CSC1097

FINAL YEAR PROJECT

ARrangeIt

Testing Documentation

Jade Hudson - 21706905

Sruthi Santhosh - 21377986

Supervisor: Dr. Hyowon Lee

30/04/2024

Table of Contents

Abstract

ARrangeIt is an Android based Augmented Reality (AR) mobile application designed to assist users in visualising furniture and decor items within their physical spaces before purchasing. By integrating ARCore and Firebase technologies, ARrangeIt enhances the online shopping experience by overlaying 3D models of furniture into real-world environments using the smartphone cameras. The application addresses major usability gaps in current AR shopping tools, focusing on Android compatibility, multi-item placement, real-time measuring ability and saving configurations.

Testing Strategy

Objective: Ensure the app is functional, user friendly and to validate the robustness of the features such as furniture placement, measurement tool and layout saving.

Approach:

- Unit Tests: Validate individual components such as Firebase data parsing and field validation.
- UI Tests: Check each interactable interface works as expected as a single activity.
- Integration Tests: Verify interactions between interfaces work as expected.
- System Tests: Validate the end-to-end user flows (Login → Place Furniture → Save Layout → Logout).
- User Testing: Gather feedback on usability from 5 participants.

Tools used:

- JUnit (Unit Tests)
- Mockito (Unit Tests)
- Espresso (UI, System and Integration Tests)
- Manual User Testing (5 people)

Challenges

Testing AR place detection (surface recognition, plane estimation, and object anchoring) is extremely difficult in automated or simulated environments because real-world conditions vary unpredictably. Unlike placement, rotation, and movement—which can be scripted with mock coordinates—accurate plane detection depends on camera input, lighting, surface textures and device sensor calibration. Emulators cannot replicate real-world depth perception, leading to unreliable test results. Additionally, ARCores plane detection behaves differently across devices due to hardware variances (LiDAR vs. non-LiDAR). As a result, our testing focuses on post-detection interactions (placement, movement, rotation) using predefined virtual surfaces, avoiding the unreliable simulation of dynamic environments. Manual adhoc testing supplements this by validating detection in real-world scenarios.

Testing Rationale (UI and System Tests: 70%, Unit Tests: 20%, User Testing: 10%):

Logic such as invalid passwords and incorrect email formats are covered by targeted unit tests, however, given that ARrangeIt is a user-centric AR application, our testing strategy prioritises UI, Manual and System Tests over lower-level unit tests. The app's core functionality (AR furniture placement, area measurement, layout saving) is driven by user actions (taps, swipes, gestures). Bugs in UI logic (misaligned buttons, broken navigation) directly impacts usability and perceived quality. Therefore we focused our strategy on espresso validation tests such as

- Navigation flows (Catalogue → AR View → Saved Layouts)
- Interactive Controls (drag-to-move, swipe to rotate)
- Error Handling (Invalid Logins, Forgot Password)

Firebase integration (authentication, database) is standardised and well-tested by Google so this was not as high of a priority. ARCore's object anchoring cannot be reliably unit-tested as it depends on camera/sensors, however UI and Manual testing allows for more freedom:

- Scripted Espresso tests for repeatable interactions ("Place 2 models → Save").
- Ad hoc testing in varied environments (carpet, tiles, low light) for AR robustness.

**** All tests in this document were last run and passed successfully on 30/04/2025 ****

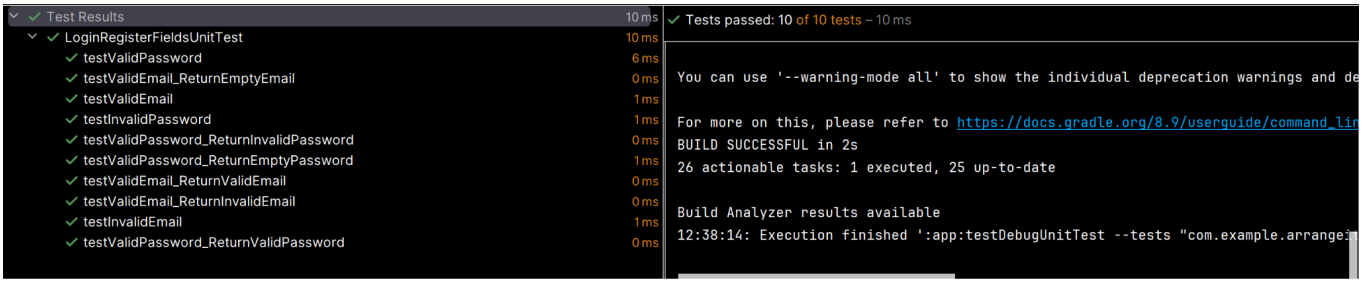
Unit Tests

Login and Register Unit Tests

Test Class: LoginRegisterFieldsUnitTest
Package: com.example.arrangeit (test)
Purpose: To verify the correctness of email and password validation logic implemented while users log in and register using the FieldValidatorHelper

