ISE321- Rotation Scheduling Project Documentation
Contributors: Chang Wang, Hangyi Li Sinan Aksu

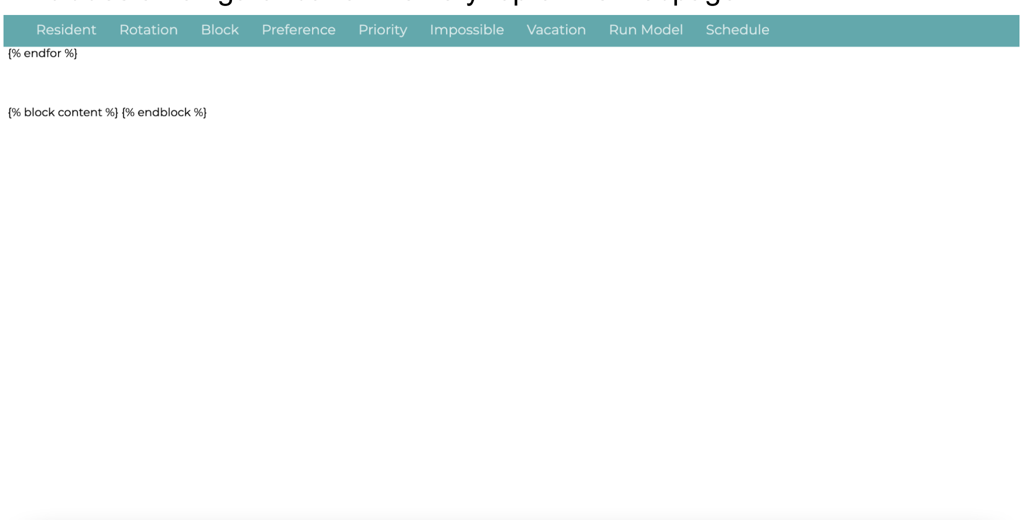
# Templates

## - Base.html

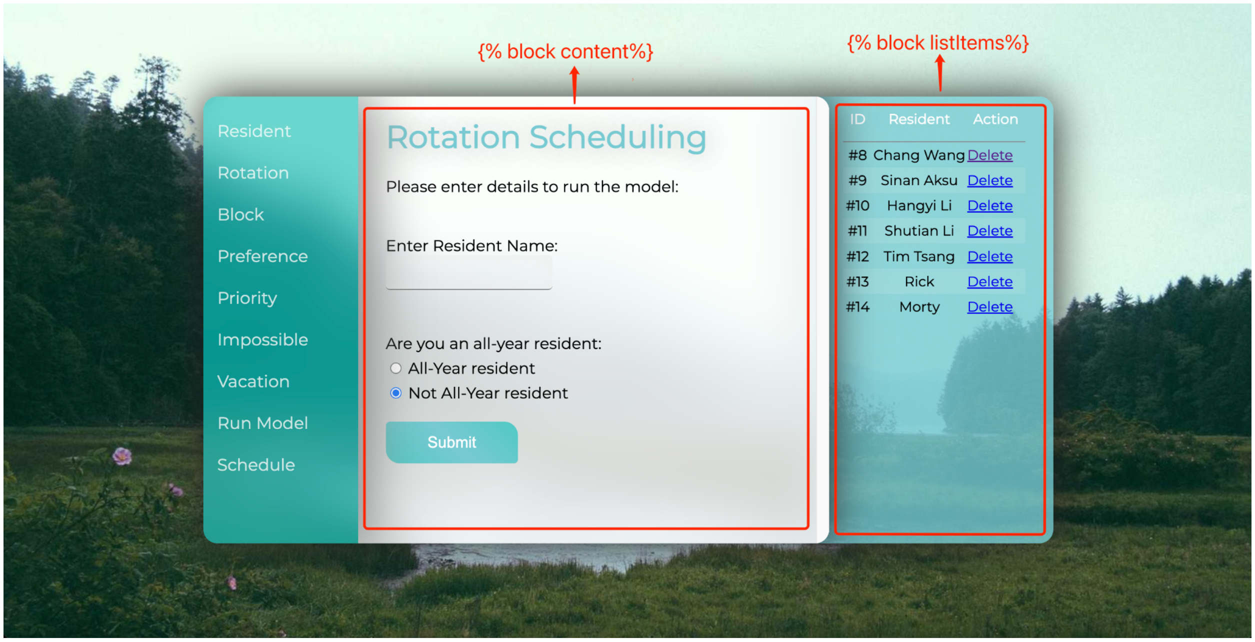
Base template for all the input interface
Style: claim the display of the interface


