Shift Schedule Builder Desktop - Client Server

Yu-Chun Lin and Kuei-Ching Yang

October 29, 2023

1 Description

Shift Schedule Builder Desktop is an application designed to simplify the task of shift arrangement for employees. Its primary aim is to provide a user-friendly interface that allows users to input their shift requirements and configurations easily and generate a shift that satisfies these shift requirements as much as possible.

The user can set up some basic shift parameters such as the number of days and the number of employees. The application will generate an empty shift that satisfies the basic requirement. In addition to basic shift parameters, users are able to add some shift requirements. The application provides several common shift requirements. For example, users are able to set up the expected number of working days in the given range of the shift. The application will automatically generate a schedule for them based on the given data and constraints. Since the scheduling problem is a multi-objective optimization problem, the users have to specify the weight of each constraints. If the user values a shift requirement a lot, the user has to specify bigger weight on that shift requirement.

The project is developed using Python and PyQt.

1.1 UI Sketch

We developed the UI by using **PyQt**. We have set up the basic parameters such as the width and the height of the main window by using the **QtDesigner**. The user interface is designed by inheriting the **QWidget**.

1.1.1 Login

In the beginning, the user has to login. The login dialog inherits QDialog. The login window will collect the username and password of the user to check and set different user. The login dialog serves as a view and controller in the application. After collecting the user's information, it will send the information to the database adapter to check if the user exists or not.

1.1.2 Main Window

There is a class MainWindow which inherits QMainWindow and creates the frame of the window. The main window enables users to create multiple shifts by adding tabs and set up the parameters. The main window is composed of several components. The UI components are shown in the figure below. The detail of the design pattern will be elaborated in the architecture design section.

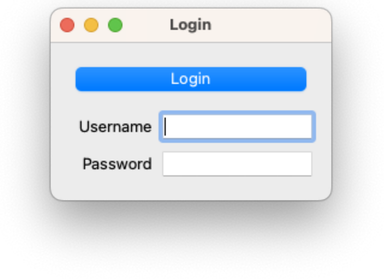


Figure 1: User Interface - Login Dialog

1.2 Main use cases

1.2.1 Login

Use case: Login

Actor: System user

Goal: To login the system

Overview: When the users open the application, they have to sign in the system. If they type their username and password correctly, they are able to sign in. However, if they type either username or password wrong, the system will pop up a window showing a message that tells the users they failed to sign in.

Actor action:

1. user: A user types username and password
2. user: The user click login button
3. system: If both username and password are correct, then the system will allow user to login; otherwise, the system pop up a message that tells the user that the given information is not correct.
4. user: If the user is fail to login, the user may click the close button to close the application; otherwise, the system will close the login dialog window and show the application's user interface.

Alternative courses:

Steps 1 and 2: The user may click the close button to close the application.

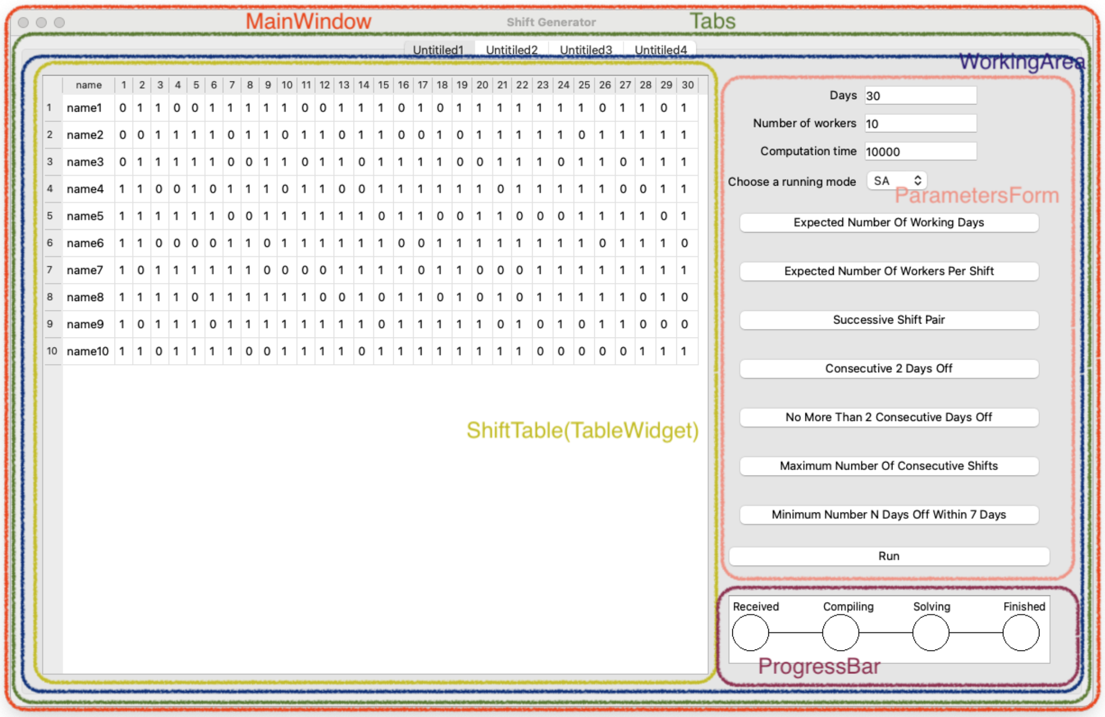
1.2.2 Sets up shift parameters

Use case: Sets up shift parameters

Actor: System user

Goal: To sets up shift parameters

Overview: Before the application generates a shift for the user, the user has to specify the shift paramters. The parameters include days, number of workers, computation time, running mode.



