



**UNIVERSITY OF  
LIMERICK**  
**OLLSCOIL LUIMNIGH**

## **Stratos - Quiz Application**

CS5722 - Software Architecture

Lecturer - J.J.Collins

**Group: Team Osmium**

<b>SHRADDHA ZADE</b>	<b>21030111</b>
<b>KOMAL IYER</b>	<b>21025398</b>
<b>SHAGIL CHAUDHARY</b>	<b>21006415</b>
<b>SARTHAK PUNJABI</b>	<b>21183147</b>

# Table of Contents

<b>1. Business Scenario</b>	4
1.1 Overview	4
<b>2. Software lifecycle and Methodologies</b>	4
<b>3. Requirements</b>	6
3.1 Functional Requirements	6
3.2 Non Functional Requirements	7
3.2.1 Security	7
3.2.2 Extensibility	7
3.2.3 Performance	7
3.3 Case tools	7
3.3.1 Microsoft Visual Studio Code	7
3.3.2 diagrams.net/ draw.io	7
3.3.3 SQLite	7
<b>4. Use Case Diagram</b>	8
<b>5. Use Case Description</b>	9
<b>6. Architectural styles and patterns</b>	12
6.1 Interceptor Architectural pattern	13
<b>7. Design Patterns</b>	14
7.1 Adapter Design Pattern	15
7.2 Bridge Pattern	17
7.3 Command Pattern	19
7.4 Factory Method Pattern	21
7.5 Memento Design Pattern	23
7.6 Facade Design Pattern	24
7.7 MVT Design Pattern	26
<b>8. Diagrams</b>	27
8.1 Architecture diagram	27
8.2 Component diagram	27
8.3 Interaction Diagram - Sequence Diagram	28
8.4 Class Diagram	28
<b>9. Added Value</b>	29
9.1 Software Quality Metrics	29
9.1.1 Sonar Lint	29
9.2 Deployment on Heroku	29
9.3 Ajax	31

9.4 Web Scraping	32
9.5 PyLint	32
9.6 . Testing	33
<b>10. Evaluation and Critique</b>	<b>34</b>
<b>11. Team Member Contribution Overview</b>	<b>35</b>
<b>12. References</b>	<b>40</b>

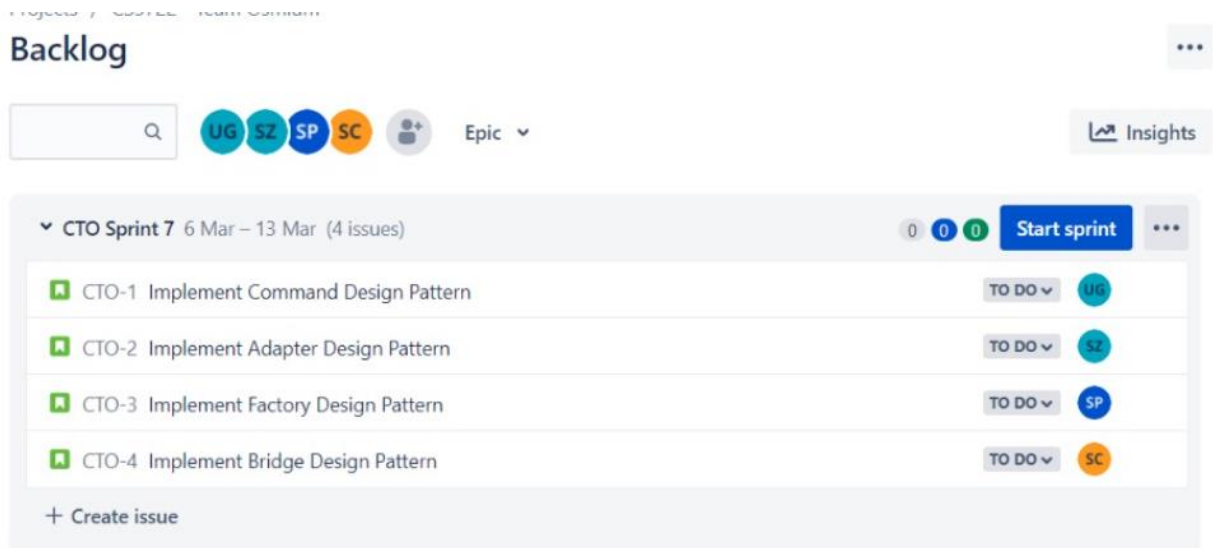
# 1. Business Scenario

## 1.1 Overview

Stratos eliminates the need for manual examinations, which saves time and effort. Apart from that, in the existing system, reviewing the answer sheets after taking the test wastes the examiners' time; however, this application will check the right response and save the examiners' time, allowing them to carry out the examination in a more efficient fashion. It is an application that was created with time restrictions in mind. The quizzes can be accessed by entering the username and password that have been saved in the database. The rules and regulations are displayed before the quiz begins, including a description of the time limit, the amount of questions to be answered, and the scoring methods. The quiz begins with one question and four options based on the category of the quiz. If the response is accurate, the score is increased by one, and there are no penalties for incorrect answers. The final score will be displayed after the submission of the quiz or after the timer has decremented to zero, and the user's score will be updated in the database.


## 2. Software lifecycle and Methodologies

Our Agile software development method is iterative, allowing our small team to concentrate on the most crucial issues. The Agile framework that we used was Jira. It helps with work visualization, which leads to better productivity and faster project development. Daily short standup sessions were done to track progress, and twice-weekly 2-hour meetings were held to review and move forward on issues that were preventing development. Pair programming was used to iterate more quickly during the development process. For communication, Google Meet, Microsoft Teams, and emails were used.




For version control, Git and GitHub were used. Members of the team worked on various modules and pushed code to suitable branches, which were later merged into the master/main branch. Pull requests were raised with every little code change to maintain the master branch of the repo and for continuous integration. Pull requests to transfer the verified code to master were accepted by all team members, which improved code review before deployment.


resolving the error

 sarthakpunjabi committed 28 days ago

Merge pull request #2 from sarthakpunjabi/loginsystem ...


 sarthakpunjabi committed 28 days ago

factory-pattern


 shagilchaudhary committed 29 days ago

Commits on Mar 28, 2022


branch with fuctionality of questions created

 shagilchaudhary committed 29 days ago


changes in binary files

 shagilchaudhary committed 29 days ago

initial setup updated

 shagilchaudhary committed 29 days ago


created initial setup

 shagilchaudhary committed 29 days ago

Login system added

SHRADDHA GANESH ZADE committed 29 days ago

Results section - Initial commit

 komal-iyer committed 29 days ago

## 3. Requirements

### 3.1 Functional Requirements

The key objective is for the administrator to be able to rapidly and simply construct assessments. It's also critical that the quizzes are timed and scores should also be calculated and displayed at the end of each examination so that both the user and the examiner can assess the results.

There are 2 main roles in the application

#### **1. User:**

New users can create an account on the application. Whenever a new user tries to attempt a test, he/ she will be redirected to the login/ signup page. A user can

1. Create a new account or login to existing account
2. Give any test if the user has successfully logged in to the system
3. Asses all the test scores
4. View/ Modify personal details like name, phone number, etc
5. A history of all the quizzes that the user has attempted

#### **2.Admin:**

1. View all tests
2. Add new tests for specific category
3. Remove tests
4. Edit/ Modify tests
5. Login to admin dashboard with a valid username and password

#### **Tools used:**

- VSCode
- GitHub
- SonarLint
- Jira

#### **Design Pattern:**

- Adapter Design Pattern
- Command Design Pattern
- Facade Design Pattern
- Factory Design Pattern
- Builder Design Pattern
- Memento Design Pattern

## Architecture Pattern:

- Interceptor Pattern

## 3.2 Non Functional Requirements

### 3.2.1 Security

JSON web tokens are used to authorize and transmit data between multiple components in the form of an object. JWT is used in the authorization header of a request to visit a certain secured endpoint, and information sent is trusted because it is digitally signed (JWT 2019).

For customer-facing endpoints, SSL certificates are required. SSL certificates provide a layer of encryption by establishing an encrypted channel between the client browser and the application frontend. SSL uses the http protocol to establish a secure connection between the browser and the web server.

### 3.2.2 Extensibility

Extensibility is the ability of the system to add new services to the existing system without affecting the already established system and with minimum effort. The system is designed using the interceptor pattern which allows new services to be added easily and transparently to the system. There won't be a need to restructure the architecture of the existing code if any new service is required to be added to the system.

