Testing Report: Island Dental Booking System

Group Members:

- [Mohamed Rayyan], [S2301279]
- [Student Name 2], [Student ID 2]
- [Student Name 3], [Student ID 3]

Date: [Date of Submission]

Module: UFCF8S-30-2 Advanced Software Development

---

1. Introduction

1.1. Purpose of this Document

This document outlines the testing strategy, test cases, and test results for the "Island Dental Booking System." The primary goal is to demonstrate that the system meets the specified requirements, functions correctly, and handles invalid data appropriately. This report provides evidence of the testing activities undertaken during the development lifecycle.

1.2. Scope of Testing

The scope of testing covers the core functionalities of the Island Dental Booking System as defined in the project scenario and requirements. This includes:

- User authentication and role-based access.
- Appointment booking, cancellation, and management.
- Doctor roster creation and management.
- Clinic, room, and service management.
- Price management based on service and shift.
- Report generation.
- Adherence to business rules (e.g., Friday unavailability, surgery room restrictions, clinic capacity).
- Data validation and error handling.

---

2. Test Strategy

2.1. Testing Levels

Our testing approach incorporated the following levels:

- **Unit Testing:** Focused on testing individual components (e.g., Django models, utility functions, individual API endpoint logic) in isolation to ensure they function as expected.
- **Integration Testing:** Focused on testing the interaction between different integrated components or modules (e.g., the flow of booking an appointment which involves models, serializers, and views).
- **System Testing:** Focused on testing the complete, integrated system against the specified requirements. This included validating GUI interactions and end-to-end workflows from a user's perspective.

2.2. Testing Types

The following types of testing were performed:

- **Functional Testing:** Verifying that all features work according to the requirements.
- **Validation Testing (Data Validation):** Ensuring the system correctly handles valid and invalid data inputs, including boundary conditions and out-of-range values. This includes checking business rule enforcement.
- **GUI Testing:** (For system testing) Ensuring the user interface elements function correctly and provide a usable experience (though the focus is on functionality over extensive UI/UX design for this module).
- **(Optional: Add if you did any specific Security Testing, Performance Testing - though likely out of scope for primary focus)**

2.3. Test Environment

- **Programming Language:** Python (Version [e.g., 3.9])
- **Framework:** Django (Version [e.g., 5.x]), Django REST Framework (Version [e.g., 3.x])
- **Database:** [e.g., SQLite for development/testing, or specify if using PostgreSQL/MySQL]
- **Testing Tools:**
    - Unit Testing: [e.g., Django's built-in TestCase, pytest-django]
    - API Testing: [e.g., Django REST Framework's APIClient, Postman (if used for exploratory API testing)]
    - Manual Testing: For GUI and end-to-end system tests.
-
- **Operating System:** [e.g., Windows 10, macOS Monterey, Linux Ubuntu 20.04]

---

3. Unit Test Cases

This section details the unit tests performed on individual components. The format generally follows: Test Case ID, Component/Method Tested, Test Description, Test Input/Preconditions, Expected Output, Actual Output, Pass/Fail.

(Note: For automated unit tests written in code, you might summarize the test suites here and refer to the actual test code in your project submission for full details. The example below is more for illustrating the thought process).

Example Table Format (repeat for each component):

Test Case ID	Component/Method Tested	Test Description	Test Input/Preconditions	Expected Output	Actual Output	Status
3.1. CustomUser Model Tests						
UT-CU-001	CustomUser.get_role_display()	Verify correct display name for 'CUSTOMER' role.	user = CustomUser(role='CUSTOMER')	"Customer"	[As observed]	[Pass/Fail]
UT-CU-002	CustomUser creation	Attempt to create a user with an invalid role.	CustomUser(username='test', role='INVALID_ROLE') then user.full_clean()	ValidationError should be raised.	[As observed]	[Pass/Fail]
...	...	...	...	...	...	...
3.2. Clinic Model Tests						
UT-CL-001	Clinic.get_available_rooms()	Test with no existing appointments on a valid day/shift.	clinic object, date (not Friday), shift (not evening if testing surgery availability)	All active rooms (respecting surgery room evening rule) should be returned.	[As observed]	[Pass/Fail]

UT-CL-002	Clinic.get_available_rooms()	Test with surgery room request during evening shift.	clinic with surgery room, date (not Friday), shift='EVENING'	Surgery room should NOT be in the list of available rooms.	[As observed]	[Pass/ Fail]
...	...	...	...	...	...	...
3.3. Room Model Tests						
UT-RM-001	Room.is_available()	Check availability for a normal room on a valid day/shift with no bookings.	room (NORMAL), date (not Friday), shift. No conflicting Appointment records.	True	[As observed]	[Pass/ Fail]
UT-RM-002	Room.is_available()	Check unavailability for a surgery room during an evening shift.	room (SURGERY), date (not Friday), shift='EVENING'.	False	[As observed]	[Pass/ Fail]
UT-RM-003	Room.is_available()	Check unavailability when an appointment is already booked for the room/date/ shift.	room, date, shift. Conflicting Appointment(status='CONFIRMED') exists.	False	[As observed]	[Pass/ Fail]