	name	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
1	name1	0	1	1	0	0	1	1	1	1	1	0	0	1	1	1	0	1	0	1	1	1	1	1	1	1	0	1	1	0	1
2	name2	0	0	1	1	1	1	0	1	1	0	1	1	0	1	1	0	0	1	0	1	1	1	1	1	0	1	1	1	1	1
3	name3	0	1	1	1	1	1	0	0	1	1	0	1	1	0	1	1	1	1	0	0	1	1	1	0	1	1	0	1	1	1
4	name4	1	1	0	0	1	0	1	1	1	0	1	1	0	0	1	1	1	1	1	1	0	1	1	1	1	1	0	0	1	1
5	name5	1	1	1	1	1	1	0	0	1	1	1	1	1	1	0	1	1	0	0	1	1	0	0	0	1	1	1	1	0	1
6	name6	1	1	0	0	0	0	1	1	0	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	0	1	1	1	0
7	name7	1	0	1	1	1	1	1	1	0	0	0	0	1	1	1	1	0	1	1	0	0	0	1	1	1	1	1	1	1	1
8	name8	1	1	1	1	0	1	1	1	1	1	1	0	0	1	0	1	1	0	1	0	1	0	1	1	1	1	1	0	1	0
9	name9	1	0	1	1	1	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1	0	1	0	1	0	1	1	0	0	0
10	name10	1	1	0	1	1	1	1	0	0	1	1	1	1	0	1	1	1	1	1	1	1	1	0	0	0	0	0	1	1	1

Days 30
Number of workers 10
Computation time 10000
Choose a running mode SA

Figure 2: User Interface - Main Window

Actor action:

1. user: A user types number of days in the parameters form.
2. system: The system regenerates a shift based on new parameter configuration.
3. user: The user types number of workers in the parameter form.
4. system: The system regenerates a shift based on new parameter configuration.
5. user: The user types the computation time in the parameter form.

Alternative courses:

Steps 1 and 2: The user may click the close button to close the application.

1.2.3 Sets up shift requirements

user case: Setup shift requirements

Actor: System user

Goal: To sets up shift requirements

Overview: Before the application generates a shift for the user, the user is able to give some shift requirements. The available shift requirements are the expected number of working days, the expected number of workers per shift, the successive shift pair, the consecutive 2 days off, no more than consecutive days off, the maximum number of consecutive shifts, and the minimum number of N days off within 7 days.

Actor action:

1. user: The user click one of the shift requirements.
2. system: The system pops up a dialog to ask the user to provide the parameter of the shift requirement.
3. user: The user provides the preferred parameters.
4. user: The user click Add button to add the parameters.

Alternative courses

Steps 3 and 4: The user may click the close button to close the dialog.

1.2.4 Generate a shift

user case: Generate a shift requirements

Actor: System user

Goal: To generate a shift

Overview: After the user finishes sets up the parameters and requirements, the user is ready to generate a shift. The user can click the run button to start generating the shift.

Actor action:

1. user: The user click the Run button to start generating a shift.
2. system: The run button becomes disabled and start generating a shift for the user.
3. system: The progress bar will keep posted the status of the task.
4. system: When finishing generating a shift, the system shows the shift on the user interface.
5. system: The run button will be unlocked so that users is able to re-run the task.

1.2.5 Create a new shift

user case: Create a new shift

Actor: System user

Goal: To create a new shift

Overview: User can create a new shift by adding a tab. Each shift can be generated concurrently.

Actor action:

1. user: The user click command+t on Mac computers or ctrl+t on windows computers.
2. system: The create a new tab for the user.

1.2.6 Save a shift

user case: Save a shift

Actor: System user

Goal: To save a shift

Overview: After acquiring a shift, the user is able to save his/her shift to the device.

Actor action:

1. user: The user click command+s on Mac computers or ctrl+s on windows computers.
2. system: The system pops up a window to ask the user to input the name of the shift and the location that the user wants to store the shift.
3. user: The user types the name and chooses a location.
4. system: The system exports the shift of the current tab to an excel sheet with given name and to the given location.

2 Database design

The data models of the application are intricate. The database not only has to store the user's information but also the shift data. Since users are allowed to set up multiple shift requirements, the number of the shift constraints will vary. Besides, the data size of the shift will vary based on the number of days and the number of workers. It is intuitive to use the NoSQL database to store and load the data; thus, we choose MongoDB to store the data. The document and the data model of the application are compatible. In the database, we create two collections to store documents, one collection is Users and the other is Shifts. The entity relationship diagram is shown below.