### 3.2.3 Performance

Heroku is used as the deployment platform for the application. It provides high availability and scalability. The Heroku platform is available across multiple AWS zones which makes it highly available.

## 3.3 Case tools

### 3.3.1 Microsoft Visual Studio Code

The integrated development used was Visual Studio Code. It is a free and optimized code editor for developing and debugging web and cloud applications. The coding was done in python, jquery, html, javascript and css using the django framework.

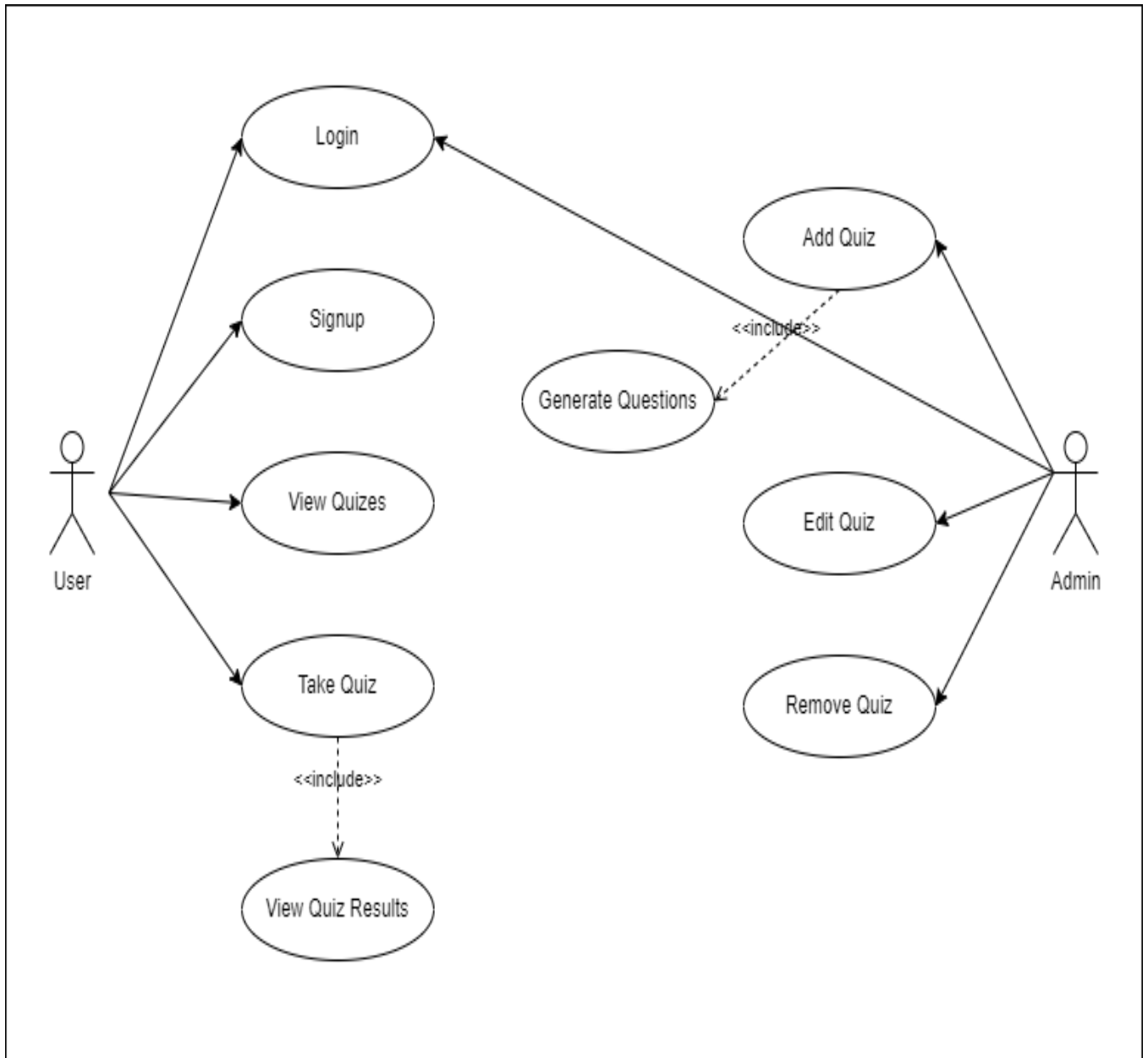
### 3.3.2 [diagrams.net/ draw.io](https://diagrams.net/)

All the diagrams were created using [diagrams.net/ draw.io](https://diagrams.net/). [diagrams.net/draw.io](https://diagrams.net/) is the world's most extensively used browser-based end-user diagramming software and is an open source technology stack for constructing diagramming apps.

### 3.3.3 SQLite

SQLite was used as a database management tool. It implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine

## 4. Use Case Diagram





## 5. Use Case Description

Use Case 1	Login	
Goal in context	User should be able to login to the system	
Scope & Level	System, Primary Task	
Preconditions	Sign up is required	
Success end conditions	The user successfully logs in to the system.	
Failed end conditions	Unable to login	
Primary, Secondary Actors	User, System	
Trigger	Login requested or clicking the login button	
Description	Step	Action
	1	User enters the username and password
	2	User submits the login credentials
	3	System verifies the user credentials and returns if account is successfully verified or not.
Extension	Step	Branching action

	3a	Invalid credentials.  System shows an error "The username or password is wrong."
--	----	--

Use Case 2	Register	
Goal in context	User should be able to register to the system	
Scope & Level	System, Primary Task	
Preconditions	The user must have a valid email id	
Success end conditions	The user is able to successfully create a new account.	
Failed end conditions	User registration failed	
Primary, Secondary Actors	User, System	
Trigger	Sign up requested or user clicks on the sign up button	
Description	Step	Action
	1	User enters details on the new user window.
	2	User submits the details

	3	System verifies the user details and returns if account is created or not
Extension	Step	Branching action
	3a	<p>1.Email id is already taken: System notifies the user</p> <p>2. Mandatory fields are not filled up: Systems highlights the mandatory fields.</p>

Use Case 3	Take the quiz	
Goal in context	User should be able to take a quiz	
Scope & Level	System, Primary Task	
Preconditions	The user must be logged in to the system	
Success end conditions	The user should be able to take the quiz successfully and be displayed the score	
Failed end conditions	The user fails to take the quiz	
Primary, Secondary Actors	User, System	
Trigger	User clicks on the quiz link	
Description	Step	Action
	1	User clicks on the quiz name

	2	User selects answers for all the questions
	3	User clicks on the submit quiz button or the timer ends
Extension	Step	Branching action
	3a	<p>1.Scores are calculated after end of the test</p> <p>2. Scores are displayed at the end of the test.</p>

## 6. Architectural styles and patterns

### 6.1 Interceptor Architectural pattern

The interceptor pattern allows out of band services to be added transparently to the existing system without restructuring the system architecture. In our project we have implemented the interceptor pattern for the logging mechanism.

Whenever any event occurs, for example, add quiz, the dispatcher calls the `interceptor.py` file and creates a context object and passes it to the interceptor. The `context_object` acts like a token and has all the internal information of the current state which will manipulate the interceptor.

Interceptor, Context Object

```
class Interceptor:
    def __init__(self) -> None:
        self.logger = None

    class Concrete_Interceptor:
        def __init__(self) -> None:
            pass

        @abstractmethod
        def get_logger(self):
            pass

class Context_object():
    def __init__(self, get_response):
        self.get_response = get_response
        self.command = []

    def __call__(self, request):
        request.log = Concrete_Interceptor().get_logger()
        response = self.get_response(request)
        # print(response)
        return response
```

The actual logging function is implemented in the concrete interceptor. Whenever there is a requirement for a new “out-of-band” service, a new interceptor can be created without making changes to the existing code.

```

class Concrete_Interceptor(Interceptor,Context_object):
    """
    This creates a logger instance of quiz application.
    It is used for demonstrating interceptor implementation.
    """
    _logger = None

    def __init__(self):
        if not self._logger:
            print("Logging Initialized...")
            self._logger = logging.getLogger("crumbs")
            self._logger.setLevel(logging.DEBUG)
            formatter = logging.Formatter(
                '%(asctime)s \t [%%(levelname)s | %(filename)s:%(lineno)s] > %(message)s'
            )
            now = datetime.datetime.now()
            dir_name = settings.LOG_LOCATION

            if not os.path.isdir(dir_name):
                os.mkdir(dir_name)

            file_handler = logging.FileHandler(
                str(dir_name) + "/log_" + now.strftime("%Y-%m-%d") + ".log"
            )
            file_handler.setFormatter(formatter)
            self._logger.addHandler(file_handler)

    def get_logger(self):
        """
        Getter for returning the interceptor object.
        """
        return self._logger

