Three-tier Web-Database Application Project

SimpApp

Introduction

SimpApp

Scenario

A group of professors is leading a new study group. Students in the study group are reading books as their homework. The professors assign books to the students.

The application helps the professors manage the assignments. It stores the necessary credential for the professors to access the application, the books, the students, and the current assignments.

Professors can login and logout to the application, view their credentials, view all the assignments and insert, delete and modify assignments.

The current implementation has no provision for inserting, deleting or modifying professors, students, and books. Professors cannot edit their credentials. There is no interactive interface for searching and querying the data. Students are not users of the system and need to email a professor when an assignment is completed. Feel free to try and add these features and other ones for your own training.

Database Schema

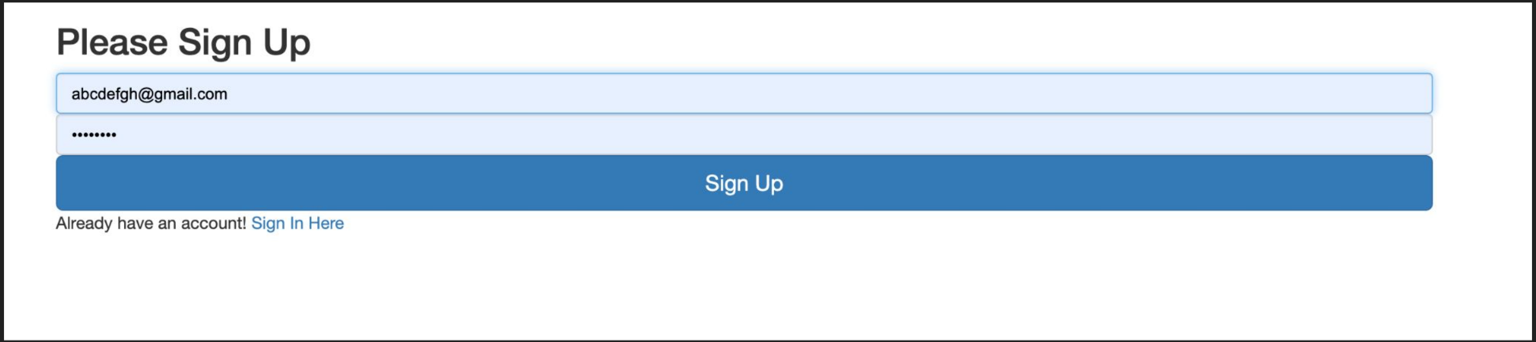
The database contains four tables: professor, student, book, and assignment.

```
CREATE TABLE IF NOT EXISTS student (email TEXT PRIMARY KEY);

CREATE TABLE IF NOT EXISTS professor (email TEXT PRIMARY KEY, password TEXT NOT NULL);

CREATE TABLE IF NOT EXISTS book (isbn TEXT PRIMARY KEY, title TEXT NOT NULL, author TEXT NOT NULL);

CREATE TABLE IF NOT EXISTS assignment (email TEXT REFERENCES student(email), isbn TEXT
REFERENCES book(isbn), PRIMARY KEY (isbn, email));
```

