System Verification and Validation Plan for Software Engineering

Team #13, ARC
Avanish Ahluwalia
Russell Davidson
Rafey Malik
Abdul Zulfiqar

November 4, 2024

Revision History

Date	Version	Notes
2024/11/04	1.0	Initial VnV Plan
2025/03/11	1.1	Unit tests added
2025/03/24	1.2	Refinements made based on critique
2025/03/25	1.3	Removed references to features we are no longer implemented; Fixed formatting issues

Contents

List of Tables

The purpose of this Verification and Validation document is to outline the testing process we'll use to make sure our project meets all its requirements and functions correctly. Verification checks that we're building the product according to our design, while validation ensures the product meets the users' needs and performs as expected. This document will describe both functional and non-functional tests, which help confirm that the project is reliable, safe, and easy to use. By following this VnV plan, we can identify and fix any issues early, ensuring the final product is high-quality and ready for users.

1 General Information

The section covers general information about the AR application, *Realm*, and this document.

1.1 Summary

The software being tested is "Realm", an AR touring platform. The platform allows users to place AR objects in the real world for others to see and allows organizations to create AR tours to provide users with high-quality experiences. This document will include tests for all requirements as well as the design plan for validation and verification.

1.2 Objectives

The primary objectives for the validation and verification of Realm are as follows:

- **Ensure accuracy of AR functionality**: One of our core goals is to build confidence in the AR features' correctness, confirming that the AR objects can be placed effectively. We also want to ensure that overlays align accurately with real-world elements, and behave as intended in various lighting and spatial conditions.

- **Verify usability and engagement**: Usability testing will focus on confirming that users can seamlessly navigate through the app, and interact with AR objects and other users. This includes ease of use,

intuitiveness of the AR controls, and overall satisfaction with the social sharing features.

- **Data security and privacy assurance**: Given the app's AR elements, it's essential to ensure that all user data, including images and interaction history, is securely managed and protected from unauthorized access.

Due to limited resources, the following objectives will not be prioritized in this phase:

- **Universal device compatibility**: Testing will focus on popular AR-enabled devices, but comprehensive performance validation across all devices and OS versions is outside the scope of this initial release.

- **Verification of third-party libraries**: The project relies on AR libraries and frameworks from external providers. We assume these libraries have been validated by their developers, allowing us to concentrate on verifying custom AR functionality and app-specific features.

1.3 Challenge Level and Extras

The challenge level for the Realm project is **general**. The extras that are going to be tackled in this project are **user documentation** and **usability testing**. We chose these two as they are deemed useful to the user's interaction with our app. The user documentation will help the user to learn to use the app simply, and the usability testing will help us to deliver a product that is acceptable and tailored to the use case of our users.

1.4 Relevant Documentation

This document contains information garnered from all the previous documentation referenced below:

- The Problem Statement is a good reference to understand the context in which all of the requirements come from. The plans in this document should reflect the goals outlined in the Problem Statement.
- The Development Plan has a list of team member roles which are should be considered when writing about a team member's testing role.
- The SRS contains all of the functional (FR) and non-functional (NFR) requirements that were found in the elicitation process and those that were found after the initial version. This document strives to have a plan for each of these requirements.
- The Hazard Analysis has many considerations for potential hazards and mitigation strategies to reduce the project risk. There are many new FRs and NFRs that were derived from this analysis that need to be verified and validated.
- The VnV Plan itself is also subject to verification as outlined by the Verification and Validation Plan Verification Plan sub-section.

2 Plan

This section outlines the structured plan for verifying and validating our project at each stage of development. It first lists the roles of the team members involved in verification, the strategies for ensuring the accuracy of our requirements, design, and implementation, as well as the tools we'll use for automated testing. Each part is designed to ensure our project meets quality standards and fulfills its intended purpose.

2.1 Verification and Validation Team

Team Member	Role	Responsibilities
Russell	Configuration Specialist	Configure Roslyn analyzers for static code analysis, ensuring they meet project needs
Abdul	Dynamic Testing Lead	Research and implement the Unity Test Framework for Unity Editor Mode, focusing on dynamic testing
Avanish	Dynamic Testing Assistant	Support Abdul in implementing the Unity Test Framework for Play Mode, ensuring comprehensive test coverage
Rafey	Static Analysis Specialist	Select and configure a static analysis tool (e.g., SonarQube), manage static code quality checks

Table 1: Verification and Validation Testing Team Roles and Responsibilities

2.2 SRS Verification Plan

For the SRS Verification Plan, we will adopt a structured approach that includes peer reviews, systematic checks, and feedback from our TA. Initially, ad hoc feedback will be gathered from team members to identify any immediate issues or ambiguities in the SRS. We will then conduct a detailed review session where each team member inspects specific sections of the document using a predefined checklist focused on clarity, completeness, and alignment with project goals. We will also meet with our TA to discuss and gather further feedback. An issue tracker will be used to document all feedback, allowing us to systematically address and resolve identified issues.

2.3 Design Verification Plan

- Internal Team Review
- Peer Review
- TA Review
- Requirements Coverage Check

2.4 Verification and Validation Plan Verification Plan

Once all the plans outlined in this section (Plan) are completed, they go will through a review process such that components that have no validation/verification method are discovered. The following process will take place:

1. Initial internal review by all group members
    - Before the deadline for this document, the group will meet to look over the entire document in the search for components of the project that were overlooked in terms of testing.
    - A checklist of all the functional and non-functional requirements from the SRS will be used to ensure each of them have a plan to be tested.
    - All plans in this sub-sections will be checked against the grading rubric. This will act as a checklist for the expected level of detail for each plan.
2. Peer-Review by an external group
    - As part of this deliverable another group will look over the plans from an outside perspective and give feedback on components we may have missed or were not sufficiently covered.
    - These suggestions will be made through GitHub Issues on the repo.
3. TA review
    - When the TA reviews this deliverable, they will provide feedback for plans that they believe could use some reconstruction.

2.5 Implementation Verification Plan

The primary method of validation for the implementation of the system will be the functional, non-functional, and unit tests described in this document. In addition to the static testing and code review specified in the non-functional tests, we will have a code walkthrough for each module to verify adherence to the design, and the completeness of the implementation. In these walkthroughs we will use our MIS as a checklist and go through module implementations checking off correctly implemented elements.

2.6 Automated Testing and Verification Tools

Refer to section 10 of the development plan for the list of tools. We will use the NUnit-based Unity Testing framework for unit testing and code coverage metrics. Continuous testing will be done by GitHub actions using Unity Builder and Unity Test Runner from the GameCI open source project.

2.7 Software Validation Plan

To validate the software throughout development, we will hold bi-weekly developer meetings. In each meeting, members will present their additions to the codebase and explain how these changes impact the functionality of the targeted components. Each team member will provide a walkthrough of the modified component through a live demo. During these meetings, the team will also ensure that no functional requirements have been inadvertently removed.

To further validate that all components function as intended, we will use formal methods, which include a suite of test cases, user testing, and surveys.

The suite of test cases will consist of *unit tests*, *module tests*, and *system tests*, designed to verify that each component and the overall system meet functionality requirements.

User testing will be conducted by individuals outside the development team, and results will be evaluated using checklists based on the functional and non-functional requirements outlined in the Software Requirements Specification (SRS).

Finally, surveys will be administered to test users, allowing them to provide feedback on user experience and satisfaction with the software's functionality.

3 System Tests

This section contains the descriptions of all system tests spanning functional and non-functional aspects of the application.

3.1 Tests for Functional Requirements

This section focuses on test cases pertaining to the functional requirements.

3.1.1 Tutorial

This section focuses on verifying the user interactions with the app's built-in tutorial. These tests ensure that users will have a fluid experience when finding and completing the tutorial. Given that this is the main instruction method provided by the app, it must work well otherwise users will likely stop using the app. To make it as comprehensible as possible, the tutorial has an interactive experience for every major feature as relayed in (*TU-FR1*) and (*TU-FR5*). All functional requirements listed under Tutorial (TU) in the SRS **SRS** have at least one test plan below.

1. **Name:** Tutorial opens on account creation
   Test ID: Test-TU1
   Control: Manual
   Initial State: User has opened the app.
   Input: User creates an account.
   Output: A pop-up appears that prompts the user to take the tutorial.
   Test Case Derivation: Users who have just created an account are likely new to the app and should have some guidance on the features through the built-in tutorial.
   How test will be performed: A tester will manually create an account within the app and check if the popup appears as expected.

2. **Name:** Tutorial can be opened through the settings screen
   Test ID: Test-TU2
   Control: Manual
   Initial State: User logged in.
   Input: User navigates to settings screen.
   Output: There is an option to open the tutorial.
   Test Case Derivation: Users may be having trouble with a major feature and need a refresher on how it works. They might also want to explore features they have not used yet.
   How test will be performed: A tester will manually navigate to the settings screen using the in-built navigation and look for the tutorial option.

3. **Name:** Tutorial involves all major features
   Test ID: Test-TU3
   Control: Manual
   Initial State: User logged in, on tutorial screen.
   Input: User traverses through every step of the tutorial.
   Output: All major app features were seen in the tutorial.
   Test Case Derivation: Ensure that the user has been introduced to the mechanics behind every major feature.
   How test will be performed: A tester will manually go through all the tutorial steps marking off each major app feature in a checklist.

4. **Name:** User can exit the tutorial at any time
   Test ID: Test-TU4
   Control: Manual
   Initial State: User logged in, on tutorial screen.
   Input: User repeatedly goes through the tutorial and attempts to exit at each unique state.
   Output: The tutorial can exit in every state.
   Test Case Derivation: A user may want to learn a subset of the major features instead of going through them all. As such, they should be able to leave the tutorial at any point instead of forcing them to complete it.
   How test will be performed: A tester will manually go through all the tutorial states and marking off which states allow a user to leave the tutorial.

5. **Name:** The tutorial is interactive in a sandbox environment
   Test ID: Test-TU5
   Control: Manual
   Initial State: User logged in, on tutorial screen.
   Input: User partially attempts each step of the tutorial by interacting with the sandbox environment.
   Output: Each step of the tutorial is interactive.
   Test Case Derivation: The user experience is improved when they can directly interact with a major feature instead of the app just displaying text describing its function.
   How test will be performed: A tester will manually go through the tutorial and attempt to interact with the specific feature at each step within the sandbox environment. They don't have to complete the interaction specified in the step but it should be apparent that the major feature is working properly in the environment.

3.1.2 Realm Testing

1. **Name:** Validate AR Object Perspective Adjustment
   Test ID: Test-RI1
   Type: Functional, Manual
   Initial State: Realm interface is open, AR objects are displayed in the user's vicinity on the camera feed.
   Input/Condition: Tester changes their position and angle in relation to an AR object.
   Output/Result: The AR object adjusts perspective appropriately, reflecting the new camera position and angle.
   How test will be performed: The tester will move around the AR object, observing whether it adjusts in real-time and maintains correct perspective relative to the user's view.

2. **Name:** Validate AR Object Clutter Management
   Test ID: Test-RI2
   Type: Functional, Manual
   Initial State: Realm interface is open, tester is in an area with overlapping AR object instances
   Input/Condition: Tester moves camera over a crowded area where multiple AR objects are present.