```

#### Dispatcher:

The dispatcher is implemented in the settings.py file which is the concrete framework which creates the dispatcher. The 'quizes.interceptor.context\_object' in the screenshot below attaches the dispatcher to the concrete interceptor.

```

MIDDLEWARE = DISPATCHER = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'quizes.Interceptor.Context_object',
]

```

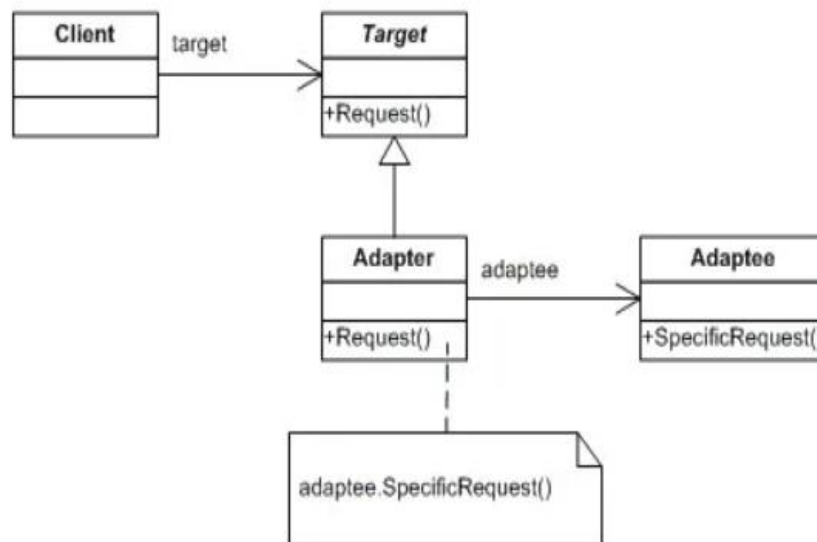
## 7. Design Patterns

### 7.1 Adapter Design Pattern

The GOF defines Adapter pattern as the one that : *Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.*

When interfaces are incompatible and functionality needs to be integrated across them without making any changes to the existing source code, this pattern is utilized.

The key aspect of this pattern is that it promotes programming to interfaces.



#### Benefits:

- It follows the Open-Closed principle. - It does not require changing existing classes or interfaces. An additional class is added which acts as an adapter between the interface and class.
- The changes in software components or core components are avoided.
- Single Responsibility Principle - The functionalities are implemented in separate classes and have fewer side effects.
- The Dependency Inversion Principle is adhered to which ensures compatibility between multiple versions.

## Implementation:

```
8  #Target
9  class GetQuestionsHTMLTarget(ListView, IGetQuestionsHTML):
10     def RenderHTML(request):
11         return render(request, "quizes/add_quiz.html", context={"choices": default_categories})
12
13  #Adapter
14  #Makes JSON's interface compatible with HTML's interface
15  class JSONToHTMLAdapter(GetQuestionsJSONAdaptee):
16     def __init__(self):
17         self.GetQuestionsJSON = GetQuestionsJSONAdaptee()
18
19     def RenderHTML(self):
20         self.GetQuestionsJSON.RenderJSON()
21
22  class QuizListView(ListView):
23     def add_quiz(request):
24         if request.META.get('HTTP_X_REQUESTED_WITH') == 'XMLHttpRequest':
25             ITEM = JSONToHTMLAdapter()
26         else:
27             JSONObj = GetQuestionsJSONAdaptee()
28             ITEM = GetQuestionsHTMLTarget(JSONObj)
29         ITEM.RenderHTML()
```



## 7.2 Bridge Pattern

The Bridge Design Pattern is a Structural Design Pattern that divides a class into two parts: abstraction and implementation, allowing them to be created separately. This encourages a loose coupling between the abstraction of a class and its implementation. This decoupling is achieved by introducing another degree of indirection, namely an interface that serves as a link between your original class and its functionality.

### Benefits:

- It preserves the Open-Closed Principle, which enhances extensibility because client/API-user code relies solely on abstraction, allowing implementation to be changed or augmented at any moment.
- Using PIMPL to implement the Bridge Design Pattern. As in the PIMPL idiom example above, we can hide the implementation details from the client.
- The Bridge Design Pattern is a more sophisticated version of the classic adage "prefer composition over inheritance."
- It's best to avoid coupling an abstraction to its implementation at build time. So that a run-time selection can be made by an implementation.

### Implementation:

```
# Create your models here.
class AbstractClass(BaseUserManager):
    '''Custom Account manager for managing roles'''
    ordering = ('email',)

    def create_user(self, email, first_name, last_name, password, **other_fields):
        '''Defining and creating user function'''
        if not email:
            raise ValueError(_("Email Is Mandatory"))
        email = self.normalize_email(email)
        user = self.model(email=email, first_name=first_name, last_name=last_name, **other_fields)
        user.set_password(password)
        user.save()
        return user

    def create_superuser(self, email, first_name, last_name, password, **other_fields):
        '''Defining and creating superuser function'''
        other_fields.setdefault("is_staff", True)
        other_fields.setdefault("is_superuser", True)
        other_fields.setdefault("is_active", True)
        other_fields.setdefault("is_staff", True)
        return self.create_user(email, first_name, last_name, password, **other_fields)
```

```

class Implementor(AbstractBaseUser, PermissionsMixin):
    '''Adding custom user models and permission'''
    class Types(models.TextChoices):
        '''Creating Proxy types and giving choices'''
        STUDENT = "CUSTOMER", "Customer"
        TEACHER = "AGENT", "Agent"
        OWNER = "OWNER", "Owner"

    type = models.CharField(_('Type'), max_length=50, choices=Types.choices, default=Types.STUDENT)

    username = models.CharField(max_length=150, unique=False)
    email = models.EmailField(_('email address'), unique=True)
    first_name = models.CharField(max_length=150)
    middle_name = models.CharField(max_length=200)
    last_name = models.CharField(max_length=150)
    phone_number = models.CharField(max_length=200)
    subscription = models.BooleanField(default=False)
    subscription_details = models.OneToOneField(Subscription,
        on_delete=models.CASCADE,
        null=True, blank=True,
    )
    is_staff = models.BooleanField(default=False)
    is_active = models.BooleanField(default=True)
    level = models.CharField(max_length=20, choices=AGENT_LEVEL, default="GOLD")
    # booking_history is not needed, as we can get this details by querying room details
    objects = AbstractClass()
    USERNAME_FIELD = "email"
    EMAIL_FIELD = "email"
    REQUIRED_FIELDS = ["first_name", "last_name"]

```

```

class ConcreteImplementor1(models.Manager):
    '''this is the manager for customer type role'''
    def get_queryset(self, *args, **kwargs):
        '''Returning querysets of proxy model customer'''
        return super().get_queryset(*args, **kwargs).filter(type=Implementor.Types.STUDENT)

    def save(self, *args, **kwargs):
        '''Overriding the save function '''
        if not self.pk:
            self.type = Implementor.Types.STUDENT
        return super().save(*args, **kwargs)

class ConcreteImplementor2(models.Manager):
    '''this is the manager for owner type role'''
    def get_queryset(self, *args, **kwargs):
        '''Returning querysets of proxy model Owner'''
        return super().get_queryset(*args, **kwargs).filter(type=Implementor.Types.TEACHER)

    def save(self, *args, **kwargs):
        '''Overriding the save function '''
        if not self.pk:
            self.type = Implementor.Types.TEACHER
        return super().save(*args, **kwargs)

class ConcreteImplementor3(models.Manager):
    '''this is the manager for agent type role'''
    def get_queryset(self, *args, **kwargs):
        '''Returning querysets of proxy model agent'''
        return super().get_queryset(*args, **kwargs).filter(type=Implementor.Types.AGENT)

    def save(self, *args, **kwargs):
        '''Overriding the save function '''
        if not self.pk:
            self.type = Implementor.Types.AGENT

```

## 7.3 Command Pattern

The Command design pattern falls under the behavioral design pattern category. In this pattern, the request is wrapped under an object as a command and passed to an Invoker object. The Invoker object searches for an object that can handle this command and sends the command to that object, which then executes it. These appropriate objects are called Receivers.