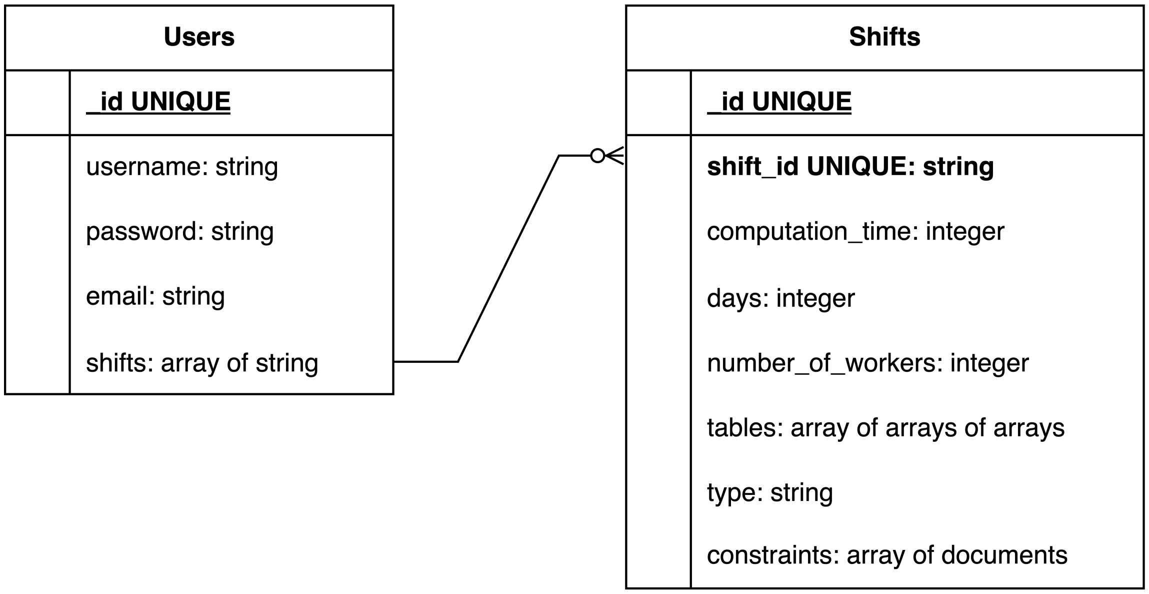


Figure 3: Entity Relationship Diagram

2.1 Users

The collection stores the basic information of a user such as username, password, email, and shifts. Here provides an example of a document. (In this prototype, we didn't use hash function to hash the password, but in the future, this feature will be added to protect user's information.)

```
[
  {
    _id: ObjectId("6518396323c0f9910c4fc915"),
    username: 'admin',
    password: 'password',
    email: 'email@gmail.com',
    shifts: [
      '66d74521-6947-4cd3-b2e9-d201140893bb',
      '0926ea12-8907-43e7-a2cb-340043eb7c12',
      '9223cea4-69f8-4dab-8832-4b48c952d91d',
      '0750a390-7f19-4341-9aa2-f41ccfc2c0f9'
    ]
  }
]
```

The description of those fields is as follows.
1. _id: A unique object id which is automatically generated by the database.
2. username : The username of the user.
3. password: The password of the user.
4. email: The email of the user.
5. shifts: A nested document that stores the shift ids owned by the user. In this example, the user owns 4 shifts. The shifts id will be used to load the shifts in Shifts collection.

2.2 Shifts

The collection stores shifts' information. The shifts' information includes shift_id, parameters, and the content of the shift. Here provides a snippet of a document in the collection.

```
[
  {
    _id: ObjectId("6519d6a87b2c1ef5a957ae5c"),
    shift_id: '594f91af-cade-47f3-aeff-a0d461285e1a',
    computation_time: 30,
    constraints: [
      {
        name: 'expected_number_of_working_days',
        parameters: { weight: '10', ewd: '20', days_off_index: [] }
      },
      {
        name: 'expected_number_of_workers_per_shift',
        parameters: { weight: '10', enwps: '8', days_off_index: [] }
      },
      {
```

```
    name: 'consecutive_2_days_leave',
    parameters: { weight: '10', days_off_index: [] }
  },
  {
    name: 'minimum_n_days_leave_within_7_days',
    parameters: { weight: '10', mndlw7d: '2', days_off_index: [] }
  }
],
days: 30,
name_list: [
  'name1', 'name2',
  'name3', 'name4',
  'name5', 'name6',
  'name7', 'name8',
  'name9', 'name10'
],
number_of_workers: 10,
tables: [
  [
    [
      1, 1, 1, 1, 0, 0, 1, 1, 1,
      1, 1, 0, 0, 1, 1, 1, 1, 1,
      1, 1, 0, 0, 0, 0, 0, 1, 1,
      1, 1, 1
    ],
    ...
    [
      1, 1, 0, 0, 1, 1, 1, 1, 1,
      1, 0, 0, 1, 1, 1, 1, 1, 0,
      0, 1, 1, 1, 1, 1, 1, 1, 1,
      0, 0, 0
    ]
  ],
  [
   [
      1, 1, 1, 1, 0, 0, 1, 1, 1,
      1, 1, 0, 0, 1, 1, 1, 1, 1,
      1, 1, 0, 0, 0, 0, 0, 1, 1,
      1, 1, 1
    ],
    ...
    [
      1, 1, 0, 0, 1, 1, 1, 1, 1,
      1, 0, 0, 1, 1, 1, 1, 1, 0,
      0, 1, 1, 1, 1, 1, 1, 1, 1,
      0, 0, 1
    ]
```

```
      ]
    ],
    type: 'DAU'
  }
]
```

1. _id: A unique object id which is automatically generated by the database.
2. shift_id: The id of the shift. The user entity will also store **shift_id** in the **shifts** field.
3. computation_time: The computation time of the shift.
4. constraints: The shift requirements. The field is a nested document that stores the name of the requirement and parameters.
5. days: The number of days
6. name_list : The name list.
7. number_of_workers: The number of workers in the shift.
8. tables: Tables is a 3-D array which can stores multiple shifts. Each shift is a 2-D array.
9. type: The type of optimizer that solves the shift scheduling optimization problem. The type field will either be 'SA' or 'DAU'. SA means the optimization algorithm is simulated annealing algorithm, and DAU means the problem is solved by using the digital annealing unit.

2.3 Sample Code

Since we use NoSQL database, we are only able to provide the sample code that we used to insert, search, and update the document. The following code snippets are parts of class DataAdapter.

2.3.1 Search user

```
user_data = self.db.Users.find_one({"username": username, "password": password})
```

2.3.2 Update user's shifts

The following code snippet update user's shifts by using update_one function.

```
user_collection = self.db.Users
user_collection.update_one(
{"username": user.getUsername()},
{"$set": {"shifts": user.getShifts()}}
)
```

2.3.3 Save a shift

The following code snippet uses update_one to save a shifts, but we set upsert=True for the case that the shift doesn't originally exist. In that case, it will insert a document.

```
collection = self.db.Shifts
inserted_data = shift.getShiftConfiguration()
inserted_data['table'] = tables
collection.update_one({"shift_id" : shift.getShiftId()}, {"$set": inserted_data}, upsert=Tr
```

2.3.4 Load a shift

The following code snippet will find a shift based on given shift_id.

```
shift = self.db.Shifts.find_one({"shift_id": shift_id})
```

3 Architectural design

The scope of the application includes the user interface, server, database, and solvers. The user interface collects user's preference of the shift. The server components processes user's requests to load/save the data and handle the computation. The database stores and loads the users' information and shifts. Solvers are used to solve the optimization problem to generate the shift that satisfies the requirements as much as possible. After acquiring the solution from cloud computation or on-device computation, Solvers will return the shift back to view/controller to show on the user interface. In this version, solvers will use web socket to do the communication. After View/controller received the shift data, it will send the data back to server to store the data into the database.

The following section will give details on architecture design for each component.

