Employee Scheduling & Roster Generation

Final Project Report

Liam Durkan – C00264405

Supervisor: Paul Barry

Date of Submission: 22/04/2024

i. Abstract

This report documents the development of my Employee Scheduling and Roster Generation system that I undertook for my Final Year Project studying software development at SETU Carlow. The projects goals were to build a system that stores staff availability, time off requests and the required shifts for a roster. The core of the application is a genetic algorithm that begins with a random assignment of employees and converges towards a solution.

The User Interface (UI) is a modern flutter application that is known for its cross-platform support. The API is a Flask API using python and the backend is Firebase for storage & authentication.

ii. Acknowledgements:

I would like to express my gratitude to my supervisor Paul Barry, for his invaluable guidance and mentorship in guiding through the development of this project.

I am also thankful to my family and friends for their support during the highs and lows of this project, their words of encouragement and belief in my capabilities have been gratefully apricated.

iii. Table of Contents

iv. Table of Figures

1. Introduction

The Final Year Project (FYP) is the most important part of final year, it is a demonstration of the abilities and skills I have learned over the past four years studying to become a software developer. This report is an account documenting the problem, objectives, design, and development process that took place to develop my Employee Scheduling & Roster Generation system. I will test and evaluate Iteration 3 to see if it meets the objectives outlined in the functional specification. Further I will reflect on my processes and actions I took in the development cycle of this project, the mistakes I made and what I learned in the process.

This projects aim is to build an Employee Scheduling & Roster management system. The system will generate rosters for a business' required shifts. Employers enter their shifts required for the week, and employees enter their availability, this combined with time off requests assigns an employee to each shift. The application also serves as a database for employees, centralizing employee contact information and storing time off requests.

2. Problem Statement

Every week millions of managers across the globe sit down to at their desk to orchestrate the beating heart of the business, the employees. Rosters come in many forms but at all serve the same purpose, to schedule an employee for a shift in work. Creating a roster can be a mundane task that involves some head scratching, most rosters are created statically using no logic to optimize the problem. Through my experience working in independent hospitality establishments, I found they all lacked

a centralized scheduling system or even procedure, beyond direct contact to a manager. The data comes in many forms some written in a diary, some requests on WhatsApp or just a passing conversation. The manager accumulates all this and produces a schedule that should be correct, if they didn't forget anything.

Given a cafe with 13 employees:

- 13 availability schedules.
- 13 staff that need a day off.
- 13 staff that have at least 1 commitment a week.

The café has 3 shifts a day and each shift could be any of the 13 employees that would be

13 * 13 * 13 = 2197 roster possibilities for one day.

A few points I would like to note:

- The business has no method to store staff availability.
- No method to document time off requests.
- The time spent manually fleshing out a roster.
- The rostering relies on one team member, due to its informal nature if anybody but that team member attempts to produce a roster they are starting from scratch.

The manager would like a system that centralizes all the scheduling and employee information by storing their employee's availability hours, leave & holiday requests and phone numbers. The required shifts for the week can be entered. Employees are assigned to shifts that is within their availability hours.

Requirements and Specifications from Functional Spec Iteration 1

Core Functionality

- **Admin Users:**
  - **Roster Generation:** The core functionality of this application is to generate rosters based on a set of constraints. The application will use an algorithm that takes the required shifts and pairs these with employee availability and time off requests.
    - Admin defines shifts the business requires for the week.
  - **Employee Management:** Employee users can be created, updated and deleted.
    - **Time Off:** Admin can approve requests for time off.
    - **Availability:** Employees can set the weekly hours they can work.
    - **Emergency Requests:** Emergency requests e.g. Illness/bereavement are requests for the current week. The system alerts the admin when it occurs accompanied by a list of employees not scheduled for the request period.
  - **Roster Management:** Rosters can be viewed and edited. Rosters can be published to employees.
    - **CRUD:** Past, current, and future rosters can be viewed & edited.
    - **Export Roster:** Rosters can be exported and published to employees**.**
    - **Republish:** If the current roster is changed during the working week, employees will receive an email notification.
- **Employee User**
  - **View Roster:** Employees can view published rosters.
    - An email notification on new roster release.
  - **Availability:** Employees can input their availability for when they can work.
    - **Availability:** Changes to availability must be approved by an admin.