Output/Result: The interface selectively displays a manageable number of AR objects without overwhelming the user's view.
How test will be performed: Tester observes the interface to confirm only a few objects are displayed at once, reducing clutter in the view.

3. **Name:** Validate AR Object Placement Accuracy
   Test ID: Test-RI3
   Type: Functional, Automated and Manual
   Initial State: Realm interface is open, test AR object instance is nearby
   Input/Condition: Test AR object instance is placed with a known alignment in the real world, and reference screenshots
   Output/Result: Test AR object appears in correct position and orientation as expected from the known alignment and reference screenshots, position also matches stored object instance data
   How test will be performed: Automated tests will compare object instance data, while a manual test will involve the tester visually confirming the position accuracy in the camera view.

4. **Name:** Validate Object Placement Workflow Control
   Test ID: Test-RI6
   Type: Functional, Manual
   Initial State: Realm interface is open.
   Input/Condition: Tester attempts to access the object placement workflow via the provided control.
   Output/Result: Tester is successfully redirected to the object placement workflow.
   How test will be performed: Tester selects the object placement control and verifies redirection to the appropriate workflow screen.

5. **Name:** Validate Nearby Tour Indication
   Test ID: Test-RI8
   Type: Functional, Manual
   Initial State: Realm interface is open, tester is near the starting point of a tour.
   Input/Condition: Tester moves within range of the tour start point.
   Output/Result: The interface displays a clear indication of the nearby tour and a link to the tour preview.

How test will be performed: Tester observes if the indication and link appear when near the tour start point.

6. **Name:** Validate Hazard Warning
   Test ID: Test-RI9
   Type: Functional, Manual
   Initial State: Realm interface is open, tester is approaching a real-world hazard.
   Input/Condition: Tester moves closer to a hazard in real space.
   Output/Result: Interface displays a clear warning when user approaches the hazard.
   How test will be performed: Tester approaches a wall with the realm interface open, and verifies that a warning appears.

7. **Name:** Validate Offline Mode for Interactive Components
   Test ID: Test-RI10
   Type: Functional, Manual
   Initial State: Realm interface is open and disconnected from the internet.
   Input/Condition: Tester attempts to interact with various components of the interface in offline mode.
   Output/Result: Interactive components function normally, but location-based features are disabled.
   How test will be performed: Tester verifies the functionality of object scanning and the unavailability of object placement, maps, and other internet-dependent features.

3.1.3 Object Placement Testing

1. **Name:** Validate Object Selection Stage
   Test ID: Test-OP1
   Type: Functional, Manual
   Initial State: Tester has progressed object placement workflow to object selection stage
   Input/Condition: Tester selects object from one of: inventory, new object scan, new prompt generation
   Output/Result: Interface successfully proceeds to sub-realm selection step with the selected object

How test will be performed: Tester initiates object placement workflow and progresses to object selection step. They then use one of the object selection methods to select an object and validate that the interface moves on to sub-realm selection with the selected object. They select the option to return to object selection, and repeat the process for all object selection methods.

2. **Name:** Validate Object Placement Stage
   Test ID: Test-OP3
   Type: Functional, Manual
   Initial State: Tester has progressed object placement workflow object placement stage
   Input/Condition: Tester rotates, resizes, and translates the object in real space, then confirms placement.
   Output/Result: Rotation, resizing, and translating are all functional, and the AR object is positioned accurately in real space with the correct orientation, aligning with the tester's intent.
   How test will be performed: Tester initiates object placement workflow and progresses to object placement step. Tester rotates, resizes, and translates the object, verifying visually that it appears in the intended location and orientation in the real-world view, and finally confirms the placement of the object.

3. **Name:** Validate Object Instance Storage
   Test ID: Test-OP4
   Type: Functional, Manual
   Initial State: Tester has just placed an AR object instance
   Input/Condition: Tester checks the AR object instance database
   Output/Result: The AR object instance that the tester placed is present with correct details, including object type, sub-realm(s), position, and orientation.
   How test will be performed: Tester completes the object placement workflow then checks the AR object instance database and verifies the presence and correctness of their newly created object instance.

4. **Name:** Validate Area Based Placement Limit
   Test ID: Test-OP5
   Type: Functional, Automated and Manual
   Initial State: Test user has sufficient AR object instances recorded

to reach the object placement limit of an area
Input/Condition: Tester attempts to place another object in the same area
Output/Result: System prevents additional placements once the area limit is reached, displaying a relevant warning.
How test will be performed: Automated test script creates AR object instance entries in database to reach limit. Tester manually attempts to place another object in selected area, and validates that they are prevented from doing so, and are presented with a warning.

5. **Name:** Validate Time Based Placement Limit
   Test ID: Test-OP6
   Type: Functional, Automated and Manual
   Initial State: Test user has sufficient AR object instances recorded to reach the time-based object placement limit for a user
   Input/Condition: Tester attempts to place another object within a short period
   Output/Result: System restricts further placements once the time-based limit is reached, displaying a relevant warning.
   How test will be performed: Automated test script creates AR object instance entries in database to reach limit. Tester manually attempts to place another object within a short period, and validates that they are prevented from doing so, and are presented with a warning.

6. **Name:** Validate Automated Retry for Failed Object Storage
   Test ID: Test-OP7
   Type: Functional, Automated
   Initial State: System is running, with simulated conditions preventing initial object storage (e.g., network issues).
   Input/Condition: User places an object, but initial storage attempt fails due to simulated conditions.
   Output/Result: System automatically retries object storage until success or retry limit reached.
   How test will be performed: Simulate a storage failure on the initial attempt, monitoring logs to confirm retry attempts are made until storage is successful, validating the success scenario. Repeat this process, but simulate continued storage failure, and verify that the number of retries attempted is equal to the retry limit.

3.1.4 Database Testing

1. **Name:** Validate Periodic Database Backup
   Test ID: Test-DB1
   Type: Functional, Automated
   Initial State: System is running, database is available, periodic backup is set up.
   Input/Condition: Periodic backup run is completed.
   Output/Result: Automated monitor verifies that the database backup is present and correct.
   How test will be performed: An automated monitor will wait for periodic backups, then restore the database from the backup in a sandbox environment and check that the data is present as expected.

2. **Name:** Validate Database Encryption
   Test ID: Test-DB2
   Type: Functional, Automated
   Initial State: System is running, database is available.
   Input/Condition: Command to check encryption status is inputted into DBMS for all databases
   Output/Result: DBMS response shows that all databases are encrypted
   How test will be performed: Will depend on the database platform used, for example on SQL Server the following query would be ran, and the output would be checked:
   SELECT db_name(database_id), encryption_state
   FROM sys.dm_database_encryption_keys;

3.1.5 Tour Management

This section focuses on testing the *Organization User* side of the tours functionality within the app. This includes creating, modifying, and publishing tours so that *General Users* can use them. The main requirement for tour management is the ability to add metadata and objects to an area (*TM-FR4*) which is covered by *Test-TM3* and *Test-TM8*. All the other functional requirements from the Tour Management (TM) section in the SRS **SRS** have at least one test plan below.

1. **Name:** *Organization Users* can access tour management screen

Test ID: Test-TM1
Control: Manual
Initial State: User logged in as a *Organization User*.
Input: User attempts to navigate to tour management screen.
Output: The tour management screen is reachable.
Test Case Derivation: Only users who are part of a verified organization will have the ability to create/edit/delete tours for their organization. *Organization Users* should have be able to access the tour management screen.
How test will be performed: A tester will manually login as a *Organization User* and attempt to see the tour management screen in the navigation.

2. **Name:** *General Users* can NOT access the tour management screen
   Test ID: Test-TM2
   Control: Manual
   Initial State: User logged in as a *General User*.
   Input: User attempts to navigate to tour management screen.
   Output: The tour management screen is hidden from user.
   Test Case Derivation: Only users who are part of a verified organization will have the ability to create/edit/delete tours for their organization. *General Users* should not have the ability to do any tour management.
   How test will be performed: A tester will manually login as a *General User* and attempt to see the tour management screen in the navigation.

3. **Name:** *Organization Users* can create a customized tour
   Test ID: Test-TM3
   Control: Manual
   Initial State: User logged in as a *Organization User*, on tour management screen.
   Input: User attempts to create a tour by inputting all the information described in *TM-FR4* and placing one of each type of object in the environment.
   Output: The tour is successfully created with the correct data.
   Test Case Derivation: *Organization Users* should be able to create customized tours with metadata and objects placed along a specified

path for a *General User* to follow.
How test will be performed: A tester will manually login as a *Organization User* and attempt to create a tour using dummy data that fits the input constraints. They will check to see if the data was set correctly.

4. **Name:** *Organization Users* can create a tour as a draft
   Test ID: Test-TM4
   Control: Manual
   Initial State: User logged in as a *Organization User*, on tour management screen.
   Input: User attempts to create a tour by inputting all the information described in *TM-FR4* and selects the option to save as a draft.
   Output: The tour is successfully created as a draft.
   Test Case Derivation: *Organization Users* should be able to create customized tours but not release it directly to the public.
   How test will be performed: A tester will manually login as a *Organization User* and attempt to create a tour using dummy data that fits the input constraints. They will then select the option at the end to save it as a draft.

5. **Name:** *Organization Users* can create a tour and directly publish it
   Test ID: Test-TM5
   Control: Manual
   Initial State: User logged in as a *Organization User*, on tour management screen.
   Input: User attempts to create a tour by inputting all the information described in *TM-FR4* and selects the option to publish the tour.
   Output: The tour is successfully created and published.
   Test Case Derivation: *Organization Users* should be able to create customized tours and release it directly to the public.
   How test will be performed: A tester will manually login as a *Organization User* and attempt to create a tour using dummy data that fits the input constraints. They will then select the option at the end to publish it.

6. **Name:** *Organization Users* can publish a draft tour
   Test ID: Test-TM6
   Control: Manual

Initial State: User logged in as a *Organization User*, on tour management screen, and has a draft tour.
Input: User navigates to the draft tour and selects publish option.
Output: The tour is successfully published.
Test Case Derivation: *Organization Users* should be able to take draft tours that they have previously worked on and publish them for use by *General Users*.
How test will be performed: A tester will manually login as a *Organization User*, publish a draft tour and look to see if it was released successfully.

7. **Name:** *Organization Users* can preview one of their tours
   Test ID: Test-TM7
   Control: Manual
   Initial State: User logged in as a *Organization User*, on tour management screen, and has a tour.
   Input: User navigates to the tour and selects the preview option.
   Output: The tour can be previewed through the lens of what a *General User* would see.
   Test Case Derivation: *Organization Users* should be able see what their tour will end up looking like when *General Users* eventually use them. This could expose any mistakes in the layout of the tour that can be fixed before release.
   How test will be performed: A tester will manually login as a *Organization User*, attempt to preview a tour, and see if the interface is accurately showing the expected tour.

