Project 2 Report

Group Members:

Codie Tamida
Divya Tanwar
Donald Novasky
Jia Wei Ng
Mc Gabriel Fernandez
Rishub Goel

Implement an authentication service

Users resources

The API should allow users to:

- Register a new user

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/register' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "first_name": "Mc Gabriel",
  "last_name": "Fernandez",
  "username": "McG",
  "password": "peepoo",
  "role": [
    "student"
  ]
}'
```

Request URL

```
http://127.0.0.1:8000/register
```

Server response

Code	Details
200	**Response body**

```
{
  "status": 201,
  "message": "McG created!",
  "data": [
    {
      "user_id": 1,
      "username": "McG",
      "first_name": "Mc Gabriel",
      "last_name": "Fernandez",
      "role": [
        "Student"
      ]
    }
  ]
}
```

Download

Response headers

```
content-length: 148
content-type: application/json
date: Thu,26 Oct 2023 06:25:35 GMT
server: uvicorn
```

```python
@app.post("/register")
def register(request: NewAccountRequest, db:sqlite3.Connection = Depends(get_db)):

    user = get_user_by_username(request.username, db)

    if user:
        raise_exception(HTTPStatus.CONFLICT, f'User with username {user["username"]} already exists')

    if gracefully_handle_db_transaction(create_user_sql_script(request), db):
        user = get_user_by_username(request.username, db)

        return create_response(HTTPStatus.CREATED, f'{user["username"]} created!', user)
```

Controller logic for /register endpoint

- Check a user's password

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/login' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "username": "McG",
  "password": "peepoo"
}'
```

Request URL

```
http://127.0.0.1:8000/login
```

Server response

Code	Details
200	Response body


```
{
  "access_token": {
    "aud": "krakend.local.gd",
    "iss": "auth.local.gd",
    "sub": "McG",
    "jti": "1",
    "roles": [
      "Student"
    ],
    "exp": 1698302933
  },
  "refresh_token": {
    "aud": "krakend.local.gd",
    "iss": "auth.local.gd",
    "sub": "McG",
    "jti": "1",
    "roles": [
      "Student"
    ],
    "exp": 1698302933
  },
  "exp": 1698302933
}
```


Download

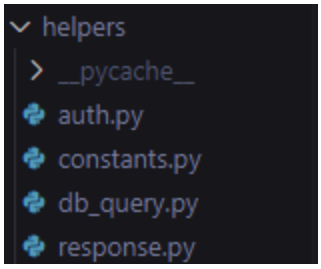
Response headers

```
content-length: 265
content-type: application/json
date: Thu,26 Oct 2023 06:28:53 GMT
server: uvicorn
```

Sample request and response body for /login endpoint

```python
@app.post("/login")
def login(request:LoginRequest, db: sqlite3.Connection = Depends(get_db)):
    user = get_user_by_username(request.username, db, hide_password =False)

    if not user:
        raise_exception(HTTPStatus.NOT_FOUND, "User not Found")

    if not verify_password(request.password, user["password"]):
        raise_exception(HTTPStatus.UNAUTHORIZED, "Username or Password is not Valid")

    return generate_claims(user["username"], user["user_id"], user["role"])
```

Controller logic for /login endpoint




```python
ALGORITHM = "pbkdf2_sha256"

def hash_password(password: str, 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)

def generate_claims(username: str, user_id: int, roles: list):
    expiration_datetime = datetime.datetime.now(tz=datetime.timezone.utc) + datetime.timedelta(minutes=20)
    claims = {
        "aud" : "krakend.local.gd",
        "iss": "auth.local.gd",
        "sub" : username,
        "jti" : str(user_id),
        "roles" : roles,
        "exp" : int(expiration_datetime.timestamp())
    }
    token = {
        "access_token" : claims,
        "refresh_token" : claims,
        "exp" : int(expiration_datetime.timestamp())
    }
    return token
```

