

16-6-2025

Version: 1.0

Test Report

Module

ICT.GP.PRJCT.V22_2425

Coach

[Redacted]

Education

Windesheim Zwolle HBO-ICT Software Engineering

Students

Bark, Ivan (s1169347)

[Redacted]

Version management

Version	Date	Description	Remarks
1.0	16-06-2025	First final version	Send for assesment

Distribution

Name	Role	Date	Version
	Coach	18-06-2025	1.0

Contents

Introduction.....	4
1. Tested Functionalities	5
2. Unit and e2e test cases	6
2.1. MachineController (End-to-End Tests).....	6
2.1.1. Scenario 1: Fetching all machines	6
2.1.2. Scenario 2: Fetching machine by ID	6
2.1.3. Scenario 3: Fetching node of a machine.....	6
2.2. NavigationService (Unit Tests).....	7
2.2.1. Scenario 1: Getting a graph	7
2.2.2. Scenario 2: Getting a node	7
2.2.3. Scenario 3: Calculating a path between two nodes	7
2.2.4. Scenario 4: Finding the nearest node	7
2.3. PathfindService (Unit Tests)	8
2.3.1. Scenario 1: Path between two connected nodes	8
2.3.2. Scenario 2: Shortest path in a 3-node graph	8
2.3.3. Scenario 3: No available path	8
2.3.4. Scenario 4: Diagonal path (3D heuristic)	8
2.4. FLV Class (Unit Tests)	9
2.4.1. Scenario 1: Initialization of Fuzzy Logic Variable	9
2.5. MQTTController (End-to-End Tests)	10
2.5.1. Scenario 1: Check MQTT client connection status.....	10
2.5.2. Scenario 2: Connect the MQTT client	10
2.5.3. Scenario 3: Disconnect the MQTT client.....	10
2.5.4. Scenario 4: Reconnect the MQTT client.....	10
2.5.5. Scenario 5: View current subscriptions	10
2.5.6. Scenario 6: Get latest sensor value.....	11
2.5.7. Scenario 7: Get debug HTML page for a sensor.....	11
2.5.8. Scenario 8: Unsubscribe from a specific topic	11
2.5.9. Scenario 9: Unsubscribe from all topics.....	11
3. Manual test approach	12
3.1. Marker detection and tracking (AprilTags)	12
3.2. Data panel visibility and updates.....	12
3.3. Navigation.....	13
4. Test Report Summary.....	14

4.1.	Unit and end-to-end tests (as of 16-06-2025)	14
4.2.	Manual tests (as of 16-06-2025)	14

Introduction

This test report describes the tests conducted on the backend of the *AR Maintenance and Inspection Assistant* project. The backend is responsible for handling data about machines, pathfinding in a 3D environment, fuzzy logic calculations, and real-time MQTT communication with sensors. Since this backend plays a crucial role in supporting the AR interface used for maintenance tasks, making sure everything works correctly was important.

To validate the functionality, both unit tests and end-to-end (e2e) tests were written. Unit tests were mainly used to check individual services like the pathfinding logic and fuzzy variable classes. The e2e tests were used to test the API endpoints and simulate actual use cases, such as fetching machine info or interacting with the MQTT system.

The tests help confirm that the backend responds correctly to valid input, handles errors properly, and behaves as expected under common scenarios. The results of these tests are summarized in this document. All tests were run locally using a development environment with a test MQTT broker and mock data where needed.

This report shows that the backend is functionally ready to support the AR assistant's core features, and it gives insight into what parts were tested and how.

1. Tested Functionalities

Unit, end-to-end and manual tests were used to validate various components of the application. These tests focus on ensuring the correctness, reliability, and consistency of both backend services, frontend-facing endpoints and the Unity AR frontend.

Tested functionalities include:

- Fetching all machines
- Fetching a machine by ID
- Fetching the node of a machine
- Handling 404 errors when machines/nodes are not found
- Fetching and validating graph data
- Fetching individual nodes
- Pathfinding between graph nodes
- Nearest node calculation
- Fuzzy logic variable (FLV) initialization and behavior

Furthermore, the frontend functionalities will be tested manually. The tested functionalities on the frontend include:

- SSE streaming
- Marker detection and tracking (AprilTags)
- Data panel visibility and updates
- Sensor graph rendering
- Machine info display (ID, name, desc.)
- Navigation

2. Unit and e2e test cases

As described in the test strategy document, unit tests and end-to-end tests will be executed. The test cases are described in this chapter. Per test case, the performed actions and expected results are listed. To execute these tests, please follow the instruction in the README.md file.