The Intent of Command Design pattern is: *To decouple the sender & receiver by creating a separate object for a set of operations.*

### **Benefits:**

- A request is encapsulated as an object.
- The Invoker has no knowledge of the Receiver or the specifics of the commands.
- The Invoker has no knowledge of the Receiver or the specifics of the commands.
- These commands are specified/implemented by the Receiver via an abstract command interface.
- As a result, it promotes the principles of "Hiding Information" and "Design by Contract."
- New commands can be added easily.
- Existing commands can be modified without disturbing others. Therefore, it promotes "Open-closed principle"

### **Application in project:**

The Command Design pattern was implemented in our project to save questions while creating a new quiz and the results of a quiz.

We also implement Undo operation to re-add the questions in the list and also provide a re-test feature to a user. The Save and Undo operations are implemented in the ConcreteCommand class and the operations are invoked by Invoker and Receiver classes respectively.

## Implementation:

```
1 //Invoker
2 class Invoker {
3     action(type) {
4         var receiver = new action();
5         receiver.action(type);
6     }
7 }
8
```

```
//Concrete Command
class ConcreteCommand {
    SetCommand(command) {
        this.command = command;
    }

    execute(data) {
        const amount = document.getElementById('num_of_question')
        const difficulty = document.querySelector('#difficult')
        const choice = document.querySelector('#cho')
        const data = {}
        const csrf = document.getElementsByName('csrfmiddlewaretoken')
        data['csrfmiddlewaretoken'] = csrf[0].value
        data['amount'] = amount.value
        data['difficulty'] = difficulty.value
        data['choice'] = choice.value
        data['type'] = "multiple"

        $.ajax({
            type: 'POST',
            url: `${url}`,
            data: data,
            success: function (response) {
                addForm.classList.add('not-visible')
                quiForm.classList.remove('not-visible')
                modalBtn.classList.remove('not-visible')
                saveBtn.classList.remove('not-visible')
                goibibo.forEach(goib => {
                    goib.classList.remove('not-visible')
                })

                const data = response.data
                maindata = data
                showData(data)
            }
        })
    }
}
```

## 7.4 Factory Method Pattern

As per the definition in Gangs of Four, *The Factory Design Pattern defines an interface for creating an object, but you let the subclasses decide which class to instantiate. Factory method lets a class defer instantiation to subclasses*

The Factory Design pattern allows to separate object construction from the rest of the code. The rest of the code is basically the code that uses the actual object.

### Benefits:

- It allows you to make an object from one of several different classes.
- It provides a standardized interface for interacting with the newly formed object.
- It allows you to keep the creation logic hidden (isolated) from the client.

### Implementation:

```
class PaymentFactory:
    """
    Factory class for determining payment methods.
    """
    def __init__(self):
        pass

    @staticmethod
    def choose_operator(payment_obj):
        """
        Method for selecting the concrete function for payment.
        """
        target_class = payment_obj.get("type") + "Mode"
        return getattr(payment_modes, target_class)(payment_obj)

class ValidatorFactory:
    """
    Factory class for determining Validation methods.
    """
    def __init__(self):
        pass

    @staticmethod
    def validate(request_details):
        """
        Method for selecting the concrete function for validating the request data.
        """
        method_list = [
            method
            for method in dir(validator.Validators)
        ]
```

```

class AgentCommissionFactory:
    """
    Factory class for calculating Agent commission.
    """
    def __init__(self):
        pass

    @staticmethod
    def calculate_commission(agent_level, rooms):
        """
        Method for selecting the concrete function for calculating commission.
        """
        target_state = agent_level.title() + "State"
        # Check if the agent is a Platinum Agent depending on number of room booked
        if agent_level != "Platinum" and len(rooms) > 10:
            target_state = "PlatinumState"
        agent = state_pattern.AgentState()
        state = getattr(state_pattern, target_state)()
        agent.set_state(state)
        return agent.calculate_commission(rooms)

```

```

class DiscountFactory:
    """
    Factory class for calculating discount on rental properties.
    """
    def __init__(self):
        pass

    @staticmethod
    def add_discount(actual_price, age):
        """
        Method for selecting the concrete function for calculating discount.
        """

```

## 7.5 Memento Design Pattern

The Memento design pattern is a software pattern for reverting an object to its earlier state. It's a part of the behavioral design pattern, which is all about algorithms and assigning duties to objects.

The intent of this pattern is : *To not violate the encapsulation principle by not exposing information outside the desired objects.*

### Benefits:

- By avoiding exposing information that should only be managed by an originator, Memento respects the boundaries of encapsulation. However, as we showed in our coding example, that should be saved outside the originator. That was saved in the caretaker object.
- By allowing the caretaker to maintain the list of states, the memento design pattern allows us to keep our client-side code simple.
- Maintaining cohesiveness is easier when the preserved state is kept separate from the key object.
- It makes it simple to add recovery and undo functionality to an application.

### Implementation:

You, 2 minutes ago | 1 author (You)

```
class Caretaker:
    def __init__(self):
        self.state_history = []
        self.restore = self

    def setState(self, obj):
        self.state_history.append(obj)

    def undo(self, obj):
        self.state_history.pop()
        self.restore = obj
```

You, 2 minutes ago | 1 author (You)

```
class Originator(Caretaker):
    def __init__(self, data_, *args):
        super().__init__(*args)
        self.data_ = data_

    def adding_quiz(self):
        try:
            access = Quiz.objects.get(name=f"{self.data_.get('meta[nameofquiz]')[0]}")
            return access

        except:
            Q, create = Quiz.objects.get_or_create(
                name=f"{self.data_.get('meta[nameofquiz]')[0]}",
                topic = f"{self.data_.get('meta[topic]')[0]}",
                number_of_questions = f"{self.data_.get('meta[numberofquestion]')[0]}",
                time = f"{self.data_.get('meta[time]')[0]}",
                required_score_to_pass = f"{self.data_.get('meta[reqscoretopass]')[0]}",
```

```
)
if create:
    for i in range(int(Q.number_of_questions)):
        print(html.unescape(f"{self.data_.get(f'data[{i}][question]')[0]}"))
        que = Question.objects.get_or_create(
            text = html.unescape(f"{self.data_.get(f'data[{i}][question]')[0]}"),
            quiz_id = Q.id
        )
        print(html.unescape(f"{self.data_.get(f'data[{i}][correct_answer]')[0]}"))
        ans = Answer.objects.get_or_create(
            text = html.unescape(f"{self.data_.get(f'data[{i}][correct_answer]')[0]}"),
            correct = True,
            question_id = que[0].id
        )
        ans1 = Answer.objects.get_or_create(text = html.unescape(f"{self.data_.get(f'data[{i}][incorrect_answer1]')[0]}"))
        ans2 = Answer.objects.get_or_create(text = html.unescape(f"{self.data_.get(f'data[{i}][incorrect_answer2]')[0]}"))
        ans3 = Answer.objects.get_or_create(text = html.unescape(f"{self.data_.get(f'data[{i}][incorrect_answer3]')[0]}"))

You, 2 minutes ago | 1 author (You)
class Memento(Caretaker):
    def __init__(self, data):
        self.data = data

    def rem(self):
        print("Entered into remove state")
        data_ = dict(self.data.lists())
        data_.pop('csrfmiddlewaretoken')
        Quiz.objects.filter(id=data_['pk'])[0].delete()
```

## 7.6 Facade Design Pattern

The Facade Pattern is a structural design pattern that seeks to hide the system's intricacies while providing a user interface through which the customer may simply use the system. In other words, it has a single class that delegates calls to other classes and provides a simplified approach to the client.