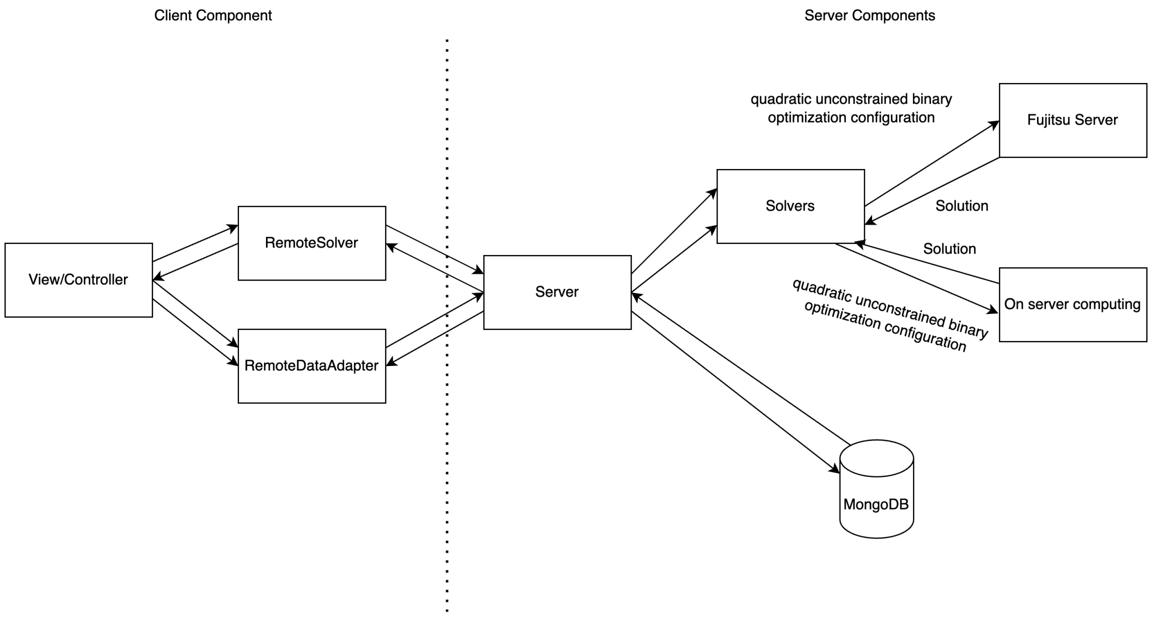


Figure 4: System Architecture

3.1 Server

We developed a tiny framework that enables us to flexibly extend the function of the server. The server's architecture is as shown in Fig. 5 The server is composed of two parts, a http server and a web socket server. When the server received an incoming request, the instance of **ProtocolTypeRouter** will do routing based on the request's protocol. For example, if the request is a http request, ProtocolTypeRouter will transfer the request to HttpServer. When the server received a request sent from ProtocolTypeRouter, both servers will do routing based on the request's URI to a certain route. Each route will handle the request and the response. The only things that developers have to do is to define the API endpoints and override the handle function of the corresponding Route to handle the request. Here provides a snippet of code that shows how to use this tiny framework.

```
class TestRoute(Route):

    def handle(self):
        data = {
            "message": "successful"
        }
        self.response.send(200, json.dumps(data), content_type="application/json")


class EchoWebsocketRoute(WebSocketRoute):

    def handle(self):
        while True:
            received_message = self._recv()
            print("Received Text: ", received_message)
            if received_message is None:
                break
            else:
                self.response.send(received_message)

if __name__ == '__main__':
    server = ProtocolTypeRouter({
        'http': HttpServer(routes=[(r'/', TestRoute)]),
        'websocket': WebSocketServer(routes=[(r'/chat', EchoWebsocketRoute)])
    })
    server.run()
```

We just need to inherit Route type and override handle function, and register the Route type to the corresponding server and specify the uri. This enables us to easily extend the function of server. Fig. 6 details the UML of the server side.

3.1.1 RemoteSolver

RemoteSolver is a class that establishes the web socket connection between UI and the server. It also processes the message sent from the server side and connect itself and the progress bar to update the status.

We wrote two remote solvers, RemoteDAUSolver and RemoteSASolver. The corresponding APIs are ws://localhost:8888/dau and ws://localhost:8888/sa. Both solvers inherit from RemoteSolver. When the function solve is called, it will use web socket to send the optimization problem to the server. Here provides a simple example of the data. The data format in the communication is in the json format.

```
{
    "type": "SA",
    "days": 30,
    "number_of_workers": 10,
```