2.1. MachineController (End-to-End Tests)

2.1.1. Scenario 1: Fetching all machines

Actions:

- Call GET /machines

Expected Results:

- Response code 200 OK
- Response body is an array
- Each machine object contains id and name properties

2.1.2. Scenario 2: Fetching machine by ID

Actions:

- Call GET /machines/:id with an existing ID

Expected Results:

- Response code 200 OK
- Machine object includes correct id and name

Error Handling:

- Call with a non-existing ID returns 404 Not Found

2.1.3. Scenario 3: Fetching node of a machine

Actions:

- Call GET /machines/:id/node

Expected Results:

- Response code 200 OK
- Response body includes nodeId

Error Handling:

- Non-existing ID returns 404 Not Found

2.2. NavigationService (Unit Tests)

2.2.1. Scenario 1: Getting a graph

Actions:

- Call getGraph with a valid ID
- Call getGraph with an invalid ID

Expected Results:

- Returns correct Graph object or null if not found

2.2.2. Scenario 2: Getting a node

Actions:

- Call getNode with valid/invalid node ID

Expected Results:

- Returns correct GraphNode or null

2.2.3. Scenario 3: Calculating a path between two nodes

Actions:

- Call getPath with valid node IDs on the same graph
- Use both SHALLOW and DEEP modes

Expected Results:

- Returns path as full node objects or only node IDs
- Throws error if:
 - One of the nodes is missing
 - Nodes belong to different graphs
 - Graph is not found

2.2.4. Scenario 4: Finding the nearest node

Actions:

- Call getNearestNode with coordinates and return type

Expected Results:

- Returns node ID for SHALLOW
- Returns full node object for DEEP
- Returns null if no node is found

2.3. PathfindService (Unit Tests)

2.3.1. Scenario 1: Path between two connected nodes

Actions:

- Setup graph with two directly connected nodes
- Run A* pathfinding

Expected Results:

- Returns correct path [A, B]

2.3.2. Scenario 2: Shortest path in a 3-node graph

Actions:

- Setup graph with nodes $A \rightarrow B \rightarrow C$
- Run A* from A to C

Expected Results:

- Returns [A, B, C]

2.3.3. Scenario 3: No available path

Actions:

- Create two nodes with no edge
- Run A*

Expected Results:

- Returns an empty array

2.3.4. Scenario 4: Diagonal path (3D heuristic)

Actions:

- Nodes with spatial diagonal placement
- Run A*

Expected Results:

- Returns correct path [A, B]

2.4. FLV Class (Unit Tests)

2.4.1. Scenario 1: Initialization of Fuzzy Logic Variable

Actions:

- Create FLV with:
 - Name
 - Labels
 - X-values

Expected Results:

- GetXvalues() returns the correct array
- GetLabels() returns the correct labels (test implied)

2.5. MQTTController (End-to-End Tests)

2.5.1. Scenario 1: Check MQTT client connection status

Actions:

- Call GET /mqtt/client/status

Expected Results:

- Status code: 200 OK
- Body contains "connected" or "not connected"

2.5.2. Scenario 2: Connect the MQTT client

Actions:

- Call GET /mqtt/client/connect

Expected Results:

- Status code: 200 OK
- Body contains "Connected"

2.5.3. Scenario 3: Disconnect the MQTT client

Actions:

- Call DELETE /mqtt/client/disconnect

Expected Results:

- Status code: 200 OK
- Body contains "Disconnected"

2.5.4. Scenario 4: Reconnect the MQTT client

Actions:

- Call GET /mqtt/client/connect again

Expected Results:

- Status code: 200 OK

2.5.5. Scenario 5: View current subscriptions

Actions:

- Call GET /mqtt/subscriptions/status

Expected Results:

- Status code: 200 OK

2.5.6. Scenario 6: Get latest sensor value

Actions:

- Call GET /mqtt/latest/:machineId/:sensor with valid values

Expected Results:

- Status code: 200 OK
- Body contains a defined text value

2.5.7. Scenario 7: Get debug HTML page for a sensor

Actions:

- Call GET /mqtt/debug/:machineId/:sensor

Expected Results:

- Status code: 200 OK
- Body contains <html

2.5.8. Scenario 8: Unsubscribe from a specific topic

Actions:

- Call DELETE /mqtt/:machineId/:sensor

Expected Results:

- Status code: 200 OK
- Body contains "Unsubscribed"

2.5.9. Scenario 9: Unsubscribe from all topics

Actions:

- Call DELETE /mqtt/all

Expected Results:

- Status code: 200 OK
- Body contains "Unsubscribed"

3. Manual test approach

In this chapter all manual tests will be detailed and explained. Per test there will be described how the test should be performed and what the expected outcome is. This will be detailed per functionality.

3.1. Marker detection and tracking (AprilTags)

Test	Actions	Expected results
Marker detection	1. Open the app	You should hear a short alert sound
	2. Wait for initialization	You should feel a short vibration
	3. Generate AprilTag with via this website - Set tag family to 36h11 - Set to ID 2	A panel should spawn
	4. Scan the tag with the app	
Marker tracking	1. Perform Marker Detection first	The spawned panel should be moving with the tag
	2. While looking at the tag with your phone, move the tag.	

3.2. Data panel visibility and updates

Test	Actions	Expected results
Data panel spawns	1. Perform Marker Detection	A panel should be spawned
SSE streaming	1. Perform Marker Detection	Sensor data should be displayed on the left panel
		The visible sensors should correspond with the machine with the same ID as the tag
		The data should be updating live
Panel positioning	1. Perform Marker Detection 2. Move up and down 3. Then move around the scanned tag	The panel should always be facing the user
		The panel should stay at the same height as the user
Sensor graph rendering	1. Perform Marker Detection	A line graph panel should be visible on the right
		The number of lines should correspond with the number of sensors
		The legend should reflect the sensors of the machine

Test	Actions	Expected results
		The data on the graph should correspond with the data visible on the left panel
Static data visibility	1. Perform Marker Detection	The machine name, ID and description should be visible
		The data should be from the machine with the same ID as the tag
Fuzzy panel	1. Perform Marker Detection with tag ID 0	One panel should be visible
		The panel should reflect a prioritised maintenance list

3.3. Navigation

Test	Actions	Expected results
Start navigation	1. Perform Marker Detection with tag ID 0	A node should be visible somewhere around you
Path following	2. Perform Start Navigation 3. Walk towards node	The node should disappear
		A new node should spawn

4. Test Report Summary

4.1. Unit and end-to-end tests (as of 16-06-2025)

Class/Tested Component	Scenario Description	Status
MachineController	GET all, by ID, and node with 404 handling	passed
NavigationService	Graph and node retrieval, pathfinding, nearest node	passed
PathfindService	A* path logic with multiple node configurations	passed
FLV Class	Initialization with values and label validation	passed
MQTTController	MQTT client status, connect/disconnect, subscriptions, sensor endpoints	passed

4.2. Manual tests (as of 16-06-2025)

Test	Expected results	Status
Marker detection	You should hear a short alert sound	passed
	You should feel a short vibration	passed
	A panel should spawn	passed
Marker tracking	The spawned panel should be moving with the tag	passed
Data panel spawns	A panel should be spawned	passed
SSE streaming	Sensor data should be displayed on the left panel	passed
	The visible sensors should correspond with the machine with the same ID as the tag	passed
	The data should be updating live	passed
Panel positioning	The panel should always be facing the user	passed
	The panel should stay at the same height as the user	passed
Sensor graph rendering	A line graph panel should be visible on the right	passed
	The number of lines should correspond with the number of sensors	passed
	The legend should reflect the sensors of the machine	passed

Test	Expected results	Status
	The data on the graph should correspond with the data visible on the left panel	passed
Static data visibility	The machine name, ID and description should be visible	passed
	The data should be from the machine with the same ID as the tag	passed
Fuzzy panel	One panel should be visible	passed
	The panel should reflect a prioritised maintenance list	failed *the list is visible, but the styling is incorrect **unfortunately, no time left to fix, should be addressed in the future
Start navigation	A node should be visible somewhere around you	failed *a line should have been visible to the node **unfortunately, no time left to fix, should be addressed in the future
Path following	The node should disappear	passed
	A new node should spawn	passed