University of Warsaw
Data Science & Business Analytics

Palina Tkachova, Anirban Das

Project Expat Assistance

Project proposal
in the field of Python/SQL
Semester 1

Warsaw, December 2022

Introduction

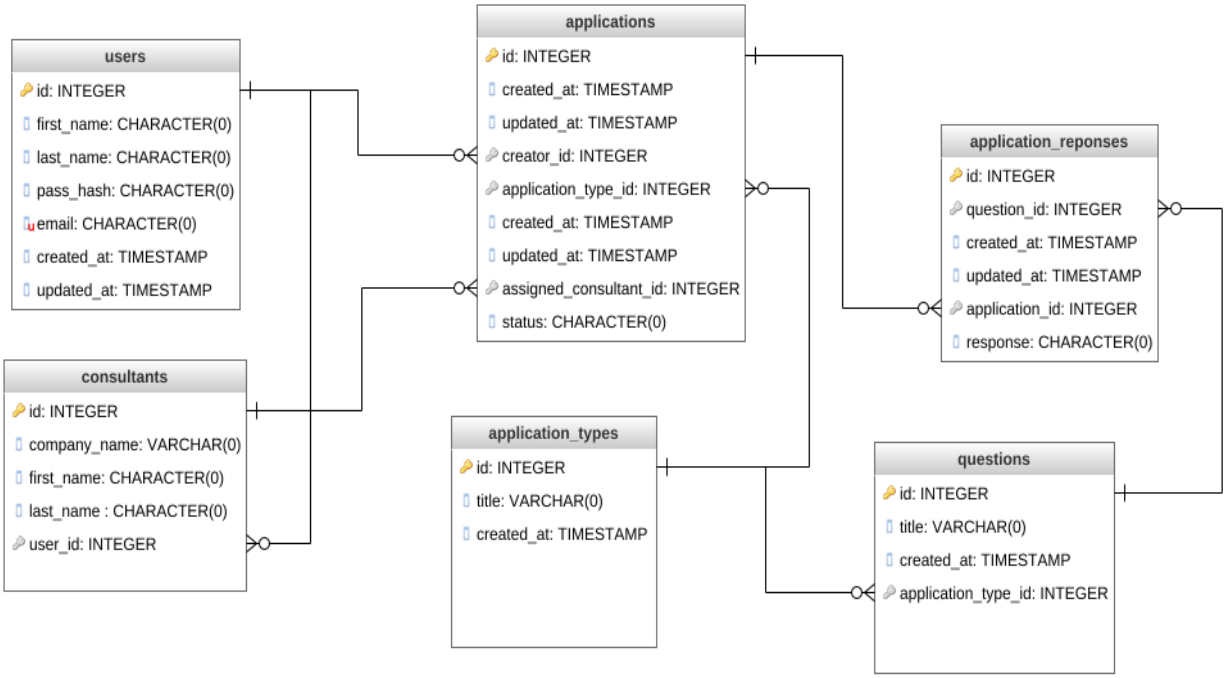
The following idea starting from the proposal is based on developing a website which will be dedicated to assist the Expats in Poland with legal advice. The main motive is to setup a dedicated platform where expats can have dedicated services based on their issues due to the ever-changing immigration policies, rules, and approach.

The backend framework was built using the Python as the coding language and the SQL as the database management system. The complete version of the programming and development will be carried out using the Jupyter Notebook as the integrated development environment (IDE) for Python and SQLlite will be used as the database engine. Additionally, Flask is chosen as the web framework for the webpage development in this project. The page optimization and development is done in html using Bootstrap, CSS and JS.

Further details on the structure of the work are discussed later in this report.

Database Scheme

The below image is the graphical representation of the database scheme.



This database scheme has been finalized and was worked on bringing out the project in reality.

The database contains 6 different tables:

- Users: This table contains all the login credentials of the users and the consultants.
- Consultants: table consists of the details of the consultants and their corresponding user ids.
- Applications: This table is updating the creator's ids, with assigned consultants along with the time of update and most importantly the status of the submission.
- Application type: This table is consisting of the various types of applications.
- Questions: Consists of a series of questions for having the user's details and the relevant observation is drawn consultants for the case.
- Application responses: The application response table is the table filled up with the responses by the users from the Questions table.

Methodology

The Flask is used as web framework for the development of the website. Starting with the libraries and packages used for the task, the code was shaped in a way that there are room for better authentication and protected routing operations.