Test Name	Objective	Test Cases	Expected Result
testValidEmail	Verify that the isValidEmail method correctly recognises valid email addresses.	“test@example.com”, “user@domain.org”	assertTrue is used and both emails should return true as they are valid formats
testInvalidEmail	Verify that isValidEmail correctly rejects the invalid emails	"invalid-email", "user@.com", "", null, "user@com"	assertFalse is used and all are invalid emails and should return false.
testValidPassword	Verify that the isValidPassword method correctly identifies valid passwords	"Password123!", "Hello@123456"	assertTrue is used and both passwords meet complexity requirements.
testInvalidPassword	Verify that isValidPassword correctly rejects weak or invalid passwords.	1. "123" (too short) 2. "" (empty), 3. null (empty) 4. "password" (no uppercase or digit or special character) 5. "PASSWORD1" (no lowercase or special character) 6. "Password!" (no digit) 7. "Password1" (no special character)	assertFalse is used and is expected to pass as these passwords are considered invalid.
testValidEmail_ReturnValidEmail	Test that validateEmail returns null for a valid email.	“test@example.com”	Expects null / no error messag
testValidEmail_ReturnEmptyEmail	Test that validateEmail detects and reports an error for the empty email field.	“”	Expects the error message “Email cannot be empty”.
testValidEmail_ReturnInvalidEmail	Test that validateEmail reports an improperly formatted email.	“invalid-email”	Expects the error message “Please enter a valid email address”.
testValidPassword_ReturnValidPassword	Test that validatePassword returns null for a valid password.	“Password1!”	Expects null / no error message.
testValidPassword_ReturnEmptyPassword	Test that validatePassword detects and reports an empty password field	“”	Expects the error message “Password cannot be empty”.
testValidPassword_ReturnInvalidPassword	Test that validatePassword detects a weak password and returns an appropriate error message.	“weak”	Expects the error message “Password must be at least 8 characters long and include an uppercase letter, a lowercase letter, a digit, and a special character.”.

Coverage: The tests cover both positive (valid inputs) and negative (invalid inputs) cases for email and password validation. Validation methods are tested for handling null and empty inputs, which are common edge cases.
Dependencies: FieldValidatorHelper must be correctly implemented for these tests to pass. JUnit 4 is used as the testing framework.
Testing Validation Achievement: Login and Registration are critical entry points to the application. Incorrect validation can cause security vulnerabilities or user frustration. These tests ensure reliability and robustness in user authentication workflows.



Furniture Catalogue Unit Tests

Test Class: FurnitureCatalogueUnitTest
Package: com.example.arrangeit (test)
Purpose: To verify the correct behavior of the furniture catalogue functionalities including item loading, filtering and sorting logic. These tests ensure reliable data handling, filtering based on user criteria and proper adapter operations.
Pre-Test Setup: Necessary UI Elements were mocked and three Furniture Items were initialised to be mocked using Mockito

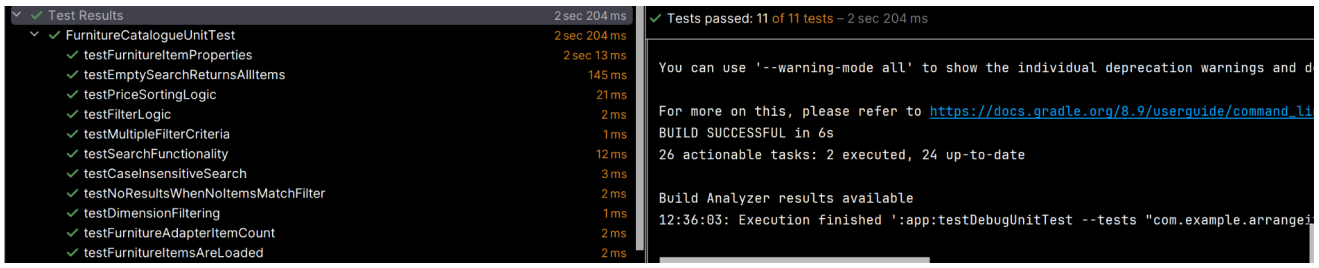
```
// mock UI
fragment.searchBar = mock(EditText.class);
fragment.colourFilterSpinner = mock(Spinner.class);
fragment.typeFilterSpinner = mock(Spinner.class);
fragment.sortByPriceSpinner = mock(Spinner.class);
```

```
testFurnitureItems = new ArrayList<>();
testFurnitureItems.add(new FurnitureItem(
        "Test Chair",
        "Chair",
        "Chair Description",
        99.99,
        "Red",
        "images/chair.png",
        "models/chair.glb",
        "Fabric",
        85.0,
        50.0,
        50.0));

testFurnitureItems.add(new FurnitureItem(
        "Test Sofa",
        "Sofa",
        "Sofa Description",
        299.99,
        "Blue",
        "images/sofa.png",
        "models/sofa.glb",
        "Leather",
        85.0,
        200.0,
        90.0));

testFurnitureItems.add(new FurnitureItem(
        "Test Table",
        "Table",
        "Table Description",
        150.0,
        "Brown",
        "images/table1.jpg",
        "models/table1.glb",
        "Wood",
        75.0,
        120.0,
        80.0));
```