8. **Name:** *Organization Users* can edit one of their tours
   Test ID: Test-TM8
   Control: Manual
   Initial State: User logged in as a *Organization User*, on tour management screen, and has a tour.
   Input: User navigates to the tour they wish to edit, selects the edit option and changes all the inputs described in *TM-FR4*.
   Output: The tour is successfully edited with the correct data.
   Test Case Derivation: *Organization Users* should be able to edit a tour's metadata and modify the objects placed along a specified path.
   How test will be performed: A tester will manually login as a *Or-*

ganization User and attempt to edit a tour using new dummy data that fits the input constraints. They will check to see if the data was set correctly.

3.1.6 Touring

This section focuses on testing the *General User* side of the tours functionality within the app. This includes the various ways a user can find tours (*TR-FR2*). There are also requirements for previewing a tour (*TR-FR3*) and using the view seen when actually going on a tour (*TR-FR4*). All the other functional requirements from the Touring (TR) section in the SRS **SRS** have at least one test plan below.

1. **Name:** *General Users* can access the touring screen
   Test ID: Test-TR1
   Control: Manual
   Initial State: User logged in as a *General User*.
   Input: User attempts to navigate to the touring screen.
   Output: The touring screen is reachable.
   Test Case Derivation: Only users who are *General Users* will have the ability to go on tours.
   How test will be performed: A tester will manually login as a *General User* and attempt to see the touring screen in the navigation.

2. **Name:** *Organization Users* can NOT access the touring screen
   Test ID: Test-TR2
   Control: Manual
   Initial State: User logged in as a *Organization User*.
   Input: User attempts to navigate to touring screen.
   Output: The touring screen is hidden from user.
   Test Case Derivation: Only users who are *General Users* will have the ability to go on tours. *Organization Users* should have no option to do any touring.
   How test will be performed: A tester will manually login as a *Organization User* and attempt to see the touring screen in the navigation.

3. **Name:** *General Users* can preview a tour
   Test ID: Test-TR3
   Control: Manual

Initial State: User logged in as *General User*, on touring screen, and a tour exits.
Input: User finds a tour and attempts to preview it.
Output: User can see the information described in *TR-FR3*.
Test Case Derivation: *General Users* should be able to preview a tour to see information like the distance and route before actually starting it. They may want to determine if the tour fits with their schedule.
How test will be performed: A tester will manually login as a *General User* and attempt to preview a tour. They will check to see if all the data outlined in *TR-FR3* is present.

4. **Name:** *General Users* can find a tour through the tour list interface
   Test ID: Test-TR4
   Control: Manual
   Initial State: User logged in as a *General User*, on touring screen, and a public tour exists.
   Input: User navigates to the tour list interface, and searches for a tour belonging to an organization.
   Output: The tour has been found.
   Test Case Derivation: *General Users* should be able to find tours through one page where the tours can grouped together by organization or location.
   How test will be performed: A tester will manually login as a *General User* and attempt to find a tour through the tour list interface that belongs to an organization.

5. **Name:** *General Users* can find a tour through a push notification when in proximity to a tour area in the real-world
   Test ID: Test-TR5
   Control: Manual
   Initial State: User logged in as a *General User* but out of the app itself, push notifications are turned on, and their is a tour in the area.
   Input: User goes close to a tour area in the real-world.
   Output: A push notification appears on the user's phone indicating that a tour is nearby and prompts them to preview it.
   Test Case Derivation: *General Users* should be able to find tours that they might not know about by just being in proximity to it. This

will expose the user to tours they may enjoy going on even when the app is not in their mind at that moment.
How test will be performed: A tester will manually login as a *General User* and close the app. They will then walk close to a tour location in the real-world and check to see if they get a push notification.

6. **Name:** *General Users* can find a tour through a QR code
   Test ID: Test-TR6
   Control: Manual
   Initial State: User logged in as a *General User* but in their phone's camera app and they have a tour QR code.
   Input: User scans the QR code through the camera app.
   Output: The camera app opens *Realm* to the preview of the corresponding tour.
   Test Case Derivation: *General Users* should be able to find tours in the app using a QR code sticker located in the real-world at the start of a tour. This allows them to quickly find the tour without searching through the list.
   How test will be performed: A tester will manually login as a *General User* and open the camera app. They will scan the QR code and see if the preview of the tour opens up in the *Realm* app properly.

7. **Name:** *General Users* can switch between the map and AR view in a tour
   Test ID: Test-TR7
   Control: Manual
   Initial State: User logged in as a *General User*, started a tour, and is in map view.
   Input: User selects option to change tour view to AR view and back.
   Output: The app switches the view to AR view and then back to map view.
   Test Case Derivation: *General Users* should be able see tours through the two views and switch between them at any point in time. The AR view is more immersive but users may want to see their progress on a larger scale through the map.
   How test will be performed: A tester will manually login as a *General User* and start a tour. They will try switching the tour view and see if the view is actually changed.

8. **Name:** *General Users* can see the map tour view
   Test ID: Test-TR8
   Control: Manual
   Initial State: User logged in as a *General User* and started a tour
   Input: User selects map view.
   Output: The user can see the map with the properties described in *TR-FR4.1*.
   Test Case Derivation: *General Users* should be able to view their progress in a tour through a map of the route with information that will aid in their understanding. They should be able to see their location overlaid on the map along with the *AR objects* locations.
   How test will be performed: A tester will manually login as a *General User* and start a tour. They will open the map view and check to see if all properties of maps outlined in *TR-FR4.1* are present.

9. **Name:** *General Users* can see the AR tour view
   Test ID: Test-TR9
   Control: Manual
   Initial State: User logged in as a *General User* and started a tour
   Input: User selects AR view.
   Output: The user can see an AR view with the properties described in *TR-FR4.2*.
   Test Case Derivation: *General Users* should have a more immersive interface similar to the realm interface used in the main app instead of having just a map. They should be able to see the *AR objects* superimposed on their screen with the historical information associated with them.
   How test will be performed: A tester will manually login as a *General User* and start a tour. They will open the AR view and check to see if all properties of maps outlined in *TR-FR4.2* are present.

3.1.7 Profile Testing

The Profile Screen Testing focuses on verifying user interactions related to profile management, password changes, and viewing of profile data. These tests ensure that users can efficiently manage their profile settings and view relevant information, which is critical for maintaining user engagement and security.

1. **Name:** Validate User Authentication
   Test ID: Test-PS1
   Control: Automated
   Initial State: App launched, login screen displayed.
   Input: User enters valid credentials.
   Output: The expected result is that the user successfully logs in and is redirected to their profile page.
   Test Case Derivation: This test is to ensure the system authenticates users with valid credentials.
   How test will be performed: Through an automated script that inputs valid user credentials and checks if redirection to the profile page is successful.

2. **Name:** Password Change Functionality
   Test ID: Test-PS2
   Control: Manual
   Initial State: User logged in, on profile settings page.
   Input: User inputs new password and confirms.
   Output: The expected result is that the system updates the password and provides a confirmation message.
   Test Case Derivation: To verify that the system allows users to change their password securely.
   How test will be performed: Tester manually changes the password and checks for confirmation of the change.

3. **Name:** View Profile Information
   Test ID: Test-PS3
   Control: Automated
   Initial State: User logged in, on profile page.
   Input: None.
   Output: The expected result is that profile information (username, password, profile picture, status) is displayed correctly.
   Test Case Derivation: Confirm that all user profile information is retrievable and displayed correctly.
   How test will be performed: Automated test that logs in as a user and verifies that all profile information is displayed as expected.

4. **Name:** Access Help Page
   Test ID: Test-PS4

Control: Manual
Initial State: User logged in, on profile page.
Input: User navigates to help page.
Output: The expected result is that a help page with FAQs and additional help information is displayed.
Test Case Derivation: Ensure the help page is accessible and provides useful information.
How test will be performed: Tester navigates to the help page and verifies the presence and accuracy of the information.

3.1.8 Settings Testing

The Settings Testing focuses on verifying user interactions related to modifying various settings, including accessibility, display, privacy, profile, and sub-realm settings. These tests ensure that users can customize their app experience according to their preferences and privacy requirements, enhancing usability and personalization.

1. **Name:** Modify Accessibility Settings
   Test ID: Test-S1
   Control: Manual
   Initial State: User logged in, viewing settings page.
   Input: User adjusts text size, enables/disables viewing of object names, or changes language.
   Output: The expected result is that accessibility settings apply as configured by the user.
   Test Case Derivation: Confirm that accessibility settings are configurable.
   How test will be performed: Tester changes each accessibility setting and verifies the changes apply.

2. **Name:** Adjust Display Settings
   Test ID: Test-S2
   Control: Manual
   Initial State: User logged in, viewing settings page.
   Input: User changes display settings such as light/dark mode or AR object visibility.
   Output: The expected result is that display settings reflect user preferences.

Test Case Derivation: To ensure display settings are customizable by the user.
How test will be performed: Tester modifies display settings and observes changes.

3. **Name:** Manage Privacy Settings
   Test ID: Test-S3
   Control: Manual
   Initial State: User logged in, viewing settings page.
   Input: User modifies privacy settings to control visibility of profile, friends list, and AR interactions.
   Output: The expected result is that privacy settings update based on user preferences.
   Test Case Derivation: Verify users can control privacy settings for their profiles and interactions.
   How test will be performed: Tester changes privacy settings and checks for corresponding updates.

4. **Name:** Update Profile Settings
   Test ID: Test-S4
   Control: Manual
   Initial State: User logged in, viewing settings page.
   Input: User changes username, password, profile picture, or status.
   Output: The expected result is that profile settings are updated and saved.
   Test Case Derivation: Ensure users can update personal profile information.
   How test will be performed: Tester updates profile settings and verifies changes.

3.1.9 Interactions with User Inventory

1. **Name:** Delete Object from Inventory
   Test ID: Test-IV1
   Control: Manual
   Initial State: App is open, tester is present on the Profile interface. Tester performs actions to open their Inventory. The Inventory is not empty.
   Input: Tester selects an object and chooses the delete option.

Output: The selected object is removed from the inventory.
Test Case Derivation: Users should have control over their inventory and be able to delete unwanted AR objects.
How test will be performed: Tester will select an object in the inventory, delete it, and check if the AR object is successfully removed.

2. **Name:** Add Object to Inventory
   Test ID: Test-IV2
   Control: Manual
   Initial State: App is open, tester is present on the Profile interface Tester performs actions to open their Inventory.
   Input: Tester adds a new object to the inventory.
   Output: The new object appears in the inventory.
   Test Case Derivation: Users should be able to expand their inventory by adding objects.
   How test will be performed: Tester will add a new object to the inventory and confirm its appearance. The object can be added through various means possible based on app features like AR object generation via prompt, scan and upload AR objects, acquiring via object sharing.

3. **Name:** Application-Provided AR Objects in Inventory
   Test ID: Test-IV3
   Control: Automatic
   Initial State: App is installed and launched for the first time. Tester navigates to the Profile interface.
   Input: Tester opens the inventory.
   Output: Inventory contains the preloaded application-provided objects.
   Test Case Derivation: The inventory should always contain a set of default objects for user reference.
   How test will be performed: An automated test verifies that default objects are loaded and present in the user's inventory on app initialization.

4. **Name:** Inventory Capacity for Personal Objects
   Test ID: Test-IV4

Control: Automatic
Initial State: Tester has 99 personal objects in inventory.
Input: Tester attempts to add an additional object.
Output: The object is successfully added, but adding another would be prevented.
Test Case Derivation: Limiting personal objects prevents overuse of storage.
How test will be performed: An automated test attempts to add objects beyond inventory limit to check if inventory maintains strict limit.

