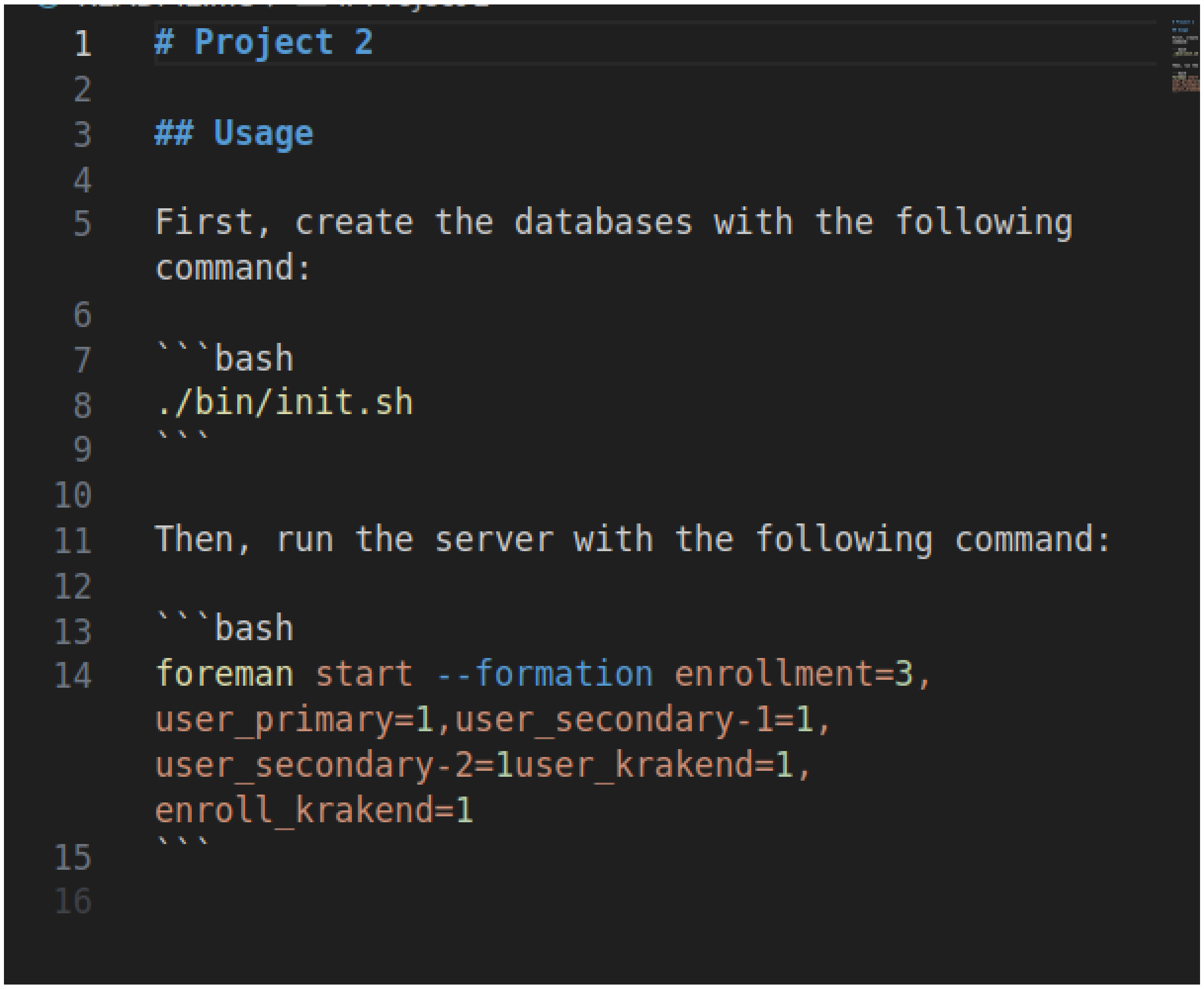
CPSC 449: Project 1 - Group 2

Juan Uriarte

CPSC 449, Section 02, Fall 2023

Running the Code / README.md

First, create the databases with the following command:



~~~
# Project 2

## Usage

First, create the databases with the following command:

```bash
./bin/init.sh
~~~

Then, run the server with the following command:

```bash
foreman start --formation enrollment=3,
user_primary=1,user_secondary-1=1,
user_secondary-2=1user_krakend=1,
enroll_krakend=1
```