...	...	...	...	...	...	...
3.4. Doctor Model Tests						
UT-DR-001	Doctor.get_availability()	Doctor has one roster entry for MORNING shift on the given date.	doctor object, date. Roster(doctor=doctor, date=date, shift='MORNING') exists.	['MORNING']	[As observed]	[Pass/Fail]
UT-DR-002	Doctor.get_availability()	Doctor has no roster entries for the given date.	doctor object, date. No Roster records for this doctor/date.	[] (empty list)	[As observed]	[Pass/Fail]
...	...	...	...	...	...	...
3.5. Service Model Tests						
UT-SV-001	Service.get_price()	Get price for a service in MORNING shift.	service object, shift='MORNING'. Price(service=service, shift='MORNING', amount=150) exists.	150.00	[As observed]	[Pass/Fail]
UT-SV-002	Service.get_price()	Attempt to get price for a shift where no price is defined.	service object, shift='INVALID_SHIFT' or a valid shift with no Price record.	0 (or appropriate default/error handling as per implementation).	[As observed]	[Pass/Fail]
...	...	...	...	...	...	...
3.6. Price Model Tests						

UT-PR-001	Price creation	Create a valid price entry.	Service object, shift='MORNING', amount=100	Price object created successfully.	[As observed]	[Pass/Fail]
UT-PR-002	Price unique constraint	Attempt to create a duplicate price entry (same service, same shift).	Two Price objects with identical service and shift.	IntegrityError or ValidationError on save/clean.	[As observed]	[Pass/Fail]
...	...	...	...	...	...	...
3.7. Roster Model Tests (Validation in clean() method)						
UT-RO-001	Roster.clean()	Attempt to create roster entry on a Friday.	date is a Friday.	ValidationError ("No service is available on Fridays.")	[As observed]	[Pass/Fail]
UT-RO-002	Roster.clean()	Doctor already has conflicting roster entry for the same date/shift.	doctor, date, shift. Another Roster entry exists for this doctor at this time.	ValidationError (conflicting roster).	[As observed]	[Pass/Fail]
UT-RO-003	Roster.clean()	(If implemented) Clinic has reached	clinic, date, shift. Roster entries already exist for max number of doctors.	ValidationError (clinic doctor capacity).	[As observed]	[Pass/Fail]

		max doctors for a shift.				
...	...	...	...	...	...	...
3.8. Appointment Model Tests (Validation in clean() method & other logic)						
UT-AP-001	Appointment.clean()	Attempt to book on a Friday.	date is a Friday.	ValidationError("No service is available on Fridays.")	[Assobserved]	[Pass/Fail]
UT-AP-002	Appointment.clean()	Attempt to book a surgery room in the evening.	room.type='SURGERY', shift='EVENING'.	ValidationError("Surgery rooms are not available during evening shifts.")	[Assobserved]	[Pass/Fail]
UT-AP-003	Appointment.clean()	Clinic has reached max patient capacity (10) for the shift.	clinic, date, shift. 10 Appointment(status='CONFIRMED') already exist.	ValidationError (clinic patient capacity).	[Assobserved]	[Pass/Fail]
UT-AP-004	Appointment.clean()	Doctor is not on roster for the selected	doctor, clinic, date, shift. No matching Roster entry.	ValidationError (doctor not on roster).	[Assobserved]	[Pass/Fail]

		clinic/date/shift.				
UT-AP-005	Appointment.save()	Verify price is calculated correctly on save.	service, shift. service.get_price(shift) returns a known value (e.g., 200).	Saved Appointment.price should be 200.00.	[Asobserved]	[Pass/Fail]
UT-AP-006	Appointment.generate_reference()	Verify unique reference is generated.	Create an appointment.	appointment.reference is not None and has expected format (e.g., "IDXXXXXXXX").	[Asobserved]	[Pass/Fail]
UT-AP-007	Appointment.cancel()	Cancel a 'CONFIRMED' appointment.	Appointment(status='CONFIRMED'). Call appointment.cancel().	appointment.status becomes 'CANCELLED'. Returns True.	[Asobserved]	[Pass/Fail]
UT-AP-008	Appointment.cancel()	Attempt to cancel an already 'CANCELLED' appointment.	Appointment(status='CANCELLED'). Call appointment.cancel().	appointment.status remains 'CANCELLED'. Returns False.	[Asobserved]	[Pass/Fail]
...	...	...	...	...	...	...
3.9. Validator Utility Tests (appointments/						