## - Base-full.html

This template is to display schedule output
It includes a navigator bar at the very top of the webpage.



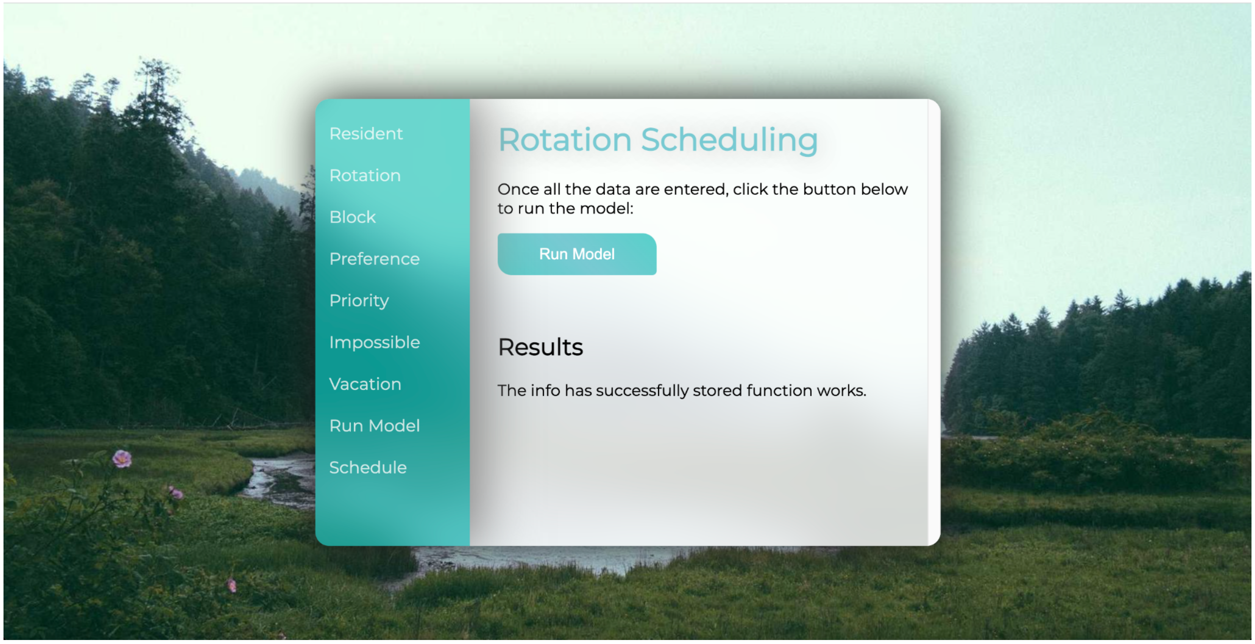
- Resident.html




The remaining input interfaces are all set in this way
Input interfaces includes: rotation, block, preference, priority, impossible, and vacation