```

The project is divided into different services while utilizing foreman to run them.
```

API Design

auth_api.py

“/register” : allows new users to register. If name already exists in the user_auth table, it returns status code 409 with “user already exists.” If name does not exist, it hashes a password and inserts the new user details into user_auth table.

“/login” : allows users to login and grabs data from user_auth table. If the user does not exist, status code 409 is shown with an error message. If a user exists, it verifies the password, if the password is incorrect status code 401 is shown with the message “invalid username or password”. Once login is successful, it generates JWT and JWS tokens. The tokens are then written to the jwt.json and jwks.json.

auth_models.py

Pydantic models consisted of:

1) User Model: id, username, password, and roles.
2) RegisterUserRequest Model: username, password, and roles
3) VerifyUserRequest Model: username, password

database.py

The database consists of four different retrieval functions:

1) list_courses(): Grabs data related to course information and related department data.
2) list_selections(): Grabs data related to section information with the related course, department, and instructor data.
3) list_enrollments(): Grabs enrollment data, user information, section, course, department, and instructor data.
4) list_waitlist(): waitlist information similar to enrollments.

enroll_api.py

list_courses
GET /courses

add_course
POST /courses

Parameters:
- body: body (AddCourseRequest, required)

get_course
GET /courses/{course_id}

Parameters:
- path: course_id (integer, required)

get_course_waitlist
GET /courses/{course_id}/waitlist

Parameters:
- path: course_id (integer, required)

list_sections
GET /sections

Parameters:
- query: course_id (integer)

add_section
POST /sections

Parameters:
- body: body (AddSectionRequest, required)

get_section

GET /sections/{section_id}

Parameters:
● path: section_id (integer, required)
delete_section

DELETE /sections/{section_id}
Parameters:
● path: section_id (integer, required)

update_section
PATCH /sections/{section_id}

Parameters:
● path: section_id (integer, required)
● body: body (UpdateSectionRequest, required)

list_section_enrollments
GET /sections/{section_id}/enrollments

Parameters:
● path: section_id (integer, required)
● query: status (any)

list_section_waitlist
GET /sections/{section_id}/waitlist

Parameters:
● path: section_id (integer, required)

list_users
GET /users

get_user
GET /users/{user_id}

Parameters:
● path: user_id (integer, required)

list_user_enrollments

GET /users/{user_id}/enrollments

Parameters:
- path: user_id (integer, required)
- query: status (any)

create_enrollment

POST /users/{user_id}/enrollments

Parameters:
- path: user_id (integer, required)
- body: body (CreateEnrollmentRequest, required)

list_user_sections
GET /users/{user_id}/sections

Parameters:
- path: user_id (integer, required)
- query: type (any)

list_user_waitlist
GET /users/{user_id}/waitlist

Parameters:
- path: user_id (integer, required)

drop_user_enrollment

DELETE /users/{user_id}/enrollments/{section_id}

Parameters:
- path: user_id (integer, required)
- path: section_id (integer, required)

drop_user_waitlist
DELETE /users/{user_id}/waitlist/{section_id}

Parameters:
- path: user_id (integer, required)
- path: section_id (integer, required)

drop_section_enrollment
DELETE /sections/{section_id}/enrollments/{user_id}

Parameters:
● path: section_id (integer, required)
● path: user_id (integer, required)

mkclaims.py

```
#!/usr/bin/env python

import os
import sys
import json
import datetime


def usage():
   program = os.path.basename(sys.argv[0])
   print(f"Usage: {program} USERNAME USER_ID ROLE...", file=sys.stderr)


def expiration_in(minutes):
   creation = datetime.datetime.now(tz=datetime.timezone.utc)
   expiration = creation + datetime.timedelta(minutes=minutes)
   return creation, expiration


def generate_claims(username, user_id, roles):
   _, exp = expiration_in(20)

   claims = {
       "aud": "krakend.local.gd",
       "iss": "auth.local.gd",
       "sub": username,
       "jti": str(user_id),
       "roles": roles,
       "exp": int(exp.timestamp()),
   }
```

```
    token = {
        "access_token": claims,
        "refresh_token": claims,
        "exp": int(exp.timestamp()),
    }

    token = json.dumps(token, indent=4)
    return token


if __name__ == "__main__":
    if len(sys.argv) < 4:
        usage()
        sys.exit(1)

    generate_claims(sys.argv[1], sys.argv[2], sys.argv[3:])
```

Generates JWT claims and tokens for users. It checks if the right number of arguments are provided. If the right number is not provided it will exit with an error. If the arguments are correct, the function "generate_claims" is called and will utilize the token. As a result, this would create a token for a specific username, user_id, and list of roles. The token will be valid for 20 minutes.

mkjwks.py