- **Requests:** Employees can submit requests for time off
- **Time Off:** Employees can submit a request for time off.
- **Swap Shifts:** Employees can request to swap a shift with another employee.
- **Emergency Request:** Employees can alert admin that they cannot work a shift due to an emergency.

Non-Core Functionality

- **Admin Users:**
  - **Admin Dashboard:** Admin users can view a dashboard displaying all related information in one place such as upcoming time off for employees, a notification panel for pending requests.
  - **Email Alerts:** Receive email notifications when requests are submitted from staff.
- **Employee Users:**
  - **Shift Request Handling:** Employees can select a preference for a particular shift. This is a soft constraint; the shift is not guaranteed but will be taken into account.
  - **Temporary availability:** Employees can add temporary availability e.g. midterm breaks.

Accomplishment of Requirements

1. **Roster Generation:** The system can generate rosters using the provided data. The roster generation takes the required shifts from the employer, the list of employees the employer wants included in the roster period. The roster generation method in the API then fetches the approved shifts request and generates the roster.

2. **Employee Management:**

1. Create Employee: Employees can be created from the Sign-Up screen, or an employer can create a new employee internally in the employees tab. Both ways register a new user with firebase and create an employee tied to the business in the firebase database.
2. Retrieve: Employees can be retrieved. Methods to retrieve singular employees take an employee id, a list of employees takes a list of employee ids, and all takes nothing. All returns employee objects.
3. Edit employee: Employees can be edited by an employer or an employee. The employee can edit all employees in the employees tab. The employee can edit their information in the profile tab.
4. Delete Employee: Employees can be deleted by an employer. When an employee is deleted it sets the 'is_deleted' flag to TRUE in the db, then on employee retrieval there is a condition to exclude deleted employees.

3. Roster Management:

1. **Create Roster:** Rosters are generated, they can then be saved to the db after generation at the users discretion.
2. **Retrieve:** Rosters can be retrieved in the rosters tab
3. **Edit Roster:** After a roster is generated, the user can use the dropdowns to selected different employees for the shift.
4. **Delete Roster:** Rosters can be deleted from the rosters tab; this sets the 'is_deleted' bool to false.
5. **Export:** Roster can be exported to a csv file; this file structure allows the employer to manipulate it any way they need.

6. **Publish Roster:** Rosters can be published/republished. Publishing a roster attaches the exported csv file to an email and delivers it using flask mail. Please note that for this product the mail is using a demo SMPT server therefore I can only address emails to my personal email address.

Roster Generation Algorithm

The core feature of the app is that is generates rosters, the process of the roster generation is assigning employees to a shift. I have experience producing rosters for small cafes so my goal was to translate this process into a form that an algorithm could process. I fleshed out on paper what the minimum data required from users would be for a viable solution. My research led me to timetabling solutions and how the genetic algorithms can be implemented to produces classroom schedules. Using these solutions as inspiration I built the algorithm around the datapoints I had and created a fitness solution that would calculate the fitness of a possible solution.

The genetic algorithm used in this application takes:

1. A list of shift objects
2. List of employee objects.
3. List of approved requests.

Using this then preforms the following steps:

1. Randomly assigns an employee to a shift.
2. Calculates the fitness (inverse of conflicts).
3. Preforms the genetic algorithm:
    a. Selection
    b. Mutation
    c. Crossover
4. Returns a solution with zero conflicts.

