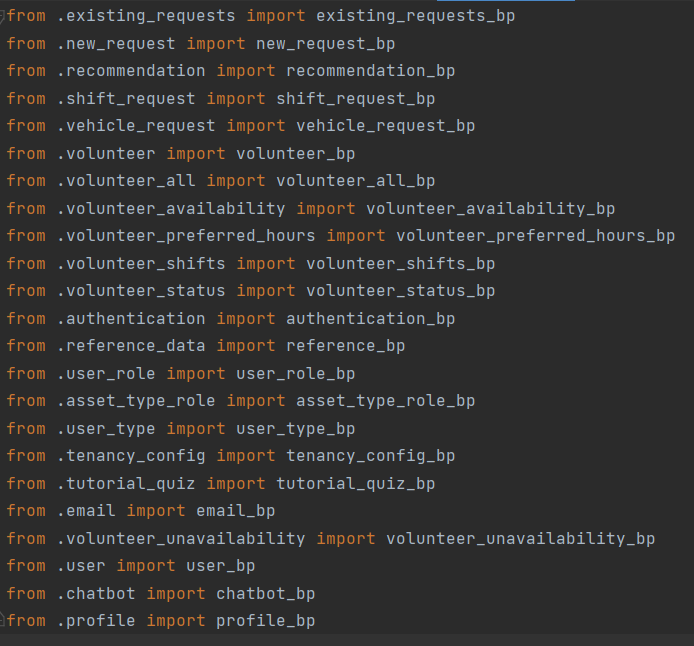
**Add\_argument** adds the parameters to url**.**

**Get\_seats** is the function which is in the repository**.**



After writing the **get** or **post** function, we need to register the API with blueprint. Define a different url for each different request. Then register that blueprint in **\_\_init\_\_.py.**