Test Name	Objective	Test Cases	Expected Result
testFurnitureItemsAreLoaded	Confirm that furniture items are properly added to both main and filtered lists.	Adds the three predefined furniture items.	Lists furnitureItems and filteredFurnitureItems where each should contain exactly 3 items and items should not be null when querying item 0 or 1.
testFurnitureAdapterItemCount	Verify that the FurnitureAdapter correctly reports the number of furniture items.	Adapter initialised with 3 furniture items.	getItemCount() should return 3.
testFilterLogic	Ensure that filtering by specific color and other attributes work correctly.	Filter items by - colour = "Blue", - type = "All", - max price = 300.0, - max height = 90.0, - max width = 250.0, - max depth = 100.0.	Only the "Test Sofa" item should be returned as it satisfies all specified criteria.
testPriceSortingLogic	Validate correct sorting of items by price, both ascending and descending.	Mocks the selection of "Price: Low to High" and “Price: High to Low” from the spinner and sorts itemsToSort in ascending order using comparators.	Ascending: "Test Chair" (cheapest) first, "Test Sofa" (most expensive) last. Descending: "Test Sofa" (most expensive) first and "Test Chair" (cheapest) last.
testMultipleFilterCriteria	Check multiple criteria filtering (e.g., type and price combined).	Filter by - colour = "All", - type = "Chair", - max price = 100.0, - max height = 100.0, - max width = 100.0, - max depth = 100.0	Only "Test Chair" should match the filter.
testNoResultsWhenNoItemsMatchFilter	Test the behavior when no furniture matches the applied filters.	Filter with colour = "Pink" (which does not exist).	Filtered result list should be empty (0 items).
testDimensionFiltering	Test dimension-specific filtering where height constraint is strict.	Filter by - colour = "All", - type = "All", - max price = 300.0, - max height = 80.0, - max width = 250.0, - max depth = 100.0	Only items matching this or lower are returned → Test Table satisfies these specifications.

testFurnitureItemProperties	Verify that the FurnitureItem properties are correctly initialised and retrievable.	Creates a FurnitureItem with specific values: (“Test Bed”, “Bed” “Test Bed Description”, 123.45, “Green”, “images/bed_image.png", “models/bed_model.glb”, "Fabric", 100.0, 200.0, 300.0)	All properties should match the values used during object creation.
testSearchFunctionality	Ensure that search functionality filters the furniture items based on user input using mocking (case-insensitive match).	Searches “chair”	Only items containing "chair" (case-insensitive) should be in the filtered list, eg “Test Chair”.
testEmptySearchReturnsAllItems	Verify that if the search bar is empty, all furniture items are displayed.	Searches “”	All items should appear in the filtered list (no filtering).
testCaseInsensitiveSearch	Confirm that search is **case-insensitive** and matches items regardless of input case.	Searches “CHAIR”	Search results should match items regardless of case differences, eg “Test Chair”.

Coverage: Positive scenarios like correct loading, correct matching when filters are satisfied, correct sorting order are validated. Negative scenarios like no items returned when filters are too strict (nonexistent colour, dimension constraints) are tested. Edge cases handled are tested such as filtering by "All" (no specific type/colour), upper-bound testing on price and dimension filtering.
Dependencies: FurnitureItem model must be correctly populated and accessible. FurnitureAdapter must correctly manage its internal list. UI components like EditText and Spinner are mocked. FirebaseFirestore instance is mocked to isolate unit tests from database dependencies. JUnit 4 is used as the framework, Mockito is used to mock context, UI elements, and database.
Testing Achievement: The furniture catalogue is a core feature of the application, offering browsing, filtering, and selection of furniture items. Accurate loading, filtering, and sorting are critical to user satisfaction and functionality. These tests ensure that catalogue operations are reliable, produce expected outputs, and respond correctly to different user selections and filters.