Demonstration

SimpApp

Sign Up/Log In

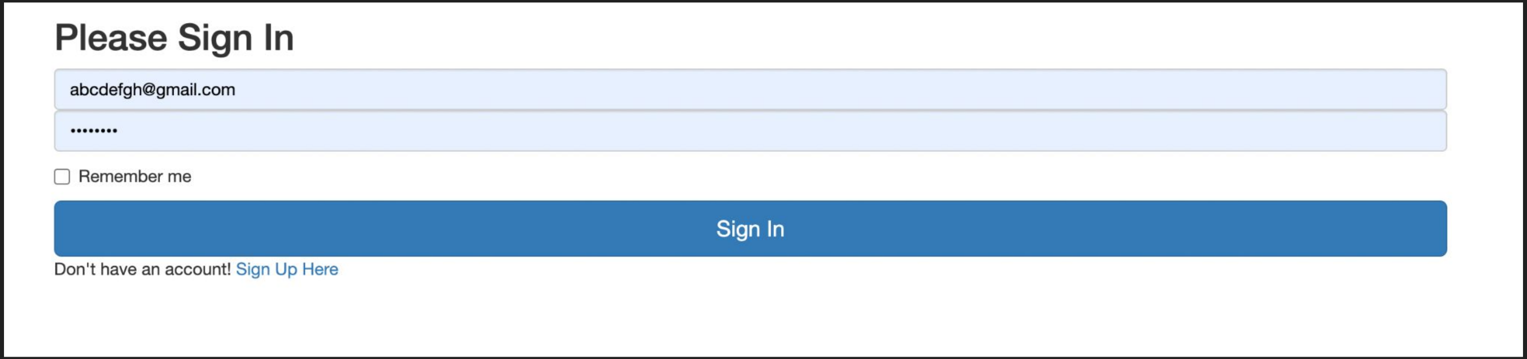


Please Sign Up

abcdefgh@gmail.com

••••••••

Already have an account! Sign In Here



Please Sign In

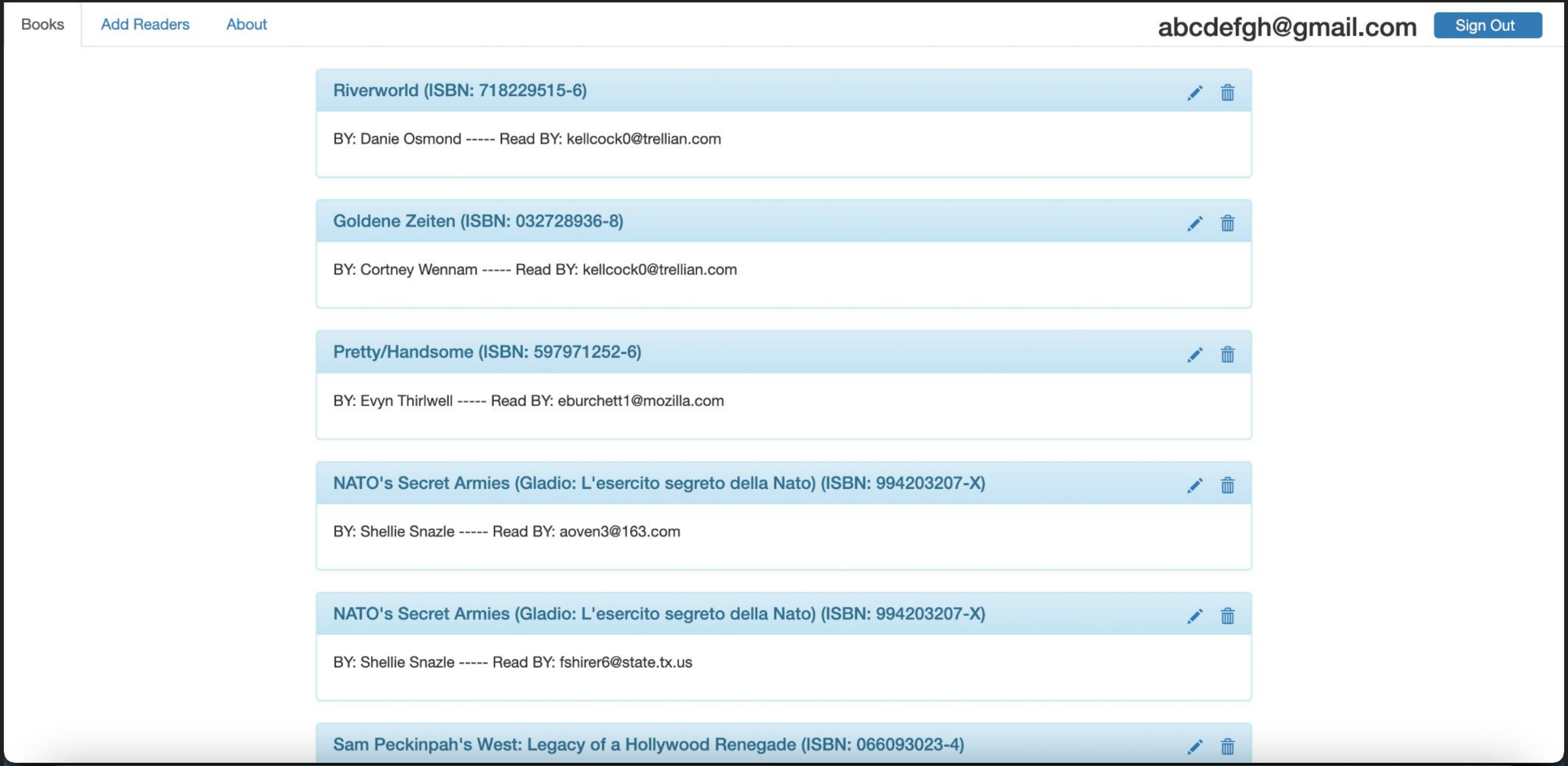
abcdefgh@gmail.com

••••••••

☐ Remember me

Don't have an account! Sign Up Here

View readings



Riverworld (ISBN: 718229515-6)

BY: Danie Osmond ----- Read BY: kellcock0@trellian.com

Goldene Zeiten (ISBN: 032728936-8)

BY: Cortney Wennam ----- Read BY: kellcock0@trellian.com

Pretty/Handsome (ISBN: 597971252-6)

BY: Evyn Thirlwell ----- Read BY: eburchett1@mozilla.com

NATO's Secret Armies (Gladio: L'esercito segreto della Nato) (ISBN: 994203207-X)

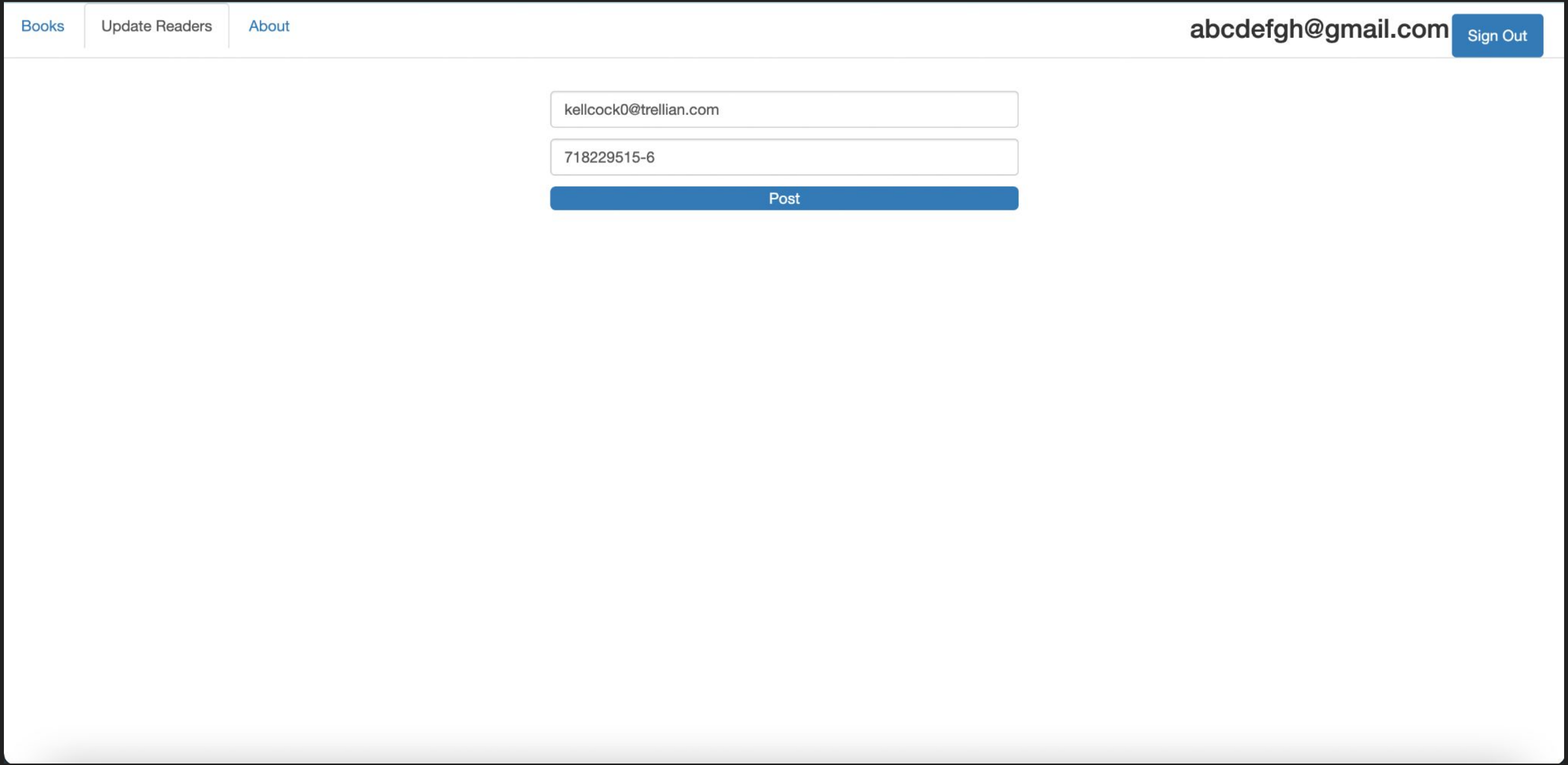
BY: Shellie Snazle ----- Read BY: aoven3@163.com

