Group 12 report
Hotel Management System

This Project is Prepared by :

We distributed the front-end and back-end among us, developed different versions and updated all on GitHub.

Name	ID
Youssef Yosry Mohamed Mahmoud	6953
Ahmed Ashraf Elsayed	6742
Mohamed Saad Essa	7214

Project URL:

GitHub Repository: https://github.com/SW-F23/f23-sw-p2-g12.git

Table of Contents :

Table of Contents

This Project is Prepared by : 1
Project URL: 1
Table of Contents : 1
Customer Statement of Requirements: 3
Website Services: 3
Reservation Management: 3
Guest Requests and Complaints 3

User Registration and Guest Profiles 4
Admin Dashboard 4
Employee Management 4
Room Availability 4
Website User Requirements: 4
Functional Requirements: 4
Non-functional Requirements: 5
Website System Requirements: 5
Functional requirements: 5
Non-functional Requirements: 6
Architecture Design: 7
Advantage of Waterfall model: 7
Disadvantage of waterfall model: 8
1. Lack of Flexibility: 8
2. Time-Consuming: 8
Processes and Phases of the project: 8
Phase 1: Requirements Gathering 8
Phase 2: Design 9
Phase 3: Development 9
Phase 4: Deployment 9
Phase 5: Maintenance 9
MVC Model Diagram: 10
Database diagrams: 11
Relational Model: 11
Entity Relation Diagram: 11
UML Diagrams: 12
Use case diagram: 12
Class Diagram: 12

State Machine Diagram: 13
User Side Diagrams: 13
1. Activity Diagram: 13
2. Sequence Diagram: 14
Admin Side Diagrams: 15
1. Activity Diagram: 15
2. Sequence Diagram: 16
Test driven development: 17

Customer Statement of Requirements:

The purpose of our initiative is to revolutionize hotel management in response to the growing demands of the thriving tourism industry. By addressing challenges related to reservations, service quality, and overall customer satisfaction.

To make things clear, a hotel management system is an integrated and innovative solution designed to streamline hotel operations, enhance customer satisfaction, and adapt to the evolving landscape of the hotel industry.

Our primary Goal is to develop a sophisticated hotel management system that optimizes internal processes and elevates customer satisfaction.

Website Services:

Reservation Management:

- Allow guests to reserve rooms for specific nights.
- Provide options for choosing room sizes: single, double, or triple.
- Offer meal plan selections, including Full Board or Half Board.

Guest Requests and Complaints

- Create a user-friendly interface for guests to submit requests, complaints, or inquiries.
- Categorize requests into different types such as room service,

maintenance, or general inquiries.
- Implement an automated acknowledgment and tracking system for guest requests.

User Registration and Guest Profiles
- Establish a secure user registration system for guests.
- Enable guests to create profiles with personal preferences, contact details, and stay history.

Admin Dashboard
- Develop an admin dashboard to manage and monitor hotel operations.
- Track real-time room availability.
- Manage reservations, including check-ins and check-outs.

Employee Management
- Create profiles for hotel staff, including relevant information such as job roles, contact details, and shift schedules.
- Provide administrative capabilities to add, modify, or remove staff profiles.

Room Availability
- Create a visual representation of room availability.
- Differentiate between room types and their status (available, booked, occupied).
- Allow easy updating of room status based on cleaning, maintenance, or reservations.

Website User Requirements:

Functional Requirements:

- **User Authentication:** Users are required to log in to their respective profile pages or sign up if no account exists.
- **Room Preferences:** Once logged in, users can seamlessly search for available rooms based on their preferences (single, triple, double)
- **Communication with Admin:** Users have the facility to submit complaints or inquiries directly to the administrator for prompt resolution.
- **Service Requests:** Following room check-in, clients can submit requests for services such as room service, maintenance, or laundry.

- **Administrative Control:** Administrators have the capability to add, modify, or delete both rooms and staff members.
- **Complaint Handling:** Admins can effortlessly view and respond to all user complaints, ensuring successful communication with clients.
- **Check-in/Check-out Management:** Admins possess control over check-in and checkout activities in rooms, ensuring a smooth process for users and the establishment.
- **Task Assignment:** Admins can assign specific tasks to workers, with these tasks being transparently displayed to clients, fostering a collaborative and responsive service environment.
- **Worker Access:** Workers can successfully log in using their unique IDs and passwords, ensuring access to the system.

Non-functional Requirements:

- **Usability:** The system should have a user-friendly interface that is intuitive and easy to navigate for both guests and hotel staff.
- **Performance:** The system should provide fast response times for actions such as room reservations, guest requests, and availability updates.
- **Reliability:** The system should be highly reliable, with minimal downtime and data loss, to ensure uninterrupted hotel operations.
- **Security:** The system should have robust security measures in place to protect guest information, user accounts, and sensitive data.
- **Accessibility:** The system should be accessible to users with disabilities, complying with accessibility standards and guidelines.
- **Scalability:** The system should be scalable to handle increasing numbers of users, rooms, and transactions as the hotel grows.
- **Training and Support:** The system should provide adequate training resources and support to ensure users can effectively utilize its features.

Website System Requirements:

Functional requirements:

- **Reservation Management:** Allow guests to reserve rooms for specific nights. As well as selecting the suitable room sizes from

single, double, or triple. We take into account the prices overhead, so we allow meal plan selections as well, that's to say choosing whether your stay is full-board or half-board or no meals provided.

- **Guest Requests and Complaints:** Submitting your feedback is crucial for our improvement, that's why we created an user friendly UI to submit your requests and complaints. - Categorize requests into different types such as room service, maintenance, or general inquiries.
- **Admin Dashboard:** We Developed an admin dashboard to manage and monitor hotel operations. It also has a real-time room availability tracker which indicates whether a room is booked or not. Most importantly it allows the management of reservations, including check-ins and check-outs.
- **Employee Management:** New employee or customer? No worries we took into consideration creation of new profiles for hotel staff and customers.
- **Room Availability:** Remember the UI we talked about, we included a visual representation of rooms availability and their type. We update room status daily, so if it ever happens that someone canceled his reservation and a room became available you can always check daily, there is a chance a room is available to book now.