```
#!/usr/bin/env python
import os
import sys
import json

from jwcrypto import jwk


def usage():
    program = os.path.basename(sys.argv[0])
    print(f"Usage: {program} KEY_ID...", file=sys.stderr)


def generate_keys(key_ids):
    keys = [jwk.JWK.generate(kid=key_id, kty="RSA", alg="RS256") for key_id
in key_ids]
    exported_keys = [
        key.export(private_key=private) for key in keys for private in
[False, True]
    ]
    keys_as_json = [json.loads(exported_key) for exported_key in
exported_keys]
    jwks = {"keys": keys_as_json}
    jwks = json.dumps(jwks, indent=4)
    return jwks


if __name__ == "__main__":
    if len(sys.argv) == 1:
        usage()
        sys.exit(1)

    generate_keys(sys.argv[1:])
```

The generate_key_function takes a list of “key_ids” as an argument and generates JWK keys for each key ID in the list of RSA algorithms correlated to RS256. The keys are then converted to json format and added to JWKS format. THE JWKS is then converted to a JSON string and returned.

model_requests.py

The following classes are :

ListUserSectionType: to specify the user relationship to a course

CreateEnrollmentRequest: refers to section ID

CreateEnrollmentResponse: It correlates fields from the Enrollment model and it has a waitlist_position field.

AddCourseRequest: Adds a new course including course code, name and ID of the Department.

AddSectionRequest: Contains course ID, classroom, capacity, timing and instructor

ListSelectionEnrollmentsItem: a list and it includes user and grade.

ListSelectionWaitlistItem: Includes user and position in the waitlist.
UpdateSectionRequest: Fields are able to update including to freeze the section and instructor ID.

RegisterUserRequest: It contains fields for username, password, and roles assigned to users.

models.py

The following models and fields are:

User: id, username, first_name, last_name

Department: id and name

Course: id, code, name and department

Section: id, course, classroom, capacity, waitlist_capacity, day, begin_time, end_time, freeze and instructor

EnrollmentStatus: enrolled, waitlisted, or dropped

Waitlist: user, section, position

register.py

```
import base64
import hashlib
import secrets
from models import *


ALGORITHM = "pbkdf2_sha256"


def hash_password(password, salt=None, iterations=260000):
   if salt is None:
       salt = secrets.token_hex(16)
   assert salt and isinstance(salt, str) and "$" not in salt
   assert isinstance(password, str)
   pw_hash = hashlib.pbkdf2_hmac(
       "sha256", password.encode("utf-8"), salt.encode("utf-8"),
iterations
   )
   b64_hash = base64.b64encode(pw_hash).decode("ascii").strip()
   return "{}${}${}${}".format(ALGORITHM, iterations, salt, b64_hash)


def verify_password(password, password_hash):
   if (password_hash or "").count("$") != 3:
       return False
   algorithm, iterations, salt, b64_hash = password_hash.split("$", 3)
   iterations = int(iterations)
   assert algorithm == ALGORITHM
   compare_hash = hash_password(password, salt, iterations)
   return secrets.compare_digest(password_hash, compare_hash)
```

The hash_password function checks the password by using the PBKDF2 to hash the password. It then returns the hashed password in a string format that includes the algorithm, iteration, count, salt, and base64 encoded hash.

The verify_password function consists of parameters password and password_hash that correlates to plain-text passwords to verify. This function uses secrets.compare_digest to compare the generated hash with the stored hash. If it matches, it returns true, otherwise false.

Bin Folder

init.sh

```
#!/bin/sh

./share/users/user_schema_init.py
./share/enroll/schema_init.py
```

Runs and initializes the schema.

etc Folder/enroll

krakend.json