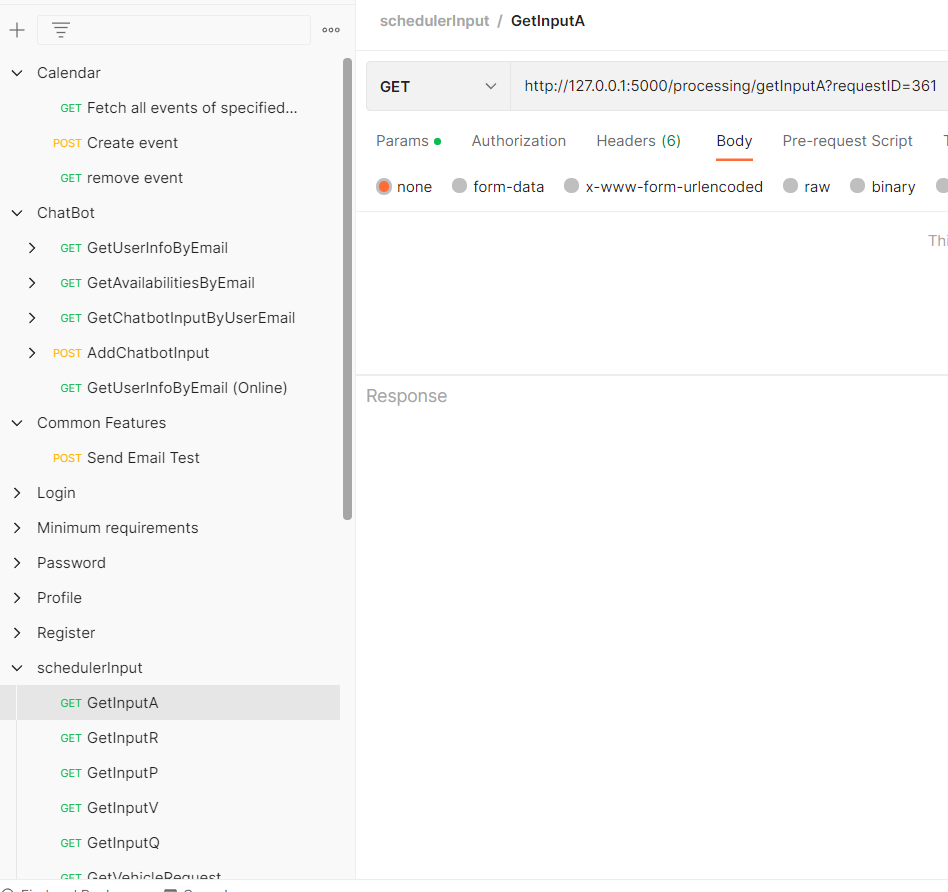
****

****

Through the above process, we can complete the writing of an API.

# APIs Test

We use the **Postman** tool to test the API. Postman plays an important role in our work process. Because we must keep the session running all the time, and cannot run a function alone. But we can indirectly test a function or API by calling the specified function through an API.



# 

# 

# APIs List

This table describes the APIs used in different features. parameters represent the data that needs to be included when using the API.

**Tutorial Feature**

| Description | Type | API | Parameters |
| --- | --- | --- | --- |
| Get question by id | GET | http://127.0.0.1:5000/quiz/getQuestionById?id=7 | id |
| Get random question | GET | http://127.0.0.1:5000/quiz/getRandomQuestion?num=10&role=Driver&difficulty=2 | number，role，difficulty |
| Check answer | GET | 127.0.0.1:5000/quiz/checkAnswer?id=2&answer=B | id，answer |
| Check mutiple answers | GET | http://127.0.0.1:5000/quiz/checkMultipleAns?id=2&ans=A | id，answer |
| Create a new question | GET | http://127.0.0.1:5000/quiz/createQuestion?question\_type=0&role=Basic&description=This is an add question test&choice=[{"id":"A","content":"yes","reason":"Congratulation"},{"id":"B","content":"no","reason":"nonono"}]&difficulty=1&answer=B | Question type，role，description，choice，difficulty，answer |
| Update a question | GET | http://127.0.0.1:5000/quiz/updateQuestion?id=93&role=Basic&description=This is an updateTest&difficulty=1&answer=B | id，role，description，difficulty，answer |
| Delete a question | GET | http://127.0.0.1:5000/quiz/deleteQuestion?id=95 | id |

**User**

| Description | Type | API | Parameters |
| --- | --- | --- | --- |
| Create a new account | POST | http://127.0.0.1:5000/authentication/register | email，password，given name，last name，phone，gender，diet，allergy |
| Edit user’s profile | POST | http://127.0.0.1:5000/profile/editProfile | id,phone,gender,allergy |
| Get one user’s profile | POST | http://127.0.0.1:5000/profile/getProfile | id |
| Send verification code to specified email address | POST | http://127.0.0.1:5000/authentication/send\_code | email |
| Check verification code | POST | http://127.0.0.1:5000/authentication/verify\_code | email，code |
| Reset Password | POST | http://127.0.0.1:5000/authentication/reset | email，new password，repeat passw |
| Get use‘s type | GET | http://127.0.0.1:5000/user-type?userId=25 | id |
| GET a list of all volunteers | GET | http://127.0.0.1:5000/user/getAllVolunteer |  |

**Scheduler**

| Description | Type | API | Parameters |
| --- | --- | --- | --- |
| Get the number of asset for a request | GET | http://127.0.0.1:5000/processing/getInputA?requestID=361 | id |
| Get the number of roles | GET | http://127.0.0.1:5000/processing/getInputR |  |
| Get the number of positions for a request | GET | http://127.0.0.1:5000/processing/getInputP?requestID=338 | id |

**Chatbot**

| Description | Type | API | Parameters |
| --- | --- | --- | --- |
| Get user’s information | GET | http://127.0.0.1:5000/user/getUserByEmail?email=dev1@test.com | email |
| Get user’s availability | GET | http://127.0.0.1:5000/user/getAvailabilityByEmail?email=dev1@test.com | email |
| Get query content | GET | http://127.0.0.1:5000/chatbot/getChatbotInputByUserEmail?email=u7170234@anu.edu.au | email |