Non-functional Requirements:

- **Compatibility:** The system should be compatible with different devices, operating systems, and web browsers commonly used by guests and hotel staff.
- **Integration:** The system should support integration with external systems such as payment gateways, accounting software, and property management systems.
- **Performance:** The system should be capable of handling concurrent user requests and large amounts of data without significant performance degradation.
- **Security:** The system should implement robust security measures, including encryption, authentication, and authorization mechanisms, to protect against unauthorized access and data breaches.

- **Scalability:** The system architecture should be designed to accommodate future growth and increased system usage without significant architectural changes.
- **Data Backup and Recovery:** The system should have mechanisms in place to regularly back up data and ensure quick recovery in the event of data loss or system failure.
- **Maintenance and Upgrades:** The system should be easily maintainable, allowing for regular updates, bug fixes, and enhancements without disrupting hotel operations.
- **Data Privacy:** The system should adhere to data privacy regulations, ensuring the confidentiality and proper handling of guest information and personal data.
- **System Monitoring:** The system should have monitoring capabilities to track system performance, detect anomalies, and generate logs for auditing and troubleshooting purposes.
- **Compliance:** The system should comply with relevant industry standards and regulations, such as data protection laws and PCI-DSS (Payment Card Industry Data Security Standard) requirements.

Architecture Design:

We Used the waterfall model for several reasons: Waterfall architecture is a sequential design method well-suited for projects with defined requirements and a clear timeline. This approach involves completing each phase of the development process before moving on to the next. For the Hotel Harmony Management System, this method offers distinct advantages.

Advantage of Waterfall model:

Well-Defined Process: The waterfall method provides a clear set of steps, making it easy to understand and follow. This reduces the likelihood of errors and delays in the development process.

Structured Approach: Waterfall is a structured process that ensures all necessary steps are completed in a logical order. This structured approach helps prevent confusion and ensures that each phase builds upon the previous one.

Disadvantage of waterfall model:

1. Lack of Flexibility:
   Waterfall is not inherently flexible, making it challenging to adapt to changes in requirements or scope. This may pose difficulties for a project like the Hotel Harmony Management System, which may encounter evolving needs.
2. Time-Consuming:
   The waterfall method can be time-consuming, especially for large or intricate projects. Given the potential size and complexity of a hotel management system, the sequential nature of waterfall may extend the development timeline.

How did we overcome the disadvantages of the model:

Perfectly finish initial phases: We did not proceed to next phases until we make ensure that previous phases are done according to the specifications.

Easy requirements definition: Requirements appear to be well defined and easy to understand. There are no conflicting requirements and no ambiguity in specifications.

Requirements are not subject to change: User will not change the requirements in the development process.

our choice to employ the waterfall model for the architectural design of the Hotel Harmony Management System was guided by its suitability for projects characterized by well-defined requirements and a clear timeline. The advantages of a well-defined process and a structured approach were evident, ensuring a systematic progression through each development phase.

Processes and Phases of the project:

Phase 1: Requirements Gathering

The first phase of the software development process is requirements gathering. This is where the team gathers information about what the website needs to do. This information can come from a variety of sources, such as user interviews, surveys, and focus groups. The goal of requirements gathering is to create a clear and concise document that describes the functionality of the website.

Phase 2: Design

The next phase is design. This is where the team creates a blueprint for the website. The design should include the overall layout of the website, as well as the look and feel of the website. The design should also take into account the user experience and usability of the website.

Phase 3: Development

The development phase is where the team actually builds the website. This is where the team takes the design and turns it into a working website. The development phase can be divided into two sub-phases:

- Coding: This is where the team writes the code for the website.
- Testing: This is where the team tests the website to make sure it works correctly.

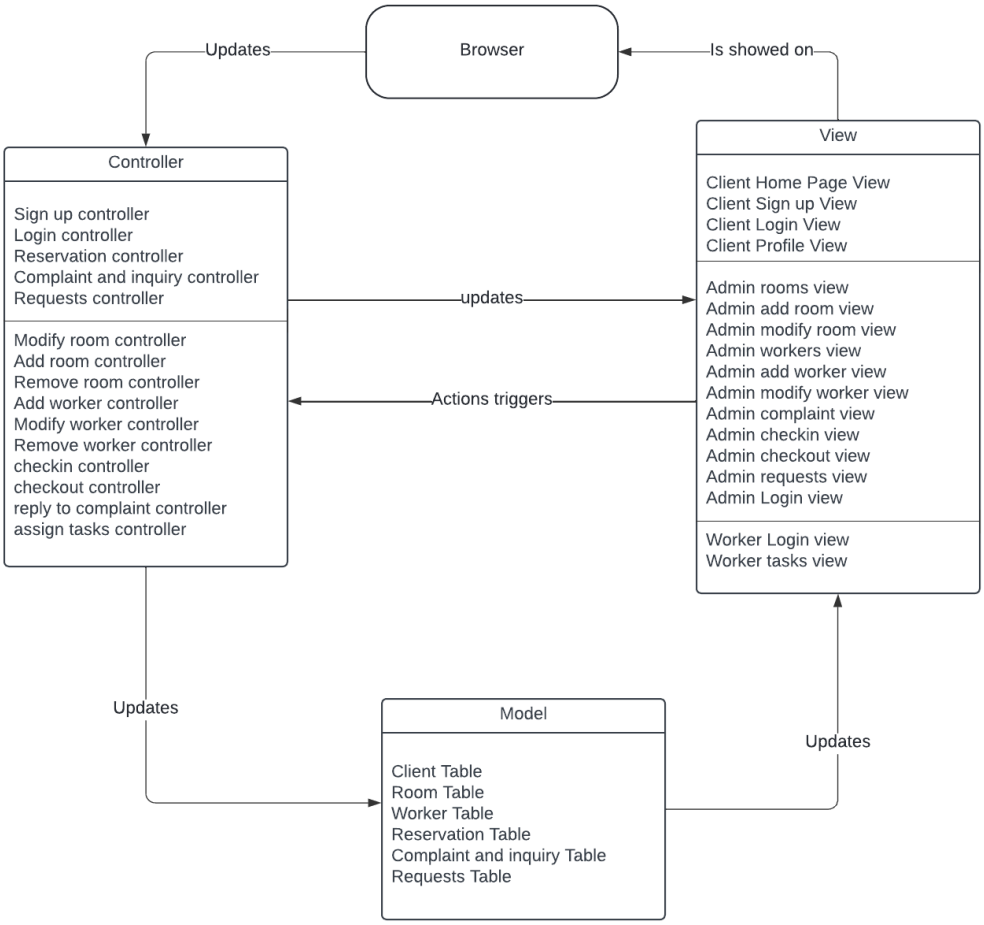
Phase 4: Deployment

The deployment phase is where the website is made available to users. This can be done by uploading the website to a web server or by hosting the website on a cloud platform.