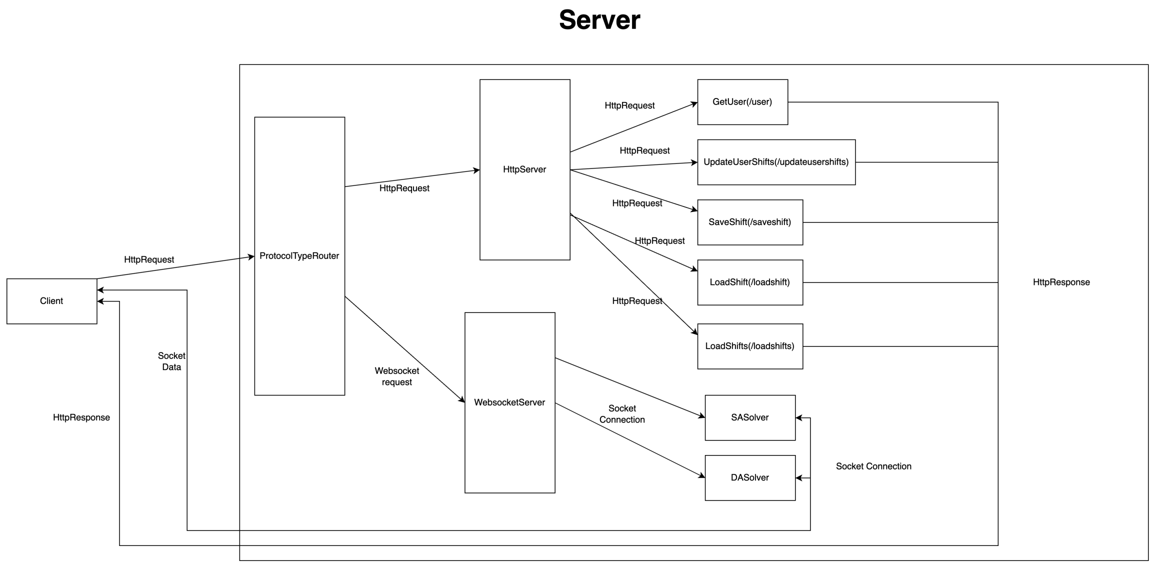


Figure 5: The architecture of the tiny framework

```
"computation_time": 10000,
"reserved_leave": [],
"constraints": [
    {
        "name": "expected_number_of_working_days",
        "parameters": {
            "weight": "10",
            "ewd": "20"
        }
    },
    {
        "name": "expected_number_of_workers_per_shift",
        "parameters": {
            "weight": "8",
            "enwps": "8"
        }
    }
],
"content": [
    {
        "name": "name1",
        "shift_array": ["0","0","1","1","0","0","1","1","1","1","1","0","0",
        "0","1","1","0","1","1","1","1","1","0","1","1","1","1","1","1","1"
        ]
    },
    {
        "name": "name2",
        "shift_array": [...]
    },
```

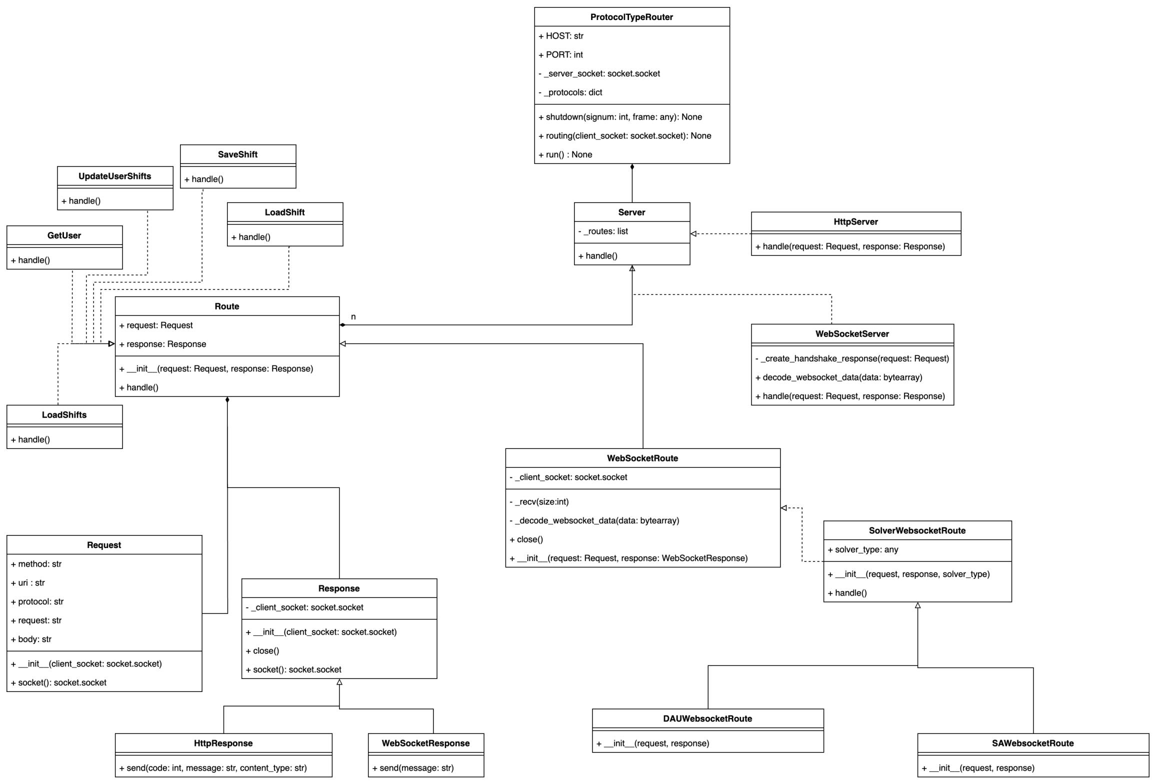

ProtocolTypeRouter
+ HOST: str
+ PORT: int
- _server_socket: socket.socket
- _protocols: dict
+ shutdown(signum: int, frame: any): None
+ routing(client_socket: socket.socket): None
+ run() : None
SaveShift
+ handle()
UpdateUserShifts
+ handle()
LoadShift
+ handle()
GetUser
+ handle()
Server
- _routes: list
+ handle()
HttpServer
+ handle(request: Request, response: Response)
Route
+ request: Request
+ response: Response
+ __init__(request: Request, response: Response)
+ handle()
n
WebSocketServer
- _create_handshake_response(request: Request)
+ decode_websocket_data(data: bytearray)
+ handle(request: Request, response: Response)
LoadShifts
+ handle()
WebSocketRoute
- _client_socket: socket.socket
- _recv(size:int)
- _decode_websocket_data(data: bytearray)
+ close()
+ __init__(request: Request, response: WebSocketResponse)
SolverWebsocketRoute
+ solver_type: any
+ __init__(request, response, solver_type)
+ handle()
Request
+ method: str
+ uri : str
+ protocol: str
+ request: str
+ body: str
+ __init__(client_socket: socket.socket)
+ socket(): socket.socket
Response
- _client_socket: socket.socket
+ __init__(client_socket: socket.socket)
+ close()
+ socket(): socket.socket
DAUWebsocketRoute
+ __init__(request, response)
SAWebsocketRoute
+ __init__(request, response)
HttpResponse
+ send(code: int, message: str, content_type: str)
WebSocketResponse
+ send(message: str)

Figure 6: The UML diagram of the server

```
        ...
    ]
}
```

In the server side, we defined a class SolverWebsocketRoute which inherits from WebSocketRoute. The class will process the request and invoke the corresponding computation service to do the optimization problem. We also defined two classes that inherit from SolverWebsockeRoute, which are DAUWebsocketRoute and SAWebsocketRoute. Both give the computation service to the base class. During the computation, the server will keep UI posted the state of the computation. For example, when the problem is compiled. It will use web socket to send

`{"message": "status", "status" : "compiling"}.`

The user interface can update the state of the computation to the user.

