Whitebox Testing

Group Name: TISM Tech
Ayaan Khan - i220832
Ayaan Mughal - i220861
Mishal Ali - i221291

Overview

Whitebox testing was conducted across all backend microservices: **User Service**, **Hotel Service**, and **Booking Service**. The goal was to achieve high coverage by testing critical functions, APIs, database interactions, and logic flows using unit and integration tests.
Testing was performed using **Jest**, **Supertest.**

What is Covered

- **Core Business Logic**:
  - Critical flows such as user registration, login, loyalty enrollment/rewards, hotel management (CRUD), room availability, booking creation, cancellation, payment processing, and group booking management were extensively tested.

- **API Endpoints**:
  - All major HTTP endpoints were unit-tested and integration-tested using **Supertest**.
  - Request validation, success responses, and error handling paths were covered.

- **Database Operations**:
  - MongoDB model interactions (using Mongoose) such as creating, updating, deleting, and querying documents were tested.
  - Mocked database behavior was used where appropriate to simulate edge cases and failures.

- **Branch Coverage**:
  - Multiple conditions such as success paths, missing data, invalid inputs, not found errors, and server errors were handled in tests.
  - Both positive and negative test scenarios were covered for major controllers.

- **Function and Statement Coverage**:

    - Most controllers, utility functions, and models have above **70%+** function and statement coverage, with some services exceeding **80%**

What is Not Covered and Why

- **Third-party Libraries**:
    - Built-in third-party libraries such as **Mongoose**, **Axios**, **bcrypt**, and **jsonwebtoken** were **mocked** or **assumed to work** correctly without direct unit testing their internals, since these are external and already well-tested libraries.

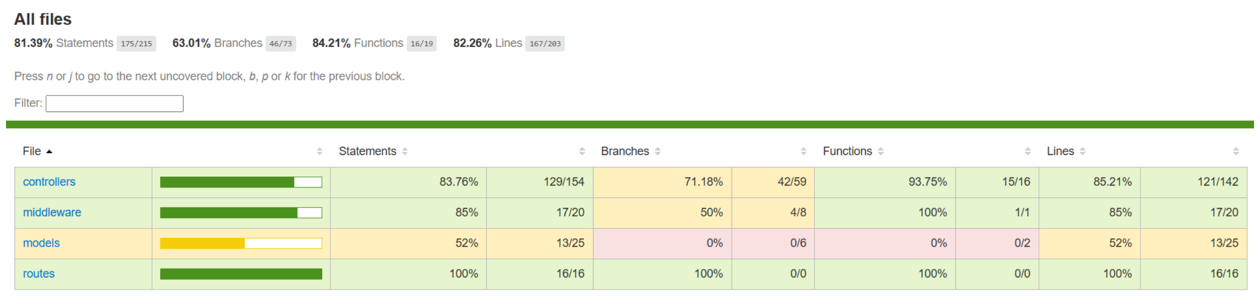
- **Asynchronous Cron Jobs**:
    - Scheduled jobs (cron jobs) that automatically free up room availability after checkout were not exhaustively tested because they run on background schedulers, and mocking long-running timed processes was not prioritized.

- **Error Paths for External Services**:
    - Some error handling paths involving external services (like Room Service or User Service) were not deeply simulated with network errors or timeouts.
    - While API call failures were mocked, full resilience against external service crashes was not fully tested.

- **Frontend**:
    - **No frontend whitebox testing** was performed in this scope. The React frontend was excluded, and only backend services were tested.

Screenshots

- **User Service**



All files

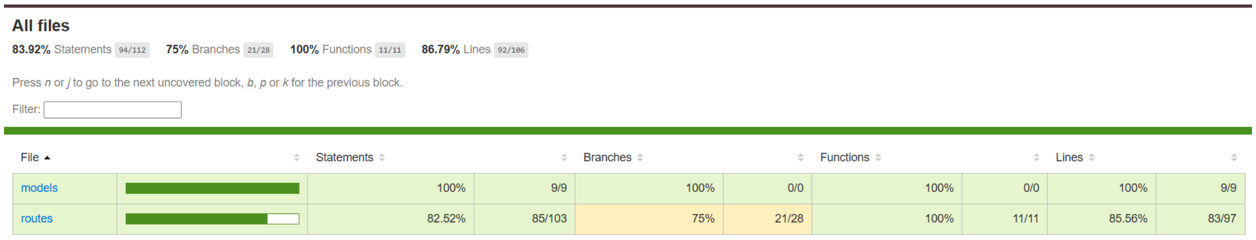
81.39% Statements 175/215 **63.01%** Branches 46/73 **84.21%** Functions 16/19 **82.26%** Lines 167/203

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File		Statements		Branches		Functions		Lines	
controllers		83.76%	129/154	71.18%	42/59	93.75%	15/16	85.21%	121/142
middleware		85%	17/20	50%	4/8	100%	1/1	85%	17/20
models		52%	13/25	0%	0/6	0%	0/2	52%	13/25
routes		100%	16/16	100%	0/0	100%	0/0	100%	16/16

- **Hotel Service**



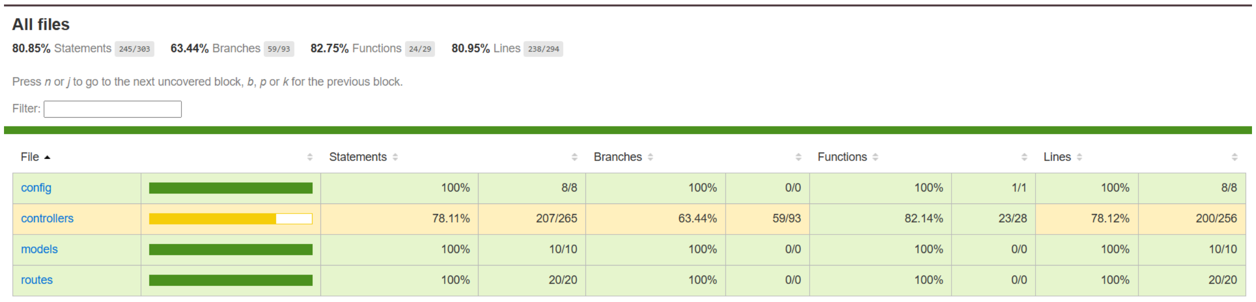
All files
83.92% Statements 94/112 **75%** Branches 21/28 **100%** Functions 11/11 **86.79%** Lines 92/106

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File ▲		Statements		Branches		Functions		Lines	
models		100%	9/9	100%	0/0	100%	0/0	100%	9/9
routes		82.52%	85/103	75%	21/28	100%	11/11	85.56%	83/97

- **Booking Service**



All files
80.85% Statements 245/303 **63.44%** Branches 59/93 **82.75%** Functions 24/29 **80.95%** Lines 238/294

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File ▲		Statements		Branches		Functions		Lines	
config		100%	8/8	100%	0/0	100%	1/1	100%	8/8
controllers		78.11%	207/265	63.44%	59/93	82.14%	23/28	78.12%	200/256
models		100%	10/10	100%	0/0	100%	0/0	100%	10/10
routes		100%	20/20	100%	0/0	100%	0/0	100%	20/20

Each report shows the achieved levels of:

- **Statement Coverage**
- **Branch Coverage**
- **Function Coverage**
- **Line Coverage**

Conclusion

Overall, the backend microservices achieved **strong unit and integration test coverage** across all critical functionalities. Some areas such as third-party libraries and background cron jobs could be tested further for complete resilience, but core business logic is well-validated through comprehensive whitebox testing.