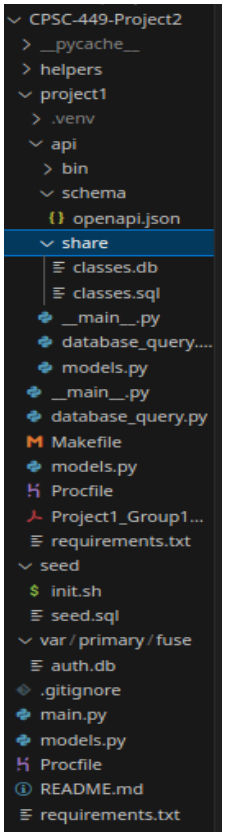
```python
helpers > db_query.py > get_user_by_username
read_dbs = ["var/secondary_2/fuse/auth.db","var/secondary_1/fuse/auth.db"]
index = itertools.cycle(range(0, len(read_dbs)))

def get_db():
    with contextlib.closing(sqlite3.connect("var/primary/fuse/auth.db")) as db:
        db.row_factory = sqlite3.Row
        yield db

def get_db_reads():
    target_db = read_dbs[next(index)]
    with contextlib.closing(sqlite3.connect(target_db)) as db:
        db.row_factory = sqlite3.Row
        yield db


def get_user_by_username(username: str, db: sqlite3.Connection, hide_password: bool = True):
    sql = """
    SELECT
    user.id as user_id,
    user.username as username,
    user.first_name as first_name,
    user.last_name as last_name,
    user.password as password,
    role.name as role
    FROM user
    LEFT JOIN user_role
    ON user_role.user_id = user.id
    LEFT JOIN role
    ON user_role.role_id = role.id
    WHERE user.username = ?
    """
    get_user = db.execute(sql, [username]).fetchall()

    if get_user:
        user = {
            "user_id" : get_user[0]["user_id"],
            "username": get_user[0]["username"],
            "first_name":get_user[0]["first_name"],
            "last_name":get_user[0]["last_name"],
            "role":[get_user[0]["role"]]
        }
        if not hide_password:
            user["password"] = get_user[0]["password"]

        for x in range(1, len(get_user)):
            user["role"].append(get_user[x]["role"])

        return user
    return None
```

```python
helpers > db_query.py > get_user_by_username
def gracefully_handle_db_transaction(sql:str, db: sqlite3.Connection):
    try:
        db.executescript(sql)
        db.commit()
        return True
    except sqlite3.IntegrityError as e:
        db.rollback()
        raise_exception(status_code=HTTPStatus.CONFLICT, message =str(e))

def create_user_sql_script(user: NewAccountRequest):
    sql_script = f'''
    INSERT INTO "user" (username, first_name, last_name, password)
    VALUES ("{user.username}", "{user.first_name}","{user.last_name}","{hash_password(user.passw
    '''
    for role in user.role:
        sql_script += f'''
        INSERT INTO "user_role" (user_id, role_id)
        VALUES ((SELECT id FROM "user" WHERE username = "{user.username}"), {ROLE[role]});
        '''

    return sql_script
```

Helper files for user query, password verification, hashing, and generating claims (claims generation logic excerpted from mkclaims.py, hashing logic from pbkdf2 resource link)

Avoiding coupling between services

The users service should have its own, separate SQLite database, with its own schema, and should run independently of the enrollment service.

- 


```sql
CREATE TABLE "role" (
    id INTEGER PRIMARY KEY,
    name VARCHAR NOT NULL
);


CREATE TABLE "user" (
    id INTEGER PRIMARY KEY,
    username VARCHAR NOT NULL UNIQUE,
    first_name VARCHAR NOT NULL,
    last_name VARCHAR NOT NULL,
    password VARCHAR NOT NULL
);

CREATE TABLE "user_role" (
    id INTEGER PRIMARY KEY,
    user_id int NOT NULL,
    role_id int NOT NULL,
    FOREIGN KEY (role_id) REFERENCES role(id),
    FOREIGN kEY (user_id) REFERENCES user(id)
);

-- role
INSERT INTO role(name) VALUES ("Registrar");
INSERT INTO role(name) VALUES ("Instructor");
INSERT INTO role(name) VALUES ("Student");
```

```sql
-- Create the Users table
DROP TABLE IF EXISTS Users;
CREATE TABLE IF NOT EXISTS Users (
    CWID INTEGER PRIMARY KEY AUTOINCREMENT,
    Name TEXT NOT NULL,
    Middle TEXT NULL,
    LastName TEXT NOT NULL,
    Role TEXT NOT NULL CHECK (role IN ('Instructor', 'Registrar', 'Student'))
);

-- Create the Class table
DROP TABLE IF EXISTS Class;
CREATE TABLE IF NOT EXISTS Class (
    CourseCode TEXT PRIMARY KEY,
    Name TEXT NOT NULL,
    Department TEXT NOT NULL
);

-- Create the Section table
DROP TABLE IF EXISTS Section;
CREATE TABLE IF NOT EXISTS Section (
    SectionNumber INTEGER NOT NULL,
    CourseCode TEXT NOT NULL,
    InstructorID INTEGER NOT NULL,
    CurrentEnrollment INTEGER NOT NULL,
    MaxEnrollment INTEGER NOT NULL,
    Waitlist INTEGER NOT NULL,
    SectionStatus TEXT NOT NULL CHECK (SectionStatus IN ('open', 'closed')),
    PRIMARY KEY (SectionNumber, CourseCode),
    FOREIGN KEY (CourseCode) REFERENCES Class (CourseCode),
    FOREIGN KEY (InstructorID) REFERENCES Users (CWID)
);


-- Create the RegistrationList table
DROP TABLE IF EXISTS RegistrationList;
CREATE TABLE IF NOT EXISTS RegistrationList (
    RecordID INTEGER PRIMARY KEY AUTOINCREMENT,
    StudentID INTEGER NOT NULL,
    CourseCode TEXT NOT NULL,
    SectionNumber INTEGER NOT NULL,
    EnrollmentDate DATETIME DEFAULT (CURRENT_TIMESTAMP),
    Status TEXT NOT NULL CHECK (Status IN ('enrolled', 'waitlisted', 'dropped')),
    FOREIGN KEY (StudentID) REFERENCES Users (CWID),
    FOREIGN KEY (CourseCode, SectionNumber) REFERENCES Section (CourseCode, SectionNumber)
);
```