NATO's Secret Armies (Gladio: L'esercito segreto della Nato) (ISBN: 994203207-X)

BY: Shellie Snazle ----- Read BY: fshirer6@state.tx.us

Sam Peckinpah's West: Legacy of a Hollywood Renegade (ISBN: 066093023-4)

Edit Readings

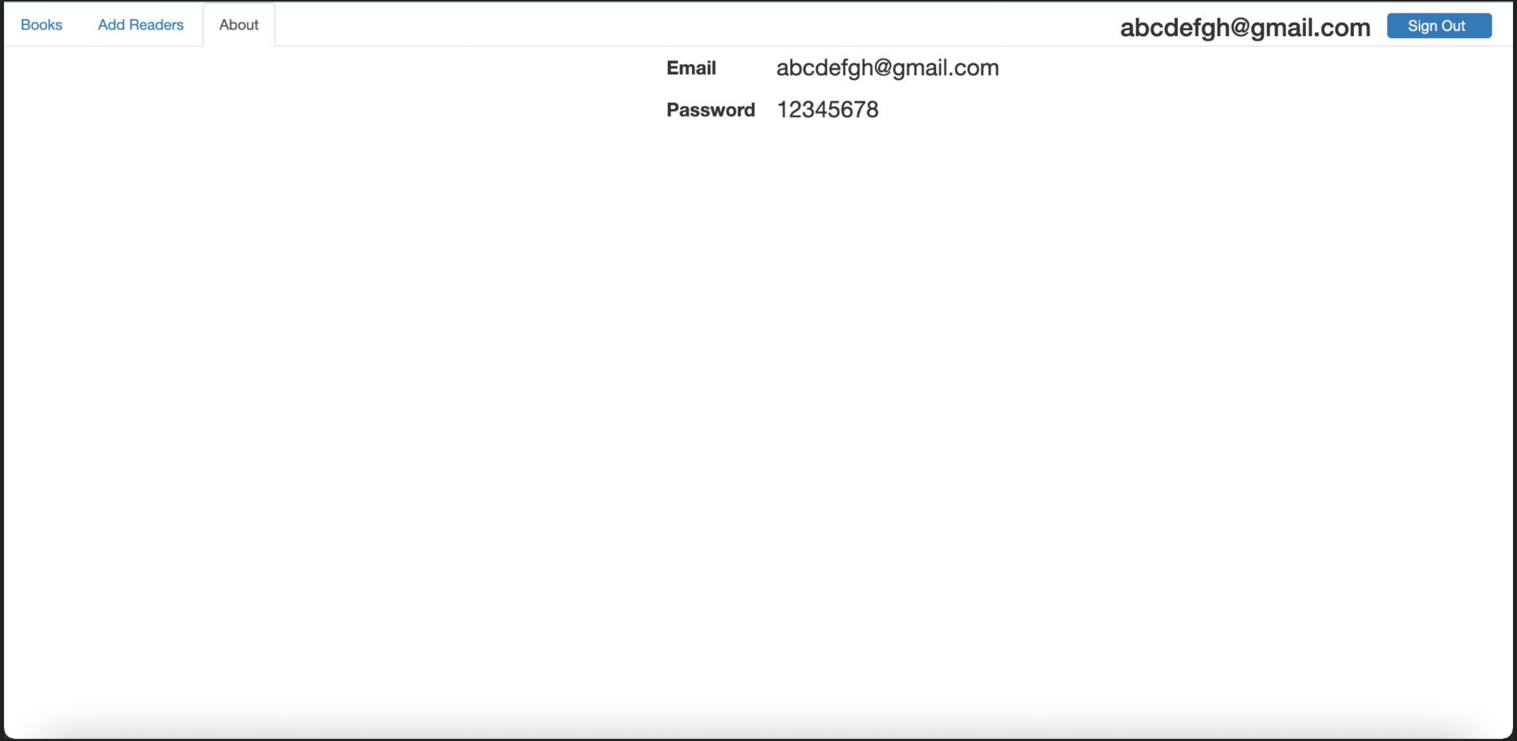


abcdefgh@gmail.com

kellcock0@trellian.com

718229515-6

View Personal Info



Email	abcdefgh@gmail.com
Password	12345678

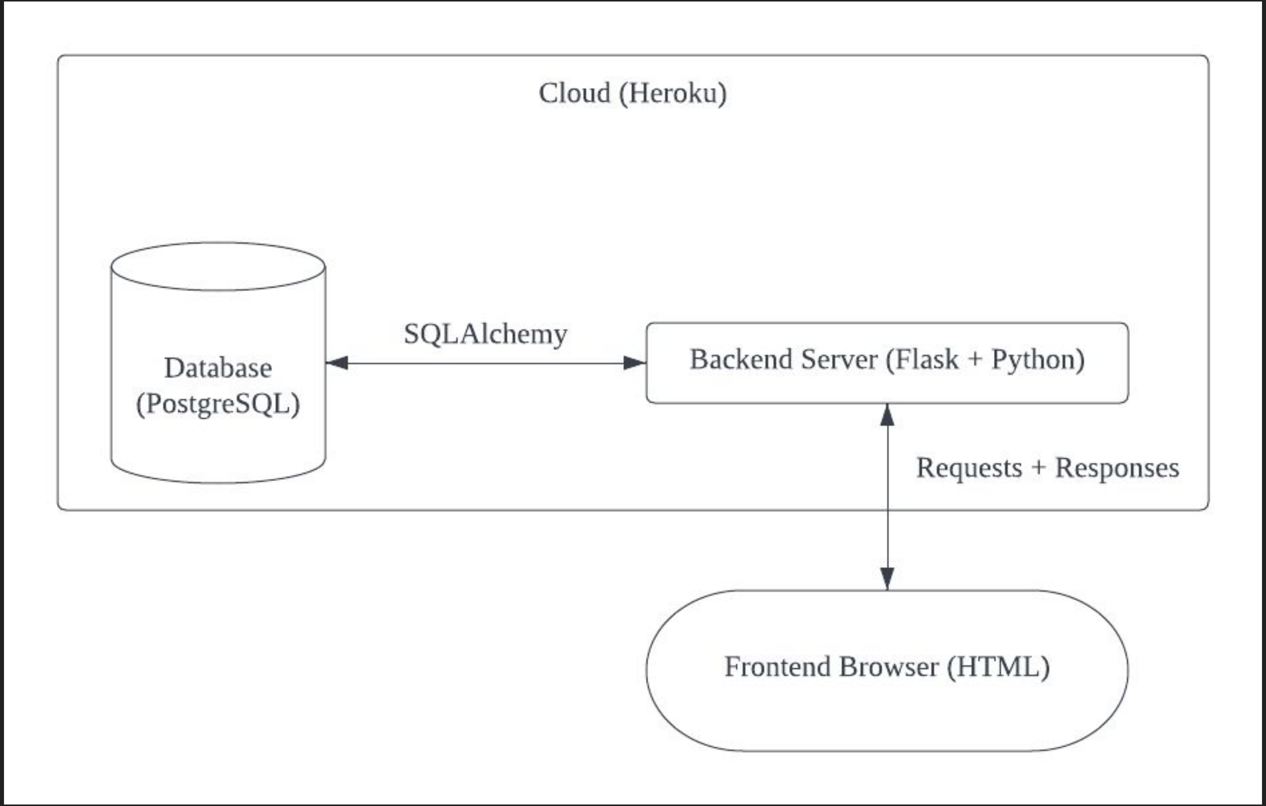
Implementation

SimpApp

Architecture

The application is a three-tier Web-database application. The client is any Web browser or application that understands HTML and HTTP. The applications server is implemented (in Python) in the Flask micro web framework and deployed to the Dynos