utils/validators.py)						
UT-VAL-001	is_friday()	Test with a known Friday date.	date(YYYY, MM, DD) which is a Friday.	True	[As observed]	[Pass/ Fail]
UT-VAL-002	is_friday()	Test with a known non-Friday date.	date(YYYY, MM, DD) which is not a Friday.	False	[As observed]	[Pass/ Fail]
UT-VAL-003	validate_surgery_room_not_in_evening()	room_type='SURGERY', shift='EVENING'	Input: 'SURGERY', 'EVENING'	ValidationError raised.	[As observed]	[Pass/ Fail]
UT-VAL-004	validate_surgery_room_not_in_evening()	room_type='NORMAL', shift='EVENING'	Input: 'NORMAL', 'EVENING'	No ValidationError.	[As observed]	[Pass/ Fail]
...	...	...	...	...	...	...
3.10. API Endpoint Unit Tests (Example - AppointmentViewSet) (Focus on serializer and view logic, mocking external calls if necessary)						
UT-API-AP-001	Create Appointment (Valid)	Customer attempts to book a valid appointment slot.	Valid POST data to /api/appointments/, authenticated as Customer. All preconditions	HTTP 201 CREATED. Response	[As observed]	[Pass/ Fail]

			(roster, room availability) met.	contains appointment details and correct price.		
UT-API-AP-002	Create Appointment (Friday)	Customer attempts to book on a Friday via API.	POST data with date as a Friday.	HTTP 400 BAD REQUEST. Error message indicating Friday unavailability.	[As observed]	[Pass/ Fail]
UT-API-AP-003	Cancel Appointment (Owner)	Customer cancels their own 'CONFIRMED' appointment.	POST to /api/appointments/{id}/cancel/, authenticated as appointment owner.	HTTP 200 OK. Response shows appointment status as 'CANCELLED'.	[As observed]	[Pass/ Fail]
UT-API-AP-004	Cancel Appointment (Not Owner)	Customer attempts to cancel another customer's appointment.	POST to /api/appointments/{id}/cancel/, authenticated as different Customer.	HTTP 403 FORBIDDEN or HTTP 404 NOT FOUND (depending on permission setup).	[As observed]	[Pass/ Fail]
...	...	...	...	...	...	...

---

4. Integration Test Cases (Examples)

This section describes tests that verify interactions between different modules or components.

Test Case ID	Test Scenario Description	Modules Involved	Preconditions	Steps	Expected Results	Actual Results	Status
IT-001	**Successful End-to-End Appointment Booking by Customer**	CustomUser, Clinic, Room, Doctor, Service, Price, Roster, Appointment models; Appointment Creation API/Logic.	Customer user exists. Clinic, Room (available), Doctor (on roster), Service, Price exist. Doctor available for selected date/shift (not Friday). Clinic has capacity.	1. Customer logs in. 2. Customer selects clinic, date, service. 3. System shows available doctors/shifts/price. 4. Customer selects doctor/shift/room. 5. Customer confirms booking.	Appointment successfully created with 'CONFIRMED' status. Correct price applied. Appointment visible in customer's history and doctor's schedule. Room/Doctor slot becomes unavailable for that time.	[As observed]	[Pass/Fail]
IT-002	**Doctor Schedule Update Reflects in Appointment Availability**	Roster, Doctor, Appointment models; Roster Management API/Logic; Appointment Availabilit	Doctor 'A' initially NOT on roster for Clinic 'X' on Date 'Y', Shift 'Z'.	1. Admin Officer adds Doctor 'A' to Roster for Clinic 'X', Date 'Y', Shift 'Z'. 2. Customer attempts to check availability for Clinic 'X', Date 'Y', Shift 'Z'.	Doctor 'A' should now appear as an available doctor for booking by the customer.	[As observed]	[Pass/Fail]

		y Check API/Logic .					
IT-003	**Manager Sets New Price and it Reflects in New Bookings**	Price, Service, Appointment models; Price Management API/Logic; Appointment Creation API/Logic .	Service 'S' has an existing price for MORNING shift.	1. Manager logs in. 2. Manager updates the MORNING shift price for Service 'S'. 3. Customer attempts to book Service 'S' for a MORNING shift.	The new price set by the Manager should be quoted to the customer and applied to the new appointment. Existing appointments with the old price should remain unchanged.	[As observed]	[Pass/Fail]
...	...	...	...	...	...	...	...

---

5. System Test Cases (GUI Focused - Examples)

This section details end-to-end tests performed via the Graphical User Interface (simulated or actual). These often mirror use cases.

(This section would be more descriptive, as GUI testing is often manual for such projects unless UI automation is implemented, which is typically beyond scope for this kind of module unless specified).