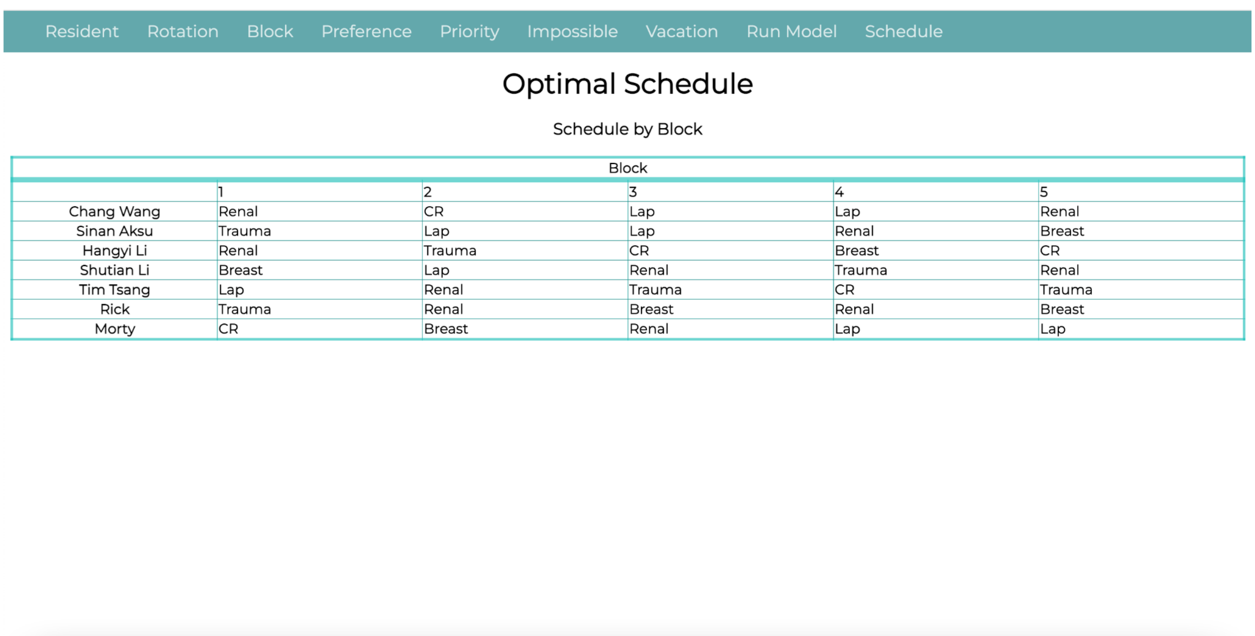
- Rotation.html
- Block.html
- Preference.html
- Priority.html
- Impossible.html
- vacation.html
- runModel.html

  The page user can ask to run the integer programming model once click "Run Model" button
  The result will be displayed on table.html




- Table.html

  Table.html is an extension of the base-full.html. It displays the output schedule by block. The file defines the style of the table frame.




## Optimal Schedule

Schedule by Block

| | Block | | | | |
|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 |
| Chang Wang | Renal | CR | Lap | Lap | Renal |
| Sinan Aksu | Trauma | Lap | Lap | Renal | Breast |
| Hangyi Li | Renal | Trauma | CR | Breast | CR |
| Shutian Li | Breast | Lap | Renal | Trauma | Renal |
| Tim Tsang | Lap | Renal | Trauma | CR | Trauma |
| Rick | Trauma | Renal | Breast | Renal | Breast |
| Morty | CR | Breast | Renal | Lap | Lap |

# Data. db:

the file to store all the user input from the interface

- Resident table
- Rotation table

# model.py:

The file that includes the integer programming model. So far, It uses data from data.db as input variables for the model.

- Import data from data.db
- Model function: includes objective, variables, and constraint function
- Constraints function: I set it as a separate function in order to easily change the constraints if we need adjustment
- Solve function: solve the model and print result
- Return results

# Datavalidation.py:

The file that validates the data stored in the database.
The file has two functions:

## 1. dataValidation(dict):

- Takes a dictionary as an input.
- Extracts all the lists from the dictionary and stores these list to their corresponding variables.
- Does the data validation.
- Returns a string that contains all the data errors caught.

## 2. checkDuplicates(list):

- Takes a list as an input.
- Check whether there are duplicate variables in the list.

The data validation consists of:

- Capitalizing people's names.
- Removing whitespaces in variable names.
- Checks whether the people, rotations or blocks lists are empty.
- Capitalizes blocks and checks their format
- Checks for duplicates in people rotations and block lists.
- In all the sets check whether all the variables exist in the people, rotations and blocks lists.