5. **Name:** Personal Object Source Verification
   Test ID: Test-IV5
   Control: Manual
   Initial State: Inventory contains personal objects generated or shared by users.
   Input: Tester opens the inventory and inspects object origins.
   Output: Each personal object is either user-generated or shared.
   Test Case Derivation: Ensures the integrity of inventory sources.
   How test will be performed: Tester inspects objects and verifies they meet origin requirements.

6. **Name:** Total Object Count in Inventory
   Test ID: Test-IV6
   Control: Automatic
   Initial State: Inventory contains objects of varying types.
   Input: Tester views the total count of objects.
   Output: The app displays the correct total number of objects.
   Test Case Derivation: Provides users with an overview of their inventory.
   How test will be performed: Automated test calculates total object count and compares it to the displayed count.

7. **Name:** Storage of 2D and 3D AR Objects
   Test ID: Test-IV7
   Control: Manual

Initial State: Inventory is empty or has objects.
Input: Tester adds both 2D and 3D AR objects to their inventory.
Output: Both 2D and 3D objects are correctly stored in inventory.
Test Case Derivation: Inventory should be versatile in managing object types.
How test will be performed: Tester can verify that both 2D and 3D objects are present in the inventory and is able to interact with them.

8. **Name:** Add to Favourite Group
   Test ID: Test-IV8
   Control: Manual
   Initial State: App is open, tester is viewing their Inventory. Inventory has objects.
   Input: Tester adds an object to the favourite group.
   Output: Object is marked as a favourite.
   Test Case Derivation: Users should highlight preferred objects for quick access.
   How test will be performed: Tester adds an object to favourites and verifies if the object is present under the favourite category.

9. **Name:** Sort Objects
   Test ID: Test-IV9
   Control: Manual
   Initial State: App is open, tester is viewing their Inventory. Inventory has objects with varied usage, favourites, and sizes.
   Input: Tester sorts objects by usage, favourites, or size.
   Output: Objects are sorted as per user selection.
   Test Case Derivation: Sorting aids in organizing objects for efficient use.
   How test will be performed: Tester selects sorting criteria and verifies that the results are indeed sorted, based on the selected criteria.

10. **Name:** Continuous Rotation for 3D Objects
    Test ID: Test-IV10
    Control: Automatic

Initial State: App is open, tester is viewing their Inventory. Inventory contains 3D objects.
Input: Tester selects option to view a 3D AR object.
Output: 3D objects are displayed in a continuous rotating state.
Test Case Derivation: Rotation helps users fully examine 3D objects.
How test will be performed: Automated test opens inventory and confirms rotation of 3D objects.

3.1.10 Maps Interface

1. **Name:** Map Location and Display of Overlays
   Test ID: Test-MP1
   Control: Manual
   Initial State: App is open, tester navigates to the Map interface.
   Input: Tester is present on the Map interface and is viewing the complete map and check displayed information.
   Output:

   (a) User's current location is displayed on the map.

   (b) Location markers appear for AR object clusters.

   (c) Markers show the count of objects in each cluster.

   (d) All sub-realm objects associated with the user are indicated on the map.

   Test Case Derivation: Ensures that the map displays the user's location, object clusters, and relevant sub-realm data accurately.
   How Test Will Be Performed:

   (a) Open the map and verify that the user's location is displayed correctly.

   (b) Confirm that AR object clusters are marked and each marker displays an accurate count of objects.

   (c) Check that all sub-realm objects connected to the user are visible on the map.

2. **Name:** Navigation and Directions on Map
   Test ID: Test-MP2
   Control: Manual
   Initial State: Tester is viewing the map with visible AR object markers.
   Input: Select a marker and initiate navigation. Terminate navigation mid-route.
   Output:

   (a) Directions to the selected marker are provided.

   (b) Navigation terminates when requested by the user.

   Test Case Derivation: Confirms that users can receive directions to selected markers and can end navigation if needed.
   How Test Will Be Performed:

   (a) Select a marker and confirm that the option to receive directions appears.

   (b) Begin navigation and verify directions are correct.

   (c) Terminate navigation and confirm that the system stops route guidance.

3. **Name:** Clutter Management and Restricted Area Identification
   Test ID: Test-MP3
   Control: Automatic
   Initial State: Tester is viewing the map with a high density of objects.
   Input: Zoom in and out on the map and navigate toward restricted areas.
   Output:

   (a) Objects are grouped to reduce clutter on the map.

   (b) Restricted areas are identified, and navigation to these areas is disallowed.

   Test Case Derivation: Ensures that map view remains clear through grouping and that restricted areas are indicated to prevent access.
   How Test Will Be Performed:

(a) Zoom in and out to verify that objects are grouped or separated appropriately to avoid clutter.

(b) Attempt to navigate to a restricted area and confirm that the system prevents navigation.

3.1.11 Custom AR Object Generation

1. **Name:** Prompt Entry and Validation
   Test ID: Test-POG1
   Control: Automatic
   Initial State: App is open, tester is present on the prompt entry screen.
   Input: Enter prompts of various lengths, with and without profanity.
   Output:

   (a) Tester can successfully enter a prompt.

   (b) The prompt is restricted to 200 characters.

   (c) Character count is displayed in real-time as user types.

   (d) Profanity is flagged and rejected.

   Test Case Derivation: Confirms that prompts adheres to length restrictions, real-time character count display, and profanity filtering.
   How Test Will Be Performed: Enter prompts within and exceeding 200 characters, observe character count display, and include some profanity to confirm filtering.

2. **Name:** Object Type Selection and Confirmation
   Test ID: Test-POG2
   Control: Manual
   Initial State: App is open, tester has entered a valid prompt and is ready to generate an object.
   Input: Select between 2D or 3D object type and confirm submission.
   Output:

   (a) The tester can select either 2D or 3D object type.

   (b) System initiates AR object generation upon confirmation.

Test Case Derivation: Ensures users can select object type before submission, and confirmation triggers generation.
How Test Will Be Performed: Choose each object type and confirm prompt submission, tester observes the initiation of the object generation (2D or 3D).

3. **Name:** AR Object Generation and Selection
   Test ID: Test-POG3
   Control: Automatic
   Initial State: App is open, tester has entered a valid prompt and confirmed initiation of object generation process.
   Input: Await object generation, then select one of the generated options.
   Output:

   (a) Multiple AR objects are generated based on the prompt.
   (b) Tester can select a preferred object from the options provided.

   Test Case Derivation: Ensures the system generates multiple objects and allows user selection.
   How Test Will Be Performed: Submit a prompt, observe the generated objects, and select one to confirm selection functionality.

4. **Name:** Add to Inventory
   Test ID: Test-POG4
   Control: Automatic
   Initial State: Tester has selected an AR object from the generated options.
   Input: Add the selected object to the personal inventory.
   Output: The chosen AR object is saved in the user's inventory.
   Test Case Derivation: Ensures that user selections can be stored in inventory for future access.
   How Test Will Be Performed: Add the object and confirm its presence in the inventory list.

5. **Name:** Generated AR Object Preview
   Test ID: Test-POG5

Control: Manual
Initial State: Tester is viewing a selected AR object in preview mode.
Input: Rotate the AR object to inspect all sides.
Output: The AR object rotates smoothly, allowing inspection from all angles.
Test Case Derivation: Confirms that users can preview generated objects by rotating them.
How Test Will Be Performed: Manually rotate the AR object to confirm smooth and complete rotation in preview mode.

3.1.12 Uploading Objects to Inventory, Post Object Scan

1. **Name:** Object Render Display and Editing
   Test ID: Test-OUI1
   Control: Manual
   Initial State: Tester has completed a scan of an object, and the scan is ready for viewing/editing.
   Input: Display the scanned object and allow for user interaction in editing mode.
   Output:

   (a) Render of scanned object is displayed to the user.

   (b) Tester can remove unneeded features from the render.

   (c) Tester can crop and resize 2D objects in the edit interface.

   (d) Tester can confirm the creation of the AR object once satisfied with edits.

   Test Case Derivation: Ensures that the user can visualize and edit the scanned object before saving it to the inventory.
   How Test Will Be Performed:

   (a) Display the scanned object render and prompt the tester to interact with it.

   (b) Attempt removal of unnecessary features, then save changes and confirm that edits are applied.

   (c) For 2D objects, perform cropping and resizing operations and confirm that modifications are correctly reflected.

(d) Complete edits and confirm creation, ensuring that all final changes are saved.

2. **Name:** Object Naming and Storage
   Test ID: Test-OUI2
   Control: Manual
   Initial State: Tester has finished editing and is ready to save the object.
   Input: Provide a name for the object, then save it to inventory with relevant metadata.
   Output:

   (a) Object name is stored, containing only ASCII characters.

   (b) Creation date, time, user information, storage size, and type (2D or 3D) are stored with the object in the inventory.

   Test Case Derivation: Confirms that all necessary metadata is correctly assigned and stored along with the object.
   How Test Will Be Performed:

   (a) Enter a name containing only ASCII characters and confirm successful entry.

   (b) Verify that all metadata, including date, time, user information, storage size, and type, is correctly saved.

   (c) Access the object in inventory to confirm that metadata is retrievable and accurate.

3. **Name:** Object Color Editing
   Test ID: Test-OUI3
   Control: Manual
   Initial State: Object is open in the edit interface after scanning has been completed.
   Input: Select specific portions of the object and apply color changes.
   Output: Color changes are applied to selected portions of the object, and the final render reflects these edits accurately.
   Test Case Derivation: Verifies that users can apply color changes to specific parts of the object for customization.
   How Test Will Be Performed:

(a) Select a portion of the object and change its color.

(b) Verify that the color change is applied only to the selected portion.

(c) Confirm that changes are saved and displayed correctly in the final render.

3.2 Tests for Nonfunctional Requirements

This section focuses on test cases pertaining to the non-functional requirements.

3.2.1 Usability Testing

1. **Name:** Validate Localization
   Test ID: Test-QS-U1
   Type: Non-Functional, Manual
   Initial State: App is open on settings page.
   Input/Condition: Language setting is changed to English, Mandarin Chinese, Hindi, Spanish, and French (The 5 most spoken languages in the world) in turn.
   Output/Result: Text in the app correctly changes to the selected language, with understandable translations.
   How test will be performed: Tester navigates to settings menu, selects one of the languages to be tested, and verifies that text in the settings page, realm interface, and maps interface are all correctly displayed in the selected language

2. **Name:** Validate User Intuitiveness and Satisfaction
   Test ID: Test-QS-U2
   Type: Non-Functional, Manual
   Initial State: App is open, and tester is logged in as a new user with no prior experience using the app.
   Input/Condition: Tester performs common workflows such as account setup, navigating between interfaces, creating AR objects, viewing and placing AR objects, and finding objects on the map, all without guidance or assistance
   Output/Result: 80% of testers complete each task and report that the app is easy and satisfying to use, and rate each workflow as highly intuitive.

How test will be performed: A group of new users will perform specified workflows and complete a post-test survey rating the intuitiveness and satisfaction of their experience. Quantitative results should show that most users rate the app as highly intuitive and satisfying to use.

