D5

CS 319

2024-25 Fall semester

S3T4 - Code Busters

Umay Dündar 22202573

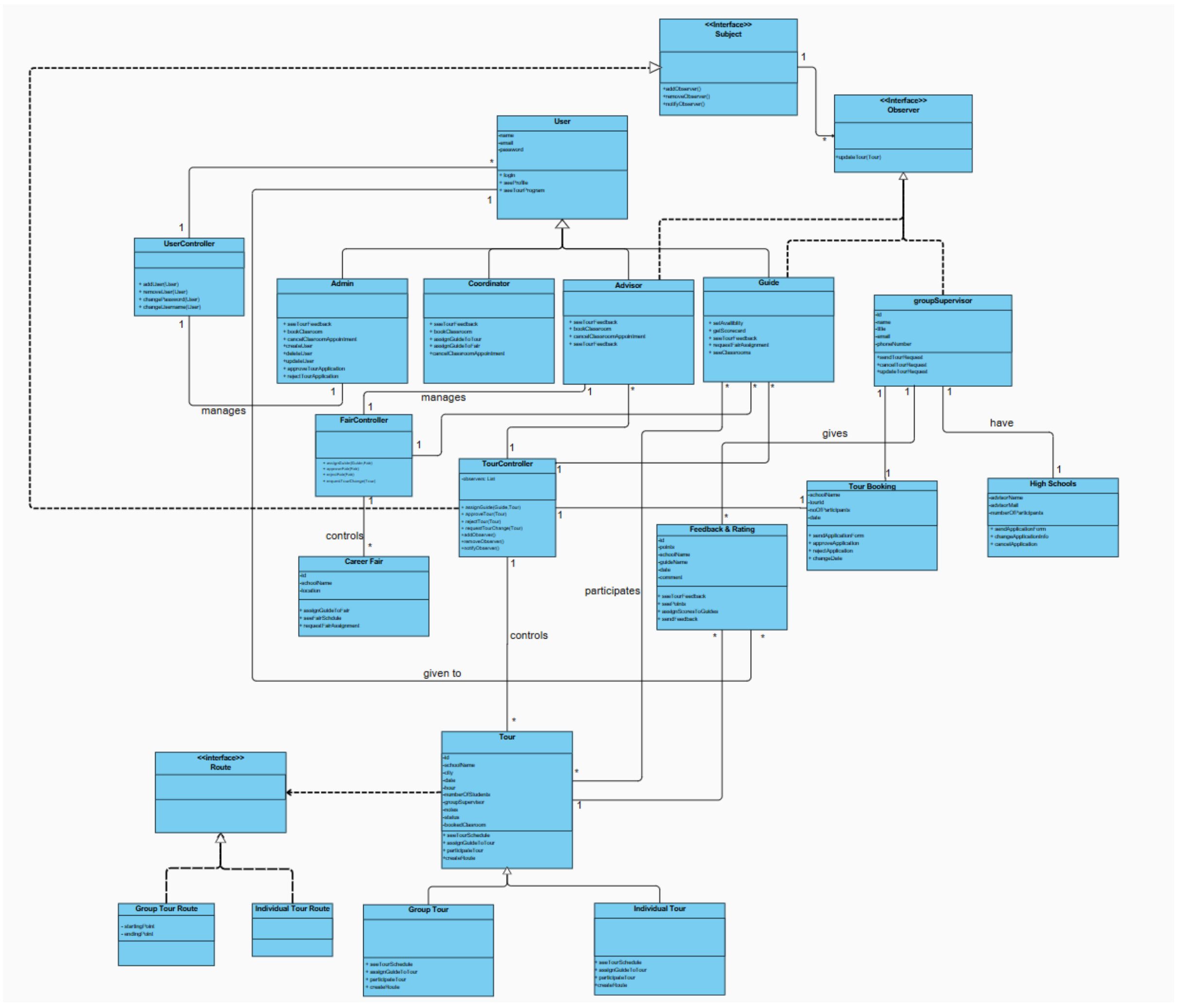
Elif Ercan 22201601

İbrahim Barkın Çınar 22003874

Kemal Onur Özkan 22201820

Contents

1. Updated Class Diagrams



- https://www.canva.com/design/DAGWBaeZScQ/QPvtkSz2dfi0fGnq0Esyg/edit?utm_content=DAGWBaeZScQ&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton

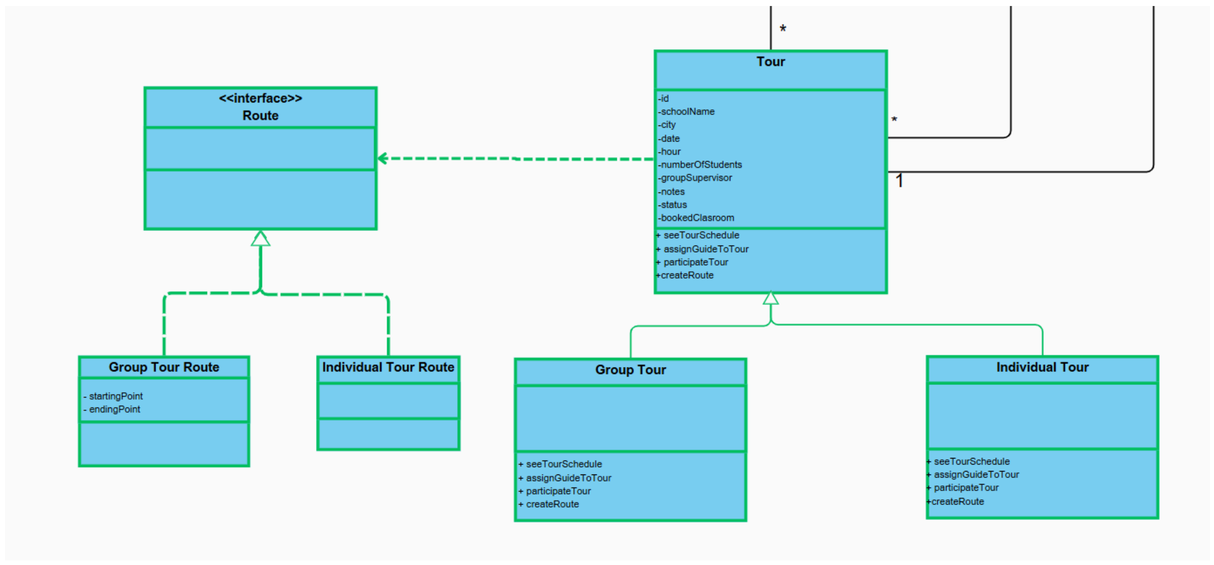
2. Design Patterns

Selected Pattern 1: Factory

Reason For Selection: In our project, we have a lot of subclasses which are derived from parent classes. Because of this, the factory design pattern suits our project. As an example, we have a main user class in our project, which represents all of the users that need to log in to the system in order to access it. Afterwards, this user class is divided into subclasses consisting of admin, coordinator, advisor and guide based on their roles and responsibilities. All of these subclasses contain all of the features of the user class and also contain their own distinct values and abilities.

Design Pattern Diagram:

The asked design pattern's diagram part is highlighted with green in the following diagram.



Design Pattern Implementation Sample:

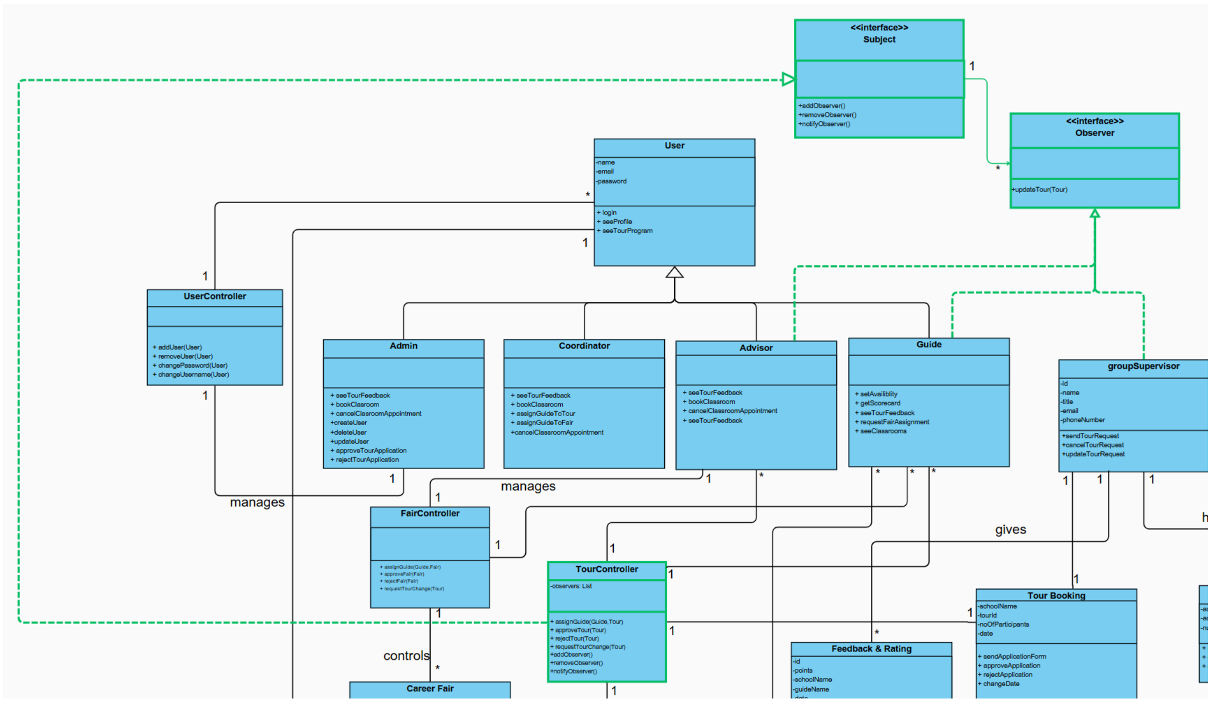
```
class User {
    constructor(name, email, password) {
        if (new.target === User) {
            throw new Error("Cannot instantiate an abstract class.");
        }
        this.name = name;
        this.email = email;
        this.password = password;
    }
}

class Admin extends User {
}

class Coordinator extends User {
}

class Advisor extends User {
}

class Guide extends User {
}

class UserFactory {
    static createUser(role, name, email, password) {
        switch (role.toLowerCase()) {
            case "admin":
                return new Admin(name, email, password);
            case "coordinator":
                return new Coordinator(name, email, password);
            case "advisor":
                return new Advisor(name, email, password);
            case "guide":
                return new Guide(name, email, password);
            default:
                throw new Error(`Invalid role: ${role}`);
        }
    }
}
```

Selected Pattern 2: Observer

Reason For Selection: In our project, the system will notify its users for important events. For instance, during tour changes, the schools are notified. Whenever a tour application is approved, rejected or a date change is requested a notification is sent to the school administrator who applied for the tour. This ensures that the communication process between the high school administrators and the tour office is automatic and requires minimal manual work. To manage these notifications and updates, we are using the Observer design pattern.

Design Pattern Diagram:

The asked design pattern's diagram part is highlighted with green in the following diagram.



Design Pattern Implementation Sample:

```
class Tour {
  constructor(schoolName, city, date, hour) {
    this.schoolName = schoolName;
    this.city = city;
    this.date = date;
    this.hour = hour;
    this.observers = [];
  }

  addObserver(observer) {
    this.observers.push(observer);
  }

  removeObserver(observer) {
    this.observers = this.observers.filter((obs) => obs !== observer);
  }

  notifyObservers(message) {
    this.observers.forEach((observer) => observer.update(message));
  }

  updateDetails(newDetails) {
    Object.assign(this, newDetails); // Update tour details
    this.notifyObservers(
      `Tour '${this.schoolName}' updated: City: ${this.city}, Date: 
${this.date}, Hour: ${this.hour}`
    );
  }

  notifyBeforeTour() {
    this.notifyObservers(
      `Reminder: Tour '${this.schoolName}' is tomorrow at ${this.hour} 
in ${this.city}.`
    );
  }
}


class GroupSupervisor {
  constructor(name, email) {
    this.name = name;
    this.email = email;
  }

  update(message) {
    console.log(`Notification for ${this.name} (${this.email}): 
${message}`);
  }
}
```

```
const supervisor1 = new GroupSupervisor('John Doe', 'john@example.com');
const supervisor2 = new GroupSupervisor('Jane Smith',
'jane@example.com');

const tour1 = new Tour('High School Visit', 'Ankara', '2023-12-01',
'9-11');

tour1.addObserver(supervisor1);
tour1.addObserver(supervisor2);
tour1.updateDetails({ city: 'Istanbul', hour: '11-13:30' });
tour1.notifyBeforeTour();
```