The flask library provided packages like *Flask, render_template, redirect, url_for, request, session, g*, which are used for the availability of the many built in functions along with rendering various .html templates for the webpage frontend visualizations, redirecting to a route or an url after satisfying certain condition, requesting procedures, adding the sessions for each login for adding an extra layer of security for the protection of the url breaching by malicious routing.

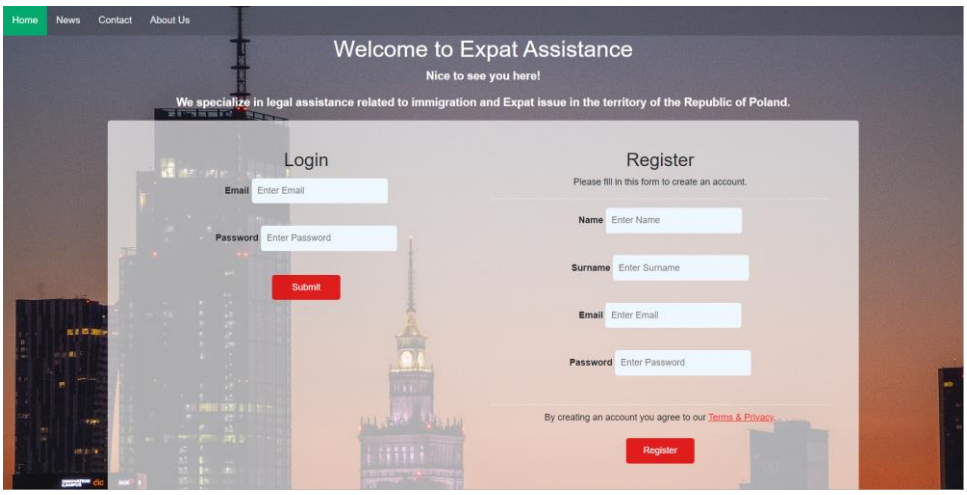
All the vital libraries include:

```
import matplotlib.pyplot as plt

import json from flask import Flask, render_template, redirect,
url_for, request, session, g

from flask_login import login_required, LoginManager, UserMixin,
login_user, current_user

from functools import wraps

import hashlib

import sqlite3

import os
```

To start with the Registration of the user, the name, surname, email and password is taken into as an input and are stored into the ***expat_project.db*** database. There are check of duplicate email addresses to ensure the security of the users, same email cannot be registered multiple times. There are methods of password hashing which is used from the *hashlib* ensuring the password is stored as a string of encrypted letters and number which makes it impossible for internal breach. Finally, the user registering is assigned with a session id (`session['user_id'] = user_id`) for the security of the platform. The registered user can stay in session performing various activities based on the type of user (normal user/consultant), when logged out, the session is cleared and further activities are obstructed until new login is done with the registered email address and password.

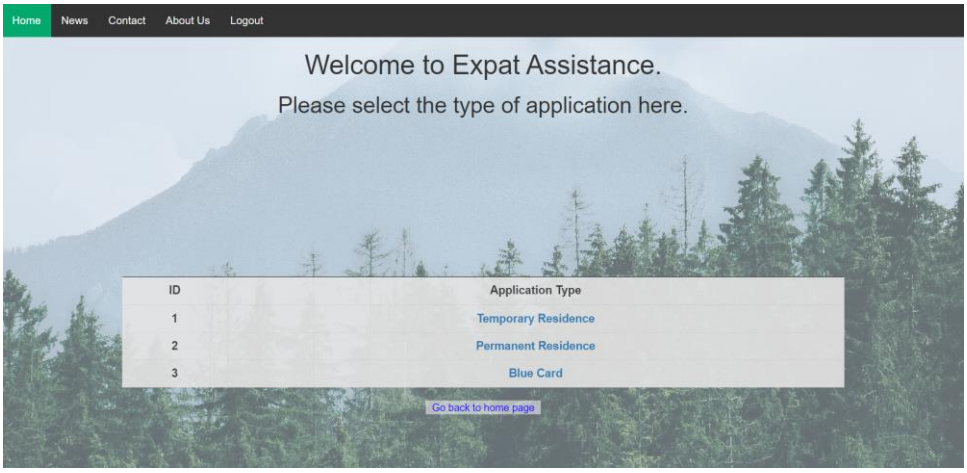
In that instance, for login of the user, the checks are established to verify id the logged in user is a normal user or a consultant.