Images showing project folder structure (project 1 moved to subfolder, class.db and auth.db are separate from one another), schema for auth.db, and schema for classes.db

respectively. Seeding scripts for both class.db and auth.db have also been tweaked to be in sync with each other.

Configure authentication through the API gateway

Configure JWT token signing

- JSON Web Key pair (JWK) was generated using the provided mkjwks.py, by passing the string “GROUP_9_SECRET_KEY”. The following public and private keys were generated

- KrakenD JWT token signing (signed with private key)

Configure JWT validation

- KrakenD validation using public key

```
{
    "endpoint" : "/api/enrollment",
    "method" : "POST",
    "input_headers": ["x_user", "x_roles"],
    "backend" : [
        {
            "url_pattern": "/enrollment",
            "method" :"POST",
            "host" : [
                "http://127.0.0.1:5100"
            ],
            "extra_config": {
                "backend/http": {
                    "return_error_details": "enrollment_service"
                }
            }
        }
    ],
    "extra_config": {
        "auth/validator": {
            "alg": "RS256",
            "jwk_local_path": "./etc/jwk_public_key.json",
            "roles_key": "roles",
            "roles_key_is_nested": false,
            "roles" : ["Student"],
            "disable_jwk_security": true,
            "operation_debug": true,
            "propagate_claims": [
                ["sub", "x_user"],
                ["roles", "x_roles"]
            ]
        }
    }
}
```

Configuration for auth/validator on /api/enrollment endpoint. The intent we had was to propagate the generated claims to the enrollment service such that the “sub” key (username) can be queried against the classes.db. However, due to how project1 was originally developed, it was very difficult to make it work as intended due to the amount of project1 files, and introducing changes broke the code in multiple places that was difficult to debug. As an alternative, we chose to add non breaking codes to verify that the claims were propagated correctly and accessible in the enrollment service.

```
async def course_enrollment(enrollment_request: EnrollmentRequest, x_user: Annotated[str | None, Header(convert_underscores=False)] = None, x_roles: Annotated[list[str] | None,
    """Allow enrollment of a course under given section for a student

    Args:
        enrollment_request (EnrollmentRequest): EnrollmentRequest model

    Raises:
        HTTPException: Raise HTTP exception when role is not authrorized
        HTTPException: Raise HTTP exception when query fail to execute in database

    Returns:
        EnrollmentRespo (parameter) x_user: str | None
    """
    print("x_user =>", x_user)
    print("x_roles =>", x_roles)
```

```
03:22:17 api.1      | x_user => jordan
03:22:17 api.1      | x_roles => ['Student']
```

POST http://localhost:5400/api/register

Params Authorization Headers (8) Body Pre-request Script Tests

none form-data x-www-form-urlencoded raw binary Gra

```
{
    "first_name" : "jordan",
    "last_name" : "ng",
    "username" : "jordan",
    "password" : "password",
    "role": ["Student"]
}
```