Web server on the Heroku cloud platform as a service. The database server is PostgreSQL 14.5 deployed on Heroku. Flask interacts with PostgreSQL using SQLAlchemy, an SQL toolkit and object-relational mapper for Python.

Architecture

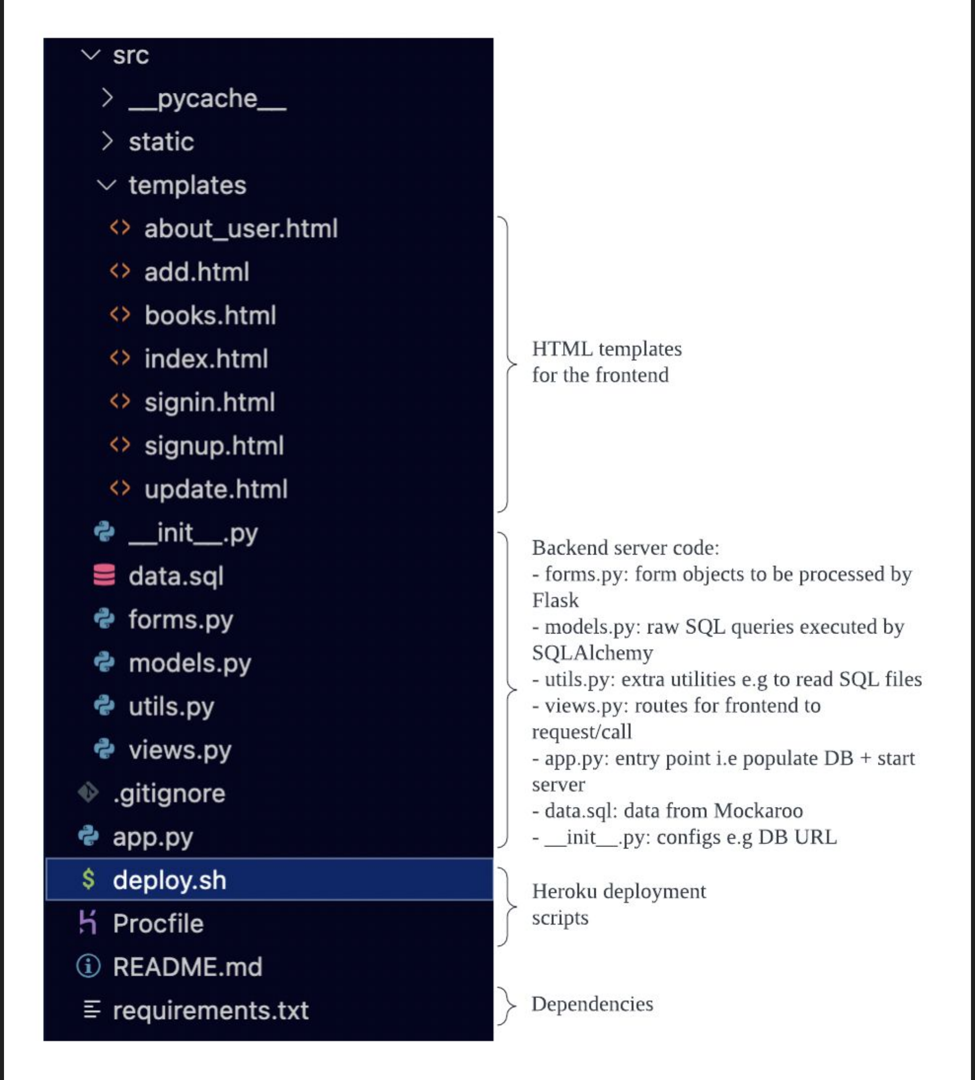


Source Code

The sample code is available in the GitHub repository:

https://github.com/daongochieu2810/flask-sqlalchemy-sandbox

Project Structure



```
src
  __pycache__
  static
  templates
    about_user.html
    add.html
    books.html
    index.html
    signin.html
    signup.html
    update.html
  __init__.py
  data.sql
  forms.py
  models.py
  utils.py
  views.py
.gitignore
app.py
deploy.sh
Procfile
README.md
requirements.txt
```