The home page is where the user either logs in or registers:



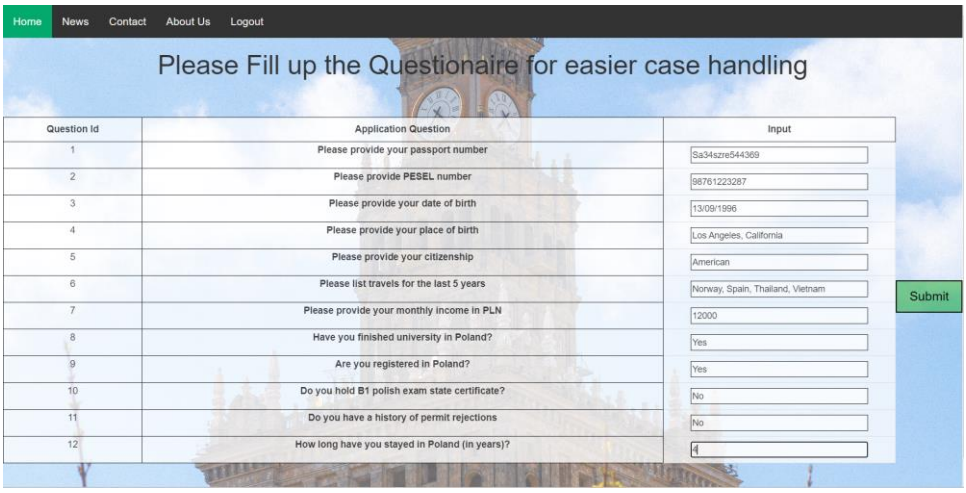
In case of the normal user is routed to the base page from where the user can either track the status of an existing case or file a new application, which leads the user for selecting new application type (Temporary Residence, Permanent Residence, Blue Card) by routing the user to

```
@app.route("/users/<int:user_id>/select_application").
```



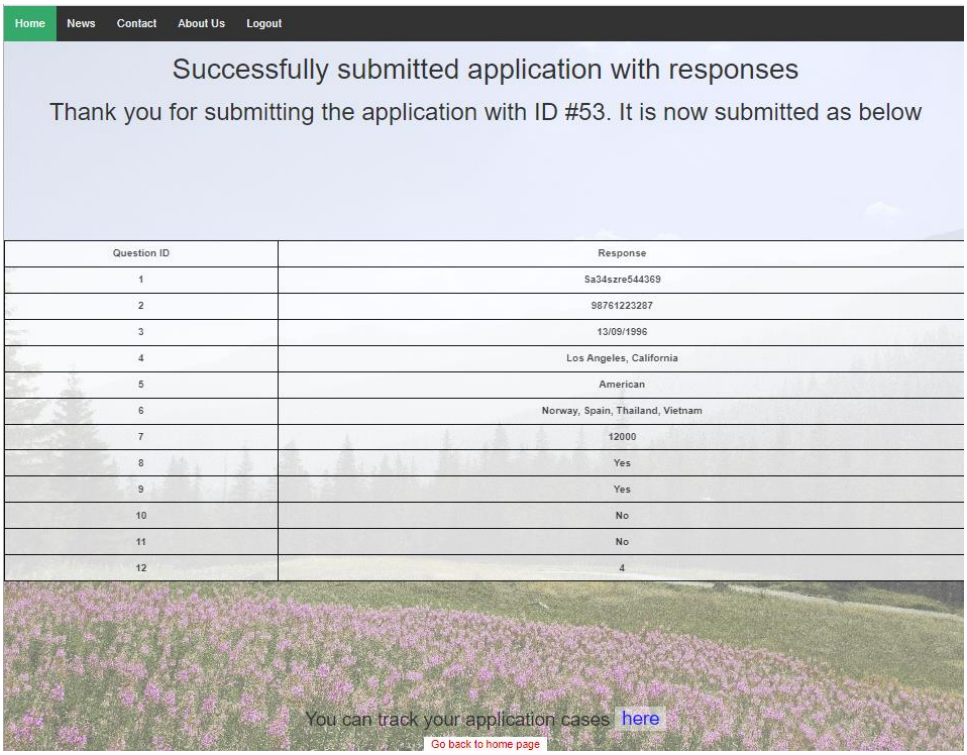
For this instance, only the Temporary Residence selection and routing is activated. On selecting the 'Temporary Residence' , the user is routed to

```
@app.route("/users/<int:user_id>/applications/<application_type_id>/
create")
```