**Calender**

| Description | Type | API | Parameters |
| --- | --- | --- | --- |
| Create an event | POST | http://127.0.0.1:5000/unavailability/createUnavailableEvent | User id, title,periodicity，start，end |
| Delete an event | GET | http://127.0.0.1:5000/unavailability/removeUnavailableEvent?eventId=89&userId=49 | event id，user id |

**Supervisor**

| Description | Type | API | Parameters |
| --- | --- | --- | --- |
| Get request information | GET | http://127.0.0.1:5000/shift/request?requestID=354 | id |
| Get a list of requests | GET | http://127.0.0.1:5000/existing\_requests |  |
| Add an asset to a reqeust | POST | http://127.0.0.1:5000/supervisor/addVehicle | request id，asset type，start date，end date |
| Delete an asset from a request | DEL | http://127.0.0.1:5000/supervisor/deleteVehicle?requestId=356&vehicleId=406 | request id，asset id |
| Delete a position from an asset | DEL | http://127.0.0.1:5000/supervisor/deleteVehicleRole?vehicleId=410&positionId=1 | asset id，position id |

# Database

In this project our database contains 12 tables：

| Table | Description |
| --- | --- |
| asset\_request | Store information about the request |
| asset\_request\_vehicle | Store the associated information of vehicle (asset) and request |
| asset\_request\_volunteer | Store the associated information of vehicle and volunteer |
| asset\_type | Store all types of vehicles |
| asset\_type\_role | Store the basic configuration of different types of vehicles |
| chatbot\_input | Store the query content entered by the user in the chatbot |
| password\_verify | Save the verification code when retrieving the password |
| qualification | save all qualifications |
| question | Store all questions related to the tutorial function |
| role | Store all volunteer types |
| unavailability\_time | Store users’ events |
| user | Store information for all volunteers and administrators |

**AI：auto increment**

**PK： Primary Key**

**FK: Foreign Key**

**asset\_request Table**

| Column | Data Format | Description |
| --- | --- | --- |
| id | int AI PK | Request id |
| user\_id | int | The user id that created the request |
| title | varchar(256) | Request title |
| status | varchar(256) | Request status（waiting or un-submitted） |
| last\_update\_datetime | datetime | Time of last modification request |
| created\_time | datetime | The time of request was created |

**asset\_request\_vehicle**

| Column | Data Format | Description |
| --- | --- | --- |
| id | int AI PK | Vehicle id |
| request\_id | Int FK(references asset\_request.id) | The request to which the vehicle is assigned |
| from | datetime | The start time when the vehicle executes the task |
| to | datetime | The end when the vehicle executes the task |
| last\_update\_datetime | datetime | The latest time to modify vehicle information |
| created\_datetime | datetime | Created time |
| asset\_type\_id | int | the type of the vehicle |

**Asset\_request\_volunteer**

| Column | Data Format | Description |
| --- | --- | --- |
| id | int AI PK | Position id |
| user\_id | Int FK(references user.id) | User id |
| vehicle\_id | Int FK(references asset\_request\_vehicle.id) | Vehicle id |
| status | varchar(256) | Pending/Rejected/Confirmed |
| last\_update\_datetime | datetime | The latest time to modify user information |
| created\_datetime | datetime | Created time |
| role\_id | Int FK(references role.id) | the role required for the position |
| qualification\_id | Int FK(references qualification.id) | Qualifications required for the position |

**Asset\_type**

| Column | Data Format | Description |
| --- | --- | --- |
| id | Int AI PK | Vehicle type id |
| code | varchar(256) | Vehicle type code |
| name | varchar(256) | Vehicle name |
| deleted | tinyint(1) | Indicates whether the type of vehicle is available |
| last\_update\_datetime | datetime | The latest time to modify vehicle type |
| created\_datetime | datetime | Created time |