Furniture Catalogue Details Unit Tests

Test Class: FurnitureDetailUnitTest
Package: com.example.arrangeit (test)
Purpose: To verify the correctness of the details page behavior when displaying and interacting with furniture details.
Pre-Test Setup: Necessary UI Elements such as Firebase, Activity and Fragment were mocked and one Furniture Item was initialised to be mocked using Mockito

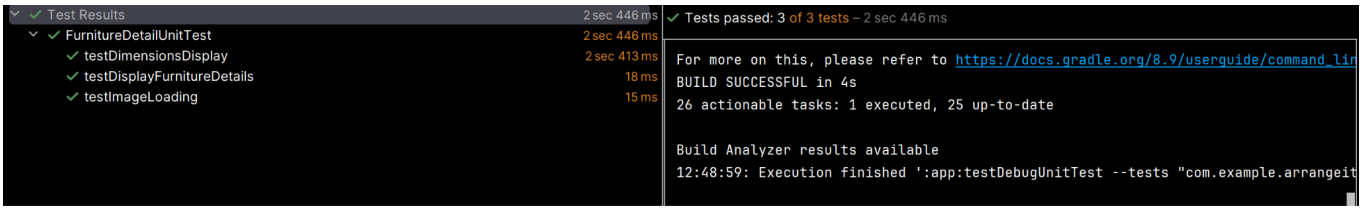
```
testFurnitureItem = new FurnitureItem(
        name: "Test Chair",
        type: "Chair",
        description: "Test Description",
        price: 129.99,
        colours: "Black",
        imageUrl: "images/test_chair.png",
        modelUrl: "models/test_chair.glb",
        texture: "Leather",
        height: 90.0,
        width: 60.0,
        depth: 55.0
);
```

```
// mock inflater
when(mockInflater.inflate(R.layout.fragment_furniture_detail, mockContainer, attachToRoot: false)).thenReturn(mockView);
when(mockBundle.getSerializable( key: "furniture_item")).thenReturn(testFurnitureItem);

// Mock activity and fragment manager
when(mockActivity.getSupportFragmentManager()).thenReturn(mockFragmentManager);
when(mockFragmentManager.beginTransaction()).thenReturn(mockTransaction);
when(mockTransaction.replace(any(Integer.class), any(Fragment.class))).thenReturn(mockTransaction);
when(mockTransaction.remove(any(Fragment.class))).thenReturn(mockTransaction);
when(mockActivity.getOnBackPressedDispatcher()).thenReturn(mockOnBackPressedDispatcher);

// mock firebase
mockFirebaseStorage = mock(FirebaseStorage.class);
mockStorageReference = mock(StorageReference.class);
mockUriTask = mock(Task.class);

doReturn(mockFirebaseStorage).when(fragment).getFirebaseStorage();
when(mockFirebaseStorage.getReference(anyString())).thenReturn(mockStorageReference);
when(mockStorageReference.getDownloadUrl()).thenReturn(mockUriTask);
when(mockUriTask.addOnSuccessListener(any())).thenReturn(mockUriTask);
when(mockUriTask.addOnFailureListener(any())).thenReturn(mockUriTask);
```

Test Name	Objective	Test Cases	Expected Result
testDisplayFurnitureDetails	Ensure the fragment correctly populates the UI components (itemName, itemDescription, itemPrice, itemHeight, itemWidth, itemDepth, itemColours, itemTexture) with the provided furniture item data.	Pass the mock furniture item values as input.	The fragment's views (TextViews for name, description, price, etc.) should reflect the data in testFurnitureItem, ensuring the fragment is correctly displaying the furniture details
testImageLoading	Verify that the FurnitureDetailFragment loads the image URL from Firebase Storage when the fragment is created.	Furniture item with the image URL "images/test_chair.png is used.	The fragment should call the getDownloadUrl() method on the Firebase StorageReference for the item image, triggering the image loading process.
testDimensionsDisplay	Verify that the dimensions (height, width and depth) of the furniture are correctly displayed in the fragment.	Mock FurnitureItem with sample dimensions (90.0, 60.0 and 55.0 cm).	The fragment should correctly display the furniture's dimensions as "Height: 90.0cm", "Width: 60.0cm", and "Depth: 55.0cm" in the respective TextViews.