HTML templates for the frontend

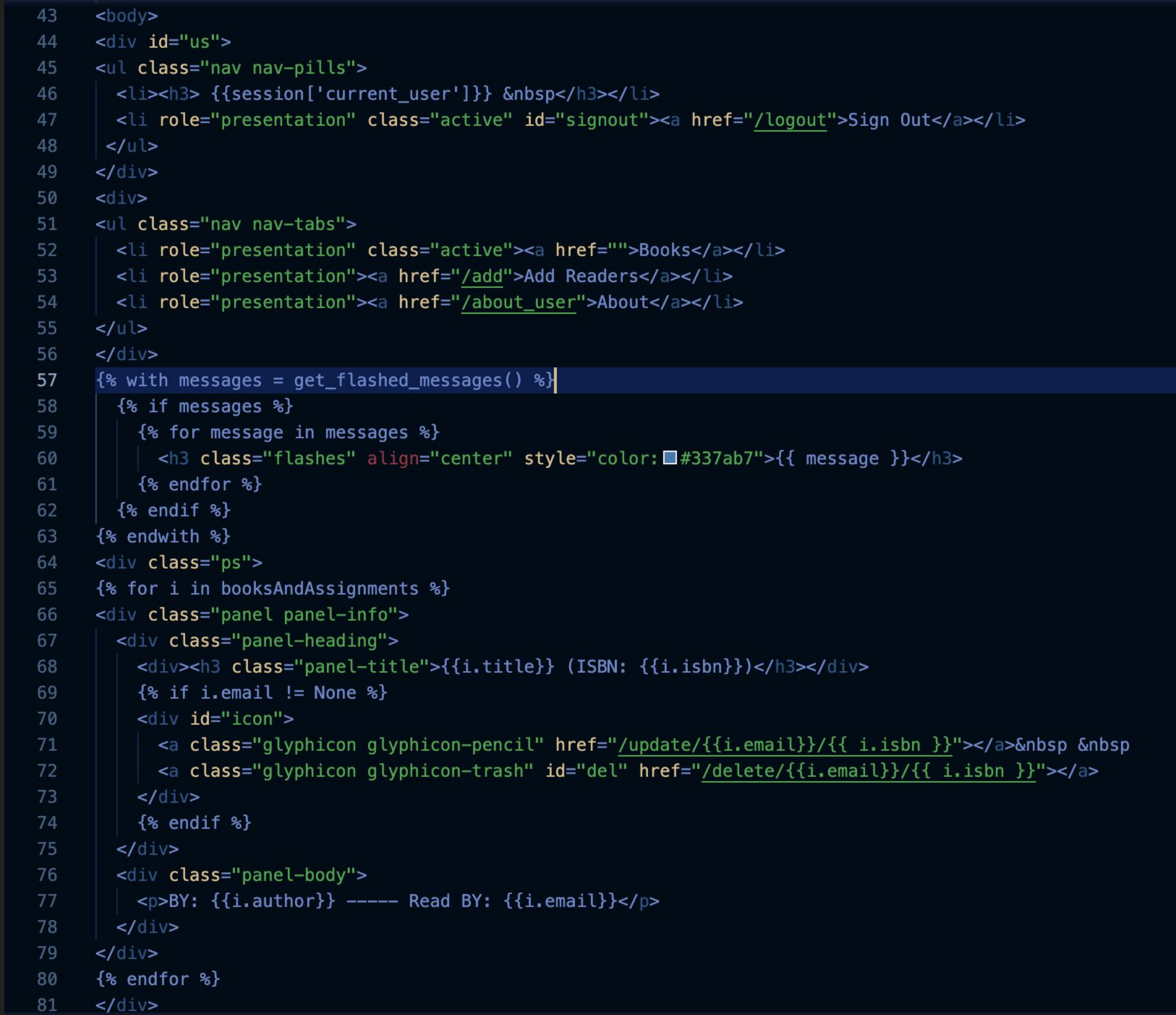
Backend server code:
- forms.py: form objects to be processed by Flask
- models.py: raw SQL queries executed by SQLAlchemy
- utils.py: extra utilities e.g to read SQL files
- views.py: routes for frontend to request/call
- app.py: entry point i.e populate DB + start server
- data.sql: data from Mockaroo
- __init__.py: configs e.g DB URL

Heroku deployment scripts

Dependencies

HTML Templates

This is not the final HTML page that will be sent to the browser. The backend Flask server processes the parts in the brackets in the templates into HTML then sends the resulting HTML to the client

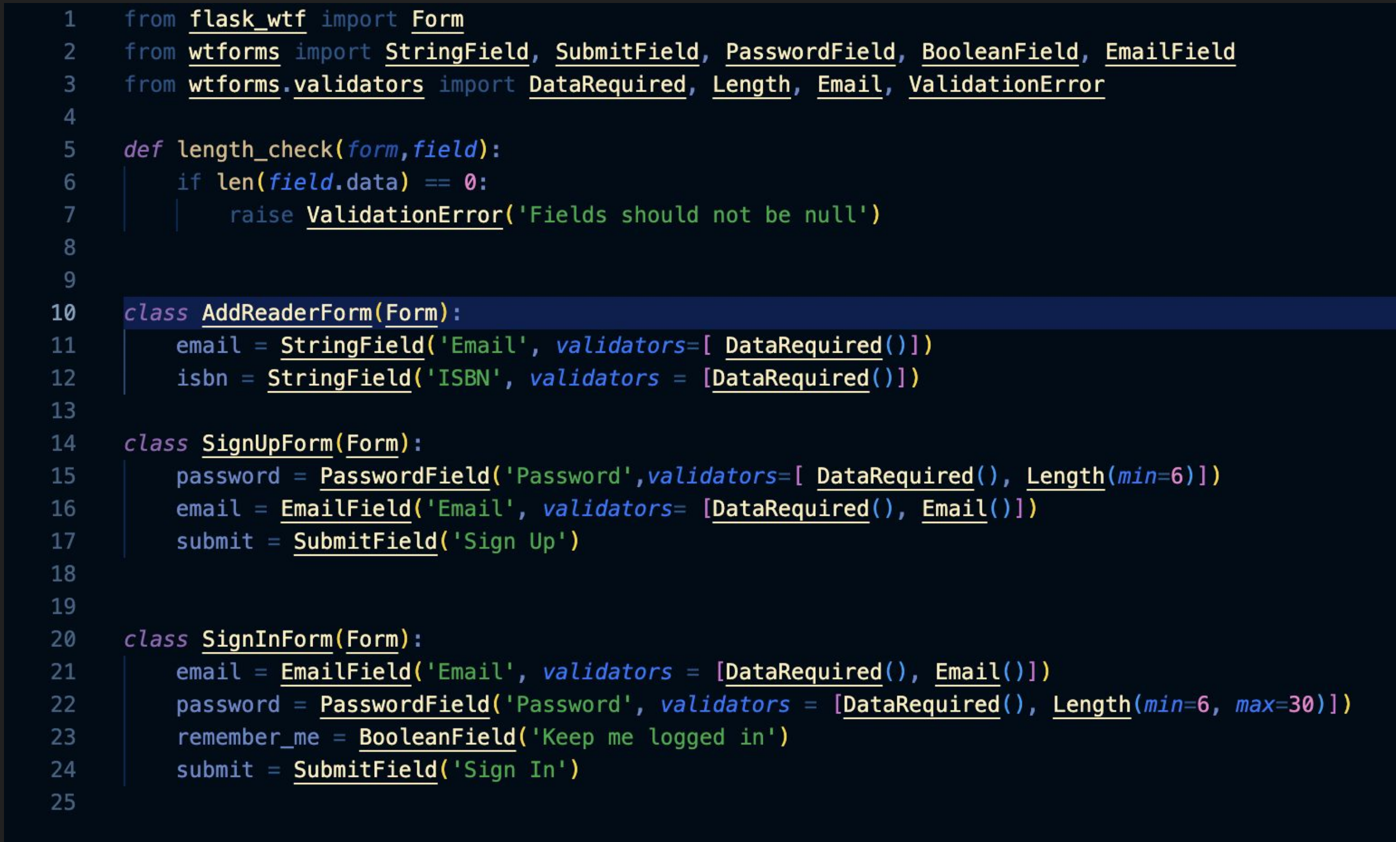


```
<body>
<div id="us">
<ul class="nav nav-pills">
  <li><h3> {{session['current_user']}} &nbsp</h3></li>
  <li role="presentation" class="active" id="signout"><a href="/logout">Sign Out</a></li>
 </ul>
</div>
<div>
<ul class="nav nav-tabs">
  <li role="presentation" class="active"><a href="">Books</a></li>
  <li role="presentation"><a href="/add">Add Readers</a></li>
  <li role="presentation"><a href="/about_user">About</a></li>
</ul>
</div>
{% with messages = get_flashed_messages() %}
  {% if messages %}
    {% for message in messages %}
      <h3 class="flashes" align="center" style="color:#337ab7">{{ message }}</h3>
    {% endfor %}
  {% endif %}
{% endwith %}
<div class="ps">
{% for i in booksAndAssignments %}
<div class="panel panel-info">
  <div class="panel-heading">
    <div><h3 class="panel-title">{{i.title}} (ISBN: {{i.isbn}})</h3></div>
    {% if i.email != None %}
    <div id="icon">
      <a class="glyphicon glyphicon-pencil" href="/update/{{i.email}}/{{ i.isbn }}"></a>&nbsp &nbsp
      <a class="glyphicon glyphicon-trash" id="del" href="/delete/{{i.email}}/{{ i.isbn }}"></a>
    </div>
    {% endif %}
  </div>
  <div class="panel-body">
    <p>BY: {{i.author}} ------ Read BY: {{i.email}}</p>
  </div>
</div>
{% endfor %}
</div>
```

