Shift Schedule Builder Desktop - Prototype

Yu-Chun Lin and Kuei-Ching Yang

October 27, 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 with 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 exist 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.

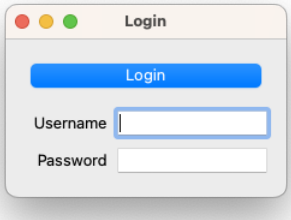

Figure 1: User Interface - Login Dialog

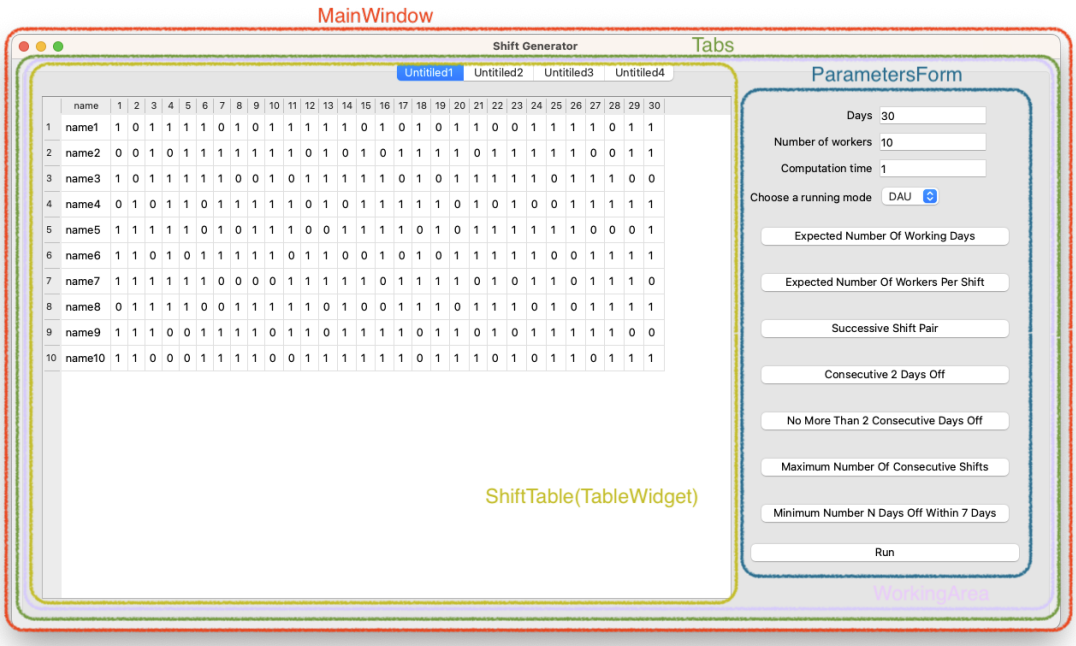

Figure 2: User Interface - Main Window

1.2 Main use cases

1. Users can login by providing their username and password.
2. Users can set up the computation parameters such as the number of days, the number of workers, the computation time, and set up the shift requirements.
3. Users are able to generate a shift by choosing an optimizer.
4. Users are able to export a shift to an excel sheet.

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.

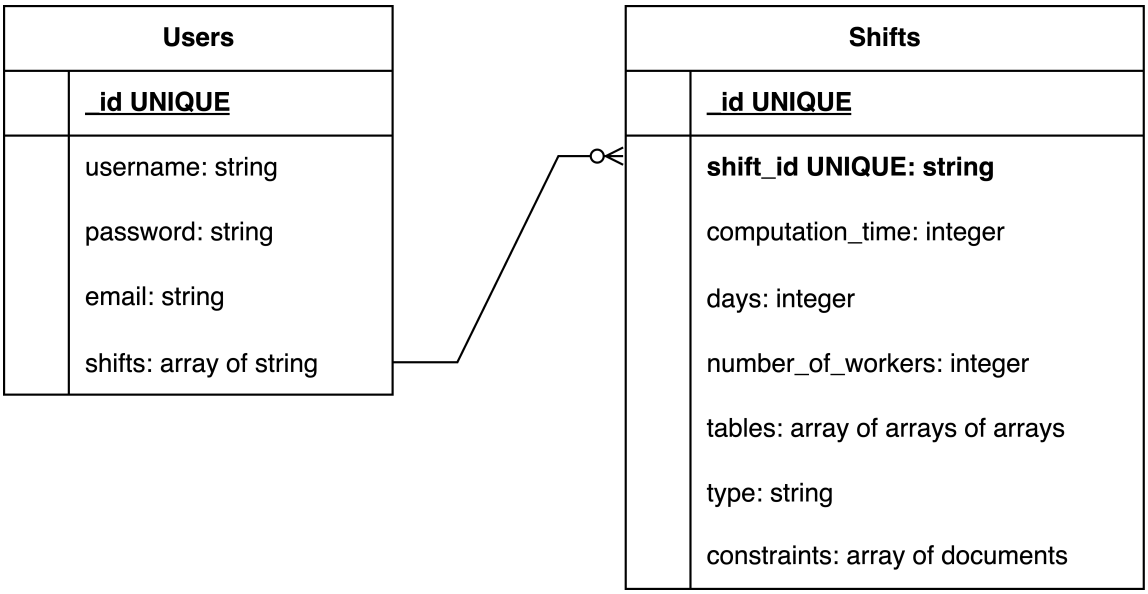

Figure 3: Entity Relationship Diagram

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.