### Benefits:

- Improve the readability and usefulness of a software library by providing a single simple API that hides interactions with more sophisticated components.
- Give more generic functions a context-specific interface.
- Serve as a springboard for a more comprehensive refactoring of monolithic or tightly connected systems in favor of loosely coupled programming



## Implementation:

```
class Questions:
    def GetQuestions(self,request,data):
        questions = []
        data = request.POST
        data_ = dict(data.lists())
        data_.pop('csrfmiddlewaretoken')
        print(data_)

        for k in data_.keys():
            question = Question.objects.get(text=k)
            print(question)
            questions.append(question)
        return questions

#Subsystem 3
You, 16 seconds ago | 1 author (You)
class Answers:
    def GetAnswers(self,request,questions):
        for q in questions:
            a_selected = request.POST.get(q.text)
            print('a_select',a_selected)
            if a_selected != "":
                question_answer = Answer.objects.filter(question=q)
                for a in question_answer:
                    if a_selected == a.text:
                        print("Done")
                return q
```

```
You, 16 seconds ago | 1 author (You)
class Operator:
    def __init__(self):
        self.Creating = Score()
        self.Generating = Questions()
        self.delivering = Answers()

    def completeOrder(self):
        self.Creating.GetScore()
```

## 7.7 MVT Design Pattern

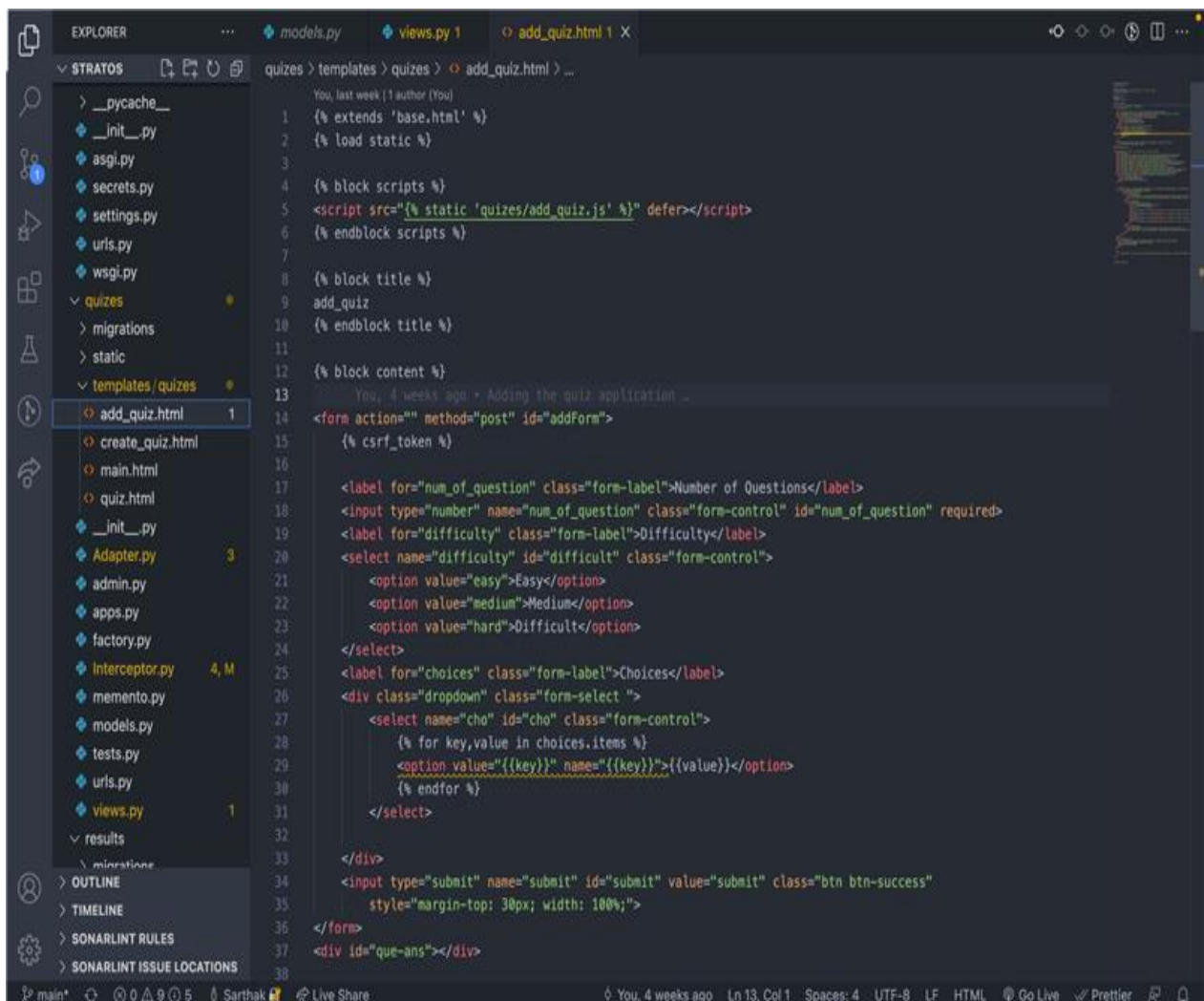
The MVT (Model View Template) is a software design pattern. It is a collection of three important components: Model View and Template. The Model helps to handle databases. It is a data access layer which handles the data.

The Template is a presentation layer which handles the User Interface part completely. The View is used to execute the business logic and interact with a model to carry data and render a template.

### Benefits:

1. Faster Development Process:
2. Ability To Provide Multiple Views:
3. Support For Asynchronous Technique
4. The Modification Does Not Affect The Entire Model

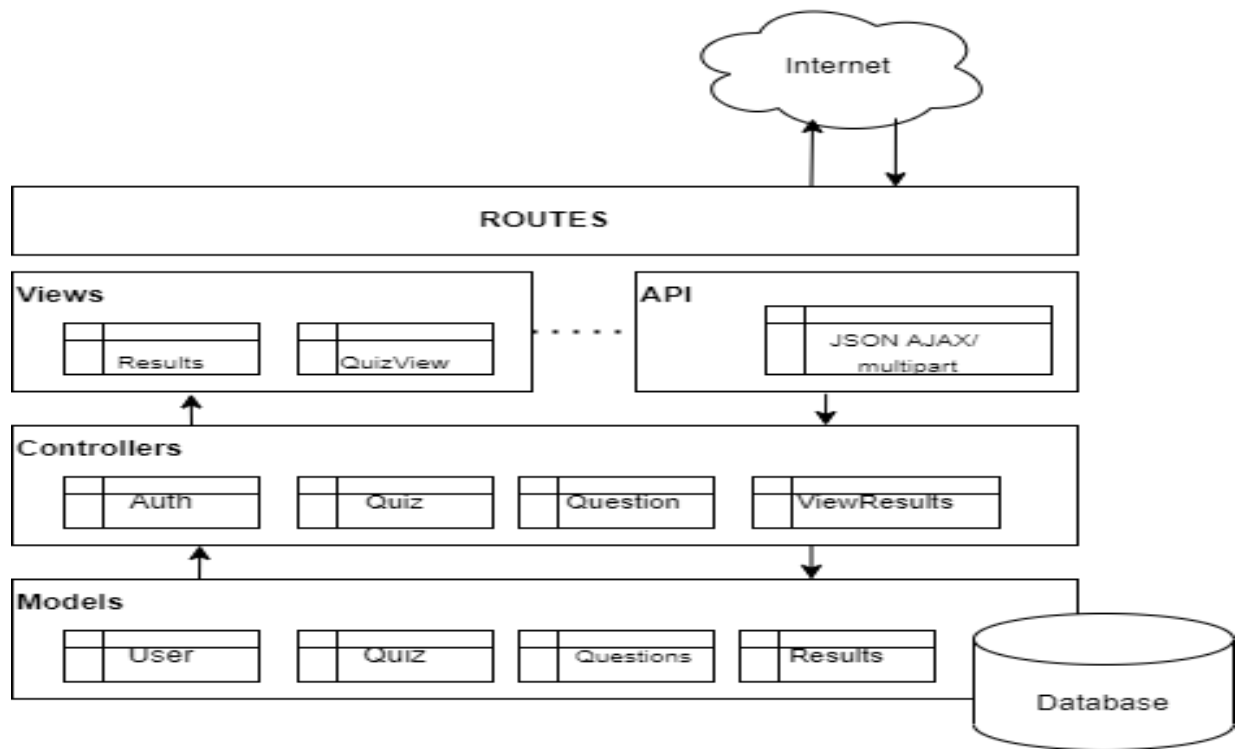
### Implementation:



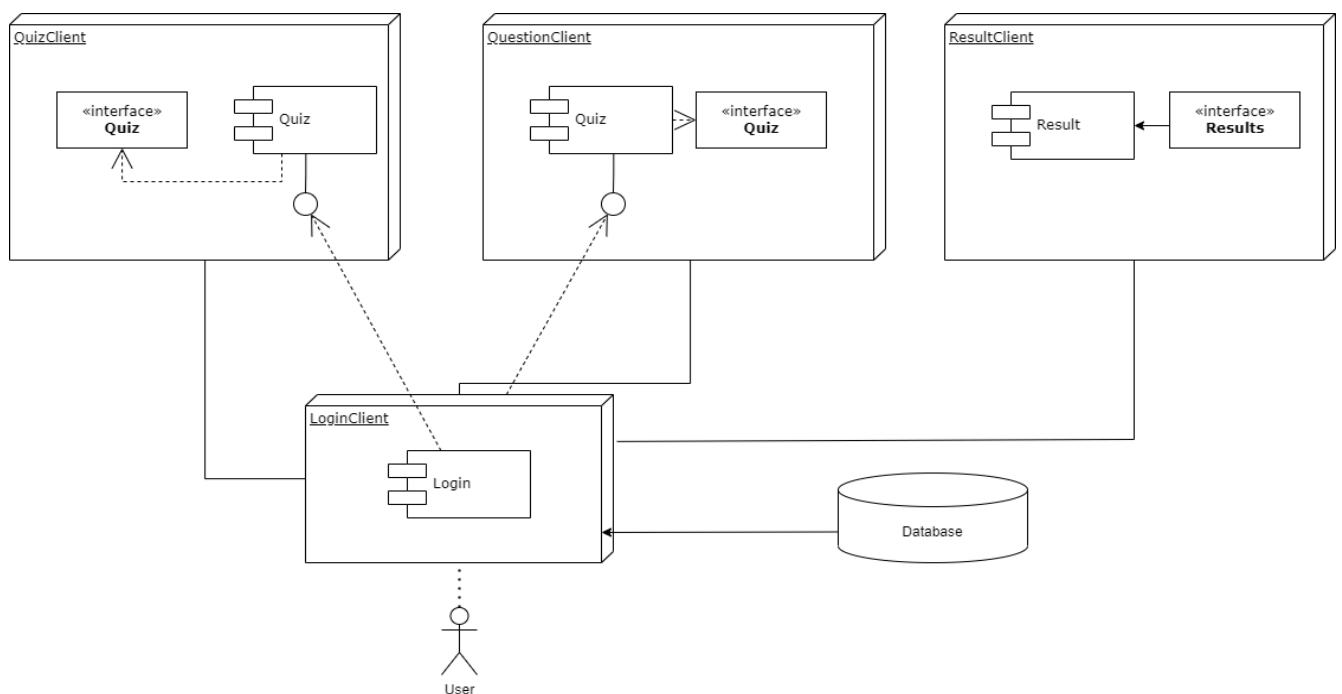
```
1 {% extends 'base.html' %}
2 {% load static %}
3
4 {% block scripts %}
5 <script src="{% static 'quizzes/add_quiz.js' %}" defer></script>
6 {% endblock scripts %}
7
8 {% block title %}
9 add_quiz
10 {% endblock title %}
11
12 {% block content %}
13 You, 4 weeks ago - Adding the quiz application ...
14 <form action="" method="post" id="addForm">
15     {% csrf_token %}
16
17     <label for="num_of_question" class="form-label">Number of Questions</label>
18     <input type="number" name="num_of_question" class="form-control" id="num_of_question" required>
19     <label for="difficulty" class="form-label">Difficulty</label>
20     <select name="difficulty" id="difficult" class="form-control">
21         <option value="easy">Easy</option>
22         <option value="medium">Medium</option>
23         <option value="hard">Difficult</option>
24     </select>
25     <label for="choices" class="form-label">Choices</label>
26     <div class="dropdown" class="form-select">
27         <select name="cho" id="cho" class="form-control">
28             {% for key,value in choices.items %}
29                 <option value="{{key}}" name="{{key}}">{{value}}</option>
30             {% endfor %}
31         </select>
32     </div>
33
34     <input type="submit" name="submit" id="submit" value="submit" class="btn btn-success"
35         style="margin-top: 30px; width: 100%;">
36 </form>
37 <div id="que-ans"></div>
38
```