3.2.2 Security Testing

1. **Name:** Encryption implementation message reading
   Test ID: Test-QS-SC1
   Type: Manual
   Initial State: Encryption algorithm is complete, user has opened the app, and requests to the server are being monitored.
   Input/Condition: A request with sensitive information has been sent to the server.
   Output/Result: The contents of the information passed is not decipherable by reading the request but the server can decrypt the information to get the original message.
   How test will be performed: A tester will open the app and monitor requests made to the server. They will check to see if the information has been encrypted or not.

2. **Name:** Encryption implementation algorithm check
   Test ID: Test-QS-SC2
   Type: Static
   Initial State: Encryption algorithm is complete.
   Input/Condition: All code relating to the encryption algorithm will be sent to a static analyzer.
   Output/Result: The analyzer will show any vulnerabilities found within the algorithm
   How test will be performed: A tester will run a static analyzer that is tasked with finding code errors in the encryption algorithm implementation that could lead to an incorrect output by the system.

3. **Name:** Verify identity before transmitting private data
   Test ID: Test-QS-SC3
   Type: Manual
   Initial State: The app is ready for security review.
   Input/Condition: All sections of the code where a user's sensitive

data is displayed is checked for a corresponding identity check.
Output/Result: The sections all have a check to verify the user's identity before divulging the private data.
How test will be performed: A tester will search through the code and find all locations where private data is being displayed. They will check to see if each of them are guarded by an identity verification check.

3.2.3 Availability Testing

1. **Name:** Validate Server Availability
   Test ID: Test-QS-A1
   Type: Non-Functional, Automated
   Initial State: Server is running, with monitoring tools actively tracking uptime.
   Input/Condition: Automated monitoring scripts track server uptime and downtime continuously over a one-week period
   Output/Result: Monitoring scripts log any downtime, with server uptime recorded at 99% or higher over the test period.
   How test will be performed: Monitoring scripts will check server availability at regular intervals and log any downtime events, ensuring the server meets the 99% availability criteria.

2. **Name:** Validate User Feedback on Server Availability
   Test ID: Test-QS-A2
   Type: Non-Functional, Manual
   Initial State: Server is running, and a test group of users has been granted access to the app.
   Input/Condition: Over a one-week period, users in the test group access the app multiple times per day, as they normally would, at varying times.
   Output/Result: No user complaints about server unavailability during the test period. Users report no issues with app access.
   How test will be performed: Test group participants will be surveyed at the end of the test period regarding any access issues they encountered. Any user-reported issues will be logged and reviewed to assess the server's availability from the user's perspective.

3.2.4 Maintainability Testing

1. **Name:** Validate API Error Message Clarity
   Test ID: Test-DI-M1
   Type: Non-Functional, Manual and Automated
   Initial State: System is running, with logging enabled for internal API calls.
   Input/Condition: Simulate the following types of system failures in internal APIs and observe the resulting error messages: database connection failure, invalid input data, service timeout.
   Output/Result: Error messages generated by the APIs clearly indicate the source and nature of the error in at least 90% of cases, helping developers quickly identify issues.
   How test will be performed: Automated scripts will be used to simulate common errors and log the resulting API responses. The resulting error messages will be manually reviewed to be evaluated on detail (e.g., error type, location, and possible causes), and clarity (accurate indication of what is causing error). Success is achieved if 90% of the messages help to identify the error source.

3.2.5 Compliance Testing

1. **Name:** Check Personal Information and Electronic Documents Act (PIPEDA) **PIPEDA** compliance
   Test ID: Test-CO1
   Type: Manual
   Initial State: App is ready for compliance review.
   Input/Condition: The app in its current state is checked against PIPEDA for compliance.
   Output/Result: The app has been verified to comply with PIPEDA.
   How test will be performed: A tester will manually parse through PIPEDA and check off all the sections that the app comports with. The app should comply with all sections.

2. **Name:** Tax records check going back six years
   Test ID: Test-CO2
   Type: Manual
   Initial State: The app is published on a app store.
   Input/Condition: Records are checked for purchases and ad-revenue

made over the course of the project's lifetime.
Output/Result: The records go back at least six years.
How test will be performed: A tester will look at the history of all revenue generated through the app and make sure the records go back to the legally required time span of 6 years.

3. **Name:** Check *Google Play* developer policy **GooglePlay** compliance
Test ID: Test-CO3
Type: Manual
Initial State: App is ready for compliance review.
Input/Condition: The app in its current state is checked against the *Google Play* developer policy for compliance.
Output/Result: The app has been verified to comply with the *Google Play* developer policy.
How test will be performed: A tester will manually parse through the *Google Play* developer policy and mark down all the sections that the app comports with. The app should comply with all sections.

4. **Name:** Check *App Store* review guidelines **AppStore** compliance
Test ID: Test-CO4
Type: Manual
Initial State: App is ready for compliance review.
Input/Condition: The app in its current state is checked against the *App Store* review guidelines for compliance.
Output/Result: The app has been verified to comply with the *App Store* review guidelines.
How test will be performed: A tester will manually parse through the *App Store* review guidelines and mark down all the sections that the app comports with. The app should comply with all sections.

3.2.6 Reusability Testing

1. **Name:** Reusable components check
Test ID: Test-DI-R1
Type: Static
Initial State: All code is available for analysis.
Input/Condition: All code is sent to a static analyzer that has indicators for code duplication.

Output/Result: The analysis will show metrics relating to the sections of code that have a high amount of duplication.
How test will be performed: A tester will run the static analysis and look at the metrics to determine if an abstract component is warranted for sections of the code that have a lot of overlap. These sections could be simplified by having them all derive from a common component.

3.2.7 Portability Testing

1. **Name:** Validate Cross-Platform Compatibility
   Test ID: Test-PT1
   Type: Non-Functional, Manual
   Initial State: Application is built for both iOS and Android platforms.
   Input/Condition: Run the app on iOS and Android devices.
   Output/Result: The app is functional and displays correctly on both platforms.
   How test will be performed: Tester will install the app on both an iOS and an Android device, verifying consistent functionality and UI.

2. **Name:** Common Codebase Validation
   Test ID: Test-PT2
   Type: Non-Functional, Code Review
   Initial State: The app's codebase is ready for review.
   Input/Condition: Inspect the codebase to ensure shared files are correctly configured with minimal platform-specific files.
   Output/Result: Codebase only differs in configuration files for platform-specific settings.
   How test will be performed: Developer will conduct a code walkthrough, focusing on configuration files to confirm minimal platform-specific variations.

3. **Name:** Build Verification on iOS and Android
   Test ID: Test-PT3
   Type: Non-Functional, Automated
   Initial State: The cross-platform codebase is ready for automated build testing.
   Input/Condition: Initiate automated builds for both iOS and Android.

Output/Result: Both builds succeed without errors.
How test will be performed: An automated CI/CD pipeline will attempt to build the app for both platforms, confirming compatibility.

3.2.8 Safety Testing

1. **Name:** Distraction to Surroundings Assessment
   Test ID: Test-SA1
   Type: Non-Functional, Survey-Based
   Initial State: App is functional and ready for user testing.
   Input/Condition: Conduct a user survey after users engage with the app in a controlled environment.
   Output/Result: Survey results show that users do not find the app dangerously distracting them from their surroundings while using it.
   How test will be performed: A group of users will be observed using the app, followed by a survey asking them to rate their distraction levels from surrounding objects. Results will be analyzed to confirm minimal distraction.

2. **Name:** No Bright Flashes or Loud Noises
   Test ID: Test-SA2
   Type: Non-Functional, Manual Inspection
   Initial State: The app is fully developed with all interfaces available for review.
   Input/Condition: Navigate through all screens and interactions within the app.
   Output/Result: No bright flashes or loud noises are present in any of the app interfaces.
   How test will be performed: Tester will manually explore the app, paying special attention to visual and audio elements, ensuring that no features could trigger discomfort or seizures in sensitive users.

3.2.9 Installation Testing

1. **Name:** Verify App Store Availability
   Test ID: Test-I1
   Type: Non-Functional, Manual
   Initial State: App has been submitted and approved on both iOS and Android app stores.

Input/Condition: Search for the app on the Apple App Store and Google Play Store.
Output/Result: The app is available for download on both app stores.
How test will be performed: Tester will verify the presence of the app by searching for it on the respective app stores and confirming it is listed and downloadable.

2. **Name:** Simple Installation Process
Test ID: Test-I2
Type: Non-Functional, Manual
Initial State: App is available on both app stores.
Input/Condition: Attempt to install the app on a device from both the Apple App Store and Google Play Store.
Output/Result: The app installs directly without any additional steps or configurations.
How test will be performed: Tester will initiate the installation from each app store, ensuring the app installs seamlessly without requiring extra configurations or settings adjustments.

3.2.10 Performance Requirements

1. **Name:** Map Rendering
Test ID: Test-QS-PE1
Type: Non-Functional, Automatic, Dynamic
Initial State: App is open, tester is present on any interface except for the Maps interface.
Input/Condition: User performs actions to navigate to the Map interface.
Output/Result: The map and its overlays are completely visible and can be interacted with.
How test will be performed: Tester will navigate to the Maps interface. Once there, the load time for all overlays will be recorded and considered to introduce improvements in their loading and rendering.

2. **Name:** Inventory Load
Test ID: Test-QS-PE2
Type: Non-Functional, Automatic, Dynamic

Initial State: App is open, and the tester is present on their profile page.
Input/Condition: User selects the option to view their entire Inventory.
Output/Result: The Inventory loads completely and can be interacted with within 1-10 seconds depending on the number of objects present in the Inventory.
How test will be performed: Tester will access the Inventory section and the application will record the load time such that the time is recorded until the user is able to interact with the objects visible. Then, the tester can observe the recorded load times and verify if the times are below the maximum threshold (10 seconds).

3. **Name:** Real-Time Render Delay
   Test ID: Test-QS-PE3
   Type: Non-Functional, Manual
   Initial State: App is open, User has intialized process to scan and create an AR object. User is actively scanning the environment.
   Input/Condition: Device settings and sensors are functioning properly. User initiates a scan of the environment.
   Output/Result: Rendered results appear within 1 second of the initial scan data.
   How test will be performed: Tester will initiate a scan to create AR object, start two timers, one for the render being displayed (*Timer1*) and the other for the scan performed by the tester (*Timer2*). Once the scanning process is finished, both timers are stopped. Then, the tester will take the difference of the two recorded times, and ensure that it does not exceed 1 second.

4. **Name:** AR Object Generation
   Test ID: Test-QS-PE4
   Type: Non-Functional, Automatic
   Initial State: App is open, tester is present on object generation interface and has entered a *valid* prompt to begin AR object generation.
   Input/Condition: Tester initiates the generation of an AR object.
   Output/Result: The AR object is fully generated and visible.

How test will be performed: Tester will initiate AR object generation and start a timer. The timer will be stopped once the AR object is fully generated and visible. The time taken is recorded to verify the process takes at most 30 seconds.