Coverage: These tests ensure that the FurnitureDetailFragment correctly handles and displays the furniture details, including textual information and image loading from Firebase Storage. The tests also verify that the fragment handles UI updates when the furniture item is passed as an argument.
Dependencies: Firebase Storage setup must be correct for image loading to work. The FurnitureItem class and FurnitureDetailFragment must be correctly implemented for the tests to function as expected. Mockito is used to mock dependencies like context, views and Firebase interactions. JUnit and Mockito are used for testing. Mockito handles the mocking of the Firebase storage interactions and UI components, while JUnit is used to run the tests and assert the correctness of the fragment's behavior.

Testing Achievement: These unit tests help ensure that the FurnitureDetailFragment functions as expected, providing users with accurate and visually appealing furniture details.



User Interface Tests

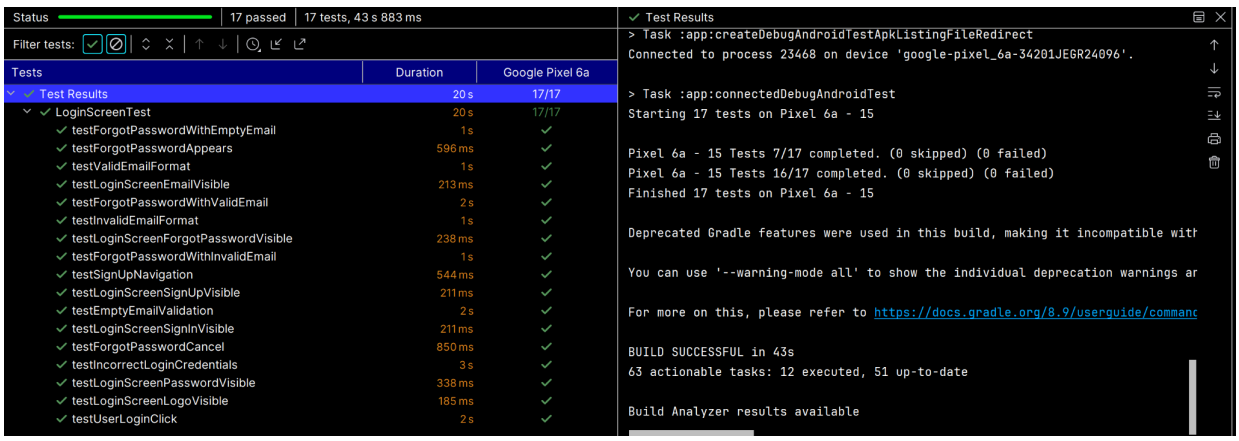
Login User Interface Test

Test Class: LoginScreenTest
Package: com.example.arrangeit (Android test)
Purpose: This class contains UI tests for the login screen of the ArrangeIt app. It ensures that various UI elements on the login screen are visible and function correctly, such as the logo, email input, password input, login button, and sign-up button. It also validates error handling for incorrect or missing input.

Test Name	Objective	Test Cases	Expected Result
testLoginScreenLogoVisible	Ensure that the logo is visible on the login screen.	Check if the logo is displayed.	The logo should be displayed
testLoginScreenEmailVisible	Ensure that the email input field is visible on the login screen.	Check if the email input field is displayed.	The email input field should be displayed.
testLoginScreenPasswordVisible	Ensure that the password input field is visible on the login screen.	Check if the password input field is displayed.	The password input field should be displayed.
testLoginScreenSignInVisible	Ensure that the "Sign In" button is visible on the login screen.	Check if the sign-in button is displayed.	The sign-in button should be displayed.
testLoginScreenSignUpVisible	Check if the sign-up button is displayed on the login screen.	Check if the sign-up button is displayed.	The sign-up button should be displayed.
testLoginScreenForgotPassword Visible	Ensure that the "Forgot Password" link is visible on the login screen.	Check if the forgot password link is displayed.	The forgot password link should be displayed.
testUserLoginClick	Test if a user can input login credentials and click the sign-in button.	Type a valid email “test@example.com” and password “Password123!”, then click the sign-in button.	The user can successfully click the sign-in button.
testSignUpNavigation	Tests navigation to the sign-up screen when the sign-up button is clicked.	Click the sign-up button and verify the presence of text indicating sign-up screen.	The sign-up screen should be displayed with “Create and Account” and “Please enter your details to register".
testEmptyEmailValidation	Test error handling when the email field is empty.	Leave the email field empty, input a valid password “Password123!” and click sign-in	An error message "Email cannot be empty" should be displayed.
testInvalidEmailFormat	Test error handling when the email format is invalid.	Input an invalid email address “invalid-email” and check for an error message.	An error message "Please enter a valid email address" should be displayed.
testValidEmailFormat	Test behavior when a valid email address is entered.	Input a valid email address "valid@example.com" and	No error message should be displayed for a valid email