This page further requests the response of the user for the questionnaire which is required for verification and case handling by the consultants. After submission, the user is directed to the route:

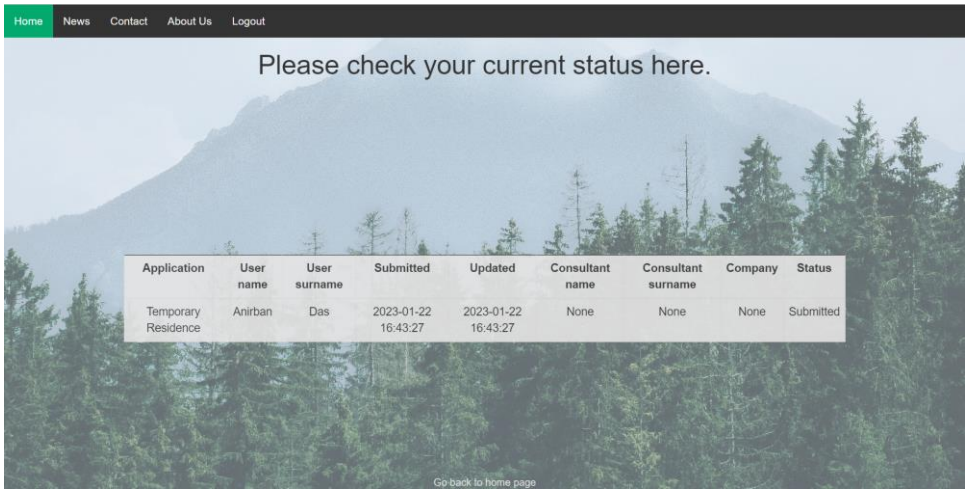
```
@app.route("/users/<int:user_id>/applications/<application_type_id>/
submit", methods=['POST'])
```



where, the user is provided with a verification table of the responses and a corresponding user ID.

From this page, the user can either switch to the homepage of track the application in

```
@app.route("/consultants/<int:consultant_id>/cases")
```

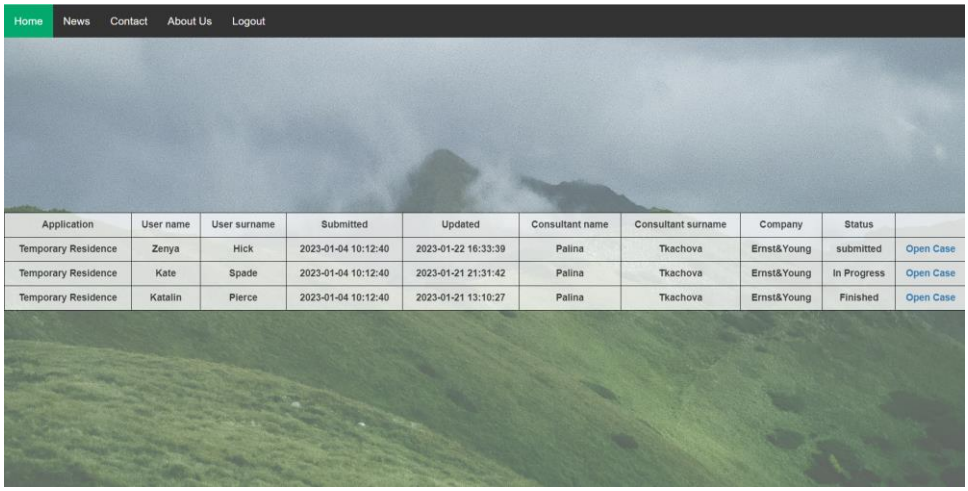


When the user login is corresponding to a consultant,

```
if results["consultant_id"] is not None:
        return redirect("/consultants/" + str(results['consultant_id'])
                        +"/cases")
```