```
{
    "$schema": "https://www.krakend.io/schema/v2.4/krakend.json",
    "version": 3,
    "endpoints": [
        {
            "endpoint": "/enoll_api/courses/",
            "method": "GET",
            "extra_config": {
                "auth/validator":{
                    "alg": "RS256",
                    "audience":["http://localhost:5200"],
                    "roles_key": "roles",
                    "roles": ["student"],
                   "jwk_local_path": "./app/public.json",
                    "cache": true,
                    "propagate_claims": [
                       ["sub", "x-user"],
                       ["roles", "x-role"]
                    ]
                    }
                },
            "backend": [
                {
                    "url_pattern": "/courses/",
                    "method": "GET",
                    "host": [
                        "http://localhost:5000",
                        "http://localhost:5001",
                        "http://localhost:5002"
                    ]
                }
            ]
        },
        {
            "endpoint": "/enoll_api/courses/",
            "method": "POST",
            "extra_config": {
                "auth/validator":{
                    "alg": "RS256",
                    "audience":["http://localhost:5200"],
                    "roles_key": "roles",
                    "roles": ["registar"],
                   "jwk_local_path": "./app/public.json",
                    "cache": true,
                    "propagate_claims": [
                       ["sub", "x-user"],
                       ["roles","x-role"]
                    ]
                    }
                },
            "backend": [
                {
                    "url_pattern": "/courses/",
                    "method": "POST",
                    "host": [
                        "http://localhost:5000",
                        "http://localhost:5001",
                        "http://localhost:5002"
                    ]
                }
            ]
        },
        {
            "endpoint": "/enroll_api/courses/{course_id}/",
            "method": "GET",
            "extra_config": {
                "auth/validator":{
                    "alg": "RS256",
                    "audience":["http://localhost:5200"],
                    "roles_key": "roles",
                    "roles": ["student"],
                   "jwk_local_path": "./etc/public.json",
                "cache": true,
                "propagate_claims": [
                   ["sub", "x-user"],
                   ["roles","x-role"]
                ]
                }
            },
        "backend": [
            {
                "url_pattern": "/coures/{course_id}/",
                "method": "GET",
                "host": [
                    "http://localhost:5000",
                    "http://localhost:5001",
                    "http://localhost:5002"
                ]
            }
        ]
    },
    {
        "endpoint": "/enroll_api/courses/{course_id}/waitlist",
        "method": "GET",
        "extra_config": {
            "auth/validator":{
                "alg": "RS256",
                "audience":["http://localhost:5200"],
                "roles_key": "roles",
                "roles": ["instructor"],
               "jwk_local_path": "./app/public.json",
                "cache": true,
                "propagate_claims": [
                   ["sub", "x-user"],
                   ["roles","x-role"]
                ]
                }
            },
        "backend": [
            {
                "url_pattern": "/courses/{course_id}/waitlist",
                "method": "GET",
                "host": [
                    "http://localhost:5000",
                    "http://localhost:5001",
                    "http://localhost:5002"
                ]
            }
        ]
    },
    {
        "endpoint": "/enroll_api/sections",
        "method": "GET",
        "extra_config": {
            "auth/validator":{
                "alg": "RS256",
                "audience":["http://localhost:5200"],
                "roles_key": "roles",
                "roles": ["student"],
               "jwk_local_path": "./app/public.json",
                "cache": true,
                "propagate_claims": [
                   ["sub", "x-user"],
                   ["roles","x-role"]
                ]
                }
            },
        "backend": [
            {
                "url_pattern": "/sections",
                "method": "GET",
                "host": [
                    "http://localhost:5000",
                    "http://localhost:5001",
                    "http://localhost:5002"
                ]
            }
        ]
    },
    {
        "endpoint": "/enroll_api/sections",
        "method": "POST",
        "extra_config": {
            "auth/validator":{
                "alg": "RS256",
                "audience":["http://localhost:5200"],
                "roles_key": "roles",
                "roles": ["registar"],
               "jwk_local_path": "./app/public.json",
                "cache": true,
                "propagate_claims": [
                   ["sub", "x-user"],
                   ["roles","x-role"]
                ]
                }
            },
        "backend": [
            {
                "url_pattern": "/sections",
                "method": "POST",
                "host": [
                    "http://localhost:5000",
                    "http://localhost:5001",
                    "http://localhost:5002"
                ]
            }
        ]
    },
    {
        "endpoint": "/enroll_api/sections/{section_id}",
        "method": "GET",
        "extra_config": {
            "auth/validator":{
                "alg": "RS256",
                "audience":["http://localhost:5200"],
                "roles_key": "roles",
                "roles": ["instructor"],
               "jwk_local_path": "./app/public.json",
                "cache": true,
                "propagate_claims": [
                   ["sub", "x-user"],
                   ["roles","x-role"]
                ]
                }
            },
        "backend": [
            {
                "url_pattern": "/sections/{section_id}",
                "method": "GET",
                "host": [
                    "http://localhost:5000",
                    "http://localhost:5001",
                    "http://localhost:5002"
                ]
            }
        ]
    },
    {
        "endpoint": "/enroll_api/sections/{section_id}",
        "method": "PATCH",
        "extra_config": {
            "auth/validator":{
                "alg": "RS256",
                "audience":["http://localhost:5200"],
                "roles_key": "roles",
                "roles": ["registar"],
               "jwk_local_path": "./app/public.json",
                "cache": true,
                "propagate_claims": [
                   ["sub", "x-user"],
                   ["roles","x-role"]
                ]
                }
            },
        "backend": [
            {
                "url_pattern": "/sections/{section_id}",
                "method": "PATCH",
                "host": [
                    "http://localhost:5000",
                    "http://localhost:5001",
                    "http://localhost:5002"
                ]
            }
        ]
    },
    {
        "endpoint": "/enroll_api/sections/{section_id}",
        "method": "DELETE",
        "extra_config": {
            "auth/validator":{
                "alg": "RS256",
                "audience":["http://localhost:5200"],
                "roles_key": "roles",
                "roles": ["registar"],
               "jwk_local_path": "./app/public.json",
                "cache": true,
                "propagate_claims": [
                   ["sub", "x-user"],
                   ["roles","x-role"]
                ]
                }
            },
        "backend": [
            {