## 8. Diagrams

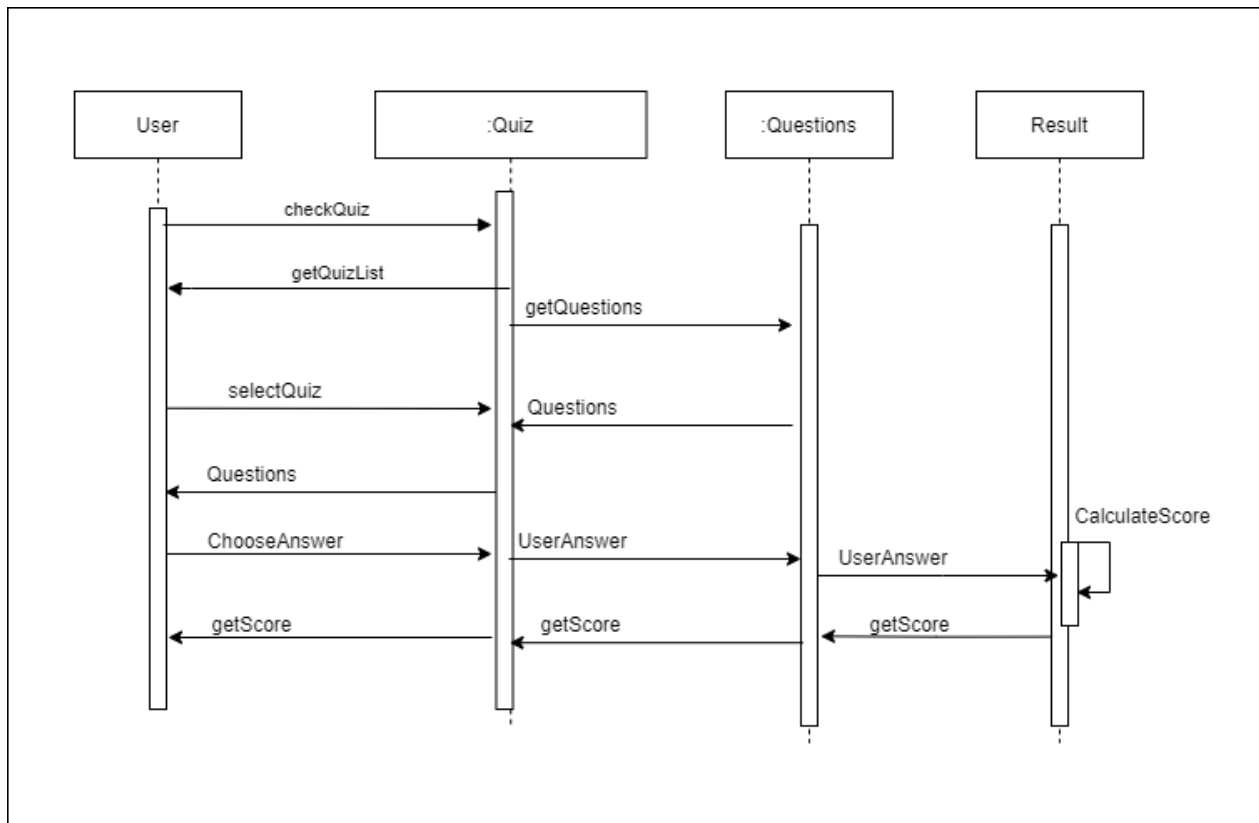
### 8.1 Architecture diagram



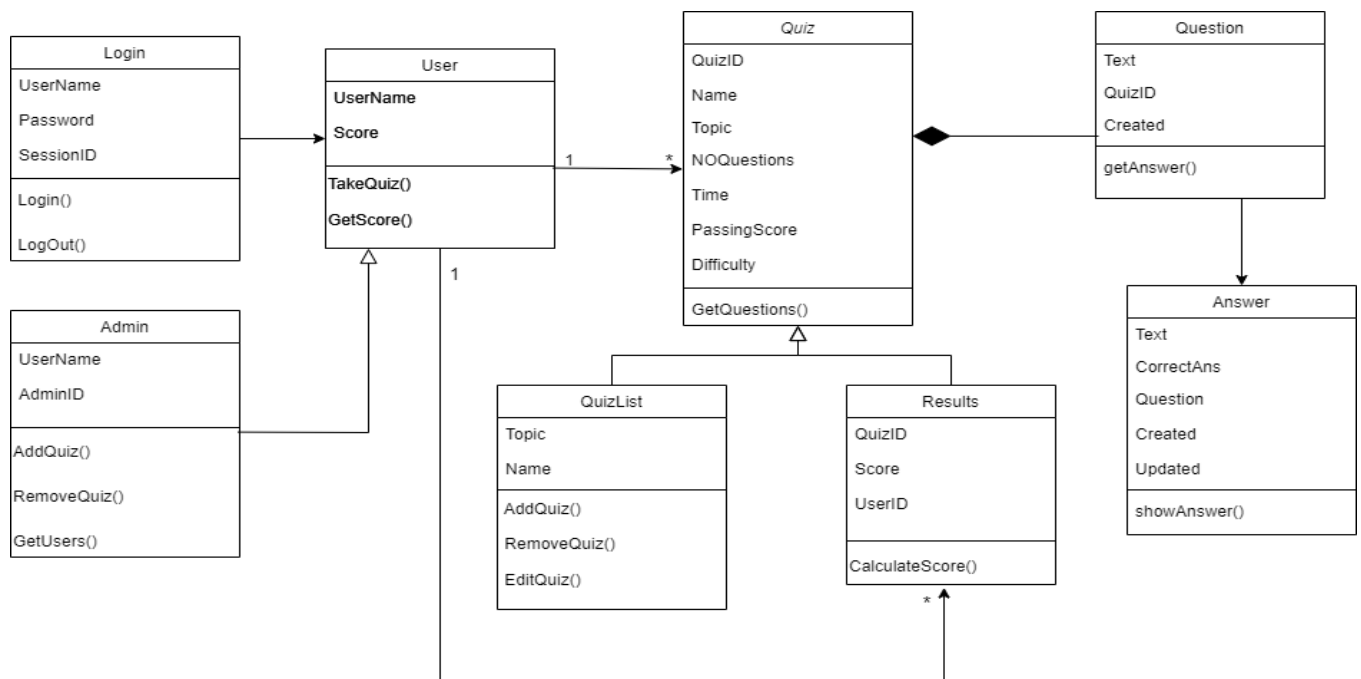
### 8.2 Component diagram



## 8.3 Interaction Diagram - Sequence Diagram



## 8.4 Class Diagram



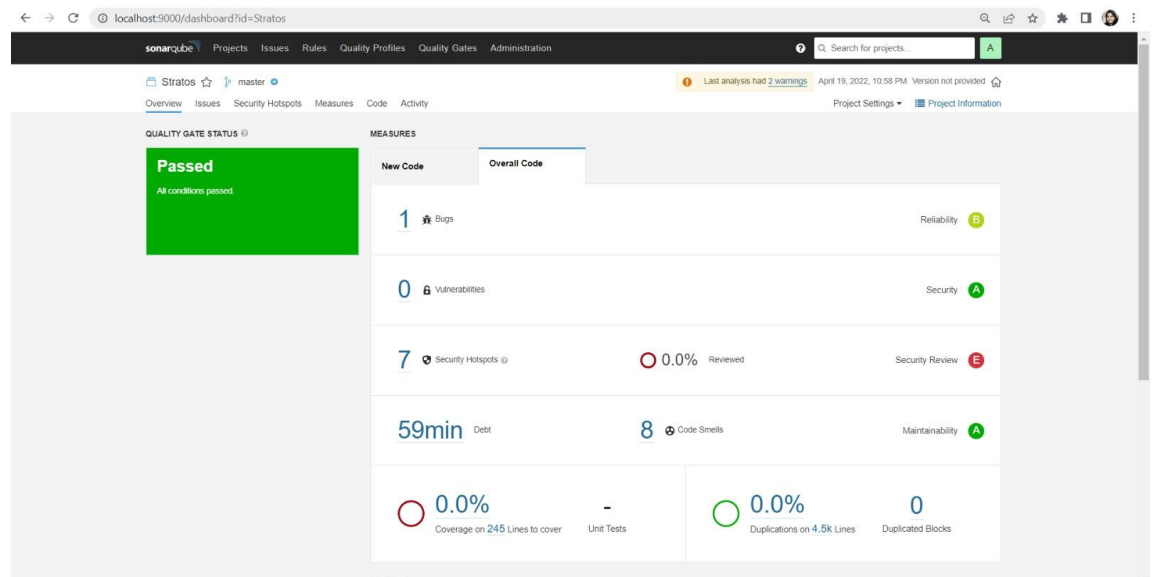
## 9. Added Value

### 9.1 Software Quality Metrics

Software Quality Metrics are performance indicators. They help us identify potential bad code smells and tell us about the overall quality of the code. We used SonarLint to identify vulnerabilities and bad code smells in the code.

#### 9.1.1 Sonar Lint

We used SonarLint in our code to detect bad code smells. We refactored wherever necessary and tried to resolve the bugs and vulnerabilities found in the Sonar Lint analysis.



### 9.2 Deployment on Heroku

The Heroku CLI is a useful tool for building and managing web apps that are hosted remotely. Once you've set up the repository you wish to deploy, you may utilize it to quickly establish a Heroku app.

Commands for deploying an application to Heroku are:  
`heroku create`

In order to deploy the latest version of the application, we can link Heroku to the Git repository. We execute the following commands:

```
git remote -v
```

```
heroku https://git.heroku.com/<heroku-app-name>.git (fetch)
```

```
heroku https://git.heroku.com/<heroku-app-name>.git (push)
```

To deploy the application, we execute the following command:  
`git push heroku main`

The deployed URL of our application is:

<https://quizforum.herokuapp.com>

The screenshot shows the Heroku dashboard for the 'stratos-quiz' application. The browser address bar displays 'dashboard.heroku.com/apps/stratos-quiz/deploy/heroku-git'. The Heroku logo and a search bar are at the top. The navigation bar shows 'Personal' > 'quiz-app' > 'stratos-quiz', with 'Open app' and 'More' buttons. The main content area is divided into two columns. The left column, titled 'Connected to a pipeline', shows the app is assigned to the 'production' pipeline in the 'quiz-app' space. It includes links to 'Manage this pipeline and this app's stage on the pipeline overview' and 'Review apps are available on the pipeline overview'. The right column, titled 'Deployment method', lists three options: 'Heroku Git Use Heroku CLI', 'GitHub Connect to GitHub', and 'Container Registry Use Heroku CLI'. Below this, the 'Deploy using Heroku Git' section provides instructions: 'Use git in the command line or a GUI tool to deploy this app.' and 'Install the Heroku CLI'. It includes a link to 'Download and install the Heroku CLI' and a note: 'If you haven't already, log in to your Heroku account and follow the prompts to create a new SSH public key.' A terminal snippet shows '\$ heroku login'. The 'Create a new Git repository' section instructs to 'Initialize a git repository in a new or existing directory'.

dashboard.heroku.com/apps/stratos-quiz/deploy/heroku-git

Salesforce Platform

HEROKU Jump to Favorites, Apps, Pipelines, Spaces...

Personal > quiz-app > stratos-quiz ☆ Open app More

Overview Resources Deploy Metrics Activity Access Settings

Connected to a pipeline

Assigned to production in quiz-app

- Manage this pipeline and this app's stage on the [pipeline overview](#)
- Review apps are available on the [pipeline overview](#)

Deployment method

- Heroku Git Use Heroku CLI
- GitHub Connect to GitHub
- Container Registry Use Heroku CLI

Deploy using Heroku Git

Use git in the command line or a GUI tool to deploy this app.

Install the Heroku CLI

Download and install the [Heroku CLI](#).

If you haven't already, log in to your Heroku account and follow the prompts to create a new SSH public key.

```
$ heroku login
```

Create a new Git repository

Initialize a git repository in a new or existing directory

## 9.3 Ajax

Ajax is a framework of JavaScript. It is a technology that allows applications to transmit and receive data asynchronously in the background without having to reload the entire page.

In order to render the selection of quizzes and user results on the same page, we use ajax calls to fetch data from the server and update a part of the webpage rather than redirecting the user to another page.

Ajax calls are used to perform Callbacks which increases speed and efficiency of the web application. The calls to the server are anonymous.

The key advantage of Ajax is to enable applications to be more responsive, faster and more user-friendly.

```
const sendData = () => {

const amount = document.getElementById('num_of_question')
const difficulty = document.querySelector('#difficult')
const choice = document.querySelector('#cho')
const data = {}
const csrf = document.getElementsByName('csrfmiddlewaretoken')
data['csrfmiddlewaretoken'] = csrf[0].value
data['amount'] = amount.value
data['difficulty'] = difficulty.value
data['choice'] = choice.value
data['type'] = "multiple"

$.ajax({
  type: 'POST',
  url: `${url}`,
  data: data,
  success: function(response){
    addForm.classList.add('not-visible')
    quiForm.classList.remove('not-visible')
    modalBtn.classList.remove('not-visible')
    saveBtn.classList.remove('not-visible')
    goibibo.forEach(goib=>{
      goib.classList.remove('not-visible')
    })

    const data = response.data
    maindata = data
    showData(data)
  },
},
```

## 9.4 Web Scraping

Web scraping is the act of gathering and processing raw data from the internet, and the Python community has developed several impressive web scraping tools.

In our application, the questions for all quizzes are collected via Web Scraping.

The python library called BeautifulSoup has been used for web scraping and extracting data via APIs.

It generates a parse tree from which data may be extracted from HTML on a website.

