**Summary**

**Unit Testing Approach**

In developing the mobile application for Grand Strand Systems, I implemented a structured unit testing approach for three primary features: **appointments**, **contracts**, and **tasks**.

1. **Appointments**: Tests validated appointment creation, modification, and deletion. I focused on constraints like future dates and preventing overlaps. For example, I tested for overlapping appointments as follows:

@Test

public void testOverlapAppointments() {

Appointment firstAppointment = new Appointment("2024-10-28 10:00", "Team Meeting");

Appointment secondAppointment = new Appointment("2024-10-28 10:30", "Client Call");

appointmentService.addAppointment(firstAppointment);

assertThrows(OverlapException.class, () -> {

appointmentService.addAppointment(secondAppointment);

});

}

1. **Contracts**: Tests checked for mandatory fields and unique identifiers. For instance, the following test verifies the presence of required fields:

@Test

public void testContractMandatoryFields() {

Contract contract = new Contract(null, "2024-12-31");

assertThrows(IllegalArgumentException.class, () -> {

contractService.createContract(contract);

});

}

1. **Tasks**: I tested task creation, updates, and deletions, ensuring valid due dates. For example, to validate due dates:

@Test

public void testTaskDueDateCannotBeInThePast() {

Task task = new Task("Test Task", LocalDate.now().minusDays(1));

assertThrows(IllegalArgumentException.class, () -> {

taskService.addTask(task);

});

}

**Alignment to Requirements**

My unit tests aligned with the software requirements, ensuring that features behaved as specified. For example, the appointment overlaps tests confirmed compliance with the requirement to prevent simultaneous appointments. Tests that checked for duplicates verified the contract’s uniqueness.

**Effectiveness of JUnit Tests**

The quality of my JUnit tests was evident in achieving over **85%** code coverage, ensuring thorough testing of standard and edge cases. This coverage effectively validated potential failures, enhancing overall product reliability.

**Writing JUnit Tests**

Writing JUnit tests involved ensuring technical soundness and efficiency. I employed assertions like assertNotNull() and assertThrows() to validate expected outcomes. For example, validating invalid appointment dates:

@Test

public void testInvalidAppointmentDate() {

assertThrows(IllegalArgumentException.class, () -> {

appointmentService.createAppointment("InvalidDate", "Description");

});

}

To enhance efficiency, I used setup methods (@BeforeEach) to initialize common objects:

@BeforeEach

public void setUp() {

contract = new Contract("ContractID123", "2024-12-31");

}

**Reflection**

**Testing Techniques Employed**

I primarily used **unit testing** **and** **boundary testing**. Unit testing allowed me to validate individual components, while boundary testing ensured that input limits were respected.

**Other Testing Techniques**

I recognized that additional techniques, such as integration and regression testing, could benefit future projects. Integration testing would evaluate the interactions between components, while regression testing would ensure that new changes did not disrupt existing functionality.

**Mindset**

Throughout testing, I maintained a cautious mindset, focusing on potential failure points and employing a systematic review process to minimize bias. Peer reviews helped validate my assumptions and provided a balanced perspective.

**Discipline and Commitment to Quality**

Maintaining discipline is crucial in software engineering to avoid technical debt. I followed my best practices, ensuring thorough documentation and a rigorous testing schedule. Implementing continuous integration with automated testing helped promptly identify issues, fostering a culture of quality.

**Conclusion**

This project underscored the significance of structured testing approaches and various testing techniques. The lessons I learned will enhance my ability to contribute effectively to future software projects.