                "url_pattern": "/sections/{section_id}",
                "method": "DELETE",
                "host": [
                    "http://localhost:5000",
                    "http://localhost:5001",
                    "http://localhost:5002"
                ]
            }
        ]
    },
    {
        "endpoint": "/enroll_api/sections/{section_id}/enrollments",
        "method": "GET",
        "extra_config": {
            "auth/validator":{
                "alg": "RS256",
                "audience":["http://localhost:5200"],
                "roles_key": "roles",
                "roles": ["registar","instructor"],
               "jwk_local_path": "./app/public.json",
                "cache": true,
                "propagate_claims": [
                   ["sub", "x-user"],
                   ["roles","x-role"]
                ]
                }
            },
        "backend": [
            {
                "url_pattern": "/sections/{section_id}/enrollments",
                "method": "GET",
                "host": [
                    "http://localhost:5000",
                    "http://localhost:5001",
                    "http://localhost:5002"
                ]
            }
        ]
    },
    {
        "endpoint": "/enroll_api/sections/{section_id}/waitlist",
        "method": "GET",
        "extra_config": {
            "auth/validator":{
                "alg": "RS256",
                "audience":["http://localhost:5200"],
                "roles_key": "roles",
                "roles": ["registar", "instructor"],
               "jwk_local_path": "./app/public.json",
                "cache": true,
                "propagate_claims": [
                   ["sub", "x-user"],
                   ["roles","x-role"]
                ]
                }
            },
        "backend": [
            {
                "url_pattern": "/sections/{section_id}/waitlist",
                "method": "GET",
                "host": [
                    "http://localhost:5000",
                    "http://localhost:5001",
                    "http://localhost:5002"
                ]
            }
        ]
    },
    {
        "endpoint": "/enroll_api/users",
        "method": "GET",
        "extra_config": {
            "auth/validator":{
                "alg": "RS256",
                "audience":["http://localhost:5200"],
                "roles_key": "roles",
                "roles": ["registar"],
               "jwk_local_path": "./app/public.json",
                "cache": true,
                "propagate_claims": [
                   ["sub", "x-user"],
                   ["roles","x-role"]
                ]
                }
            },
        "backend": [
            {
                "url_pattern": "/users/",
                "method": "GET",
                "host": [
                    "http://localhost:5000",
                    "http://localhost:5001",
                    "http://localhost:5002"
                ]
            }
        ]
    },
    {
        "endpoint": "/enroll_api/users/{user_id}",
        "method": "GET",
        "extra_config": {
            "auth/validator":{
                "alg": "RS256",
                "audience":["http://localhost:5200"],
                "roles_key": "roles",
                "roles": ["registar"],
               "jwk_local_path": "./app/public.json",
                "cache": true,
                "propagate_claims": [
                   ["sub", "x-user"],
                   ["roles","x-role"]
                ]
                }
            },
        "backend": [
            {
                "url_pattern": "/users/{user_id}",
                "method": "GET",
                "host": [
                    "http://localhost:5000",
                    "http://localhost:5001",
                    "http://localhost:5002"
                ]
            }
        ]
    },
    {
        "endpoint": "/enroll_api/users/{user_id}/enrollments",
        "method": "GET",
        "extra_config": {
            "auth/validator":{
                "alg": "RS256",
                "audience":["http://localhost:5200"],
                "roles_key": "roles",
                "roles": ["instructor"],
               "jwk_local_path": "./app/public.json",
                "cache": true,
                "propagate_claims": [
                   ["sub", "x-user"],
                   ["roles","x-role"]
                ]
                }
            },
        "backend": [
            {
                "url_pattern": "/users/{user_id}/enrollments",
                "method": "GET",
                "host": [
                    "http://localhost:5000",
                    "http://localhost:5001",
                    "http://localhost:5002"
                ]
            }
        ]
    },
    {
        "endpoint": "/enroll_api/users/{user_id}/enrollments",
        "method": "POST",
        "extra_config": {
            "auth/validator":{
                "alg": "RS256",
                "audience":["http://localhost:5200"],
                "roles_key": "roles",
                "roles": ["student"],
               "jwk_local_path": "./app/public.json",
                "cache": true,
                "propagate_claims": [
                   ["sub", "x-user"],
                   ["roles","x-role"]
                ]
                }
            },
        "backend": [
            {
                "url_pattern": "/users/{user_id}/enrollments",
                "method": "POST",
                "host": [
                    "http://localhost:5000",
                    "http://localhost:5001",
                    "http://localhost:5002"
                ]
            }
        ]
    },
    {
        "endpoint": "/enroll_api/users/{user_id}/sections",
        "method": "GET",
        "extra_config": {
            "auth/validator":{
                "alg": "RS256",
                "audience":["http://localhost:5200"],
                "roles_key": "roles",
                "roles": ["instructor"],
               "jwk_local_path": "./app/public.json",
                "cache": true,
                "propagate_claims": [
                   ["sub", "x-user"],
                   ["roles","x-role"]
                ]
                }
            },
        "backend": [
            {
                "url_pattern": "/users/{user_id}/sections",
                "method": "GET",
                "host": [
                    "http://localhost:5000",
                    "http://localhost:5001",
                    "http://localhost:5002"
                ]
            }
        ]
    },
    {
        "endpoint": "/enroll_api/users/{user_id}/waitlist",
        "method": "GET",
        "extra_config": {
            "auth/validator":{
                "alg": "RS256",
                "audience":["http://localhost:5200"],
                "roles_key": "roles",
                "roles": ["student"],
               "jwk_local_path": "./app/public.json",
                "cache": true,
                "propagate_claims": [
                   ["sub", "x-user"],
                   ["roles","x-role"]
                ]
                }
            },
        "backend": [
            {
                "url_pattern": "/users/{user_id}/waitlist",
                "method": "GET",
                "host": [
                    "http://localhost:5000",
                    "http://localhost:5001",
                    "http://localhost:5002"
                ]
            }
        ]
    },
    {
        "endpoint": "/users/{user_id}/enrollments/{section_id}",
        "method": "DELETE",
        "extra_config": {
            "auth/validator":{
                "alg": "RS256",
                "audience":["http://localhost:5200"],
                "roles_key": "roles",
                "roles": ["student"],
               "jwk_local_path": "./app/public.json",
                "cache": true,
                "propagate_claims": [
                   ["sub", "x-user"],
                   ["roles","x-role"]
                ]
```