**Asset\_type\_role**

| Column | Data Format | Description |
| --- | --- | --- |
| id | int AI PK | Record id |
| asset\_type\_id | Int FK(references asset\_type.id) | Vehicle type id |
| seat\_number | int | Seat id in a vehicle |
| role\_id | Int FK(references role.id) | The role required by the seat |
| created\_datetime | datetime | Created\_datetime |

**Chatbot\_input**

| Column | Data Format | Description |
| --- | --- | --- |
| id | int AI PK | Record id |
| created\_time | datetime | Created time |
| user\_email | varchar(256) FK(references user.id) | User email |
| content | varchar(1000) | The content queried by user in chatbot |

**Password\_verify**

| Column | Data Format | Description |
| --- | --- | --- |
| id | int AI PK | Record id |
| email | varchar(256) FK(reference user.email) | The email of the user who wants to retrieve the password |
| code | varchar(256) | Verification code |
| created\_datetime | datetime | Created datetime |
| expired\_datetime | datetime | Verification code validity period |

**Qualification**

| Column | Data Format | Description |
| --- | --- | --- |
| id | int AI PK | Qualification id |
| name | varchar(256) | Qualification name |
| deleted | tinyint(1) | Indicates whether the qualication is available |
| last\_update\_datetime | datetime | The latest time to modify qualification |
| created\_datetime | datetime | Created time |

**Question**

| Column | Data Format | Description |
| --- | --- | --- |
| id | bigint AI PK | Question id |
| created\_time | datetime | Created time |
| update\_time | datetime | The latest time to modify question |
| question\_type | enum(’SINGLE’,’MULTIPLE’) | Question type |
| role | varchar(256) FK(reference role.name) | The role that the question belongs to |
| description | varchar(1024) | Question Description |
| choice | text | Description of each Choice |
| difficulty | tinyint | Difficulty level of the question |
| answer | varchar(20) | Answer |
| status | tinyint(1) | Indicates whether the question is available |
| answered\_time | int | The number of times the question was answered |
| correct\_time | int | Number of times the question was answered correctly |

**Role**

| Column | Data Format | Description |
| --- | --- | --- |
| id | int AI PK | role id |
| name | varchar(256) | Role name |
| deleted | tinyint(1) | Indicates whether the role is available |
| last\_update\_time | datetime | The latest time to modify role |
| created\_datetime | datetime | Created time |
| code | code | Role code |

**Unavailability\_time**

| Column | Data Format | Description |
| --- | --- | --- |
| eventId | int AI PK | Record id |
| userId | int PK FK(reference user.id) | User id |
| title | varchar(256) | Event title |
| periodicity | int | Periodicity = 1: daily schedule  Periodicity = 2: weekly schedule  Periodicity = 3: one-off schedule |
| start | datetime | Event start time |
| end | datetime | Event end time |
| status | tinyint(1) | Indicates whether the event is available |

**User**

| Column | Data Format | Description |
| --- | --- | --- |
| id | int AI PK | User id |
| user\_type | enum(‘VOLUNTEER’,’ADMIN’,’ROOT\_ADMIN’) | User type |
| password | varchar(256) | encrypted password |
| incorrect\_password\_count | int | The number of times the user has logged in with an incorrect password |
| last\_sign\_in\_datetime | datetime | Last login time |
| first\_name | varchar(256) | First name |
| last\_name | varchar(256) | Last name |
| mobile\_number | varchar(256) | Mobile number |
| email | varchar(256) | Email |
| preferred\_hours | int | Preferred task time |
| experience\_years | int | Experience yesrs |
| possible\_roles | json | Available roles |
| qualifications | json | Available qualifications |
| availabilities | json | Availability |
| last\_update\_datetime | datetime | The latest time to modify user information |
| created\_datetime | datetime | Created time |
| gender | enum(‘Male’,’Female’) | Gender |
| diet | enum('meals','vegetarian','kosher','halal','vegan') | Diet habits |
| allergy | varchar(256) | allergy food |