3 Architectural design

The scope of the application includes the user interface, database, and solvers. The user interface collects user's preference of the shift. 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. View/controller will also store the data into the MongoDB. The following section will give details on architecture design for each component.

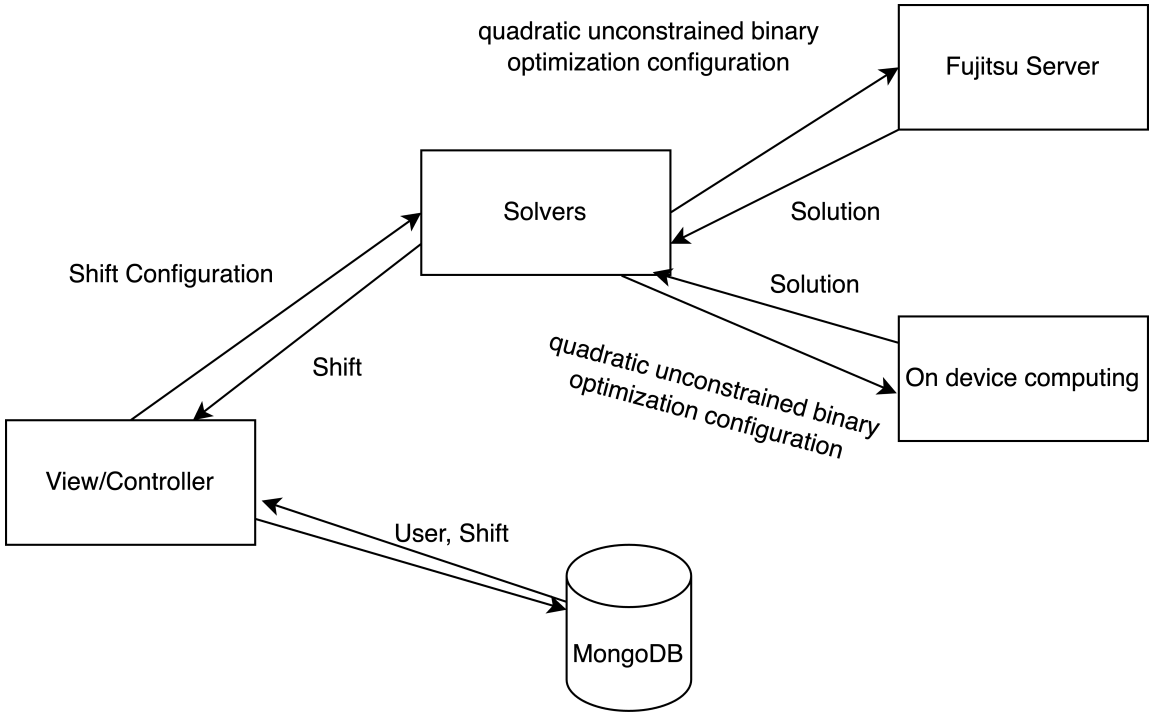

Figure 4: System Architecture

3.1 UI

As shown in Figure 2. The main window is composed of several UI components. The relationship between each component is presented in the Figure 5.

3.2 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.