```
                    }
                },
            "backend": [
                {
                    "url_pattern":
"/users/{user_id}/enrollments/{section_id}",
                    "method": "DELETE",
                    "host": [
                        "http://localhost:5000",
                        "http://localhost:5001",
                        "http://localhost:5002"
                    ]
                }
            ]
        },
        {
            "endpoint": "/sections/{section_id}/enrollments/{user_id}",
            "method": "DELETE",
            "extra_config": {
                "auth/validator":{
                    "alg": "RS256",
                    "audience":["http://localhost:5200"],
                    "roles_key": "roles",
                    "roles": ["registar", "instructor"],
                   "jwk_local_path": "./app/public.json",
                    "cache": true,
                    "propagate_claims": [
                       ["sub", "x-user"],
                       ["roles","x-role"]
                    ]
                    }
                },
            "backend": [
                {
                    "url_pattern":
"/sections/{section_id}/enrollments/{user_id}",
                    "method": "DELETE",
                    "host": [
                        "http://localhost:5000",
                        "http://localhost:5001",
                             "http://localhost:5002"
                         ]
                    }
               ]
          }
     ]
}
```

etc Folder/user

krakend.json

```
{
   "$schema": "https://www.krakend.io/schema/v2.4/krakend.json",
   "version": 3,
   "endpoints": [
        {
            "endpoint":"/auth_api/register/",
            "method":"POST",
            "backend" : [
                {
                    "url_pattern":"/register/",
                    "method":"POST",
                    "host":["http://localhost:5100"],
                    "encoding":"json"
                }
            ]
        },
        {
            "endpoint":"/auth_api/login/",
            "method":"GET",
            "backend":[
                {
                    "url_pattern":"/login/",
                    "method":"GET",
                    "host":["http://localhost:5100"],
                    "encoding":"json"
                }
            ]
        }
   ]
}
```

“/auth_api/register/”: clients will use this path when making requests to Krakend. The method is POST that will make requests to the url pattern “/register/” endpoint. The host will be “http://localhost:5100”. The encoding type will be json.

“/auth_api/login/”: clients will use this path through Krakend. The method is GET with similar backend services like above except for the url pattern which will be “/login/”.

primary.yml

```
fuse:
 dir: "./var/user/primary/fuse"
 allow-other: false

data:
 dir: "./var/user/primary/data"
 compress: true

http:
 addr: ":20202"

lease:
 type: "static"
 hostname: "127.0.0.1"
 advertise-url: "http://127.0.0.1:20202"
 candidate: true

exec: "uvicorn --port $PORT app.auth_api:app --reload"
```

secondary-1.yml

```
fuse:
 dir: "./var/user/secondary-1/fuse"
 allow-other: false

data:
 dir: "./var/user/secondary-1/data"
```

```
compress: true

http:
addr: ":20203"

lease:
type: "static"
advertise-url: "http://127.0.0.1:20202"
candidate: false
```

secondary-2.yml

```
fuse:
dir: "./var/user/secondary-2/fuse"
allow-other: false

data:
dir: "./var/user/secondary-2/data"
compress: true

http:
addr: ":20204"

lease:
type: "static"
advertise-url: "http://127.0.0.1:20202"
candidate: false
```

share Folder/enroll

schema_init.py