Info + credentials for created user in user.db

Testing

```
03:47:11 krakend.1 | [GIN] 2023/10/28 - 03:47:11 | 401 |        48.3µs |             ::1 | POST     "/api/enrollment"
03:47:19 api.1     | 2023-10-28 03:47:19.412 | INFO     | project1.api.database_query:check_user_role:72 - Checking user role
03:47:19 api.1     | x_user => jordan
03:47:19 api.1     | x_roles => ['Student']
03:47:19 api.1     | INFO:     ::1:0 - "POST /enrollment HTTP/1.1" 401 Unauthorized
03:47:19 krakend.1 | [GIN] 2023/10/28 - 03:47:19 | 200 |    3.112399ms |             ::1 | POST     "/api/enrollment"
```

Request to api/enrollment endpoint with no authorization header, and with authorization header (bearer $token)

Configure load balancing for the enrollment service

Created a /bin/run_formation.sh shell script to run formation

```
bin
$ run_formation.sh
$ run.sh
```

```
bin > $ run_formation.sh
1   foreman start --formation auth_service_primary=1,auth_service_secondary_1=1,auth_service_secondary_2=1,enrollment_service=3,krakend=1
```

Foreman Start starts a single instance of KrakenD and the user service, and 3 instances of the enrollment service.

```
Procfile
1   auth: uvicorn --port 8000 main:app --reload
2   enrollment_service_1: uvicorn --port 5100 project1.api.__main__:app --reload
3   enrollment_service_2: uvicorn --port 5101 project1.api.__main__:app --reload
4   enrollment_service_3: uvicorn --port 5102 project1.api.__main__:app --reload
5   krakend: echo ./etc/krakend.json | entr -nrz krakend run --config ./etc/krakend.json --port 5400
6   auth_service_primary: ./litefs mount --config ./etc/primary.yml
7   auth_service_secondary_1: ./litefs mount --config ./etc/secondary_1.yml
8   auth_service_secondary_2: ./litefs mount --config ./etc/secondary_2.yml
```

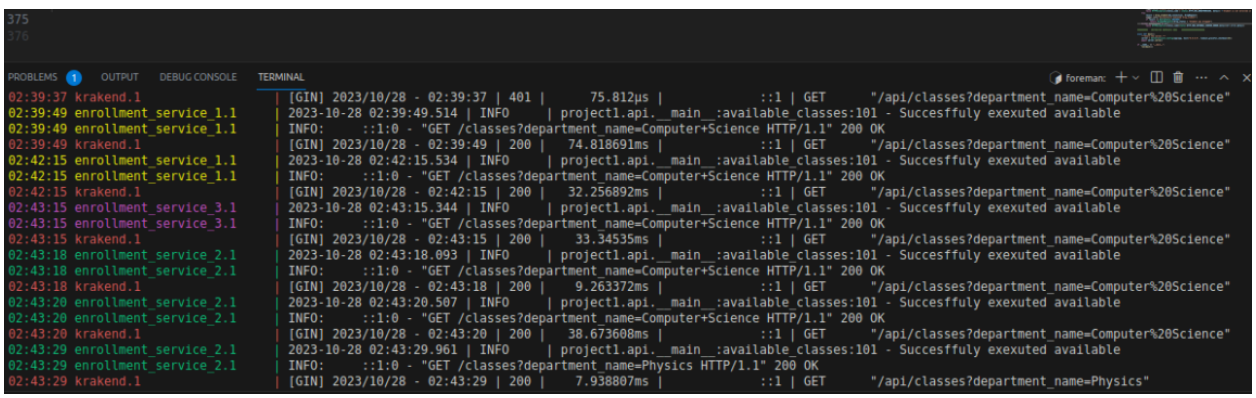
```
(.venv) codie@codie-Lenovo-300e:~/Desktop/CPSC-449-Project2$ foreman start
03:09:35 auth.1                    | started with pid 412169
03:09:35 enrollment_service_1.1    | started with pid 412170
03:09:35 enrollment_service_2.1    | started with pid 412171
03:09:35 enrollment_service_3.1    | started with pid 412172
03:09:35 krakend.1                 | started with pid 412173
03:09:35 auth_service_primary.1    | started with pid 412174
03:09:35 auth_service_secondary_1.1 | started with pid 412177
03:09:35 auth_service_secondary_2.1 | started with pid 412178
```

KrakenD load balancing was configured to round-robin between the 3 instances of enrollment services.