Phase 5: Maintenance

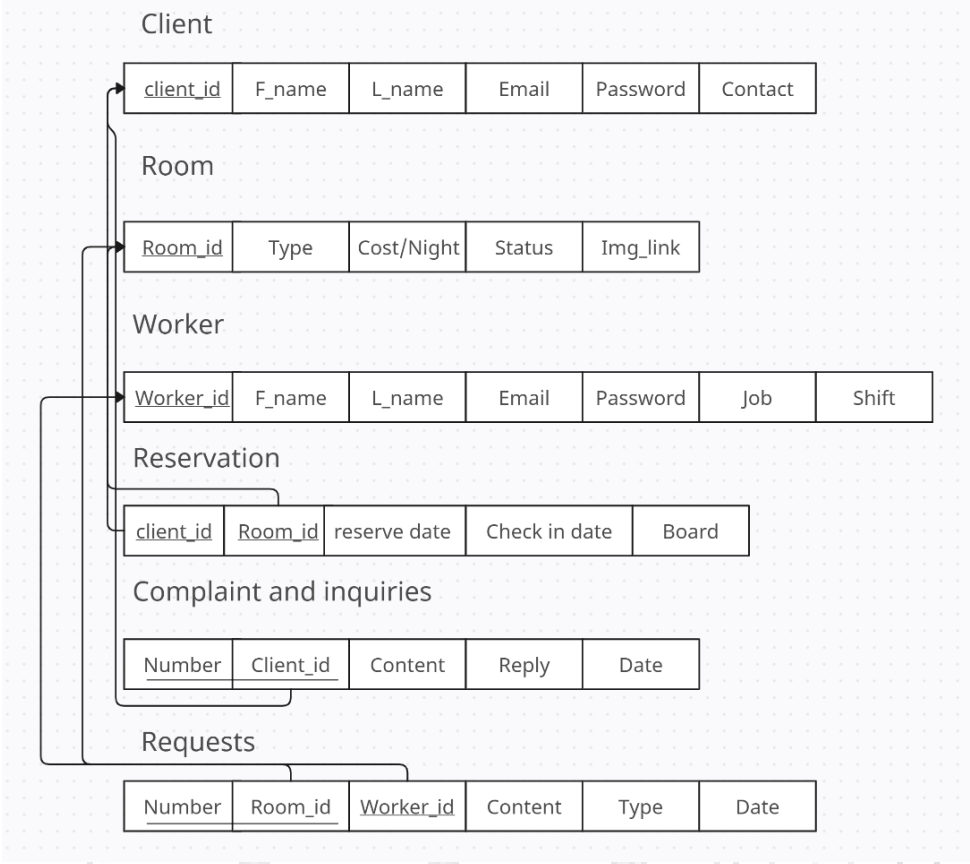
The maintenance phase is where the team keeps the website up and running. This includes tasks such as fixing bugs, adding new features, and performing security updates.

MVC Model Diagram:

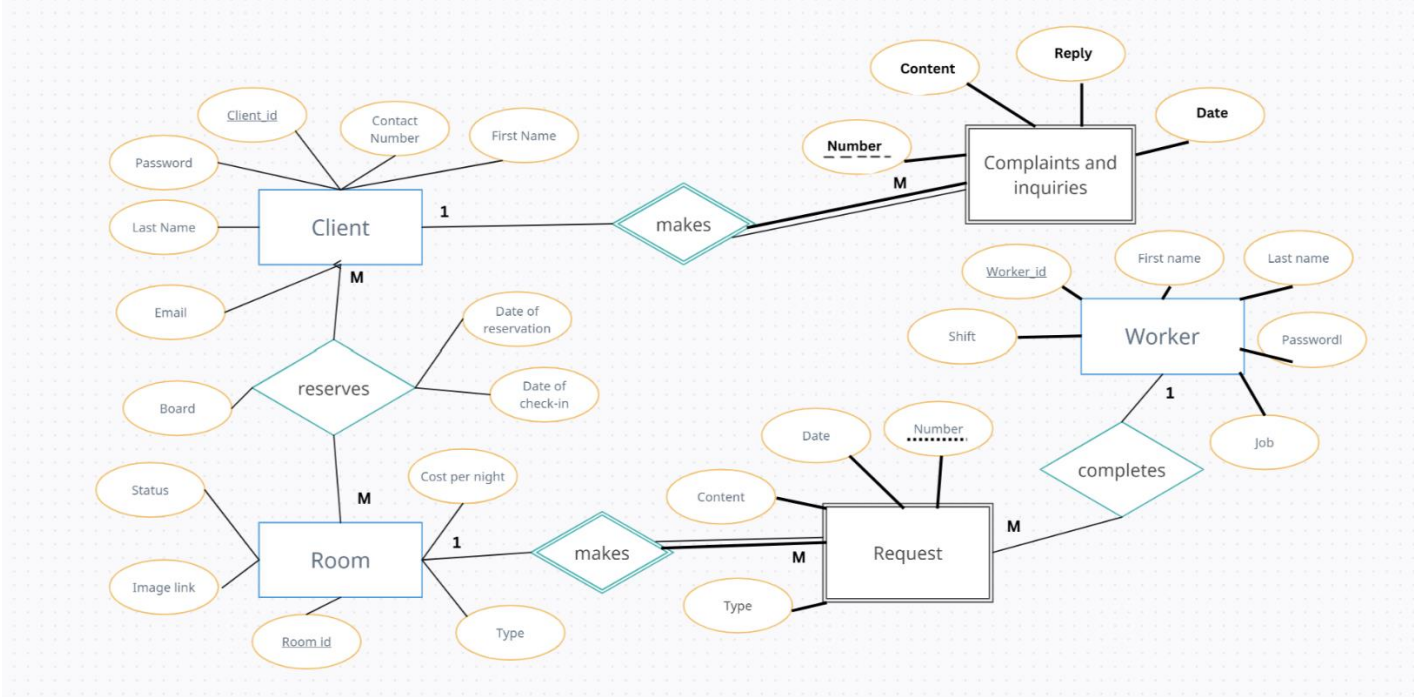


Database diagrams:

Relational Model:

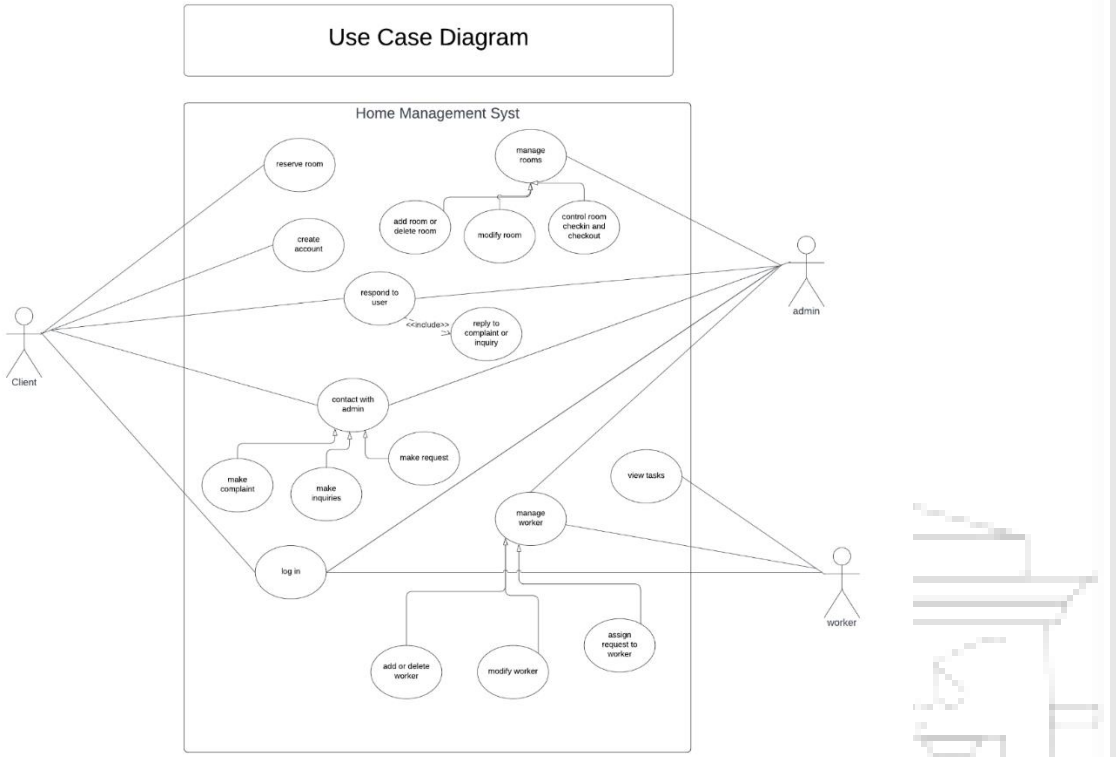


Entity Relation Diagram:

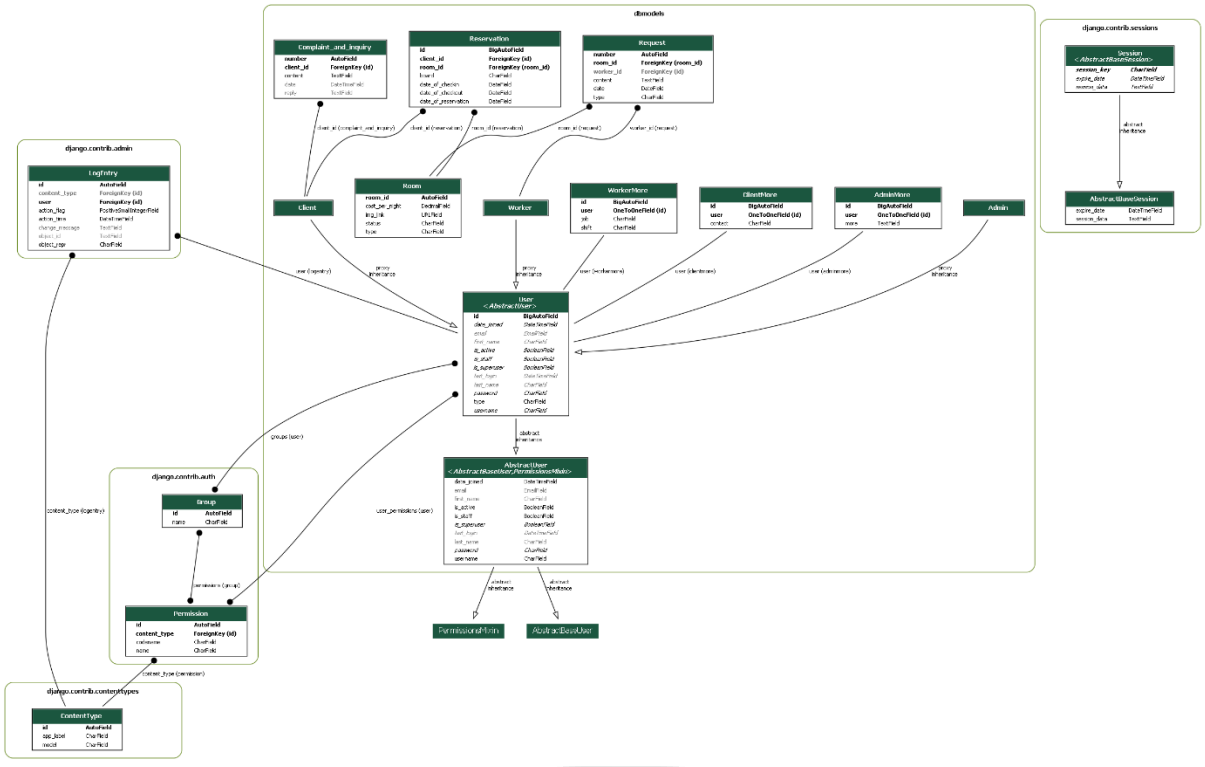


UML Diagrams:

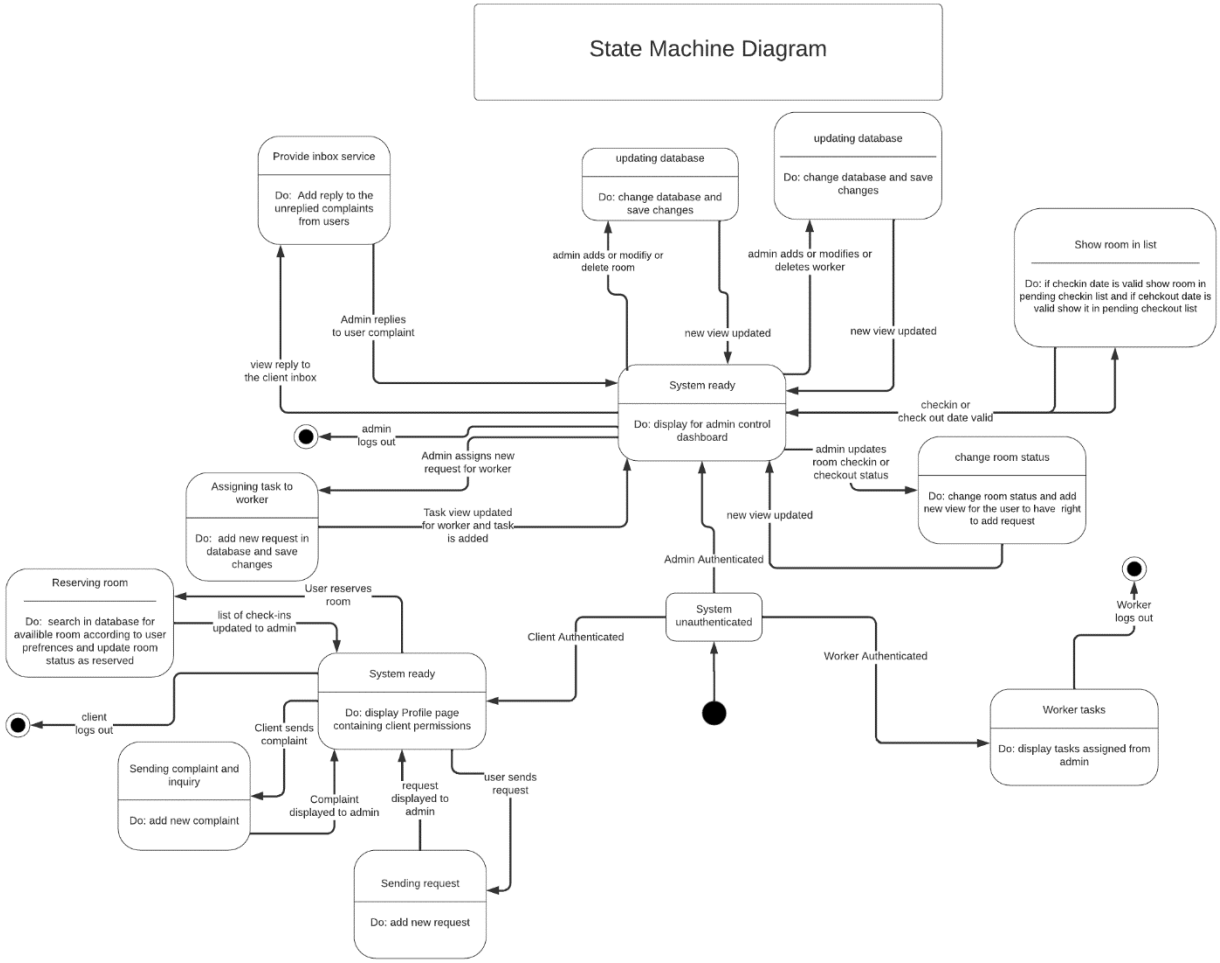
Use case diagram:



Class Diagram:

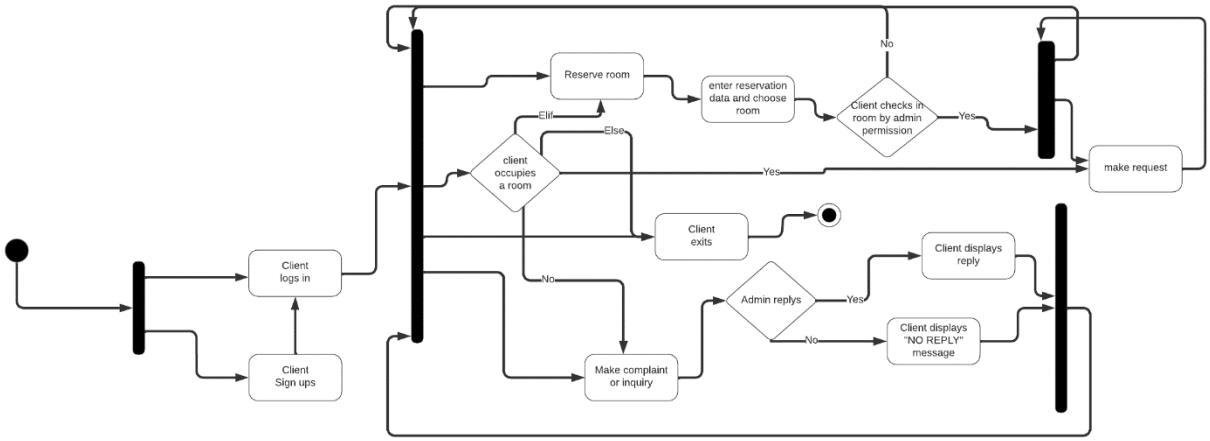


State Machine Diagram:

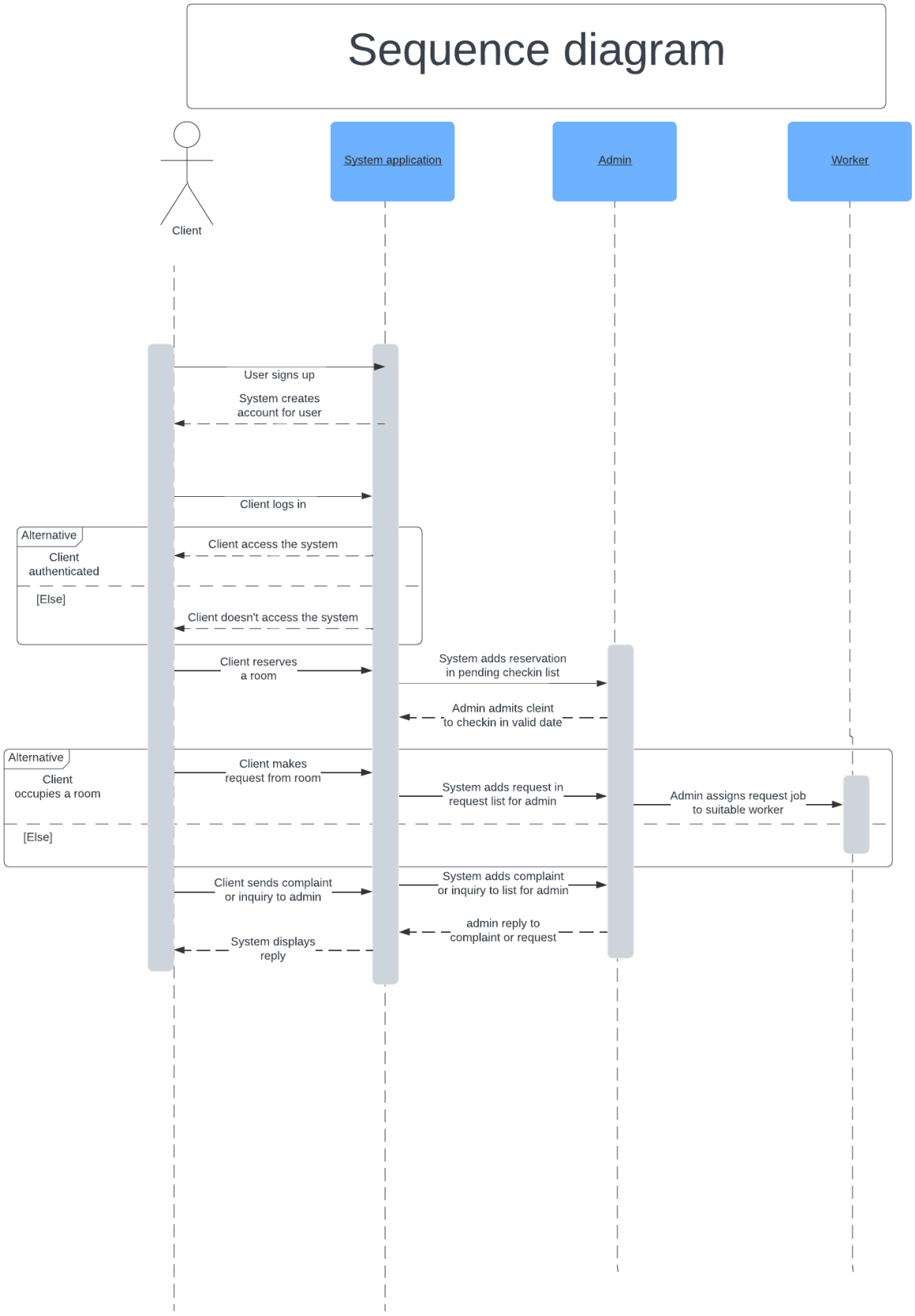


User Side Diagrams:

1. Activity Diagram:

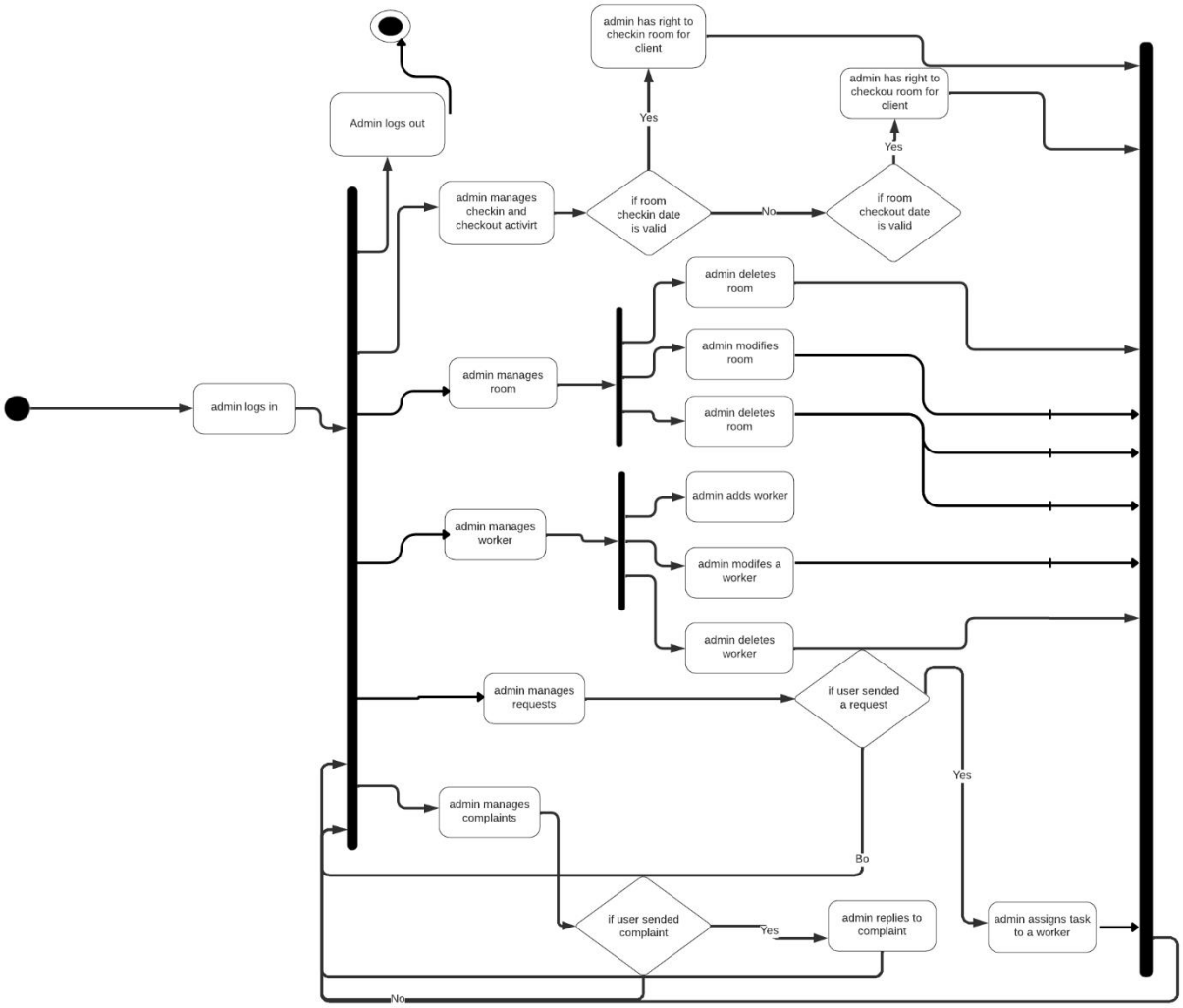


2. Sequence Diagram:

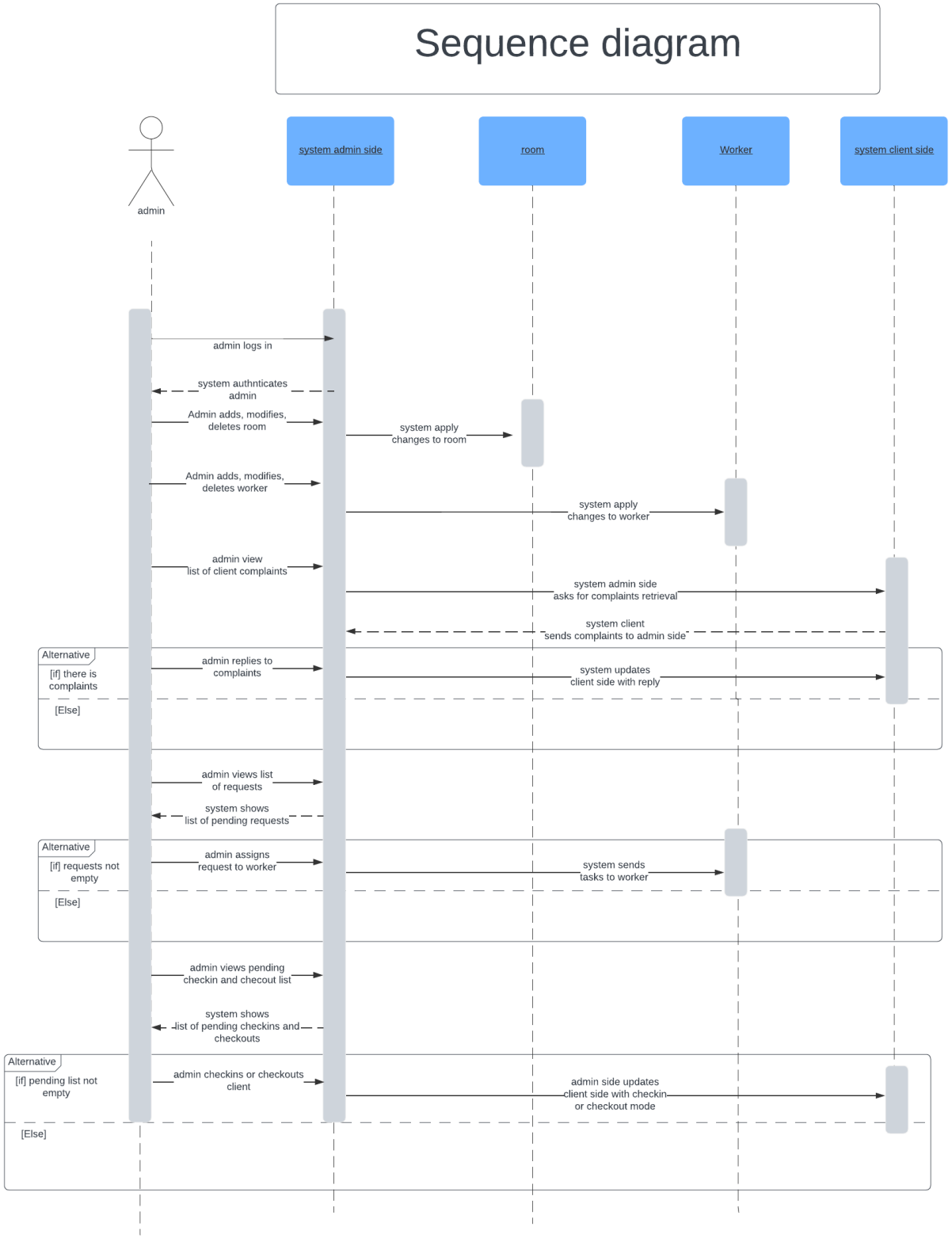


Admin Side Diagrams:

1. Activity Diagram:



2. Sequence Diagram:



Test driven development:

We Installed the coverage django extension in order to get a baseline of the amounts of tests needed to cover the mvc architecture.

Initial Run:

Coverage report: 43%

coverage.py v7.4.0, created at 2023-12-28 20:37 +0200

Module	statements	missing	excluded	coverage
database__init__.py	0	0	0	100%
database\settings.py	20	0	0	100%
database\urls.py	3	0	0	100%
dbmodels__init__.py	0	0	0	100%
dbmodels\admin.py	8	0	0	100%
dbmodels\apps.py	4	0	0	100%
dbmodels\forms.py	192	106	0	45%
dbmodels\migrations__init__.py	0	0	0	100%
dbmodels\models.py	106	21	0	80%
dbmodels\tests.py	1	0	0	100%
dbmodels\urls.py	4	0	0	100%
dbmodels\views.py	278	231	0	17%
manage.py	12	2	0	83%
Total	**628**	**360**	**0**	**43%**

coverage.py v7.4.0, created at 2023-12-28 20:37 +0200

Initial Testing of models.py:

Coverage for **dbmodels\models.py**: 80%

106 statements | 85 run | 21 missing | 0 excluded

« prev ^ index » next coverage.py v7.4.0, created at 2023-12-28 20:37 +0200

```python
from django.db import models
from django.contrib.auth.models import AbstractUser
from django.utils.translation import gettext_lazy as _
# To query data from now on for a specific type of user, use the following: user.more.<field_name> such as client.more.contact or worker.more.job
# To query data from now on for a user which includes all types of users (Admin, Client, Worker), use the following usertype.<field_name> such as
# Worker.objects.all() or Client.objects.all().fname etcc , the extra fields are acesses through the more but the user is accessed through it's name
# fields for the user are : first name , last name , email , password , username , type

class User(AbstractUser):
    class Types(models.TextChoices):
        ADMIN = "ADMIN", "Admin"
        CLIENT = "CLIENT", "Client"
        WORKER = "WORKER", "Worker"

    base_type = Types.ADMIN

    # What type of user are we?
    type = models.CharField(
        _("Type"), max_length=50, choices=Types.choices, default=base_type
    )
```