```
#!/usr/bin/env python3
import argparse
import sqlite3
import os

parser = argparse.ArgumentParser(
prog="schema_init.py",
description="Initialize the SQLite database schema",
)
parser.add_argument("-i", "--input", help="Input schema file",
default="./share/enroll/schema.sql")
parser.add_argument("-f", "--file", help="SQLite database file",
default="./var/enroll/enroll.db")

args = parser.parse_args()

schema_sql_file = open(args.input, "r")
schema_sql = schema_sql_file.read()

schema_testdata_sql_file = open(args.input.replace(".sql",
"_testdata.sql"), "r")
schema_testdata_sql = schema_testdata_sql_file.read()

if os.path.isfile(args.file):
answer = input("Database file already exists. Overwrite? (y/n) ")
if answer.lower() == "y":
os.remove(args.file)
else:
print("Aborting...")
exit(1)

conn = sqlite3.connect(args.file)

c = conn.cursor()
c.executescript(schema_sql)

insertTestData = input("Insert test data? (y/n) ")
if insertTestData.lower() == "y":
c.executescript(schema_testdata_sql)

conn.commit()
conn.close()
```

schema_testdata.sql

```
INSERT INTO users (username, first_name, last_name) VALUES
```

```
('alice', 'Alice', 'Smith'),
('bob', 'Bob', 'Jones'),
('carol', 'Carol', 'Williams'),
('dave', 'Dave', 'Brown'),
('eve', 'Eve', 'Johnson'),
('frank', 'Frank', 'Miller'),
('grace', 'Grace', 'Davis'),
('harry', 'Harry', 'Wilson'),
('isabel', 'Isabel', 'Taylor'),
('jack', 'Jack', 'Garcia'),
('kim', 'Kim', 'Martinez'),
('larry', 'Larry', 'Anderson'),
('mary', 'Mary', 'Lopez'),
('nick', 'Nick', 'Robinson'),
('olivia', 'Olivia', 'Walker'),
('peter', 'Peter', 'Perez'),
('qiana', 'Qiana', 'Johnson'),
('robert', 'Robert', 'Brown'),
('sarah', 'Sarah', 'Davis'),
('thomas', 'Thomas', 'Miller'),
('ursula', 'Ursula', 'Wilson');

INSERT INTO departments VALUES
(1, 'Computer Science'),
(2, 'Engineering'),
(3, 'Mathematics');

INSERT INTO courses VALUES
(1, 'CPSC 449', 'Web Back-End Engineering', 1),
(2, 'MATH 150A', 'Calculus I', 3);

INSERT INTO sections VALUES
(1, 1, 'CS102', 30, 15, 'Tuesday', '7pm', '9:45pm', 2, 0, 0),
(2, 1, 'CS104', 30, 15, 'Wednesday', '4pm', '6:45pm', 2, 0, 0),
(3, 2, 'MH302', 35, 15, 'Monday', '12pm', '2:45pm', 4, 0, 0),
(4, 2, 'MH107', 32, 15, 'Thursday', '9am', '11:30am', 4, 0, 0);

INSERT INTO enrollments VALUES
(5, 1, 'Enrolled', NULL, '2023-09-15'),
(6, 1, 'Enrolled', NULL, '2023-09-15'),
(7, 1, 'Enrolled', NULL, '2023-09-15'),
(8, 1, 'Enrolled', NULL, '2023-09-15'),
(9, 2, 'Enrolled', NULL, '2023-09-15'),
(10, 2, 'Dropped', NULL, '2023-09-15'),
(11, 2, 'Enrolled', NULL, '2023-09-15'),
(12, 3, 'Enrolled', NULL, '2023-09-15'),
(13, 3, 'Dropped', NULL, '2023-09-15'),
(14, 4, 'Enrolled', NULL, '2023-09-15'),
(5, 3, 'Enrolled', NULL, '2023-09-15'),
(6, 4, 'Enrolled', NULL, '2023-09-15'),
(7, 2, 'Enrolled', NULL, '2023-09-15');

-- For waitlist table
INSERT INTO waitlist VALUES
(8, 3, 1, '2023-09-15'),
(9, 2, 1, '2023-09-15'),
(10, 4, 2, '2023-09-15'),
(11, 1, 2, '2023-09-15'),
(12, 3, 3, '2023-09-15'),
(13, 1, 3, '2023-09-15'),
(14, 2, 4, '2023-09-15');
```

schema.sql

```
CREATE TABLE users (
id INTEGER PRIMARY KEY AUTOINCREMENT,
username TEXT NOT NULL,
first_name TEXT NOT NULL,
last_name TEXT NOT NULL
);

CREATE TABLE departments (
id INTEGER PRIMARY KEY AUTOINCREMENT,
name TEXT NOT NULL
);

CREATE TABLE courses (
id INTEGER PRIMARY KEY AUTOINCREMENT,
code TEXT NOT NULL,
name TEXT NOT NULL,
department_id INTEGER NOT NULL REFERENCES departments (id)
);