Sample endpoint using round robin to distribute load between enrollment services instances

Testing

Several requests were made and they were routed to different service instances



Add read replication to the users service

Configure database replicas

Read replication added to the users service, 2 replicas added: auth_service_secondary_1.1, and auth_service_secondary_2.1

```
(.venv) codie@codie-Lenovo-300e:~/Desktop/CPSC-449-Project2$ foreman start
00:48:56 auth.1                     | started with pid 404556
00:48:56 api.1                      | started with pid 404557
00:48:56 krakend.1                  | started with pid 404558
00:48:56 auth_service_primary.1     | started with pid 404559
00:48:56 auth_service_secondary_1.1 | started with pid 404562
00:48:56 auth_service_secondary_2.1 | started with pid 404567
00:48:56 auth_service_primary.1     | LiteFS v0.5.7, commit=c477633c9b17981ccbfa0b86d28b9cd6b7202888
00:48:56 auth_service_secondary_1.1 | LiteFS v0.5.7, commit=c477633c9b17981ccbfa0b86d28b9cd6b7202888
00:48:56 auth_service_secondary_2.1 | LiteFS v0.5.7, commit=c477633c9b17981ccbfa0b86d28b9cd6b7202888
```

Each process is configured separately in the Procfile instead of running –formation

```
Procfile
auth: uvicorn --port 8000 main:app --reload
api: uvicorn --port 5100 project1.api.__main__:app --reload
krakend: echo ./etc/krakend.json | entr -nrz krakend run -c ./etc/krakend.json --port 5400
auth_service_primary: ./litefs mount --config ./etc/primary.yml
auth_service_secondary_1: ./litefs mount --config ./etc/secondary_1.yml
auth_service_secondary_2: ./litefs mount --config ./etc/secondary_2.yml
```

Update the users service

Users service was modified, the current code allows write operations to the primary node, while read operations are cycled between the two secondary nodes

```python
read_dbs = ["var/secondary_2/fuse/auth.db","var/secondary_1/fuse/auth.db"]
index = itertools.cycle(range(0, len(read_dbs)))

def get_db():
    with contextlib.closing(sqlite3.connect("var/primary/fuse/auth.db")) as db:
        db.row_factory = sqlite3.Row
        yield db

def get_db_reads():
    target_db = read_dbs[next(index)]
    with contextlib.closing(sqlite3.connect(target_db)) as db:
        db.row_factory = sqlite3.Row
        yield db
```

Get_db dependency injection functions for read and write operations

```python
@app.post("/register")
def register(request: NewAccountRequest, db:sqlite3.Connection = Depends(get_db)):

    user = get_user_by_username(request.username, db)

    if user:
        raise_exception(HTTPStatus.CONFLICT, f'User with username {user["username"]} already exists')

    if gracefully_handle_db_transaction(create_user_sql_script(request), db):
        user = get_user_by_username(request.username, db)

        return create_response(HTTPStatus.CREATED, f'{user["username"]} created!', user)

@app.post("/login")
def login(request:LoginRequest, db: sqlite3.Connection = Depends(get_db_reads)):
    user = get_user_by_username(request.username, db, hide_password =False)

    if not user:
        raise_exception(HTTPStatus.NOT_FOUND, "User not Found")

    if not verify_password(request.password, user["password"]):
        raise_exception(HTTPStatus.UNAUTHORIZED, "Username or Password is not Valid")

    return generate_claims(user["username"], user["user_id"], user["role"])
```

main.py

```yaml
etc > ! secondary_1.yml
fuse:
  dir: "var/secondary_1/fuse"
  allow-other: false
  debug: false

data:
  dir: "var/secondary_1/data"
  compress: true

http:
  addr: ":20203"

lease:
  type: "static"
  hostname: "secondary_1"
  advertise-url: "http://127.0.0.1:20202"
  candidate: false
```

Secondary_1.yml (replica 1)

```yaml
fuse:
  dir: "var/secondary_2/fuse"
  allow-other: false
  debug: false

data:
  dir: "var/secondary_2/data"
  compress: true

http:
  addr: ":20204"

lease:
  type: "static"
  hostname: "secondary_2"
  advertise-url: "http://127.0.0.1:20202"
  candidate: false
```

Secondary_2.yml (replica 2)

```yaml
fuse:
  dir: "var/primary/fuse"
  allow-other: false
  debug: false

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

http:
  addr: ":20202"

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

exec: "uvicorn main:app --reload --port 8000 "
```

primary.yml