5. **Name:** AR Object Fallback Mode
   Test ID: Test-QS-PE5
   Type: Non-Functional, Manual
   Initial State: App is open on a low-performance device.
   Input/Condition: Tester attempts to view AR objects within the app.
   Output/Result: The app renders AR objects with minimal lag or provides a fallback mode (low-resolution objects) for accessibility.
   How test will be performed: Tester will use a low-performance device to attempt rendering AR objects. Performance will be observed to confirm the app enables a fallback mode for compatibility such as lower resolution for AR objects and limited functionality in terms of app features (AR object generation, Scan and Upload AR objects).

3.2.11 Reliability Requirements

1. **Name:** Database Failure/Corruption
   Test ID: Test-QS-RE1
   Type: Non-Functional, Dynamic, Automatic
   Initial State: The app is running with a stable connection to the primary database, and all user data is accessible without issues. (No failure or corruption has occurred in the database)
   Input: Inject random data or errors into the test database to trigger failure.
   Output:

   (a) Database recovers automatically or through a set of recovery steps.

   (b) If there is user data loss, only **2%** of user data will be lost after the recovery.

   (c) System returns to normal operation, allowing all users to access their data without issues.

How Test Will Be Performed:

(a) Set up a testing environment where the database can be backed up and forcefully corrupted.

(b) Initiate a database corruption/failure.

(c) Ensure that the system detects and logs the failure and attempts recovery.

(d) After the recovery process is finished, review the data with a backed-up copy for multiple test user accounts to confirm that large amounts of user data have not been erased.

(e) Compute the data loss percent to ensure it does not exceed the stated limit.

3.2.12 Distribution Requirements

1. **Name:** Device Compatibility
   Test ID: Test-DI-D1
   Type: Non-Functional, Automatic
   Initial State: The app is not yet installed on test devices. The test devices include mobile devices running iOS 16.0+ and Android 12+ and even older Android and iOS versions.
   Input/Condition: Attempt to download and install the app on the user device.
   Output/Result:

   (a) The app installs successfully on devices running iOS 16.0+ and Android 12+.

   (b) The app functions as expected post-installation on the device.

   How test will be performed: Install the app on a device, app checks if the device's OS version satisfies the requirements.

2. **Name:** Regional Availability
   Test ID: Test-DI-D2
   Type: Non-Functional, Manual
   Initial State: The app is available in the app store.
   Input/Condition: Attempt to access and download the app from app

stores within Canada and the USA.
Output/Result: The app is accessible and can be downloaded by users in both Canada and the USA. **How test will be performed:** Using devices connected to networks in Canada and the USA, check that the app is accessible, downloadable, and installable from each app store.

3. **Name:** Recommended Age Requirement Display
   Test ID: Test-DI-D3
   Type: Non-Functional, Manual
   Initial State: The app is available in the app store, ready to be viewed by users.
   Input/Condition: Open the app store listing for the app.
   Output/Result: The app store listing clearly shows a recommended age requirement of 16+. **How test will be performed:** View the app's store page on various devices and confirm the presence of the age recommendation.

4. **Name:** User Data Storage in North America
   Test ID: Test-DI-D4
   Type: Non-Functional, Manual
   Initial State: User data is being created and saved through app usage.
   Input/Condition: Review server locations where user data is stored.
   Output/Result: All user data is stored within North America. **How test will be performed:** Verify if app servers used for user data storage and processing are located within North American.

3.3 Traceability Between Test Cases and Requirements

Test-ID	Test Name	Requirements
Test-RI1	Validate AR Object Perspective Adjustment	RI-FR1.1
Test-RI2	Validate AR Object Clutter Management	RI-FR1.2
Test-RI3	Validate AR Object Placement Accuracy	RI-FR1.2
Test-RI6	Validate Object Placement Workflow Control	RI-FR3
Test-RI8	Validate Nearby Tour Indication	RI-FR5
Test-RI9	Validate Hazard Warning	RI-FR6
Test-RI10	Validate Offline Mode for Interactive Components	RI-FR7
Test-OP1	Validate Object Selection Stage	OP-FR2.1
Test-OP3	Validate Object Placement Stage	OP-FR2.3
Test-OP4	Validate Object Instance Storage	OP-FR1
Test-OP5	Validate Area Based Placement Limit	OP-FR3.1
Test-OP6	Validate Time Based Placement Limit	OP-FR3.2
Test-OP7	Validate Automated Retry for Failed Object Storage	OP-FR1
Test-DB1	Validate Periodic Database Backup	DB-FR1
Test-DB2	Validate Database Encryption	DB-FR2
Test-QS-U1	Validate Localization	QS-U1
Test-QS-U2	Validate User Intuitiveness and Satisfaction	QS-U2
Test-QS-A1	Automated Server Availability Monitoring	QS-A1
Test-QS-A2	User Feedback on Server Availability	QS-A1
Test-DI-M1	Validate API Error Message Clarity	DI-M1

Table 2: Mapping of Tests to Requirements (I)

Test-PS1	Validate User Authentication	PS-FR1
Test-PS2	Password Change Functionality	PS-FR3
Test-PS3	View Profile Information	PS-FR4
Test-PS4	Access Help Page	PS-FR6
Test-S1	Modify Accessibility Settings	S-FR1
Test-S2	Adjust Display Settings	S-FR2
Test-S3	Manage Privacy Settings	S-FR3
Test-S4	Update Profile Settings	S-FR4
Test-PT1	Validate Cross-Platform Compatibility	DI-P1
Test-PT2	Common Codebase Validation	DI-P2
Test-PT3	Build Verification on iOS and Android	DI-P1
Test-SA1	Distraction Level Assessment	QS-SA1
Test-SA2	No Bright Flashes or Loud Noises	QS-SA2
Test-I1	Verify App Store Availability	DI-I1
Test-I2	Simple Installation Process	DI-I2

Table 3: Mapping of Tests to Requirements (II)

4 Unit Test Description

This section contains descriptions of the unit tests derived from the MIS.

4.1 Settings Module Testing

1. **Name:** Key Validation
   Test ID: Test-SM1
   Type: Functional, Automated
   Initial State: Settings module is initialized
   Input/Condition: One key that is a valid settings key, and another that is not
   Output/Result: The function should return true for the valid key, and false for the invalid key
   How test will be performed: The **ValidateKey** method on the settings module will be called with the inputs

2. **Name:** Ensure Valid Profile Details
   Test ID: Test-SM2
   Type: Functional, Automated

Test-TM1	*Organization Users* can access tour management screen	TM-FR1
Test-TM2	*General Users* can NOT access the tour management screen	TM-FR1
Test-TM3	*Organization Users* can create a customized tour	TM-FR4
Test-TM4	*Organization Users* can create a tour as a draft	TM-FR2
Test-TM5	*Organization Users* can create a tour and directly publish it	TM-FR3
Test-TM6	*Organization Users* can publish a draft tour	TM-FR3
Test-TM7	*Organization Users* can preview one of their tours	TM-FR5
Test-TM8	*Organization Users* can edit one of their tours	TM-FR6
Test-TR1	*General Users* can access the touring screen	TR-FR1
Test-TR2	*Organization Users* can NOT access the touring screen	TR-FR1
Test-TR3	*General Users* can preview a tour	TR-FR3
Test-TR4	*General Users* can find a tour through the tour list interface	TR-FR2.1
Test-TR5	*General Users* can find a tour through a push notification when in proximity to a tour area in the real-world	TR-FR2.2

Table 4: Mapping of Tests to Requirements (III)

Test-TR6	*General Users* can find a tour through a QR code	TR-FR2.3
Test-TR7	*General Users* can switch between the map and AR view in a tour	TR-FR4
Test-TR8	*General Users* can see the map tour view	TR-FR4.1
Test-TR9	*General Users* can see the AR tour view	TR-FR4.2
Test-QS-SC1	Encryption implementation message reading	QS-SC1
Test-QS-SC2	Encryption implementation algorithm check	QS-SC1
Test-QS-SC3	Verify identity before transmitting private data	QS-SC2
Test-CO1	Check Personal Information and Electronic Documents Act (PIPEDA) **PIPEDA** compliance	CO1
Test-CO2	Tax records check going back six years	CO2
Test-CO3	Check *Google Play* developer policy **GooglePlay** compliance	CO3
Test-CO4	Check *App Store* review guidelines **AppStore** compliance	CO4
Test-DI-R1	Reusable components check	DI-R1
Test-OS1	Surroundings Scan	OS-FR1, OS-FR4
Test-OS2	Object Type Selection	OS-FR2
Test-OS3	Real-Time Render Display	OS-FR3
Test-OS4	Confirm Scanning Completion	OS-FR5
Test-IV1	Delete Object from Inventory	IV-FR1
Test-IV2	Add Object to Inventory	IV-FR2
Test-IV3	Application-Provided AR Objects in Inventory	IV-FR3
Test-IV4	Inventory Capacity for Personal Objects	IV-FR4
Test-IV5	Personal Object Source Verification	IV-FR5

Table 5: Mapping of Tests to Requirements (IV)

Test-IV6	Total Object Count in Inventory	IV-FR6
Test-IV7	Storage of 2D and 3D AR Objects	IV-FR7
Test-IV8	Add to Favourite Group	IV-FR8
Test-IV9	Sort Objects	IV-FR9
Test-IV10	Continuous Rotation for 3D Objects	IV-FR10
Test-MP1	Map Location and Display of Overlays	MP-FR1, MP-FR2, MP-FR3, MP-FR4, MP-FR5
Test-MP2	Navigation and Directions on Map	MP-FR6, MP-FR7, MP-FR9
Test-MP3	Clutter Management and Restricted Area Identification	MP-FR8, MP-FR10
Test-POG1	Prompt Entry and Validation	POG-FR1, POG-FR2, POG-FR3, POG-FR4
Test-POG2	Object Type Selection and Confirmation	POG-FR5, POG-FR6
Test-POG3	AR Object Generation and Selection	POG-FR7, POG-FR8
Test-POG4	Add to Inventory	POG-FR9
Test-POG5	Generated AR Object Preview	POG-FR10
Test-OUI1	Object Render Display and Editing	OUI-FR1, OUI-FR2, OUI-FR3, OUI-FR4
Test-OUI2	Object Naming and Storage	OUI-FR5 OUI-FR6, OUI-FR7, OUI-FR8, OUI-FR9, OUI-FR10, OUI-FR11,
Test-OUI3	Object Color Editing	OUI-FR12

Table 6: Mapping of Tests to Requirements (V)

Test-QS-PE1	Map Rendering	QS-PE1
Test-QS-PE2	Inventory Load	QS-PE2
Test-QS-PE3	Real-Time Render Delay	QS-PE3
Test-QS-PE4	AR Object Generation	QS-PE4
Test-QS-PE5	AR Object Fallback Mode	QS-PE5
Test-QS-RE1	Database Failure/Corruption	QS-RE1
Test-DI-D1	Device Compatibility	DI-D1
Test-DI-D2	Regional Availability	DI-D2
Test-DI-D3	Recommended Age Requirement Display	DI-D3
Test-DI-D4	User Data Storage in North America	DI-D4

Table 7: Mapping of Tests to Requirements (VI)

Initial State: Settings module is initialized
Input/Condition: A valid user settings object
Output/Result: The function should return an object matching the schema of the valid user settings object
How test will be performed: The **FetchProfileDetails** method on the settings module will be called with the inputs

4.2 Help Module Testing

1. **Name:** String Search
   Test ID: Test-HM1
   Type: Functional, Automated
   Initial State: Help module is initialized
   Input/Condition: Partial and full keywords matching help items
   Output/Result: The search result outputs should match the precomputed expected search results for the provided search terms
   How test will be performed: The **SearchHelp** method on the help module will be called with the inputs

4.3 Collision Detection Module Testing

1. **Name:** Detect Collision
   Test ID: Test-CD1
   Type: Functional, Automated

Initial State: Collision detection module is initialized
Input/Condition: Mock AR tracking data and device accelerometer data representing hazardous and non-hazardous scenarios
Output/Result: The function should output true for potential collisions, and false otherwise
How test will be performed: The **DetectCollision** method on the collision detection module will be called with the inputs

4.4 Tour Proximity Module Testing

1. **Name:** Detect Nearby Tour
   Test ID: Test-TP1
   Type: Functional, Automated
   Initial State: Tour proximity module is initialized
   Input/Condition: Mock GPS data representing device, and tour positions
   Output/Result: The function should output a list containing all tours within a certain distance, and excluding the rest
   How test will be performed: The **DetectNearbyTours** method on the tour proximity module will be called with the inputs The scope of the unit testing is limited to helper functions present in scripts used by Unity objects. The unit tests will be written in C# and will be executed using the Unity Test Framework. The tests will be written to ensure that the helper functions are working as expected and are free of bugs.