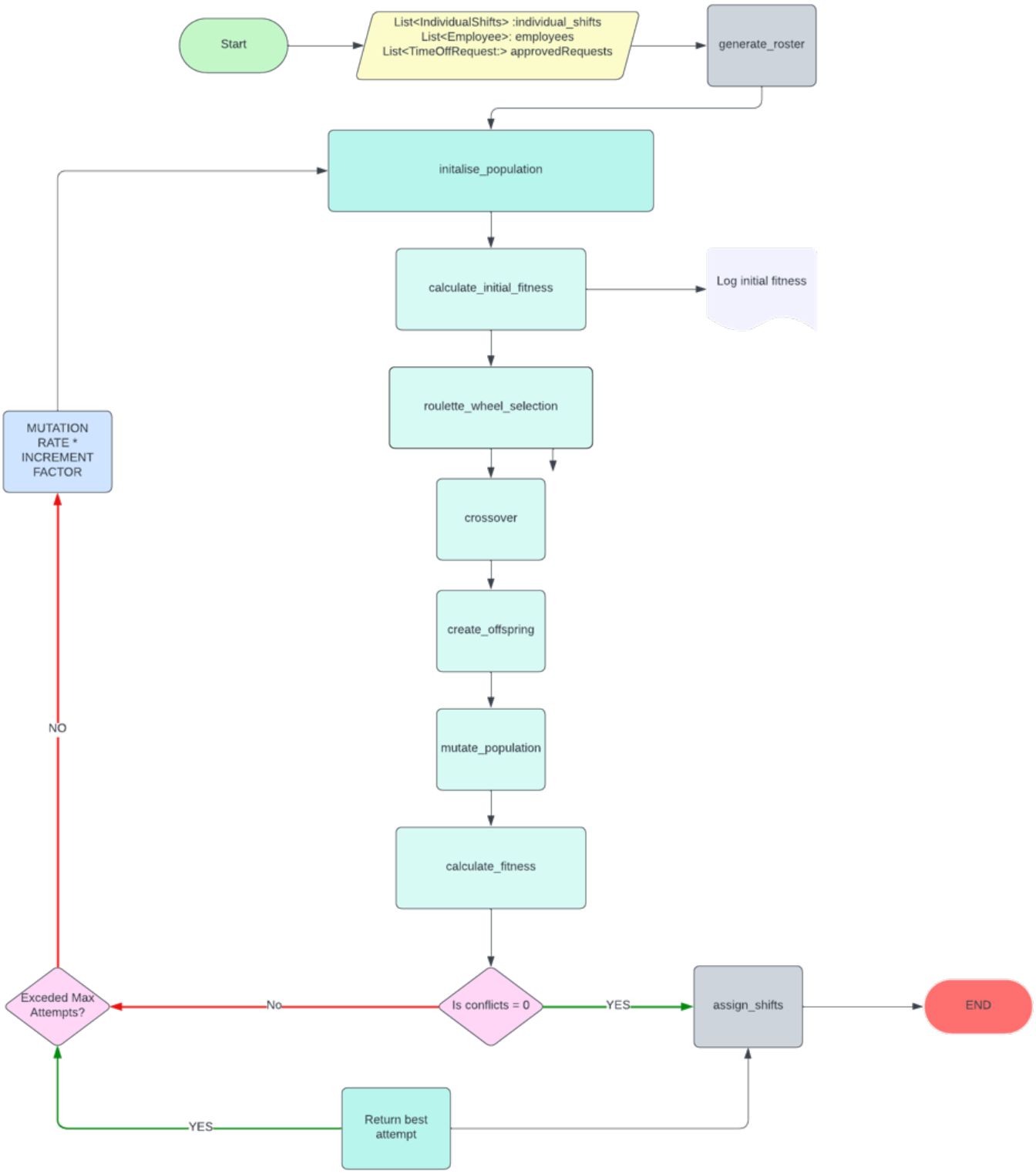
Start
List<IndividualShifts> :individual_shifts
List<Employee>: employees
List<TimeOffRequest:> approvedRequests
generate_roster
initalise_population
calculate_initial_fitness
Log initial fitness
roulette_wheel_selection
crossover
create_offspring
mutate_population
calculate_fitness
Is conflicts = 0
YES
assign_shifts
END
No
Exceded Max Attempts?
NO
MUTATION RATE * INCREMENT FACTOR
YES
Return best attempt

Figure 1

initalise_population(population_size, employees, num_shifts):

- **population size:** The number of chromosomes to generate.
- **employees**: The list of employee objects for this roster
- **num_shifts**: The number if shifts in the roster.
- This method creates a chromosome which is like an array. The chromosome has indexes where x is the number of shifts.

shift 1	shift 2	shift 3	shift 4	shift 5	shift 6	shift 7	shift 8	shift 9	shift 10

- Randomly assigns an employee id to each index.

emp2	emp2	emp3	emp1	emp2	emp1	emp3	emp3	emp1	emp2

- The method will return x chromosomes representing a roster solution where x is the population_size.
- In my program I am returning 10 that progress to the next stage.

calculate_fitness():

- Calculates the fitness of a chromosome, representing the number of conflicts.
- Fitness is calculated by summing the number of conflicts, it is inversely proportional to the total number of conflicts so the lower the conflicts the higher the fitness.