		ensure no error message is shown.	address.
testForgotPasswordAppears	Test if the forgot password UI elements appear when clicked.	Click the "Forgot Password" link and verify the visibility of the email input, reset button and cancel button.	Forgot password screen elements should be visible.
testForgotPasswordCancel	Test if the cancel button on the forgot password screen works correctly.	Click the cancel button and ensure the login screen is displayed again.	The login screen should appear again after canceling.
testForgotPasswordWithEmptyEmail	Test if an error message is shown when trying to reset the password with an empty email.	Click "Forgot Password", leave the email field empty, and click reset.	Error message "Email cannot be empty" should be displayed.
testForgotPasswordWithInvalidEmail	Test if an error message is shown when trying to reset the password with an invalid email.	Click "Forgot Password", enter an invalid email "not-an-email" and click reset.	Error message "Please enter a valid email address" should be displayed.
testIncorrectLoginCredentials	Test behavior when incorrect login credentials are entered.	Enter incorrect email "wrong@example.com" and password "WrongPass123!" and click sign-in.	The login screen should remain displayed (indicating failed login). A toast message will also indicate this visually but this is difficult to pick up in espresso tests.
testForgotPasswordWithValidEmail	Test if the forgot password process works when a valid email is entered.	Enter a valid email "testing@example.com", click the reset button and check if the login screen is displayed again.	The login screen should be displayed after successful reset.

Coverage: These tests cover UI elements visibility (logo, input fields, buttons, etc). They also cover user interactions (clicking buttons, entering text), Validation of input fields (empty email, invalid email, correct email), Navigation flows (sign-up, forgot password) and Error handling for login and reset password actions.
Dependencies: JUnit, Espresso, ActivityScenarioRule, GrantPermissionRule.
Testing Achievements: These tests verify that the app's login flow is working as expected. It ensures correct behavior for user input validation. It checks that navigation to sign-up and forgot password screens works correctly. It confirms that error messages are displayed when the user enters incorrect or incomplete information.



Register User Interface Test

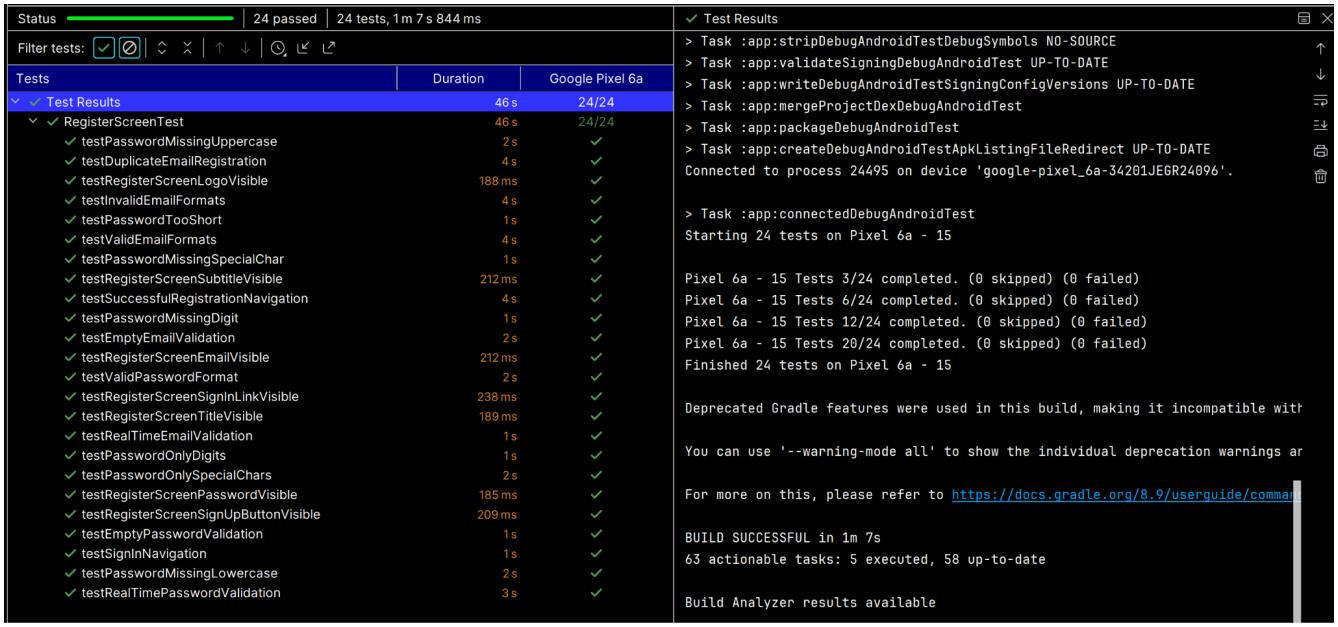
Test Class: RegisterScreenTest
Package: com.example.arrangeit (Android test)
Purpose: This class contains UI tests for the register screen and it ensures that various UI elements on the screen are visible and function correctly such as the logo, email input, password input, sign up button and login button. It also validates error handling for incorrect or missing input.