Example Test Case Format:

Test Case ID: ST-CUST-BOOK-001
Test Case Title: Customer successfully books an available appointment.
User Role: Customer
Preconditions:

1. Customer user customer_test with password password123 exists.
2. Male Clinic exists and is active.

3. Room 101 (Normal) in Male Clinic is available for [Specific Future Date - Not Friday] during MORNING shift.
4. Dr. John Doe is on roster for Male Clinic on [Specific Future Date] during MORNING shift.
5. Teeth Cleaning service exists with a price defined for MORNING shift (e.g., 150).
6. Male Clinic has fewer than 10 appointments for the specified date/shift.
   Test Steps:
7. Navigate to the login page.
8. Enter username customer_test and password password123. Click Login.
9. Navigate to the "Book Appointment" section/form.
10. Select Male Clinic from the clinic dropdown.
11. Select [Specific Future Date] from the date picker.
12. Select Teeth Cleaning from the service dropdown.
13. Select MORNING shift.
14. (System should display Dr. John Doe as available and Room 101, and the price 150).
15. Select Dr. John Doe.
16. Select Room 101.
17. Click "Confirm Booking" button.
    Expected Result:
18. A success message "Appointment booked successfully" is displayed.
19. A booking receipt is shown/available with a unique reference, correct doctor, service, date, time, room, and price.
20. The appointment appears in the customer's "My Appointments" list with "CONFIRMED" status.
    Actual Result: [Describe what actually happened]
    Status: [Pass/Fail]
    Notes/Screenshots: [Optional: e.g., "Screenshot of confirmation page attached as Appendix A"]

Test Case ID: ST-ADMIN-ROSTER-001
Test Case Title: Admin Officer successfully adds a doctor to the duty roster.
User Role: Administrative Officer
Preconditions:

1. Admin Officer user admin_officer_test exists.
2. Dr. Jane Smith exists.
3. Kulhudhufushi Clinic exists.
4. Dr. Jane Smith is NOT currently on roster for Kulhudhufushi Clinic on [Specific Future Date - Not Friday] for AFTERNOON shift.
   Test Steps:
5. Login as admin_officer_test.
6. Navigate to "Manage Doctor Rosters".
7. Select "Add New Roster Entry" (or similar).

8. Select Dr. Jane Smith for Doctor.
9. Select Kulhudhufushi Clinic for Clinic.
10. Select [Specific Future Date] for Date.
11. Select AFTERNOON for Shift.
12. Save the roster entry.
    Expected Result:
13. Success message indicating roster entry created.
14. The new roster entry is visible in the roster list for Dr. Jane Smith / Kulhudhufushi Clinic.
15. If a customer now checks availability for that clinic/date/shift, Dr. Jane Smith should be listed.
    Actual Result: [Describe what actually happened]
    Status: [Pass/Fail]

(Add more System Test Cases for other key functionalities and user roles, including tests for business rule violations like trying to book on Friday, surgery room in evening, exceeding clinic capacity, etc.)

---

6. Test Execution Summary & Outputs

6.1. Unit Test Execution (e.g., Pytest Output)

(If you used an automated unit testing framework like pytest-django or Django's built-in test runner, include a summary of the execution here. You can paste a snippet of the command-line output or a summary report.)

Example:
All unit tests were executed using the pytest command.

```
    ============================= test session starts
==============================
platform ... -- Python ...
plugins: django-4.5.2
collected XX items

appointments/tests/test_models.py .....                  [XX%]
appointments/tests/test_api.py .....                    [YY%]
appointments/tests/test_validators.py .....              [ZZ%]

====================== XX passed in X.XXs ========================
```

Summary: A total of XX unit tests were executed, and all XX tests passed.
(Attach full test runner output as an appendix if extensive, or link to it in the code submission).

6.2. Manual System Test Execution Notes

(Provide a summary of the manual system testing execution. If you tracked pass/fail for each ST case above, you can summarize here.)

Example:
Manual system testing was performed by group members following the System Test Cases outlined in Section 5.

- ST-CUST-BOOK-001: Passed
- ST-ADMIN-ROSTER-001: Passed
- ST-CUST-BOOK-FRIDAY-FAIL-001 (Test for booking on Friday): Passed (System correctly prevented booking)
- ... (list other key system tests and their overall status)

All critical path functionalities were tested and verified to be working as per requirements. Minor UI alignment issues were noted but are outside the primary functional scope of this assessment.

---

7. Conclusion

The testing activities conducted for the Island Dental Booking System, encompassing unit, integration, and system tests, have demonstrated that the system generally meets the specified functional requirements and adheres to the defined business rules. Automated unit tests provide a good foundation for regression testing. Manual system tests have validated key user workflows.

Areas for future improvement in testing could include:

- Increased unit test coverage for API error handling scenarios.
- Implementation of automated UI tests for key user journeys.
- Formal performance and security testing if the system were to be deployed in a production environment.