- The conflicts that are penalized are:
    - Employee is available during the shift times.
    - Employee is not working more than their max shifts per week or the global maximum set by employer.
    - Employee is only working one shift per day.
    - Shift is not during approved time off.
    - Recently added; Penalise if employees are unevenly distributed.
- Returns the fitness value and the number of conflicts in the chromosome.

calculate_initial_fitness(initial_population, individual_shifts, employees):

- This is a helper method as calculate_fitness can only take 1 chromosome at a time, so it takes the x chromosomes that were initialized in initialize population and puts them through one by 1.
- Returns a list of x chromosomes with their fitness and conflicts values.

roulette_wheel_selection(population, num_selection):

- Takes a population of chromosomes, each chromosome has a fitness, the probability of a chromosome being picked is directly related to the weight of its fitness.
- Spins a wheel, the higher the fitness the more likely of getting chosen.
- Selects and returns x chromosomes where x is *'num_selection'*.

crossover(parent1,parent2)

- parent1 & parent2: chromosomes chosen in roulette wheel.
- Random number between 1 - chromosome length is generated.
- parent1 & parent2 are split at the crossover point and appended to the other chromosome.

  random = 4

emp2	emp2	emp3	emp1	emp2	emp1	emp3	emp3	emp1	emp2
emp1	emp3	emp3	emp2	emp1	emp3	emp2	emp1	emp3	emp2

emp2	emp2	emp3	emp1	emp1	emp3	emp2	emp1	emp3	emp2
emp1	emp3	emp3	emp2	emp2	emp1	emp3	emp3	emp1	emp2

create_offspring(selected_for_crossover, individual_shifts, employees)

- '*selected_for_crossover*': List of dictionaries representing selected chromosomes crossover.
- '*individual_shifts*': Shift objects representing the shifts available for scheduling.
- '*employees*': Employee objects available for current rostering persion

- This function takes the two selected chromosomes from the roulette wheel, calls the crossover method, calculates the fitness of the returned chromoseomes and returns them as a list of dictionaries.

mutate_population(population, employees, mutation_rate)

This is responsible for increasing random changes to add diversity to the population.

- *population*: It takes the current population of chromosomes as input.
- *employees*: It requires information about the employees, which may be used for generating new genes during mutation.
- *mutation_rate*: The probability of an employee assigned to a shift being swapped for a different employee.
- The loop iterates through each index in the chromosome, if a random probability is less than the '*mutation _rate'* a random employee is swapped in.

algorithm(next_generation, individual_shifts, employees, num_selection, population_size):

- *'next_generation'*: Population of chromosomes representing a shift.
- *'individual_shifts'*: Individual shifts that need to be assigned to employees.
- '*employees*': The list of selected employee objects for the rostering period.
- '*num_selection*': The number of chromosomes selected for crossover in each generation.
- '*population_size*': The size of the population of chromosomes.

This function repeats the selection, crossover, mutation & fitness check, each time creating new possible solutions converging towards 0 conflicts.

assign_shifts(chromosome, individual_shifts, employees):

- 'chromosome': The solution returned from the algorithm with zero conflicts and a fitness of 1.0.
- This method iterates through the chromosome, from the 'employee_id' representing an employee assigned to a shift in the chromosome, it populates the employee attribute if the individual shirt with the employee object.

generate_roster(individual_shifts, employees, approvedRequests)

- 'approved_requests': List of approved request objects
- *individual_shifts'*: Individual shifts that need to be assigned to employees.
- '*employees*': The list of selected employee objects for the rostering period.

This is the main of the roster generation algorithm, it calls all the other methods and keeps track of the runs.

1. A for loop runs a loop x times. For this iteration we have maximum set to 5.
2. Initialise population function
3. Calculate initial fitness function
4. Run the algorithm that converges till conflicts are 0
5. If no result has returned && below '*max_generations'*
6.