Test Name	Objective	Test Case and Input	Expected Result
testRegisterScreenLogoVisible	Ensure the app logo is visible on the register screen	Check if the logo is displayed.	Logo is displayed
testRegisterScreenEmailVisible	Verify that the email input field is visible	Check if the email field is displayed.	Email input field is displayed
testRegisterScreenPasswordVisible	Verify that the password input field is visible	Check if the password field is displayed.	Password input field is displayed
testRegisterScreenSignUpButtonVisible	Verify that the sign-up button is visible	Check if the sign up button is displayed.	Sign-up button is displayed
testRegisterScreenSignInLinkVisible	Verify that the sign-in link is visible	Check if the sign in button is displayed.	Sign-in link is displayed
testRegisterScreenTitleVisible	Verify that the screen title is visible	Check if the screen title “Create an Account” is displayed.	Title "Create an Account" is displayed
testRegisterScreenSubtitleVisible	Verify that the screen subtitle is visible	Check if the subtitle “Please enter your details to register” is displayed.	Subtitle "Please enter your details to register" is displayed
testSignInNavigation	Ensure that clicking the Sign In link navigates to the login screen	Click Sign In link	"Welcome back" screen is displayed
testEmptyEmailValidation	Validate error when email is empty	Empty email field “” and valid password “Password123!” entered, then click Sign Up	Error: "Email cannot be empty" is displayed.
testInvalidEmailFormats	Validate error for invalid email formats	Enter various invalid emails - "invalidemail", - "missing@dot", - "missing.domain@", - "@missing.namepart", - "spaces in@email.com",	Error: "Please enter a valid email address" is displayed.
testValidEmailFormats	Validate no error for correct email formats	Enter valid emails - "test@example.com", - "firstname.lastname@example.com", - "email@subdomain.example.com", - "1234567890@example.com",	No errors shown
testEmptyPasswordValidation	Validate error when password is empty	Valid email “test@example.com”, with an empty password, then click Sign Up	Error: "Password cannot be empty" will be displayed

testPasswordTooShort	Validate password length requirements	Enter short password like "short"	Password error displayed “Password must be at least 8 characters long and include an uppercase letter, a lowercase letter, a digit, and a special character.”.
testPasswordMissingUppercase	Validate password missing uppercase letter	Enter password like "alllowercase1!"	Password error displayed “Password must be at least 8 characters long and include an uppercase letter, a lowercase letter, a digit, and a special character.”.
testPasswordMissingLowercase	Validate password missing lowercase letter	Enter password like "ALLUPPERCASE1!"	Password error displayed “Password must be at least 8 characters long and include an uppercase letter, a lowercase letter, a digit, and a special character.”.
testPasswordMissingDigit	Validate password missing a digit	Enter password like "NoDigitsHere!"	Password error displayed “Password must be at least 8 characters long and include an uppercase letter, a lowercase letter, a digit, and a special character.”.
testPasswordMissingSpecialChar	Validate password missing special character	Enter password like "MissingSpecial1"	Password error displayed “Password must be at least 8 characters long and include an uppercase letter, a lowercase letter, a digit, and a special character.”.
testPasswordOnlyDigits	Validate password made of only digits	Enter password like "12345678"	Password error displayed “Password must be at least 8 characters long and include an uppercase letter, a lowercase letter, a digit, and a special character.”.
testPasswordOnlySpecialChars	Validate password made of only special characters	Enter password like "!@#$%^&*"	Password error displayed “Password must be at least 8 characters long and include an uppercase letter, a lowercase letter, a digit, and a special character.”.
testValidPasswordFormat	Validate no error for a strong password	Enter "StrongPass123!"	No error shown

testRealTimeEmailValidation	Check real-time email validation errors update correctly	Enter invalid “invalid” then valid “valid@example.com”email	Error shown “Please enter a valid email address” then removed when corrected email entered.
testRealTimePasswordValidation	Check real-time password validation errors update correctly	Enter weak password “weak” and then a strong password “StrongPass123!”	Error shown “Password must be at least 8 characters long and include an uppercase letter, a lowercase letter, a digit, and a special character.” then removed when correctly formatted password entered.
testSuccessfulRegistrationNavigation	Ensure successful registration and navigates to next screen	Enter a unique email "test" + System.currentTimeMillis() + "@example.com" and valid password “Password123!”	"Welcome back" and login screen is displayed, indicating registration was successful
testDuplicateEmailRegistration	Validate error for duplicate email	Enter an already registered email “test@gmail.com”	Error: "This email is already in use. Please use a different email." is displayed.