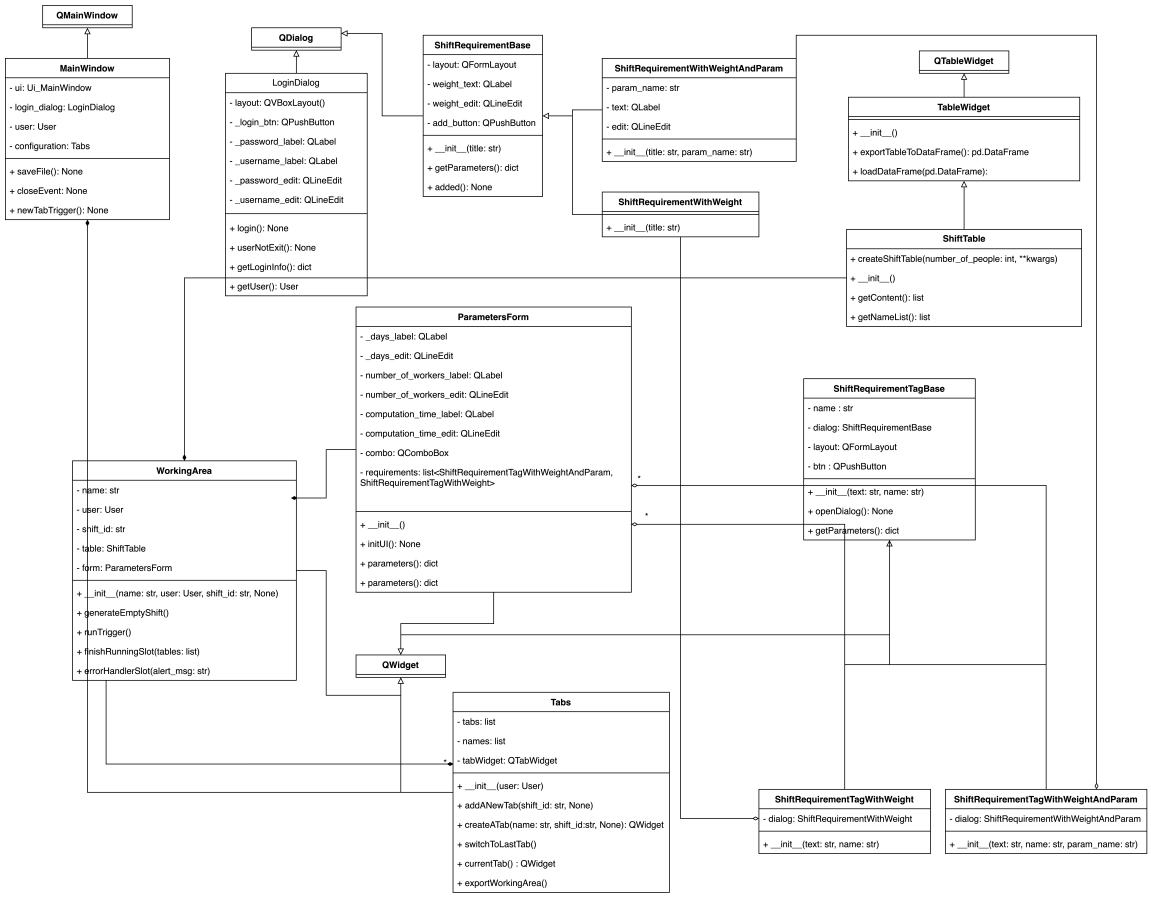

Figure 5: UI's UML

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 Figure 6.

3.2.1 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.

3.2.2 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.

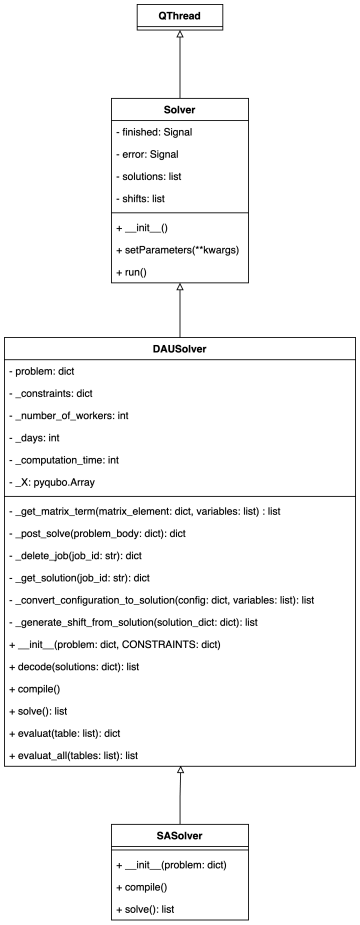

Figure 6: UML of solvers

3.2.3 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, $(0|1)^*010(0|1)^*$.

3.2.4 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:

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

.

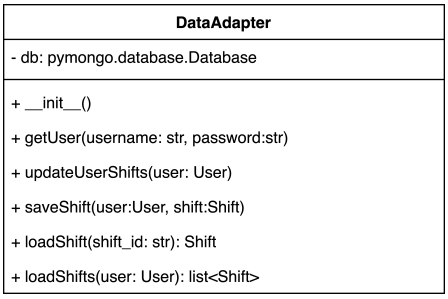

Figure 7: Data Adapter UML

3.2.5 No Consecutive 2 Days Leave

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

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

.

3.2.6 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

3.2.7 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.

3.3 Database

We design two models, User, Shift, and a data adapter to save and load the models between code and the documents in the MongoDB. The scope of the models is only between the controller and the data adapter. The fields in the model and the fields in the documents are nearly the same, respectively for both models. Please refer to Figure 7. for the UML of data adapter class.

4 Runnable Prototype with GUI and Video

https://youtu.be/uJIEUGbZPPI