Forms

These are forms to capture user input

The underlying logic is provided by Flask WTF - a library for Flask



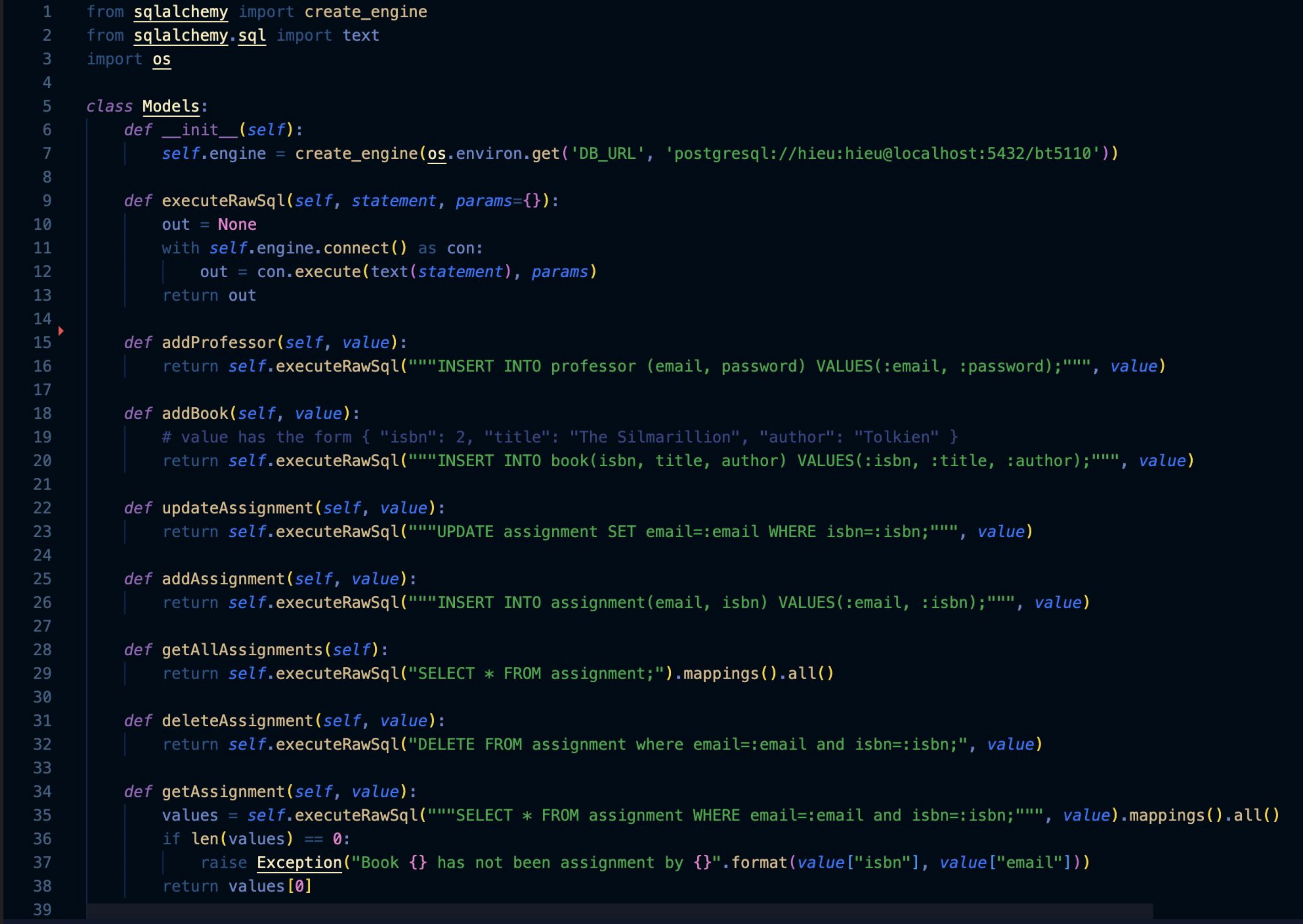
```python
from flask_wtf import Form
from wtforms import StringField, SubmitField, PasswordField, BooleanField, EmailField
from wtforms.validators import DataRequired, Length, Email, ValidationError

def length_check(form,field):
    if len(field.data) == 0:
        raise ValidationError('Fields should not be null')


class AddReaderForm(Form):
    email = StringField('Email', validators=[ DataRequired()])
    isbn = StringField('ISBN', validators = [DataRequired()])

class SignUpForm(Form):
    password = PasswordField('Password',validators=[ DataRequired(), Length(min=6)])
    email = EmailField('Email', validators= [DataRequired(), Email()])
    submit = SubmitField('Sign Up')


class SignInForm(Form):
    email = EmailField('Email', validators = [DataRequired(), Email()])
    password = PasswordField('Password', validators = [DataRequired(), Length(min=6, max=30)])
    remember_me = BooleanField('Keep me logged in')
    submit = SubmitField('Sign In')