3.1.2 Data Server

The data server uses HTTP protocol to communicate with the user interface. There are 5 APIs written in the server. Each API is handled by a Route class which inherits from class Route and override the function handle.

Get the user

API Endpoints: /user

Description: Through /user URI, the server accepts the post method. The client posts username and password to the server. The server will use the function getUser defined in the DataAdapter to get the user owned by the collection User. Then, the server will return the user in JSON.

Method: POST

Example Request Data:
```
{
        "username":"admin",
        "password":"password"
    }
```

Example Response Data:
```
{
        "username": "admin",
        "password": "password",
        "email": "admin@gmail.com",
        "shifts": [
            "10a67b34-dede-4a99-97ef-8f0b76927947"
        ]
    }
```

Update user shifts

API Endpoints: /updateusershifts

Description: Through /updateusershifts URI, the client uses the post method to send the user's information to the server. The server will use the function updateUserShifts defined in the DataAdapter to update the shift_id array which is owned by the user and stored in the collection User.

Method: POST

Example Request Data:
```
{
            "username": "admin",
            "password": "pasword",
            "email": "admin@gmail.com",
            "shifts": [
                "10a67b34-dede-4a99-97ef-8f0b76927947",
                "0fc9f879-b236-40c4-a6f6-b5f3d5c7ddab"
            ]
    }
```

Example Response Data: `no response data`

Save a shift

API Endpoints: /saveshift

Description: Through /saveshift URI, the client uses the post method to send the user's information and the shift's information to the server. The server will use the function saveShift defined in the DataAdapter to save the parameters owned by the shift and stored in the collection Shifts.

Method: POST

Example Request Data: {
```
        "shift_id": "10a67b34-dede-4a99-97ef-8f0b76927947",
        "computation_time": 10000,
        "constraints": [],
        "days": 30,
        "name_list": [ "name1", "name2", "name3", "name4", "name5",
            "name6", "name7", "name8", "name9",  "name10" ],
        "number_of_workers": 10,
        "reserved_leave": [],
        "tables": [
    [ [ 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1,
     1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 4 1, 1, 1 ], ...
    [ 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1,
    1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0 ] ] ],
        "type": "DAU",
    }
```

Example Response Data: `no response data`

Load a shift

API Endpoints: /loadshift

Description:

Method: POST

Example Request Data: {
```
        "shift_id":"10a67b34-dede-4a99-97ef-8f0b76927947"
    }
```

Example Response Data: {
```
        "shift_id": "10a67b34-dede-4a99-97ef-8f0b76927947",
        "computation_time": 10000,
        "constraints": [],
        "days": 30,
        "name_list": [ "name1", "name2", "name3", "name4", "name5",
            "name6", "name7", "name8", "name9",  "name10" ],
        "number_of_workers": 10,
        "reserved_leave": [],
        "tables": [
    [ [ 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1,
     1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 4 1, 1, 1 ], ...
    [ 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1,
    1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0 ] ] ],
        "type": "DAU",
    }
```

Load shifts

API Endpoints: /loadshifts

Description: Through /loadshifts URI, the server accepts the post method. The client posts the user's information to the server to load all shifts for the user.

The server will use the function loadShifts defined in the DataAdapter to get all shifts owned by the user. The function loadShifts will return a list of Shift instances. Finally, the server will convert each shift in the list to a dictionary variable and store the data in a list. Then, return the data back to the client.

Method: POST

Example Request Data: `{`

```
            "username": "admin",
            "password": "pasword",
            "email": "admin@gmail.com",
            "shifts": [
                "10a67b34-dede-4a99-97ef-8f0b76927947",
                "0fc9f879-b236-40c4-a6f6-b5f3d5c7ddab"
            ]
    }
```

Example Response Data: `{`

```
      "shift_list":[
    {
        "shift_id": "10a67b34-dede-4a99-97ef-8f0b76927947",
        "computation_time": 10000,
        "constraints": [],
        "days": 30,
        "name_list": [ "name1", "name2", "name3", "name4", "name5",
            "name6", "name7", "name8", "name9",  "name10" ],
        "number_of_workers": 10,
        "reserved_leave": [],
        "tables": [
    [ [ 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1,
     1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 4 1, 1, 1 ], ...
    [ 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1,
    1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0 ] ] ],
        "type": "DAU",
    },
    {
        "shift_id": "0fc9f879-b236-40c4-a6f6-b5f3d5c7ddab ",
        "computation_time": 10000,
        "constraints": [{
                        "name": "expected_number_of_workers_per_shift",
                        "parameters": {
                            "weight": "20",
                            "enwps": "10",
                            "days_off_index": []
                        },
                    {
                        "name": "no_consecutive_leave",
                        "parameters": {
                            "weight": "10",
                            "days_off_index": []
                        }
                    }
                    }
    ],
        "days": 30,
        "name_list": [ "name1", "name2", "name3", "name4", "name5",
            "name6", "name7", "name8", "name9",  "name10" ],
        "number_of_workers": 10,
        "reserved_leave": [],
        "tables": [
    [ [ 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1,
     1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 4 1, 1, 1 ], ...
    [ 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1,
    1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0 ] ] ],
        "type": "SA",
    }
      ]
    }
```

3.2 Database

We design two models, User and Shift, and class DataAccess to manipulate the data in the Application. Class User corresponds to a document in the collection Users and class Shift corresponds to a document in the collection Shifts. Please refer to Fig. 9. for the UML of data adapter class. Class DataAccess is an interface that will be implemented by both DataAdapter and RemoteDataAdapter.