CREATE TABLE sections (
id INTEGER PRIMARY KEY AUTOINCREMENT,
course_id INTEGER NOT NULL REFERENCES courses (id),
classroom TEXT, -- NULL if online
capacity INTEGER NOT NULL,
waitlist_capacity INTEGER NOT NULL,
day TEXT NOT NULL,
begin_time TEXT NOT NULL,
end_time TEXT NOT NULL,
instructor_id INTEGER NOT NULL REFERENCES users (id),
freeze BOOLEAN NOT NULL DEFAULT FALSE,
deleted BOOLEAN NOT NULL DEFAULT FALSE
);

CREATE TABLE enrollments (
user_id INTEGER NOT NULL REFERENCES users (id),
section_id INTEGER NOT NULL REFERENCES sections (id),
status TEXT NOT NULL,
grade TEXT,
date DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
PRIMARY KEY (user_id, section_id)
);

CREATE TABLE waitlist (
user_id INTEGER NOT NULL REFERENCES users (id),
section_id INTEGER NOT NULL REFERENCES sections (id),
position INTEGER NOT NULL,
date DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
PRIMARY KEY (user_id, section_id)
);
```

share Folder/users

user_schema_init.py

```
#!/usr/bin/env python3
import argparse
import sqlite3
import os

parser = argparse.ArgumentParser(
   prog="user_schema_init.py",
   description="Initialize the SQLite database schema",
)
parser.add_argument(
   "-i", "--input", help="Input schema file",
default="./share/users/user_service_schema.sql"
)
parser.add_argument("-f", "--file", help="SQLite database file",
default="./var/user/primary/fuse/auth.db")

args = parser.parse_args()

schema_sql_file = open(args.input, "r")
schema_sql = schema_sql_file.read()

schema_testdata_sql_file = open(args.input.replace(".sql",
"_testdata.sql"), "r")
schema_testdata_sql = schema_testdata_sql_file.read()

if os.path.isfile(args.file):
   answer = input("Database file already exists. Overwrite? (y/n) ")
   if answer.lower() == "y":
       os.remove(args.file)
   else:
       print("Aborting...")
       exit(1)

conn = sqlite3.connect(args.file)

c = conn.cursor()
c.executescript(schema_sql)
```

```
insertTestData = input("Insert test data? (y/n) ")
if insertTestData.lower() == "y":
   c.executescript(schema_testdata_sql)

conn.commit()
conn.close()
```

user_service_schema_testdata.sql

```
INSERT INTO users_auth VALUES (1, 'kylew', 'pass', 'student');

INSERT INTO jwks VALUES (1,'kylew','1234567890');
```

user_service_schema.sql

```
CREATE TABLE users_auth (
   id INTEGER PRIMARY KEY AUTOINCREMENT,
   username TEXT NOT NULL UNIQUE,
   password TEXT NOT NULL,
   roles TEXT NOT NULL
);

CREATE TABLE jwks (
 id INTEGER PRIMARY KEY AUTOINCREMENT,
 key_id TEXT UNIQUE,
 jwk TEXT
);
```

var/user/primary/data

clusterid

LFSC002B336B101C702F

var/user/secondary-1/data

clusterid

LFSC002B336B101C702F

var/user/secondary-2/data

LFSC002B336B101C702F

.gitignore

```
venv
.env
.venv
.vscode
__pycache__
enroll.db
enroll.db*
auth.db
auth.db*
```

Procfile

```
enrollment: uvicorn --port $PORT app.enroll_api:app --reload
user_primary: ./bin/litefs mount -config ./etc/user/primary.yml -fuse.debug -tracing
user_secondary-1: ./bin/litefs mount -config ./etc/user/secondary-1.yml -fuse.debug -tracing
user_secondary-2: ./bin/litefs mount -config ./etc/user/secondary-2.yml -fuse.debug -tracing
user_krakend: echo ./etc/user/krakend.json | entr -nrz krakend run -c ./etc/user/krakend.json -p $PORT
enroll_krakend: echo ./etc/enroll/krakend.json | entr -nrz krakend run -c ./etc/enroll/krakend.json -p
$PORT
```

Enrollment: starts the enrollment application using uvicorn. It then runs the application that has app.enroll_api on a port.

Auth: similar to enrollment application.

User_primary, user_secondary-1, user_secondary-2 will mount a file system using litefs with different configurations.

User_krakend: it monitors changes with the krakend.json file for the user service using entr. If the file changes, the krakend server is restarted with the new configuration.

Enroll_krakend: similar to user_krakend but this is for enroll_krakend.

requirments.txt

```
annotated-types==0.5.0
anyio==3.7.1
fastapi==0.103.2
h11==0.14.0
idna==3.4
pydantic_core==2.10.1
sniffio==1.3.0
starlette==0.27.0
uvicorn==0.23.2
jwcrypto==1.5.0
```