4.5 Local Database Manager

1. **Name:** fetchData
   Test ID: Test-LDM-FD
   Type: Automatic
   Initial State: N/A
   Input/Condition: A string Query to access database entries that need to be fetched from local database
   Output/Result: The data expected by the caller (could be binary, string, integer, float, etc.)
   Test Case Derivation: This test case is derived from the requirement to fetch data from the local database.

How test will be performed: This function will be called and the tester will check if data is retrieved with no errors. The tester must also verify if the data retrieved from the database matches the expected data.

2. **Name:** saveData
   Test ID: Test-LDM-SD
   Type: Automatic
   Initial State: N/A
   Input/Condition: A key to access the data in the future and the data that needs to be added to the database.
   Output/Result: No exceptions are thrown and the data can be observed in the local database through the dev environment.
   Test Case Derivation: This test case is derived from the requirement to save data to the local database.
   How test will be performed: The function will be used by the tester to save a new data entry in the local database. If no errors are shown, the test has passed otherwise the function must be corrected to prevent any errors.

3. **Name:** updateData
   Test ID: Test-LDM-UD
   Type: Automatic
   Initial State: Data entries must exist in the local database
   Input/Condition: A key referring to the data needed to be updated and the new content that replaces the existing data entry in the local database.
   Output/Result: No output is expected and the new content can be seen under the key used to update the entry in the local database.
   Test Case Derivation: This test case is derived from the requirement to update data in the local database.
   How test will be performed: The tester will call the function with the key and new content and after the function is called, they can observe whether the data under the key passed into the function matches the new content they wanted in the database.

4. **Name:** deleteData
   Test ID: Test-LDM-DD
   Type: Automatic
   Initial State: Data entries must exist in the local database
   Input/Condition: Key used to access the data that needs to be deleted
   Output/Result: No output is expected and the data entry should not exist in the local database.
   Test Case Derivation: This test case is derived from the requirement to delete data from the local database.
   How test will be performed: The function will be called with the appropriate key and the tester will then check if the data entry corresponding to the input key still exists. If it does not, the function has passed the test.

5. **Name:** syncWithServer
   Test ID: Test-LDM-SWS
   Type: Automatic
   Initial State: N/A
   Input/Condition: No input is needed
   Output/Result: No output is expected. The data present in the local database is up to date and matches with the server database.
   Test Case Derivation: This test case is derived from the requirement to sync the local database with the server database.
   How test will be performed: The tester will call the function and check if all the entries between the local database and the server database match (This could be done using a script to compare all entries in the database).

6. **Name:** isCacheStale
   Test ID: Test-LDM-ICS
   Type: Automatic
   Initial State: N/A
   Input/Condition: No input is needed
   Output/Result: Output True or False based on whether the local database is outdated compared to the server database.

How test will be performed: The tester will call the function on an outdated local database and check if the function correctly returns True. The function will also be executed on an up-to-date local database with an expected output of False.

4.6 Server Database Manager

1. **Name:** fetchData
   Test ID: Test-SDM-FD
   Type: Automatic
   Initial State: N/A
   Input/Condition: A string Query to access database entries that need to be fetched from server database
   Output/Result: The data expected by the caller (could be binary, string, integer, float, etc.)
   Test Case Derivation: This test case is derived from the requirement to fetch data from the server database.
   How test will be performed: This function will be called and the tester will check if data is retrieved from the server database with no errors. The tester must also verify if the data retrieved from the database matches the expected data.

2. **Name:** saveData
   Test ID: Test-SDM-SD
   Type: Automatic
   Initial State: N/A
   Input/Condition: A key to access the data in the future and the data that needs to be added to the database.
   Output/Result: No exceptions are thrown and the data can be observed in the server database through the dev environment.
   Test Case Derivation: This test case is derived from the requirement to save data to the server database.
   How test will be performed: The function will be used by the tester to save a new data entry in the server database. If no errors are shown, the test has passed otherwise the function must be corrected to prevent any errors.

3. **Name:** updateData
   Test ID: Test-SDM-UD
   Type: Automatic
   Initial State: Data entries must exist in the server database
   Input/Condition: A key referring to the data needed to be updated and the new content that replaces the existing data entry in the server database.
   Output/Result: No output is expected and the new content can be seen under the key used to update the entry in the server database.
   Test Case Derivation: This test case is derived from the requirement to update data in the server database.
   How test will be performed: The tester will call the function with the key and new content and after the function is called, they can observe whether the data under the key passed into the function matches the new content they wanted in the database.

4. **Name:** deleteData
   Test ID: Test-SDM-DD
   Type: Automatic
   Initial State: Data entries must exist in the server database
   Input/Condition: Key used to access the data that needs to be deleted
   Output/Result: No output is expected and the data entry should not exist in the server database.
   Test Case Derivation: This test case is derived from the requirement to delete data from the server database.
   How test will be performed: The function will be called with the appropriate key and the tester will then check if the data entry corresponding to the input key still exists. If it does not, the function has passed the test.

5. **Name:** syncWithLocal
   Test ID: Test-SDM-SWL
   Type: Automatic
   Initial State: N/A

Input/Condition: A list of key-to-data entries that are present in the local database and not the server one.
Output/Result: The data present in the server database matches with the local database.
Test Case Derivation: This test case is derived from the requirement to sync the server database with the local database.
How test will be performed: The tester will call the function with the data not present in the local database, and check if all the entries between the local database and the server database match (This could be done using a script to compare all entries in the database).

6. **Name:** logSyncOperation
   Test ID: Test-SDM-LSO
   Type: Automatic
   Initial State: A sync operation is executed between the local and server database
   Input/Condition: A boolean status that can be retrieved from the module's `status` flag
   Output/Result: No output is expected and the log should be written to the application's log terminal.
   Test Case Derivation: This test case is derived from the requirement to log sync operations between the local and server databases.
   How test will be performed: The tester will call the function with the appropriate status and check if the log is written to the application's log terminal.

4.7 REST API Communication

1. **Name:** sendRequest
   Test ID: Test-RAC-SR
   Type: Automatic
   Initial State: N/A
   Input/Condition: A string representing an endpoint, a string representing the HTTP method (GET, POST, PUT, DELETE) and a string that has the request body.
   Output/Result: The response from the server.

Test Case Derivation: This test case is derived from the requirement to send a request to the API server.
How test will be performed: The tester will call the function with the appropriate parameters and check if the response from the server is as expected.

2. **Name:** parseResponse
   Test ID: Test-RAC-PR
   Type: Automatic
   Initial State: An HTTP request must be correctly retrieved after requesting.
   Input/Condition: A raw HTTP response.
   Output/Result: Outputs the JSONified HTTP response.
   Test Case Derivation: To check if the Raw Response from the HTTP request can be converted into a JSON object that is easier to work with.
   How test will be performed: The tester will call the function with the raw HTTP response and check if the JSON object is returned.

3. **Name:** setHeaders
   Test ID: Test-RAC-SH
   Type: Automatic
   Initial State: N/A
   Input/Condition: A string representing the header used for subsequent HTTP requests.
   Output/Result: No output is expected.
   Test Case Derivation: This test case is derived from the requirement to set headers for the HTTP request.
   How test will be performed: The tester will call the function with the appropriate header and check if the header is set for the subsequent HTTP requests.

4. **Name:** handleAuthentication
   Test ID: Test-RAC-HA
   Type: Automatic
   Initial State: N/A

Input/Condition: A string token representing the authentication token.
Output/Result: No output is expected
Test Case Derivation: This test case is derived from the requirement to handle authentication for the HTTP request.
How test will be performed:

5. **Name:** checkServerStatus
   Test ID: Test-RAC-CSS
   Type: Automatic
   Initial State: N/A
   Input/Condition: A boolean status that can be retrieved from the module's `status` flag
   Output/Result: Returns an Enum representing the server state.
   Test Case Derivation: This test case is derived from the requirement to check the server status.
   How test will be performed: The tester will call the function and check if the server status is returned as expected.

6. **Name:** buildURL
   Test ID: Test-RAC-BU
   Type: Automatic
   Initial State: N/A
   Input/Condition: Takes the endpoint as a string and the query parameters as a list of strings.
   Output/Result: Returns the complete URL with the base URL, the endpoint and the query parameters.
   Test Case Derivation: This test case is derived from the requirement to build a URL for the HTTP request.
   How test will be performed: The tester will call the function with the appropriate parameters and check if the URL is built as expected.

7. **Name:** logRequest
   Test ID: Test-RAC-LR
   Type: Automatic
   Initial State: An HTTP request has been made.
   Input/Condition: A string representing the request details.

Output/Result: No output is expected and the log should be written to the application's log terminal.
Test Case Derivation: This test case is derived from the requirement to log requests made to server.
How test will be performed: The tester will call the function with the appropriate request details and check if the log is written to the application's log terminal.

4.8 Realm Interface

1. **Name:** renderObjects
   Test ID: Test-RI-RO
   Type: Automatic
   Initial State: N/A
   Input/Condition: A list of 3D objects that need to be rendered.
   Output/Result: No output is expected.
   Test Case Derivation: This test case is derived from the requirement to render objects in the realm.
   How test will be performed: The tester will call the function with the appropriate 3D objects and check if the objects are rendered in the realm.

4.9 Authentication Module

4.9.1 SendNotification

Name: SendNotification
Test ID: Test-AM-SN
Type: Manual
Initial State: System is running, and the test environment has a known valid user ID.
Input/Condition: User ID, notification message
Output/Result: Bool - Success or Fail
Test Case Derivation: Testing that notifications are being sent. We also consider possible edge cases (e.g., invalid user ID or empty message) to ensure correct behavior or exception handling.