5.3 Entity-Relationship Diagram

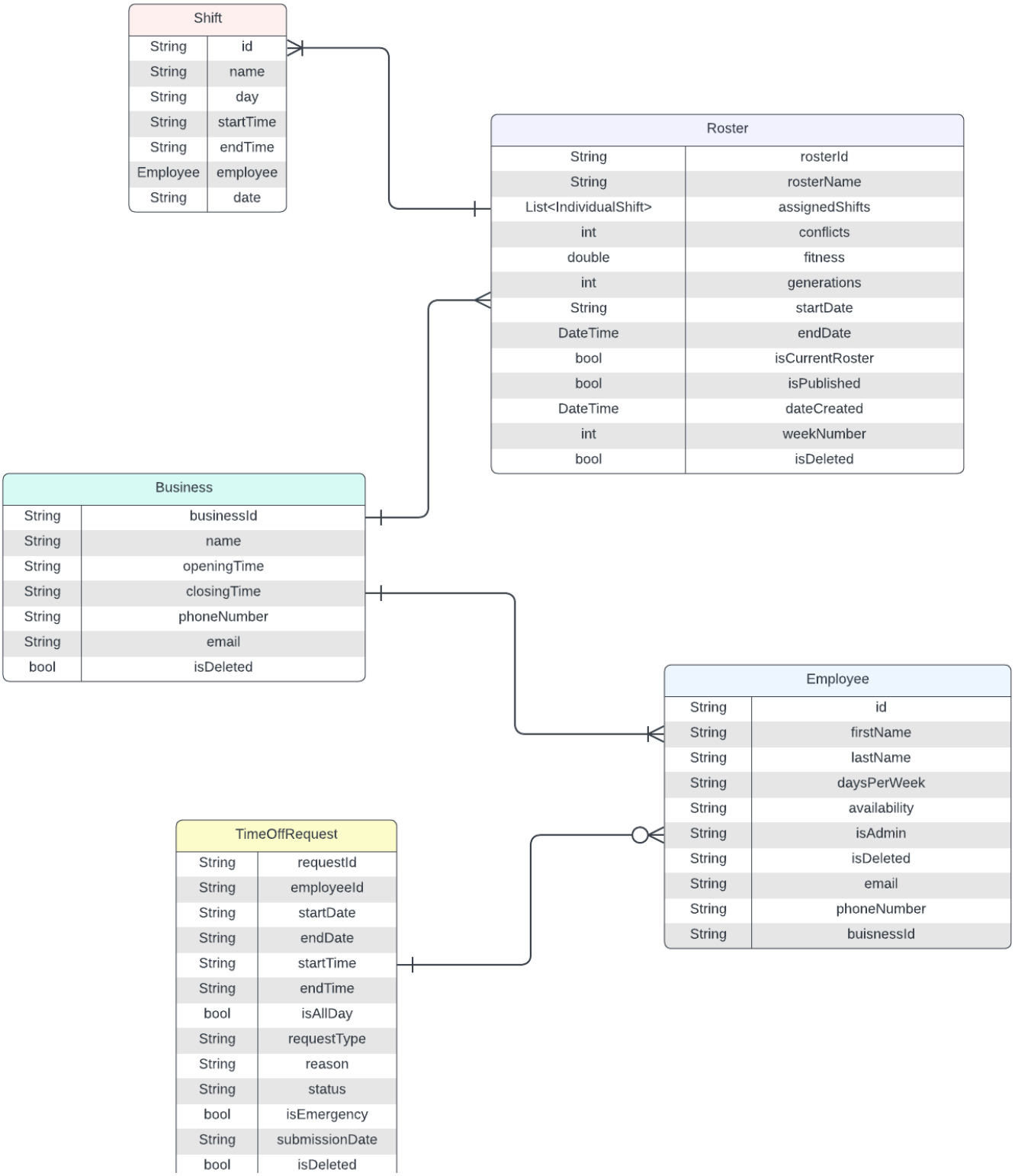

Figure 2

6. Implementation

Front End: The front end of this application is developed using Flutter. While I had prior experience with other front end languages such as react and windows forms the cross-platform support, quick UI and modern interfaces interested me.

API: Implemented using Python3 and Flask, the API serves as the core to my application. The API is currently hosted on Render Web Service. To streamline development & deployment processes the API automatically updates & redeploys when a change is pushed to the GIT repo.

Database: My project is using Firestore as the database for this project, I chose firebase as it has integrate with flutters cross-platform support providing supported access to their systems.

Authentication: Username & Password authenticated is being handled by firebase authentication offering secure and reliable access control.