3.2.1 DataAdapter

The DataAdapter will establish a connection to the MongoDB database. We can load and save the document into the collections. Followings describe the function defined in the class DataAdapter.

getUser: We can get the user's information by providing the username and password to the function. The function will search the document in the collection Users with the given data. If the function successfully gets a document from the collection, it will generate an instance of class User and return the object; otherwise, return None.

updateUserShifts: We can update the user's information by providing the document in the collection Users to the function. The function will search and update the document in the collection Users with the given data.

saveShift: We can update the shift's information by providing the document in the collections Users and Shifts to the function. The function will search and update the document in the collection Shifts with the given data.

loadShift: We can get the shift's information by providing the shift_id to the function. The function will search the document in the collection Shifts with the given data. If the function successfully gets a document from the collection, it will generate an instance of class Shift and return the object.

loadShifts: We can get the several shifts' information by providing the document in the collection Users to the function. The function will search the document in the collection Shifts with the given data. The function will get several documents from the collection and generate a list to store the documents and return the object.

3.3 RemoteDataAdapter

The RemoteDataAdapter serves as a connection between user interface and the server. When the user load and save data through the

getUser: The function will use the HTTP POST method to post the username and password, which are the parameters of the function, to the server. When the function receives the response from the server, the server will generate an instance of class User and return.

updateUserShifts: The function will use the HTTP POST method to post the document in the collection Users, which is the parameter of the function, to the server. When the function receives the response from the server, the server will update the document in the collection Users with the given data.

saveShift: The function will use the HTTP POST method to post the document in the collections Users and Shifts, which are the parameters of the function, to the server. When the function receives the response from the server, the server will update the document in the collection Shifts with the given data.

loadShift: The function will use the HTTP POST method to post the shift_id, which is the parameter of the function, to the server. When the function receives the response from the server, the server will generate an instance of class Shift and return.

loadShifts: The function will use the HTTP POST method to post the document in the collection Users, which is the parameter of the function, to the server. When the function receives the response from the server, the server will generate a list to store the documents and return the object.

3.4 UI

As shown in Fig. 2. The main window is composed of several UI components. The relationship between each component is presented in the Fig. 7.

3.5 Solvers(Algorithms)

The shift scheduling problem is modeled by the quadratic unconstrained binary optimization (QUBO) problem. 0 represents a day off and 1 represents a working day. Once the user adds shift requirements, they implicitly formulate their QUBO problem. The QUBO problem is compiled in user's computer and solved by either the digital annealer or the simulated annealing algorithm on users' device. After the device finishes computation, the solution is then post-processed to generate the final schedule. Digital annealer is a computation service provided by Fujitsu Ltd. Digital annealer is a quantum-inspired device that is good at solving optimization problems.

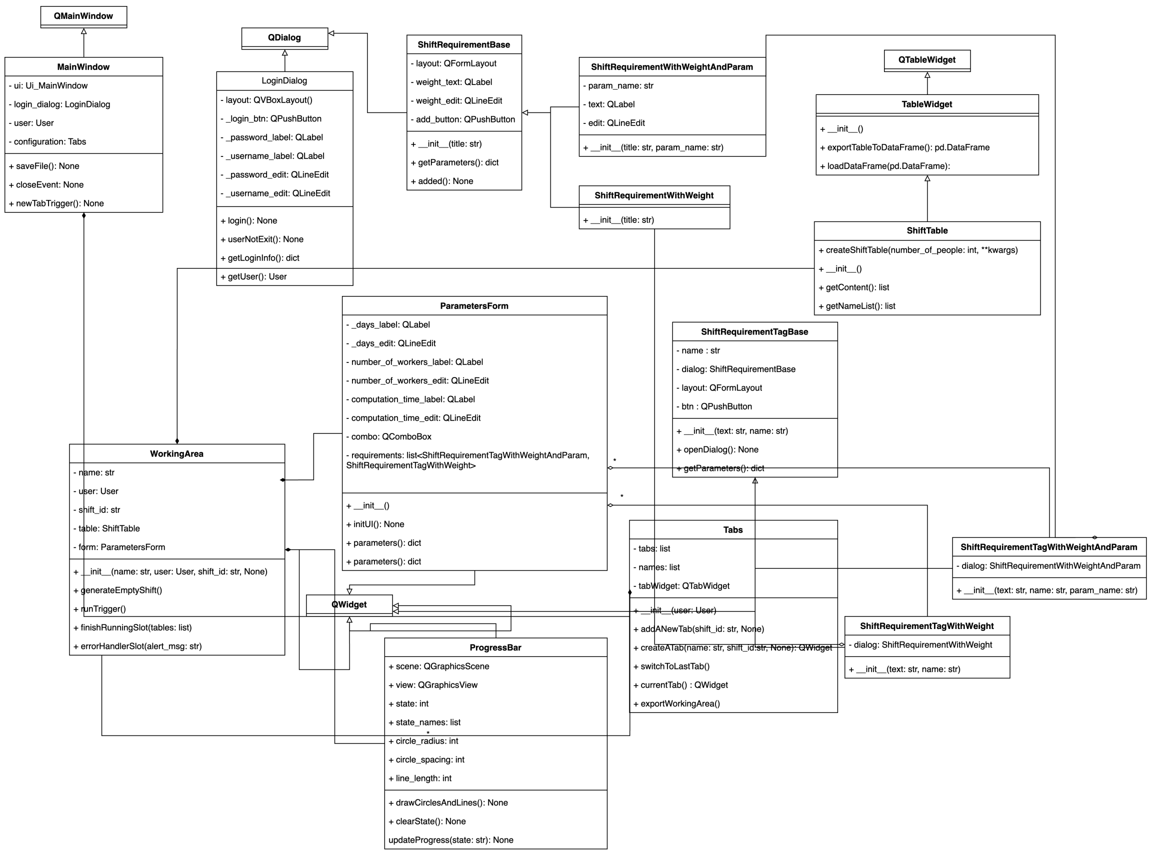


Figure 7: UI's UML

In this prototype, the application offers limited shift requirements for users to choose from. The available shift requirements are listed and explained in detail below. The source code of the mathematic constraints is written in this file constraints.py.