Navigation, searching, and changing these parse trees are all possible with BeautifulSoup.

## 9.5 PyLint

Pylint is a Python code checking tool that looks for errors, tries to impose a coding standard, and detects code smells. It can also look for specific type issues, make suggestions for how to modify specific blocks, and provide information about the code's complexity.

```
(env1) PS C:\Users\admin\Documents\JJ-Final-Proj-V1\Stratos\stratos> pylint loginsystem
***** Module admin
loginsystem\admin.py:4:0: E0402: Attempted relative import beyond top-level package (relative-beyond-top-level)
***** Module forms
loginsystem\forms.py:7:4: R0903: Too few public methods (0/2) (too-few-public-methods)
loginsystem\forms.py:18:4: R0903: Too few public methods (0/2) (too-few-public-methods)
***** Module models
loginsystem\models.py:95:15: C0209: Formatting a regular string which could be a f-string (consider-using-f-string)
loginsystem\models.py:107:12: W0201: Attribute 'type' defined outside __init__ (attribute-defined-outside-init)
loginsystem\models.py:120:12: W0201: Attribute 'type' defined outside __init__ (attribute-defined-outside-init)
loginsystem\models.py:133:12: W0201: Attribute 'type' defined outside __init__ (attribute-defined-outside-init)
loginsystem\models.py:141:4: R0903: Too few public methods (0/2) (too-few-public-methods)
loginsystem\models.py:150:4: R0903: Too few public methods (0/2) (too-few-public-methods)
loginsystem\models.py:158:4: R0903: Too few public methods (0/2) (too-few-public-methods)
***** Module urls
loginsystem\urls.py:3:0: E0611: No name 'views' in module '' (no-name-in-module)

-----
Your code has been rated at 8.50/10 (previous run: 8.50/10, +0.00)

(env1) PS C:\Users\admin\Documents\JJ-Final-Proj-V1\Stratos\stratos> 
```

```
(env1) PS C:\Users\admin\Documents\JJ-Final-Proj-V1\Stratos\stratos> pylint results
***** Module results.admin
results\admin.py:11:0: C0304: Final newline missing (missing-final-newline)
results\admin.py:9:0: W0105: String statement has no effect (pointless-string-statement)
***** Module results.migrations.0001_initial
results\migrations\0001_initial.py:1:0: C0103: Module name "0001_initial" doesn't conform to snake_case naming style (invalid-name)

-----
Your code has been rated at 8.80/10 (previous run: 2.69/10, +6.11)

(env1) PS C:\Users\admin\Documents\JJ-Final-Proj-V1\Stratos\stratos> 
```



```
(env1) PS C:\Users\admin\Documents\JJ-Final-Proj-V1\Stratos\stratos> pylint quizapplication
***** Module quizapplication.secrets
quizapplication\secrets.py:2:0: C0305: Trailing newlines (trailing-newlines)
quizapplication\secrets.py:1:0: C0114: Missing module docstring (missing-module-docstring)
quizapplication\secrets.py:1:0: C0103: Constant name "secret" doesn't conform to UPPER_CASE naming style (invalid-name)
***** Module quizapplication.settings
quizapplication\settings.py:120:0: C0303: Trailing whitespace (trailing-whitespace)
quizapplication\settings.py:150:0: C0303: Trailing whitespace (trailing-whitespace)
quizapplication\settings.py:151:0: C0303: Trailing whitespace (trailing-whitespace)

-----
Your code has been rated at 8.64/10 (previous run: 8.64/10, +0.00)

(env1) PS C:\Users\admin\Documents\JJ-Final-Proj-V1\Stratos\stratos> 
```

## 9.6 . Testing

We tested various aspects of our project using different test cases. For example, in the screenshot below, we tested if the quiz was created successfully or not.

We also tested other factors such as:

1. Fetching the questions from API (trivia api)
2. Test Creation
3. Score Calculation

```
4 import datetime
5 from django.test import TestCase
6 from django.db import models
7 from .. import models as quiz_models
8
9
10 class TestModels(TestCase):
11     """
12     Class for defining unit test cases methods.
13     """
14     def setUp(self):
15         """
16         Constructor for each unit test case
17         """
18         class QuizModel(models.Model):
19             """
20             Assigns a manager for dummy test model.
21             """
22             objects = quiz_models.Quiz()
23
24         self.DummyModel = QuizModel
25         self.assertIsInstance(self.QuizModel.objects, quiz_models.QuizModel)
26
27     def test_model_str(self):
28         """
29         Unit test case for string format of model object.
30         """
31         quiz_obj = Quiz.objects.create(
32             start_date=datetime.date.today(),
33             end_date=datetime.date.today(),
34             questions=500,
35             quiz_type=2,
36             total_score=100,
37             userId=5
38         )
39         self.assertEqual(str(quiz_obj), "Test User")
40
```

## 10. Evaluation and Critique

- Optimized code due to implementation of multiple design patterns
- Identified and fixed bad code smells by using SonarLint
- During the process of improving code quality to prevent bad code smells, the code performed better and was optimized.
- Although implementing the Interceptor Design Pattern was difficult, we were able to add logging service into our web application without making major changes to the existing code
- The services (Logging, Quiz application, Questions, Results) were loosely coupled and can be extended easily.

## 11. Team Member Contribution Overview

Team Member	Lines of Code CS5722
Sarthak Punjabi (SP)	882
Shraddha Zade (SZ)	750
Komal Iyer (KI)	766
Shagil Chaudhary (SC)	720

quizes		
Class	LOC	Author
add_quiz.js	68	SP
main.js	180	SZ
quiz.js	150	SC
style.css	96	KI

loginsystem		
Class	LOC	Author
__init__.py	78	SZ
admin.py	54	SP
apps.py	64	SC
models.py	47	SZ
tests.py	80	KI

loginsystem		
Class	LOC	Author
manage.py	89	SZ
views.py	95	KI

questions		
Class	LOC	Author
__init__.py	74	KI
admin.py	92	SZ
apps.py	112	SC
models.py	180	SP
tests.py	60	KI

quiz application		
Class	LOC	Author
__init__.py	179	SZ
asgi.py	97	SP
secrets.py	54	SP
settings.py	32	SC
urls.py	90	SC

quiz application		
Class	LOC	Author
__init__.py	179	SZ
asgi.py	97	SP
secrets.py	54	SP
settings.py	32	SC
wsgi.py	86	KI

static\quizes		
Class	LOC	Author
add_quiz.js	180	SP
main.js	48	SZ
quiz.js	150	SZ

templates\quizes		
Class	LOC	Author
add_quiz.html	45	SC
main.html	39	KI
quiz.html	56	SP
create_quiz.html	34	SC

quizes		
Class	LOC	Author
__init__.py	25	SZ
admin.py	37	SP
apps.py	62	SZ
models.py	71	SC
tests.py	35	KI
urls.py	23	SP
views.py	45	KI
command.js	137	KI
adapter.py	150	SZ
factory.py	100	SC
interceptor.py	250	SP
memento.py	120	SC
facade.py	80	SP

results		
Class	LOC	Author
__init__.py	48	SC
admin.py	36	KI
apps.py	23	SZ
models.py	120	SC
tests.py	95	KI
views.py	100	KI
styles.css	47	SP
base.html	95	SP

## 12. References

- [1] Memento Design pattern - <https://hackernoon.com/memento-design-pattern-overview-4r7p3wol>
- [2] Bridge Design pattern - <https://www.scaler.com/topics/bridge-design-pattern/>
- [3] Factory Design pattern - <https://www.buggybread.com/2016/12/design-pattern-advantages-and.html>
- [4] Adapter Design pattern - <https://www.dineshonjava.com/adapter-design-pattern/>
- [5] Facade Design pattern - <https://spectrumstutz.com/dp/facade-design-pattern/>
- [6] Interceptor Design pattern - <http://software-pattern.org/Interceptor>
- [7] Design patterns overview -  
Nechypurenko, A., 2016. Using design patterns to improve aspect reusability and dynamics. *Aspects, Components, and Patterns for Infrastructure Software*, p.38.
- [8]Refactoring Guru (2022) Builder Design Pattern available:  
<https://refactoring.guru/design-patterns/builder>
- [9]GeeksforGeeks (2022) Adapter Design Pattern available:  
<https://www.geeksforgeeks.org/adapter-pattern>
- [10] Hunt, John. "Gang of Four Design Patterns." (2013).
- [11] <https://oxylabs.io/blog/python-web-scraping>
- [12] <https://pythonwife.com/facade-design-pattern-with-python/>