User & System Requirements

System: Currently the application can be accessed two different ways.

Webapp can be accessed from browser on desktop or on mobile device. IOS app is compiled for devices on IOS 11+

The UI uses a responsive layout widget to determine if the screen width. If the screen width is greater than 650 pixels the desktop UI is displayed, if it is below 650 the mobile UI is displayed

User Requirements:

1. Business user sets up businesses account, enters business opening & closing time.
2. Employee user signs up under a business. Once they enter their availability it can be factored into the rosters.

7. Testing and Validation

- During the development of my project, I used postman to build and test API endpoints. This was an application I used throughout my internship and not only drastically sped up the development process but allowed me to create a series of postman tests to verify the expected output of http request.

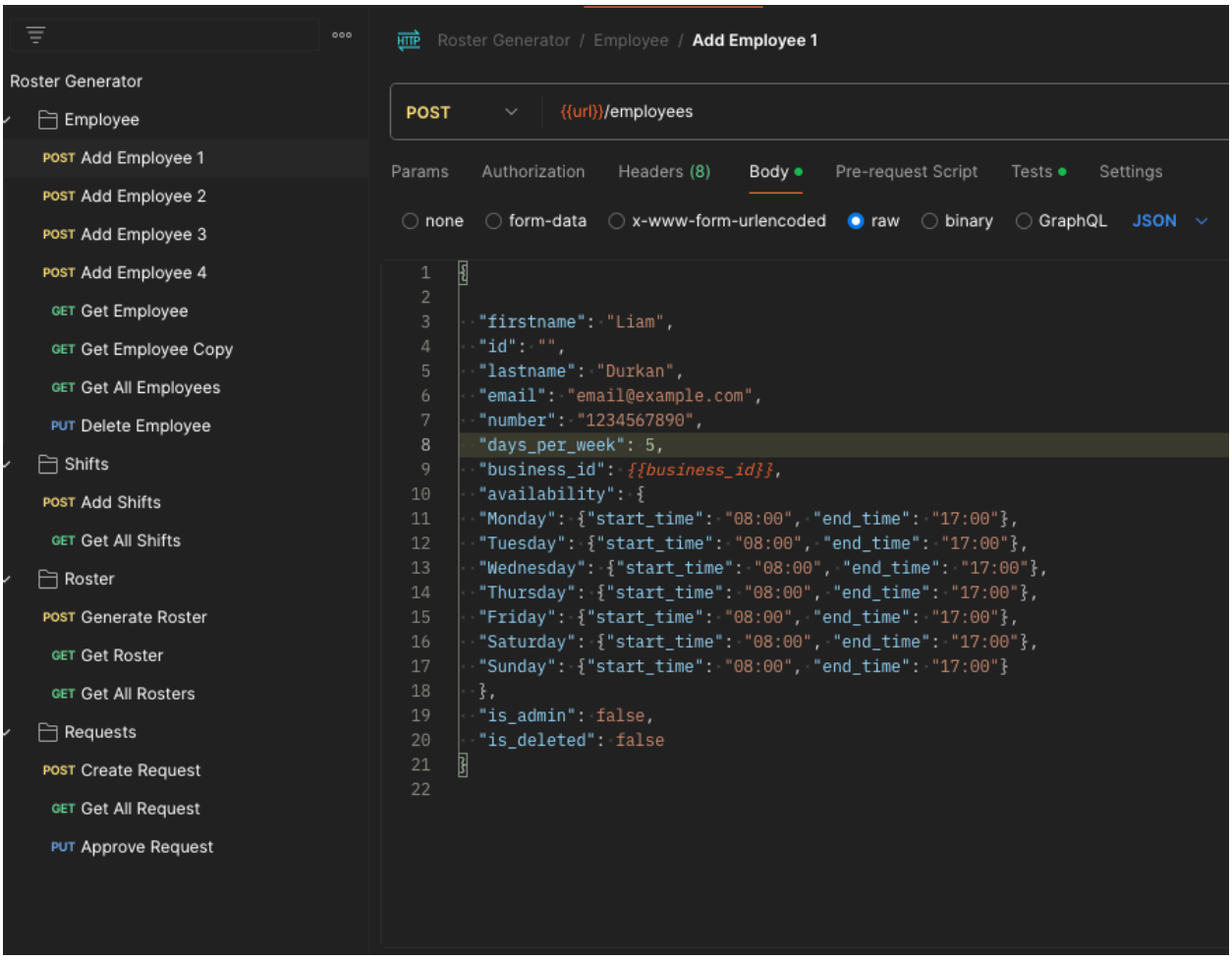

Figure 3

- An important part of this application for me was the UI and how it fells to use. No matter how good the back end is, an employee & employees don't see that part and struggle to see beyond a bad UI. My UI has been tested on web using chrome & on an iPhone.
- To test the roster generation process I created a series of situations setting employees to available and unavailable, days and then generated a roster with these, I check to make sure that the employee was not scheduled on the day. The fitness function on the roster generation is itself a test as it is checking to make sure that a possible solution has zero conflicts.

8. Results

The system has met all the core requirements except one. One of the core requirements was for the employees to be able to make a request to swap a shift with another employee. This requirement came from a selling point that would attract employees to use the application. After a lot of deliberation between adhering to my functional specification I chose the route to deviate and implement a method for employers to change the roster, using dropdowns with employees' names after generation. This new feature then checks if the selected employee is available for that shift, with visual cues if not. This is a prime example of agile methodology in the development lifecycle. As I wrote the spec I had to channel the mindset of what an employer would want from this application. I, the employer, did not know what I wanted from my product and thought that the generation would be sufficient. Sometimes an employer will want to put an employee on a specific shift, this could be for many reasons involving the schedule of the business that day. Therefore, cutting the staff swap shift request in favor of editing a generated roster is a more valuable asset to the overall application, and adding the visual cues that that employee cannot work the shift is a bonus feature by taking advantage of the data we already have in our system.

The roster generation algorithm was initially built in jupter notebook and then refactored into the system. It has undergone a lot of tweaking to try and prevent it getting stuck in a local maximum or minimum where it will not find a possible solution. To counteract this I