# main.py

This file contains 4 main important aspects of the program:

## 1 database setup

Data includes Resident data, Rotation data, Block data, Preference data, Priority data, Impossible data, Vacation data

Current database :

1

```
Store_Resident_data
residentId
name
allYear
```

2

```
Store_Rotation_data
 rotationId
 rotationName
 mustDo
 busy
 p_min
 p_max
```

3

```
Store_Pref_data
 prefId
 residentname
 rotationName
 block
```

4

```
Store_Priority_data
 prifId
 residentname
 rotationName
 block
```

5

```
Store_Impo_data
 impoId
 residentname
 rotationName
 block
```

6

```
Store_Vacation_data
 vacId
 residentname
 block
```

Template:
In order to add more table in the database:

1 create one more function in the main.py file with the signature

```
class Store_{table_name}_data(db.Model):
```

2 add table name and corresponding table columns (id and other attributes)
Example:

```
__tablename__ =
Id = db.Column('Id', db.Integer, primary_key = True)
name = db.Column('name', db.String(50))
```

This will be the attribute in the table

## 2 create data in the database

Template:
In order to add features to create data in the database:

1 create one more function in the main.py file with the signature

```
@app.route('/', methods=['GET', 'POST'])
def{table_name}():
```

2 add the following codes to get the input from the user input box
Example:

```
result = False
```

```
if not os.path.exists(os.path.join(basedir, 'data.db')):
  db.create_all()
if request.method == 'POST':
  form = request.form

  # Store data to table
  name = request.form.get('name',False)
```

This has to match the attribute in the table

3 commit the change to the database in order to modify/add the data

```
  db.session.add(Store_{table_name}_data(name))
  db.session.commit()

  # render result
  result = calculate(form)
datas = Store_{table_name}_data.query.all()
```

4 render the index.html file stored in the templates folder

```
return render_template('resident.html', result=result, datas=datas)
```

# 3 view data in the database

Inorder to view the data in the database, we need the GET request

Template:

1 create one more function in the main.py file with the signature

```
@app.route('/myData', methods=['GET'])
def myData():
```

2 query the data from the database:

```
rotationDatas = Store_Rotation_data.query.all()
return render_template('myData.html', rotationDatas=rotationDatas)
```

# 4 delete data in the database

```
#delete the resident information and redirect to the same page
```

Inorder to delete the data in the database
Template:

1 create one more function in the main.py file with the signature

```
@app.route('/deleteResident/<int:id>')
def deleteResident(id):
```

2 get the line of data by querying its id
Example:

```
 resident_to_delete = Store_Resident_data.query.get_or_404(id)
```

This has to match the id in the table

3 use the id to find the data and commit the change to the database in order to delete the data

```
    try:
   db.session.delete(resident_to_delete)
   db.session.commit()
   return redirect('/')
 except:
   return "Deletion Problem"
```

# 5 update data in the database

Inorder to delete the data in the database, we need the GET and POST request
Template:
1 create one more function in the main.py file with the signature

```
@app.route('/updateResident/<int:id>', methods=['GET', 'POST'])
def updateResident(id):
```

2 get the line of data by querying its id
Example:

```
 resident_to_update = Store_Resident_data.query.get_or_404(id)
```

This has to match the id in the table

3 use the id to find the data and commit the change to the database in order to update the data

```
 f request.method == 'POST':
# Store data to table
resident_to_update.name = request.form.get('residentName',False)
resident_to_update.allYear = request.form.get('allYear',False)
try:
  db.session.commit()
```

4 Redirect to the view page

```
return redirect("/")
   except:
     return "There is a problem"
```

5 Render and return the page

```
 else:
   return render_template('updateResident.html',
resident_to_update=resident_to_update)
```

6 we also need to add the html page for out updating purpose:
The template is located in the Template folder
The html page will receive the update request and revert back to the function to update the database