```

Models

The Models object creates a connection to the Postgres DB (`self.engine`)

It has a function to execute raw SQL queries (`executeRawSql`)

The underlying logic is provided by SQLAlchemy



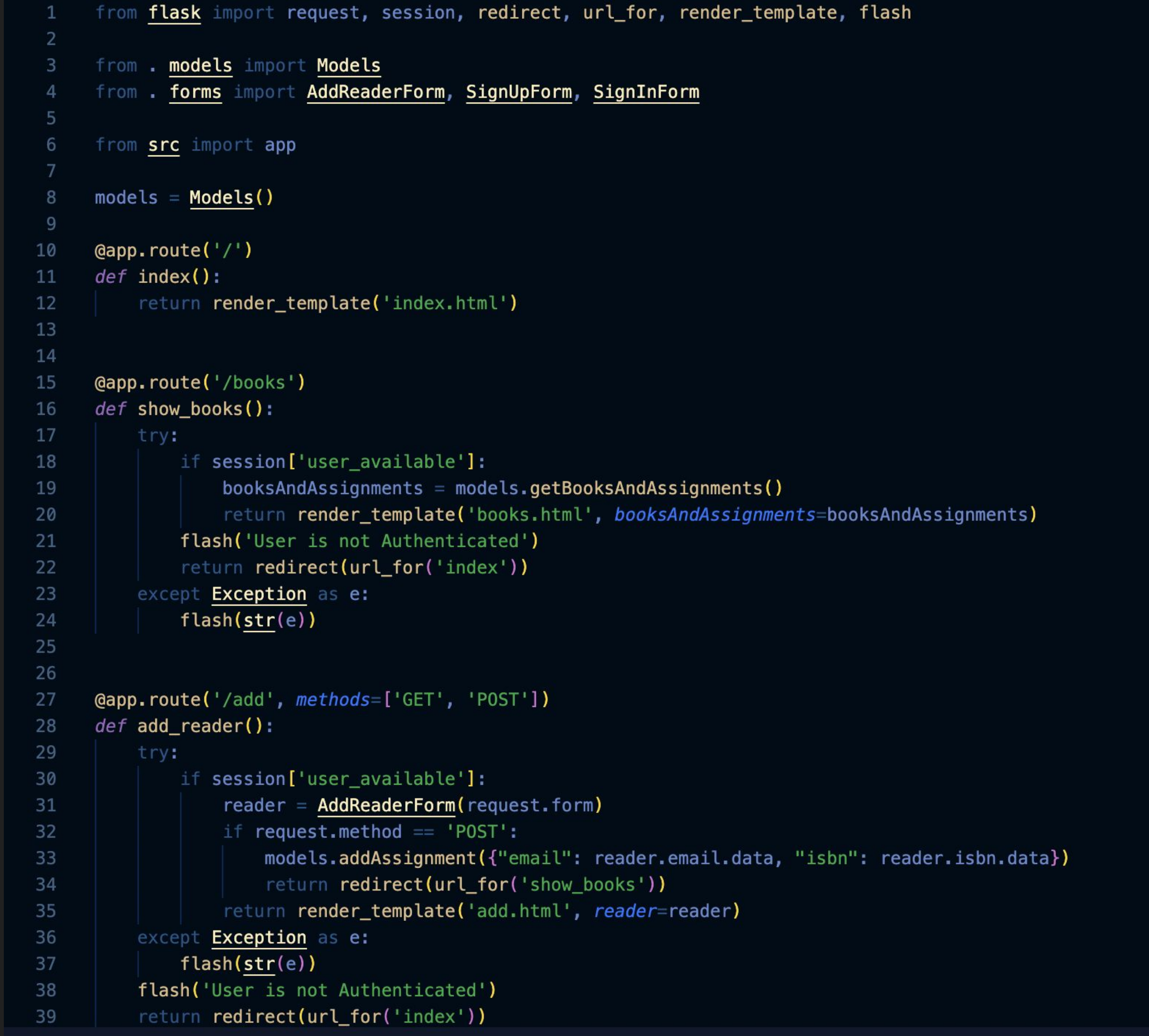
```python
from sqlalchemy import create_engine
from sqlalchemy.sql import text
import os

class Models:
    def __init__(self):
        self.engine = create_engine(os.environ.get('DB_URL', 'postgresql://hieu:hieu@localhost:5432/bt5110'))

    def executeRawSql(self, statement, params={}):
        out = None
        with self.engine.connect() as con:
            out = con.execute(text(statement), params)
        return out

    def addProfessor(self, value):
        return self.executeRawSql("""INSERT INTO professor (email, password) VALUES(:email, :password);""", value)

    def addBook(self, value):
        # value has the form { "isbn": 2, "title": "The Silmarillion", "author": "Tolkien" }
        return self.executeRawSql("""INSERT INTO book(isbn, title, author) VALUES(:isbn, :title, :author);""", value)

    def updateAssignment(self, value):
        return self.executeRawSql("""UPDATE assignment SET email=:email WHERE isbn=:isbn;""", value)

    def addAssignment(self, value):
        return self.executeRawSql("""INSERT INTO assignment(email, isbn) VALUES(:email, :isbn);""", value)

    def getAllAssignments(self):
        return self.executeRawSql("SELECT * FROM assignment;").mappings().all()

    def deleteAssignment(self, value):
        return self.executeRawSql("DELETE FROM assignment where email=:email and isbn=:isbn;", value)

    def getAssignment(self, value):
        values = self.executeRawSql("""SELECT * FROM assignment WHERE email=:email and isbn=:isbn;""", value).mappings().all()
        if len(values) == 0:
            raise Exception("Book {} has not been assignment by {}".format(value["isbn"], value["email"]))
        return values[0]

```

Views

Views generate pages that the client sees by putting together the templates, the forms, and the SQL queries in the model

Each view/page is accompanied by a route e.g /books, and the logic processing the corresponding HTML template