have recently introduced a mutation rate that increases as by a factor of 1.2 each time the algorithm runs, which is capped at a maximim of 5. This addition has increased the speed of the algorithm and reduced failed attempts.

9. Conclusion

In conclusion this project has successfully developed a roster generation algorithm that meets the specified objectives. The application offers businesses a centralized system to manager their employees scheduling an rostering needs, saving hours dealing with roster related

Through the project I have achieved the following key findings:

1. Implemented a roster generation algorithm that efficiently assigns shifts to employees by converging towards a solution with 0 conflicts.
2. Developed a modern user-friendly interface that is cross platform, available as a webapp or on IOS.
3. Implemented a solution for employees to submit time off requests and for employers to approve/deny them.

- **Future Work**
    1. Employees can request to swap a shift with another employee.
    2. Calendar on dashboard showing employer upcoming requests.

3. Android app to increase availability across the market.
4. Distribution of employees across shifts customization.
5. Soft constraints such as staff having low priority preferences for days off.
6. Seed future roster generations with previous ones. This will create consistency across rosters.

10. Reflection and Learning

- The FYP is my biggest and proudest accomplishment in my career to date. It is a representation of my capabilities as a developer and has the skillset I have acquired. Many of the skills I have improved on with this project are technical, but the most important skills I take away from this project was the process of development. The development process was such a learning experience. Countless errors, mistakes and wrong decisions got frustrating, some days I made zero progress with code, maybe even less than when I began, but looking back all those hours spent debugging and researching was hours of consuming knowledge, which is invaluable. An example of this is my journey with Flutter. Seven months ago I had never heard of flutter, although I had experience in react the cross-platform support was a major selling point to me. I dived into my final year project and the first step was to learn a new language. There was an initial learning curve to it as expected, my first UI screen was not only clunky to use but terribly coded, as I became more versed in the language, I found myself refactoring perfectly functional screens that resulted in no UI changes, just to improve the structure, reflecting my skills improving.

While I had experience in maintaining C# APIs, this was the first python one I built from the ground up. I thoroughly enjoyed the development of the API and conceptualizing the data flow within the system, it broadened my horizon on how many of our systems today are simply moving and manipulating data, and sparked in interest in me to see what other problems I can solve with my technical knowledge.

This project has been an immense learning experience that has made an impact on my future career choice. I've come to recognize my interests and abilities and want to direct them into back-end development, from this I am taking clarity and a confidence as a developer.