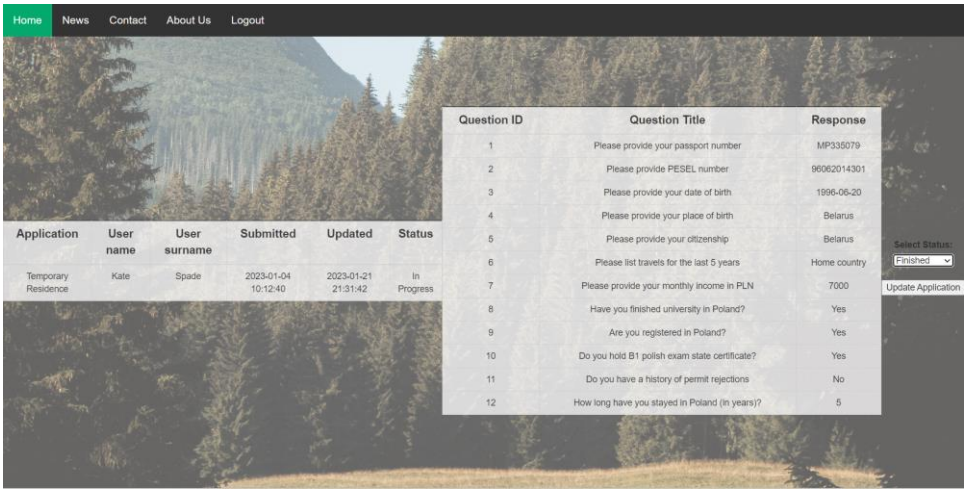
they are routed to:

```
@app.route("/consultants/<int:consultant_id>/cases")
```



where, the consultants can check the assigned users, work by opening their case and updating the status on current procedure at

```
@app.route("/consultants/<int:consultant_id>/cases/<application_id>",
methods=["GET", "POST"])
```



which in turn reflects to the `@app.route("/consultants/<int:consultant_id>/cases")`

for the user.

Finally, the route `@app.route("/About")` shows the graph of the total number of cases and the cases that are being worked on and updated on a dynamic basis. Every time the page is clicked and routed to, the status from the database is updates and saved as a image in

```
plt.savefig('static/status_graph.png')
```

which is finally accessed and showed for any user.

Final Queries

For table Creation:

```
CREATE TABLE "users" (
	"id" INTEGER,
	"first_name" TEXT,
	"last_name" TEXT,
	"email" TEXT UNIQUE,
	"created_at" DATETIME DEFAULT CURRENT_TIMESTAMP,
	"pass_hash" TEXT,
	PRIMARY KEY("id" AUTOINCREMENT) );
CREATE TABLE "consultants" (
	"id" INTEGER,
	"first_name" TEXT,
	"last_name" TEXT,
	"email" TEXT,
	"company_name" TEXT,
	"profile_photo" BLOB,
	"user_id" INTEGER,
	PRIMARY KEY("id" AUTOINCREMENT) );
CREATE TABLE "applications" (
	"id" INTEGER,
	"application_type_id" INTEGER,
	"creator_id" DATETIME,
	"assigned_consultant_id" INTEGER,
	"created_at" DATETIME DEFAULT CURRENT_TIMESTAMP,
	"updated_at" DATETIME DEFAULT CURRENT_TIMESTAMP,
	"status" TEXT DEFAULT 'Submitted',
	PRIMARY KEY("id" AUTOINCREMENT) );
```

```sql
CREATE TABLE "application_types" (
	"id" INTEGER,
	"title" TEXT,
	"created_at" DATETIME DEFAULT CURRENT_TIMESTAMP,
	PRIMARY KEY("id" AUTOINCREMENT) );

CREATE TABLE "application_responses" (
	"id" INTEGER,
	"created_at" DATETIME DEFAULT CURRENT_TIMESTAMP,
	"updated_at" DATETIME DEFAULT CURRENT_TIMESTAMP,
	"question_id" INTEGER,
	"application_id" INTEGER,
	"response" TEXT,
	"attachments" BLOB,
	PRIMARY KEY("id" AUTOINCREMENT) );

CREATE TABLE "questions" (
	"id" INTEGER,
	"application_type_id" INTEGER,
	"title" TEXT,
	"created_at" DATETIME DEFAULT CURRENT_TIMESTAMP,
	"required" INTEGER,
	PRIMARY KEY("id" AUTOINCREMENT) );

/* Run triggers separately after database is created*/
CREATE TRIGGER updated_applications
AFTER UPDATE ON applications FOR EACH ROW BEGIN UPDATE applications
SET updated_at = CURRENT_TIMESTAMP WHERE id = OLD.id;
END;
```

```
CREATE TRIGGER updated_responses

AFTER UPDATE ON application_responses

FOR EACH ROW BEGIN UPDATE application_responses SET updated_at =
CURRENT_TIMESTAMP WHERE id = OLD.id;

END;
```

