Learning Outcome 3: Apply a Wide Variety of Testing Techniques and Compute Test Coverage and Yield

3.1. Range of Techniques

Functional Testing:

- Verify that all features behave as expected under normal operating conditions.
- Test end-to-end workflows, including the validateOrder method, pathfinding algorithms, and system responses to input errors.

Edge Case Testing:

- Test boundary values for parameters like LngLat coordinates, invalid regions, order totals, and maximum pizza count (e.g., minimum and maximum allowed values, empty inputs, and invalid bank details).
- Simulate extreme cases, such as orders from locations outside the delivery region or a lot of noflyzones that put the drone in awkward positions.

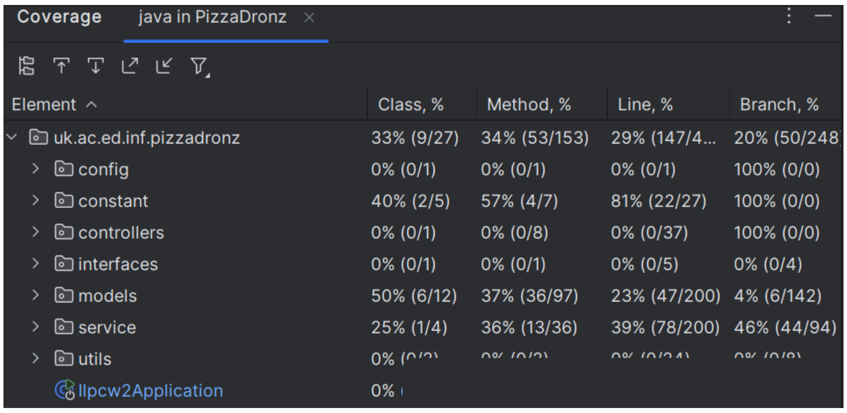
Performance Testing:

- Measure response times and resource utilisation for the delivery path calculation and functionalities.

3.2. Evaluation Criteria for the Adequacy of Testing

Test Coverage:

- The main testing file was OrderValidationCheckerTest, which fetched all of the orders from the API and gave the results. Although we were aiming for a coverage of 80%, due to the time restrictions for this coursework only OrderValidation had automatic tests that resulted in a coverage of 33%. All other endpoints were tested manually through postman and although not ideal, I evaluated that it was the best decision due to the time constraint.



Coverage   java in PizzaDronz

Element	Class, %	Method, %	Line, %	Branch, %
uk.ac.ed.inf.pizzadronz	33% (9/27)	34% (53/153)	29% (147/4...	20% (50/248
config	0% (0/1)	0% (0/1)	0% (0/1)	100% (0/0)
constant	40% (2/5)	57% (4/7)	81% (22/27)	100% (0/0)
controllers	0% (0/1)	0% (0/8)	0% (0/37)	100% (0/0)
interfaces	0% (0/1)	0% (0/1)	0% (0/5)	0% (0/4)
models	50% (6/12)	37% (36/97)	23% (47/200)	4% (6/142)
service	25% (1/4)	36% (13/36)	39% (78/200)	46% (44/94)
utils	0% (0/2)	0% (0/2)	0% (0/24)	0% (0/8)
llpcw2Application	0%			

Risk Coverage:

- I decided to Prioritise tests for the most critical feature, order validation, since this was the most complex, other endpoints needed less testing as they were more simple and had much less input.
- This was done by executing unit tests for OrderValidation and manual tests for the pathfinding and other endpoints.