How test will be performed:

1. Manually invoke **SendNotification** with a valid user ID and a non-empty message.
2. Verify that the function returns true (success).
3. Repeat with an invalid user ID or empty message to ensure it returns false or throws an exception.
4. Document all outcomes and compare against expected results.

4.9.2 FetchNotifications

Name: FetchNotifications
Test ID: Test-AM-FN
Type: Manual
Initial State: System is running, user is logged in, and the test environment has a known valid user ID.
Input/Condition: User ID
Output/Result: List of notifications
Test Case Derivation: Testing that a list of notifications can be received. We also consider edge cases to ensure correct behavior or exception handling.
How test will be performed: Manually invoke FetchNotifications with a valid user ID that is known to have notifications and verify that the function returns the expected list of notifications.

4.9.3 MarkNotificationRead

Name: MarkNotificationRead
Test ID: Test-AM-MNR
Type: Manual
Initial State: System is running, user is logged in, and the test environment has a known valid notification ID.
Input/Condition: Notification ID
Output/Result: Bool - Success or Fail
Test Case Derivation: Testing that a notification can be marked as read. We also consider edge cases to ensure correct behavior or exception handling.
How test will be performed: Receive a notification and mark it as read. Check if the database reflected this notification as read.

4.9.4 DeleteNotification

Name: DeleteNotification
Test ID: Test-AM-DN
Type: Manual
Initial State: System is running, user is logged in, and the test environment has a known valid notification ID that can be deleted.
Input/Condition: Notification ID
Output/Result: Bool - Success or Fail
Test Case Derivation: Testing that a notification can be deleted. We also consider edge cases (e.g., invalid notification ID) to ensure correct behavior or exception handling.
How test will be performed: Receive a notification and then go in and delete it and see if it is still stored or if it is actually deleted.

4.9.5 UpdateNotificationSettings

Name: UpdateNotificationSettings
Test ID: Test-AM-UNS
Type: Manual
Initial State: System is running, user is logged in, and the test environment has a known valid user ID and default notification settings.
Input/Condition: User ID, settings data
Output/Result: Bool - Success or Fail
Test Case Derivation: Testing that a user's notification settings can be updated. We also consider edge cases (e.g., invalid settings data) to ensure correct behavior or exception handling.
How test will be performed: Modify some settings regarding notifications (such as notification sound), and then manually receive a notification and verify that the change worked.

4.10 Object Render Module

4.10.1 FetchRenderSettings

Name: FetchRenderSettings
Test ID: Test-AM-FRS
Type: Manual
Initial State: System is running with a default or previously configured set of render settings.
Input/Condition: -
Output/Result: Dictionary of current render settings
Test Case Derivation: Testing that the function retrieves the current render settings without requiring any input. We also consider the default configuration scenario to ensure correct behavior.
How test will be performed:

1. Invoke FetchRenderSettings in a running system.
2. Confirm that it returns a valid dictionary of render settings matching the current configuration.

4.11 Touring Module

4.11.1 StartTour

Name: StartTour
Test ID: Test-TM-ST
Type: Manual
Initial State: System is running, user is logged in, and a valid tour exists but has not started.
Input/Condition: Tour ID
Output/Result: Bool - Success or Fail
Test Case Derivation: Testing that a tour can be started with a valid Tour ID. We also consider edge cases (e.g., invalid Tour ID) to ensure correct behavior or exception handling.
How test will be performed:

1. Invoke **StartTour** with a valid Tour ID and confirm it returns `true`.
2. Invoke **StartTour** with an invalid Tour ID and confirm it throws an InvalidTourIDException.

4.11.2 PauseTour

Name: PauseTour
Test ID: Test-TM-PT
Type: Manual
Initial State: System is running, user is logged in, and a tour is currently active.
Input/Condition: -
Output/Result: Bool - Success or Fail
Test Case Derivation: Testing that an active tour can be paused without any specific input.
How test will be performed:

1. Start a tour using **StartTour**.
2. Invoke **PauseTour** and confirm it returns `true`.
3. Verify that the tour's state changes to a paused state in the system.

4.11.3 EndTour

Name: EndTour
Test ID: Test-TM-ET
Type: Manual
Initial State: System is running, user is logged in, and a tour is currently active or paused.
Input/Condition: -
Output/Result: Bool - Success or Fail
Test Case Derivation: Testing that an active or paused tour can be ended.
How test will be performed:

1. Start a tour using **StartTour**
2. Invoke **EndTour** and confirm it returns true.

4.11.4 FetchTourDetails

Name: FetchTourDetails
Test ID: Test-TM-FTD
Type: Manual

Initial State: System is running, user is logged in, and at least one valid tour ID is known.
Input/Condition: Tour ID
Output/Result: Tour Object
Test Case Derivation: Testing that the function retrieves detailed information about a specific tour.
How test will be performed: Invoke **FetchTourDetails** with a valid Tour ID and verify it returns the correct Tour Object.

4.12 Tour LIst Module

4.12.1 FetchTourList

Name: FetchTourList
Test ID: Test-TM-FTL
Type: Manual
Initial State: System is running, user is logged in.
Input/Condition: None
Output/Result: Array of Tours
Test Case Derivation: Testing that a list of all available tours can be fetched without any specific input.
How test will be performed:

1. Invoke **FetchTourList** in a running system.
2. Verify that it returns an *Array of Tours* containing all currently available tours.

4.12.2 SearchTours

Name: SearchTours
Test ID: Test-TM-STQ
Type: Manual
Initial State: System is running, user is logged in, and there are multiple tours available.
Input/Condition: Search Query
Output/Result: Array of Tours
Test Case Derivation: Testing that the function can return tours filtered by a given search query. We also consider edge cases, like an empty or

irrelevant query.
How test will be performed:

1. Invoke **SearchTours** with a valid query (e.g., a known keyword).
2. Verify that the returned *Array of Tours* matches the search criteria.
3. Invoke **SearchTours** with an empty or irrelevant query to ensure it returns an empty array or no matching results.

4.13 Tour Management Module

4.13.1 CreateTour

Name: CreateTour
Test ID: Test-TM-CT
Type: Manual
Initial State: System is running, user is logged in, and the environment allows creation of new tours.
Input/Condition: Tour Object
Output/Result: Bool - Success or Fail
Test Case Derivation: Testing that a new tour can be created with valid tour data. We also consider edge cases (e.g., invalid tour data) to ensure correct behavior or exception handling (e.g., **InvalidTourDataException**).
How test will be performed:

1. Invoke **CreateTour** with a valid Tour Object and verify it returns true.
2. Invoke **CreateTour** with invalid tour data to ensure it throws InvalidTourDataException.

4.13.2 UpdateTour

Name: UpdateTour
Test ID: Test-TM-UT
Type: Manual
Initial State: System is running, user is logged in, and at least one valid tour exists.
Input/Condition: Tour ID, Updated Tour Data

Output/Result: Bool - Success or Fail
Test Case Derivation: Testing that an existing tour can be updated with new data. We also consider edge cases (e.g., invalid Tour ID) to ensure correct behavior or exception handling (e.g., **InvalidTourIDException**).
How test will be performed:

1. Invoke UpdateTour with a valid Tour ID and valid updated tour data. Verify it returns true.

2. Invoke UpdateTour with an invalid Tour ID to ensure it throws InvalidTourIDException.

4.13.3 DeleteTour

Name: DeleteTour
Test ID: Test-TM-DT
Type: Manual
Initial State: System is running, user is logged in, and at least one valid tour ID is known.
Input/Condition: Tour ID
Output/Result: Bool - Success or Fail
Test Case Derivation: Testing that an existing tour can be deleted with a valid tour ID. We also consider edge cases (e.g., invalid Tour ID) to ensure correct behavior or exception handling (e.g., **InvalidTourIDException**).
How test will be performed:

1. Invoke DeleteTour with a valid Tour ID and confirm it returns true.

2. Invoke DeleteTour with an invalid Tour ID to ensure it throws InvalidTourIDException.

5 Appendix

This is where you can place additional information.

5.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

5.2 Usability Survey Questions

- For the following statements, please indicate your level of agreement between Strongly Disagree, Disagree, Neither Agree nor Disagree, Agree, and Strongly Agree:
  - Navigating between interfaces is intuitive
  - Placing objects is intuitive
  - Scanning objects is intuitive
  - Generating objects by prompt is intuitive
  - Embarking on tours is intuitive
  - Creating tours is intuitive (ONLY FOR ORG USERS)
  - Managing tours is intuitive (ONLY FOR ORG USERS)
  - Changing user settings is intuitive
  - Interacting with other's objects is intuitive
  - Reporting other's objects is intuitive
  - Using the app is generally satisfying
  - Using the app is distracting from the surroundings
  - The app is compatible with, and works well on, my device (Android or IOS)
- Did you experience any service interruptions, including but not limited to excessive load times for elements like navigation and object placement, while testing the app?

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Lifelong Learning.

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?

   Making the tests were very easy after we had the initial template down and distributed the work effectively amongst ourselves based on the functional and non-functional requirements.

2. What pain points did you experience during this deliverable, and how did you resolve them?

   One pain point we experienced was that we had a lot to do for this deliverable that we couldn't work on the next one, which is POC. Ultimately, what we decided to do was to focus mainly on this, because the deadline is earlier, and focus mainly on getting our project set up for POC. That way, we at least have started on the next deliverable.

   Another thing was deciding which tests to automate and which to make manual. We resolved this by seeing which test would be easiest or quickest to implement. If a manual test was easiest, we would choose that, but if a manual test would take long, we'd make it automated.

3. What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project? Examples of possible knowledge and skills include dynamic testing knowledge, static testing knowledge, specific tool usage, Valgrind etc. You should look to identify at least one item for each team member.

   The team will need to become familiar with C# tooling like unit test frameworks and compatible static analysis tools.

   For dynamic testing, the team will likely use the *Unity Test Framework* since it is already integrated into the Unity game engine which powers the project. There are also Roslyn analyzers build for the .NET platform that are integrated into IDEs that are connected to Unity. These are useful as static analyzers that can check for errors in the code even if it still compiles.

   As a tentative plan, we will have Russell look into configuring the Roslyn analyzers that come packaged with .NET so that it meets our needs. Abdul and Avanish will research the *Unity Test Framework* for the Unity *Editor Mode* and *Play Mode* respectively. Rafey will pick a static analysis tool like SonarQube for us to use the code.

4. For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill? Of the identified approaches, which will each team member pursue, and why did they make this choice?

   One approach (especially given the popularity of Unity/.NET/SonarQube) is to watch videos online where people talk about the testing within the context of Unity projects. This could give us a prospective on how other people implement testing. Another approach is to read through the documentation associated with the topic at hand. This is very useful when a there is a smaller user base for a tool and no associated videos or examples.

   Russell will read documentation since most of his work surrounds configuration which requires a description for each configurable part. This is easily searchable when sifting through documentation.

   Abdul and Avanish will watch videos (if they can find some good ones)

about the *Unity Test Framework*. Since it is a very popular among game developers who use Unity, there should be many walkthroughs on how to use it. By watching multiple videos, they should get a sense of how people usually implement the framework for testing.

Rafey will use Google and look through internet forums to see what other developers recommend when it comes to static analysis for C#. He will also end up looking at the documentation for these tools to decide which one works the best for our use case.