Completed Testing of models.py:

Coverage for **dbmodels\models.py**: 100%

128 statements | 128 run | 0 missing | 0 excluded

« prev ^ index » next coverage.py v7.4.0, created at 2023-12-28 22:13 +0200

```python
from django.db import models
from django.contrib.auth.models import AbstractUser
from django.utils.translation import gettext_lazy as _
from django.contrib.auth.base_user import BaseUserManager
from django.contrib.auth.models import AbstractBaseUser, PermissionsMixin
from django.db import models
# To query data from now on for a specific type of user, use the following: user.more.<field_name> such as client.more.contact or worker.more.job
# To query data from now on for a user which includes all types of users (Admin, Client, Worker), use the following usertype.<field_name> such as
# Worker.objects.all() or Client.objects.all().fname etcc , the extra fields are acesses through the more but the user is accessed through it's name
# fields for the user are : first name , last name , email , password , username , type

class UserManager(BaseUserManager):
    def create_user(self, email, username, password=None, type=None, **extra_fields):
        if not email:
            raise ValueError('The Email field must be set')
        if not username:
            raise ValueError('The Username field must be set')

        user = self.model(
            email=self.normalize_email(email),
            username=username,
            type=type,
            **extra_fields
        )
        user.set_password(password)
```

Initial testing for forms:

Coverage for **dbmodels\forms.py**: 52%

199 statements | 103 run | 96 missing | 0 excluded

« prev ^ index » next coverage.py v7.4.0, created at 2023-12-29 16:16 +0200

```python
from django import forms
from .models import *
from django import forms
from .models import Worker


class WorkerCreationForm(forms.Form):
    username = forms.CharField(max_length=150)
    email = forms.EmailField()
    password = forms.CharField(widget=forms.PasswordInput)
    confirm_password = forms.CharField(widget=forms.PasswordInput)
    job = forms.CharField(max_length=100)
    shift = forms.CharField(max_length=50)

    def clean_username(self):
        username = self.cleaned_data['username']
        # Add custom username validation if needed
        return username

    def clean_email(self):
        email = self.cleaned_data['email']
        # Add custom email validation if needed
        return email
```

Completed Testing for forms:

Coverage for **dbmodels\forms.py**: 99%

202 statements | 199 run | 3 missing | 0 excluded

« prev ^ index » next coverage.py v7.4.0, created at 2023-12-29 21:17 +0200

```python
from django import forms
from .models import *
from django import forms
from .models import Worker
from django.contrib.auth.hashers import make_password
from django.contrib.auth import authenticate
import re


class WorkerCreationForm(forms.Form):
    username = forms.CharField(max_length=150)
    email = forms.EmailField()
    password = forms.CharField(widget=forms.PasswordInput)
    confirm_password = forms.CharField(widget=forms.PasswordInput)
    job = forms.CharField(max_length=100)
    shift = forms.CharField(max_length=50)

    def clean_username(self):
        username = self.cleaned_data['username']

        return username

    def clean_email(self):
        email = self.cleaned_data['email']

```

Initial coverage for views:

Coverage for **dbmodels\views.py**: 22%

279 statements | 61 run | 218 missing | 0 excluded

« prev ^ index » next coverage.py v7.4.0, created at 2023-12-28 23:18 +0200

```
1  from django.urls import reverse
2  from django.http import HttpResponse
3  from django.utils import timezone
4  from django.contrib.auth.decorators import login_required
5  from django.contrib.auth import authenticate, login
6  from django.core.exceptions import ObjectDoesNotExist
7  from django.db import IntegrityError
8  from .forms import *
9  from .models import *
10 from datetime import datetime, timedelta , date
11 from django.http import JsonResponse
12 from django.views.decorators.http import require_POST
13 from django.shortcuts import render, get_object_or_404, redirect
14
15
16
17 def home(request):
18     return render(request, 'dbmodels/home.html')
19
20 def view_workers(request):
21     workers = Worker.objects.all()
22     return render(request, 'dbmodels\workers.html', {'workers': workers})
23
24 def view_complaints_and_inquiries(request):
```

Nearly complete coverage for views:

Coverage for **dbmodels\views.py**: 57%

255 statements | 145 run | 110 missing | 0 excluded

« prev ^ index » next coverage.py v7.4.0, created at 2023-12-29 12:17 +0200

```
1  from django.urls import reverse
2  from django.http import HttpResponse, HttpResponseNotAllowed
3  from django.utils import timezone
4  from django.contrib.auth.decorators import login_required
5  from django.contrib.auth import authenticate, login
6  from django.core.exceptions import ObjectDoesNotExist
7  from django.db import IntegrityError
8  from .forms import *
9  from .models import *
10 from datetime import datetime, timedelta , date
11 from django.http import JsonResponse
12 from django.views.decorators.http import require_POST
13 from django.shortcuts import render, get_object_or_404, redirect
14
15
16
17 def home(request):
18     return render(request, 'dbmodels/home.html')
19
20 def view_workers(request):
21     workers = Worker.objects.all()
22     return render(request, 'dbmodels\workers.html', {'workers': workers})
23
24 def view_complaints_and_inquiries(request):
```

Nearly Complete Coverage but refactoring prevented more tests from being implemented before the deadline.

Coverage report: 78%

filter...

coverage.py v7.4.0, created at 2023-12-29 12:17 +0200

Module	statements	missing	excluded	coverage
database__init__.py	0	0	0	100%
database\settings.py	20	0	0	100%
database\urls.py	3	0	0	100%
dbmodels__init__.py	0	0	0	100%
dbmodels\admin.py	8	0	0	100%
dbmodels\apps.py	4	0	0	100%
dbmodels\forms.py	192	90	0	53%
dbmodels\migrations\0001_initial.py	6	0	0	100%
dbmodels\migrations\0002_rename_extra_field_for_admin_adminmore_more_and_more.py	5	0	0	100%
dbmodels\migrations__init__.py	0	0	0	100%
dbmodels\models.py	146	6	0	96%
dbmodels\tests.py	322	4	0	99%
dbmodels\urls.py	4	0	0	100%
dbmodels\views.py	255	110	0	57%
manage.py	12	2	0	83%
Total	**977**	**212**	**0**	**78%**

coverage.py v7.4.0, created at 2023-12-29 12:17 +0200