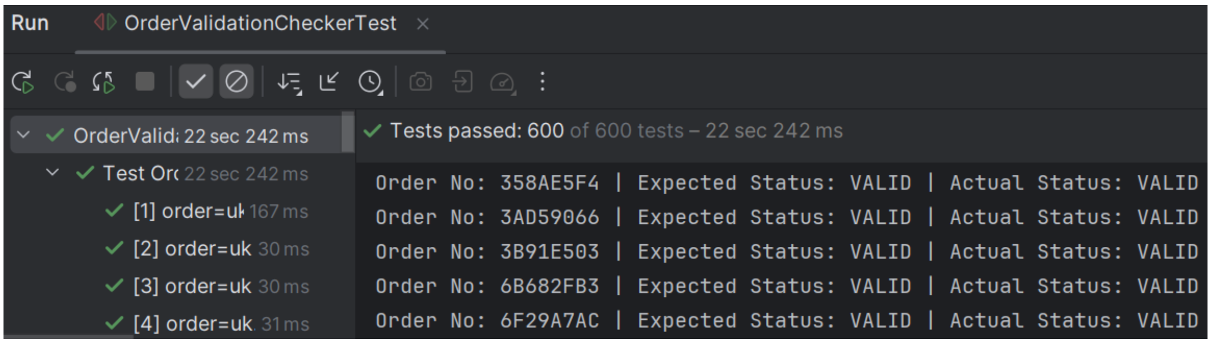
3.3. Results of Testing

Performance Metrics:

- Successfully executed 600 unit tests for validateOrder functionality.
- Verified system handles delivery path calculations for all 7 defined restaurant paths without errors.

Pass/Fail Rates:

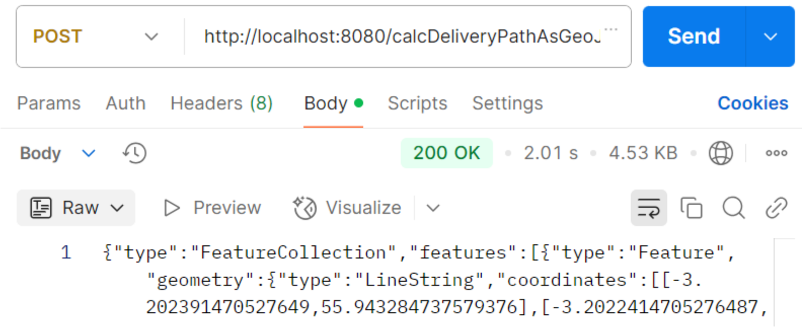
- 100% of unit tests passed successfully.



Tests passed: 600 of 600 tests – 22 sec 242 ms

```
Order No: 358AE5F4 | Expected Status: VALID | Actual Status: VALID
Order No: 3AD59066 | Expected Status: VALID | Actual Status: VALID
Order No: 3B91E503 | Expected Status: VALID | Actual Status: VALID
Order No: 6B682FB3 | Expected Status: VALID | Actual Status: VALID
Order No: 6F29A7AC | Expected Status: VALID | Actual Status: VALID
```

- 6/7 restaurant paths had a runtime of < 1 second (one restaurant that was much further away had a runtime of ~2 seconds). These results are much better than the target of 60s.



```
{"type":"FeatureCollection","features":[{"type":"Feature",
  "geometry":{"type":"LineString","coordinates":[[-3.
  202391470527649,55.943284737579376],[-3.2022414705276487,
```

3.4. Evaluation of Results

Unit Test Adequacy:

- Unit tests covered the majority of expected outputs and scenarios.
- Identified additional scenarios requiring tests, such as handling invalid cases that were not thought of.

Pathfinding Robustness:

- Pathfinding tests were limited to predefined routes; dynamic and edge-case scenarios were not robustly tested.
- Identified the need for additional tests covering alternate route calculations, recalculations during failures.

Manual testing:

- Manual testing was done throughout the entire system to verify endpoints such as uuid, and calcdeliverypath which was not possible to test systematically.
- The addition of manual testing on top of unit tests helped provide comfort that all functional and non functional requirements from learning outcome 1 were achieved.

Possible improvements

1. **Expand Unit Test Cases to increase coverage:**
    - Path finding route scenarios to test more than was possible manually.
    - Unit testing for other endpoints such as