Each constraint in the project represents a class. The constraint inherits a base class which is defined as follows:

```
class ConstraintFunction(object):

    def __init__(self, X:Array, **kwargs):
        self._X = X
        self._weight = float(kwargs['weight'])

    def hamiltonian(self) -> Model:
        pass

    def weighted_hamiltonian(self) -> Model:
        return self._weight * self.hamiltonian()

    def evaluate(self, table) -> dict:
        pass
```

In the project, there are two solvers available. One is digital annealer, and the other is simulated annealing solver. The UML digram is shown in Fig. 8.

3.5.1 The Expected Number of Working Days

This requirement could be added to define the number of working days in the shift range. Workers in the shift are expected to have equal working days. The mathematical formulation for N workers and D days shift is as follow:

$$H = \sum_{i}^{N} \left(\sum_{j}^{D} x_{ij} - \alpha \right)^2 ,$$

where α is the expected number of working days, and H is the hamiltonian.

3.5.2 The Expected Number of Workers per Shift

The following constraint defines the expected number of workers in each shift. This constraint is common and used to balance the workforce and the workload each day. The constraint is modeled by summing up the variables on each shift and minus the expected number of workers and, finally, double themselves and make them quadratic. The mathematical formulation is as follows:(The situation is the same as the one used above)

$$H = \sum_{j}^{D} \left(\sum_{i}^{N} -\beta \right)^2 ,$$

where β is the expected number of workers each shift.

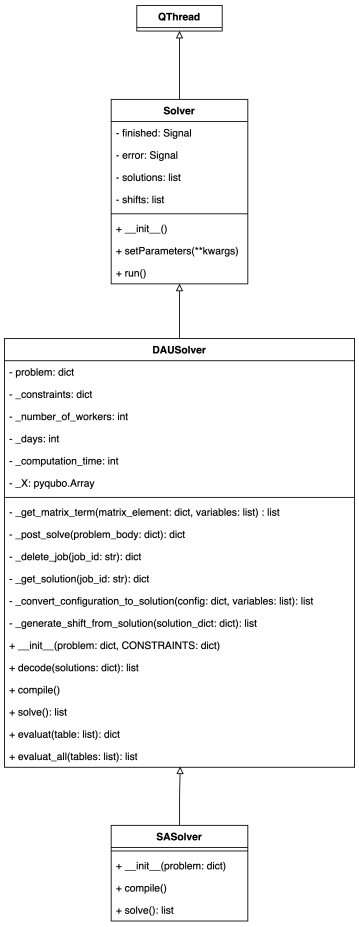

QThread
Solver
- finished: Signal
- error: Signal
- solutions: list
- shifts: list
+ __init__()
+ setParameters(**kwargs)
+ run()
DAUSolver
- problem: dict
- _constraints: dict
- _number_of_workers: int
- _days: int
- _computation_time: int
- _X: pyqubo.Array
- _get_matrix_term(matrix_element: dict, variables: list) : list
- _post_solve(problem_body: dict): dict
- _delete_job(job_id: str): dict
- _get_solution(job_id: str): dict
- _convert_configuration_to_solution(config: dict, variables: list): list
- _generate_shift_from_solution(solution_dict: dict): list
+ __init__(problem: dict, CONSTRAINTS: dict)
+ decode(solutions: dict): list
+ compile()
+ solve(): list
+ evaluat(table: list): dict
+ evaluat_all(tables: list): list
SASolver
+ __init__(problem: dict)
+ compile()
+ solve(): list

Figure 8: UML of solvers

3.5.3 Successive Shift Pair

This constraint is placed to make each worker has at least two consecutive shift. Here's the situation we don't want. The concept is shown below in regular expression form ((0|1)*0|0*)1(0*|0(0|1)*)

To be simple, the mathematical formulation separates the boundary situations, (0|1)*01 and 10(0|1)*, and the general situation, (0|1)*010(0|1)*.

3.5.4 Consecutive 2 days Leave

This is a soft constraint for days off preference. When this constraint is employed, the algorithm would try to arrange the days off together.

The mathematical formulation is as follows:

$$H = \sum_{i}^{N} \sum_{j}^{D-1} (1 - x_{i,j} * x_{i,j+1})$$

.

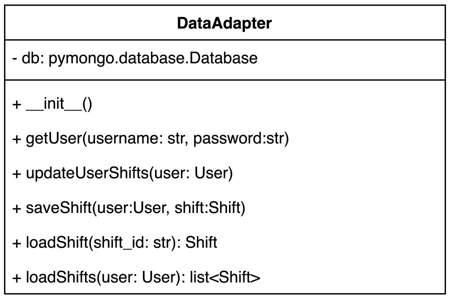

DataAdapter
- db: pymongo.database.Database
+ __init__()
+ getUser(username: str, password:str)
+ updateUserShifts(user: User)
+ saveShift(user:User, shift:Shift)
+ loadShift(shift_id: str): Shift
+ loadShifts(user: User): list<Shift>

Figure 9: Data Adapter UML

3.5.5 No Consecutive 2 Days Leave

This constraint is an opposite version of the above constraint. The mathematical formulation is as follows:

$$H = \sum_{i}^{N} \sum_{j}^{D-1} [(1 - x_{i,j}) * (1 - x_{i,j+1})]$$

.

3.5.6 The Maximum Consecutive Shifts

The constraint set up a limit on the maximum number of consecutive shifts.

This constraint could be modeled by using the supplementary variables. However, this would result in increasing the problem scale. This technique would probably double or triple the number of original variables. Moreover, the solver, Fujitsu's digital annealer, doesn't know the meaning of each variable after compiling the solution into the accepted data; thus, after the annealing process, it would be possible to find a better solution.

The digital annealer computation service offers users to post the inequality.

The inequalities would be an array and append the following inequality.

$$\sum_{j}^{j+\gamma+1} x_{ij} \leq \gamma, \ \forall i \in [1, N], \ j \in [1, D - \gamma - 1]$$

where γ is the maximum consecutive shifts

3.5.7 Minimum N-days leave within 7 days

The constraint is for employees' days off welfare. The employees working in graveyard shifts need to have days off in a range.

4 Runnable Prototype with GUI and Video

https://youtu.be/3ukQoSu0d6I