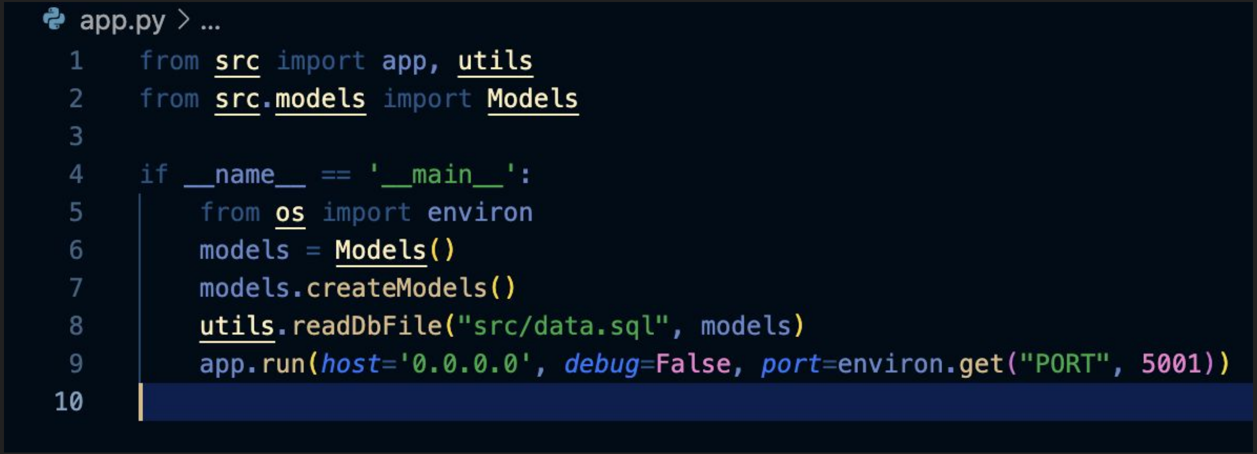
Error messages are displayed if needed



```python
from flask import request, session, redirect, url_for, render_template, flash

from . models import Models
from . forms import AddReaderForm, SignUpForm, SignInForm

from src import app

models = Models()

@app.route('/')
def index():
    return render_template('index.html')


@app.route('/books')
def show_books():
    try:
        if session['user_available']:
            booksAndAssignments = models.getBooksAndAssignments()
            return render_template('books.html', booksAndAssignments=booksAndAssignments)
        flash('User is not Authenticated')
        return redirect(url_for('index'))
    except Exception as e:
        flash(str(e))


@app.route('/add', methods=['GET', 'POST'])
def add_reader():
    try:
        if session['user_available']:
            reader = AddReaderForm(request.form)
            if request.method == 'POST':
                models.addAssignment({"email": reader.email.data, "isbn": reader.isbn.data})
                return redirect(url_for('show_books'))
            return render_template('add.html', reader=reader)
    except Exception as e:
        flash(str(e))
    flash('User is not Authenticated')
    return redirect(url_for('index'))
```

Others

app.py is the entry point of the project which starts the server, creates the Models object, populates the database with fake data from Mockaroo



app.py > ...

```python
from src import app, utils
from src.models import Models

if __name__ == '__main__':
    from os import environ
    models = Models()
    models.createModels()
    utils.readDbFile("src/data.sql", models)
    app.run(host='0.0.0.0', debug=False, port=environ.get("PORT", 5001))

```

Set-up

SimpApp

GitHub

Download he sample code from the GitHub repository:

https://github.com/daongochieu2810/flask-sqlalchemy-sandbox

Running the Project Locally

- Prerequisites: Git, Python 3, Heroku CLI
- In models.py:
    - The local database URL currently is `postgresql://hieu:hieu@localhost:5432/bt5110`
    - Replace this with your URL: `postgresql://<your db username>:<your db password>@localhost:5432/<your db name>`
- To run the project locally
    - MacOS/Linux: `chmod a+x deploy.sh; ./deploy.sh`
    - Windows:
        - python -m venv .venv (first time only)
        - .venv/Scripts/activate
        - python -m pip install --upgrade pip (first time only)
        - pip install -r requirements.txt (first time only)
        - python app.py

Heroku

Create your Heroku account: https://dashboard.heroku.com

Setup a project/application:

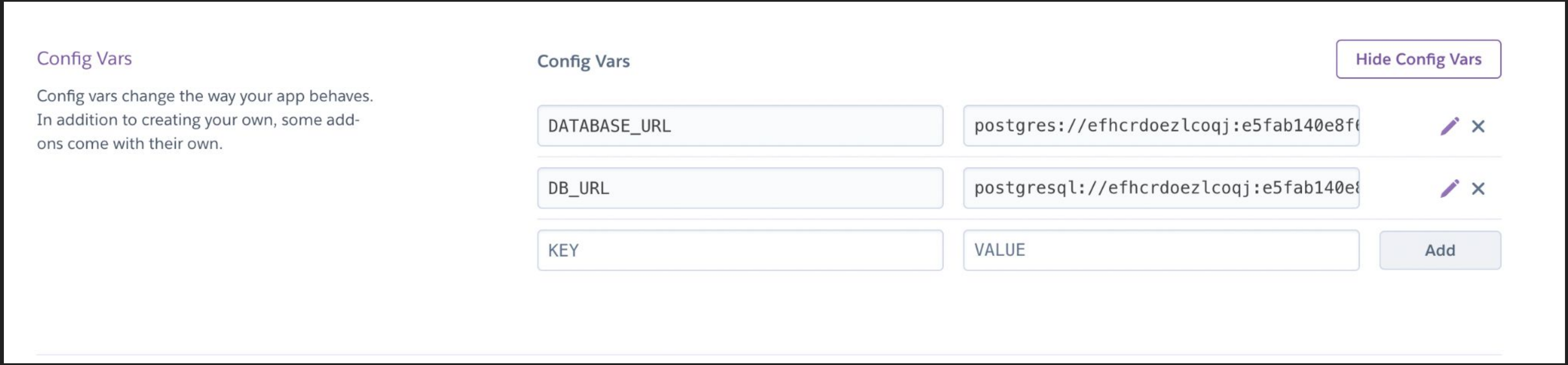
- Install Heroku CLI: https://devcenter.heroku.com/articles/heroku-cli
- Create a new project: `heroku apps:create example` (under your project directory)

Add your collaborators (project teams are up to three members)

  - `heroku access:add <collaborator@gmail.com>`
  - Collaborator can pull the repo with `heroku git:clone -a <bt5110-project-name>`

Update DB_URL (from `postgres` to `postgresql`) on Heroku

  - Go to Dashboard -> Project -> Settings -> Config Vars



Config vars change the way your app behaves. In addition to creating your own, some add-ons come with their own.

DATABASE_URL	postgres://efhcrdoezlcoqj:e5fab140e8f
DB_URL	postgresql://efhcrdoezlcoqj:e5fab140e
KEY	VALUE