<u>For execution in the Flask, listed below are the queries for various tasks:</u>

At first, the get_db_connection() is created which returns the 'conn' for further query execution

```
def get_db_connection():

      conn = sqlite3.connect('expat_project.db')

      conn.row_factory = sqlite3.Row

      conn.set_trace_callback(print)

      return conn
```

For Login:

```
cursor.execute('SELECT users.id, consultants.id as "consultant_id",
users.email, users.pass_hash FROM users LEFT JOIN consultants ON
consultants.user_id = users.id where users.email=? AND users.pass_hash=?',
(email, hashed_password))
```

For Registration:

```
c.execute("INSERT INTO users (first_name,last_name,email,pass_hash) VALUES
(?,?,?,?)",(first_name,last_name,email,hashed_password))

c.execute("SELECT id FROM users WHERE email=?",(email,))
```

For Application Selection:

```
conn.execute("SELECT id, title FROM application_types").fetchall()
```

For existing user:

```
conn.execute("""SELECT t.title, u.first_name as "user_first_name",
u.last_name as "user_last_name", a.created_at, a.updated_at, c.first_name
as "consultant_first_name", c.last_name as "consultant_last_name",
c.company_name, a.status FROM applications AS a LEFT JOIN application_types
as t ON a.application_type_id = t.id LEFT JOIN users AS u ON a.creator_id =
u.id LEFT JOIN consultants AS c ON a.assigned_consultant_id = c.id WHERE
a.creator_id = ?""",(user_id,)).fetchall()
```

For new application:

```
conn.execute('SELECT title, id, required FROM questions WHERE
application_type_id = ' + application_type_id ).fetchall()
```

For successful submission:

```
application_id = c.execute("INSERT INTO applications (application_type_id,
creator_id) VALUES (?,?)",(application_type_id,user_id)).lastrowid
```

```
application_response_id = c.execute("INSERT INTO application_responses
(question_id, application_id, response) VALUES (?, ?, ?)", (question_id,
application_id, response[question_id])).lastrowid
```

For consultant cases:

```
conn.execute("""SELECT a.id, t.title, u.first_name AS "user_first_name",
u.last_name AS "user_last_name", a.created_at, a.updated_at, c.first_name
AS "consultant_first_name", c.last_name AS "consultant_last_name",
c.company_name, a.status FROM applications as a LEFT JOIN application_types
as t ON a.application_type_id = t.id LEFT JOIN users as u ON a.creator_id =
u.id LEFT JOIN consultants as c ON a.assigned_consultant_id = c.id WHERE
a.assigned_consultant_id = ?""", (consultant_id,)).fetchall() conn.close()
```

For updates on cases:

```
c.execute("UPDATE applications SET status = ? WHERE id =
?",(application_status,application_id))
```

For consultant associated with application:

```
conn.execute("""SELECT a.id, t.title, u.first_name AS "user_first_name",
u.last_name AS "user_last_name", a.created_at, a.updated_at, c.first_name
AS "consultant_first_name", c.last_name AS "consultant_last_name",
c.company_name, a.status FROM applications as a LEFT JOIN application_types
as t ON a.application_type_id = t.id LEFT JOIN users as u ON a.creator_id =
u.id LEFT JOIN consultants as c ON a.assigned_consultant_id = c.id WHERE
a.assigned_consultant_id = ? AND a.id = ? """,(consultant_id,
application_id)).fetchall()
```

```
responses = conn.execute("""SELECT ar.*, q.title FROM application_responses
as ar LEFT JOIN questions q ON q.id = ar.question_id WHERE
ar.application_id = ? """, (application_id)).fetchall() conn.close()
```

For About Page:

```
cursor.execute("SELECT status, COUNT(*) FROM applications GROUP BY status")
```

Distribution of Work

	Anirban Das	**Palina Tkachova**
1	Code/Database for login and Registration.	Code/Database for Application list and Home
2	Code/Database for Application page	Code/Database for Application form page
3	Code for Consultant Overview	Database design for Consultant Overview
4	Visualization of the assigned pages	Visualization of the assigned pages
5	Documentation in Report and Presentation	Documentation in Report and Presentation