Coverage: This provides coverage and verifies the presence and correct rendering of key UI elements such as the logo, email and password input, sign-up button, sign-in link, and relevant titles. It also ensures proper form validation by checking for correct email and password formats. Navigation logic is tested through the functionality of the sign-in link and by confirming successful sign-up redirects. Additionally, error handling is thoroughly covered, including checks for missing fields, invalid input formats and attempts at duplicate registration.
Dependencies: Espresso for UI interaction, JUnit4 for test structure, ActivityScenarioRule for launching activities and Custom Matchers (hasTextInputLayoutErrorText and testPasswordValidation from Helpers folder)
Testing Achievements: Verified that all essential UI elements are present and visible. Confirmed field validation logic for both Email and Password fields. Ensured navigation flows between Registration and Login screens work. Ensured that real-time feedback is correctly provided to users during input. Tested both success and failure scenarios for registration flow. Confirmed error messages match expected behavior under various invalid inputs



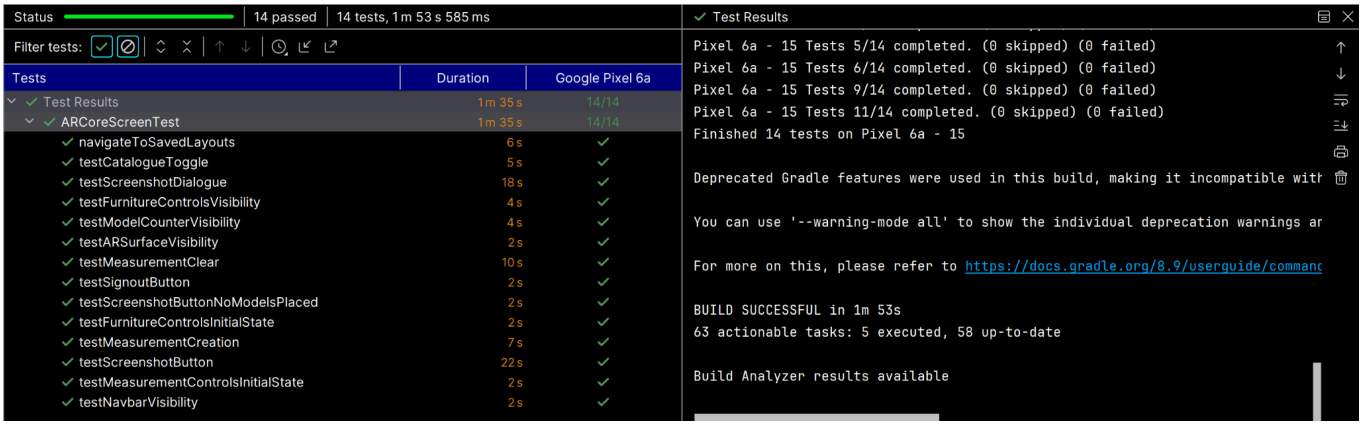
ARCore User Interface Test

Test Class: ARCoreScreenTest
Package: com.example.arrangeit (Android test)
Purpose: Verify UI functionality and user interactions in the ARCore screen of the ArrangeIt application.
Pre-setup: A Glide setup integrating (custom GlideIdlingResource and GlideTestRule) are used to ensure Espresso waits for Glide image loading to complete during UI tests by monitoring background jobs and disabling animations or transformations.

Test Name	Objective	Test Cases and Input	Expected Result
testNavbarVisibility	Verify the bottom navigation bar and its icons are visible	Launch ARCore screen and check the visibility of navigation bar elements	Bottom navigation bar is displayed with Saved Layouts, Catalogue, Measure Tool and LogOut icons.
testARSurfaceVisibility	Confirm AR surface renders properly	Initialise ARCore session	AR fragment container is visible
testFurnitureControlsInitialState	Validate initial state of the furniture manipulation controls	Launch AR view with no placed models	Furniture controls and model counter are hidden
testCatalogueToggle	Ensure catalogue loads correctly.	Click catalogue icon in navbar	Catalogue fragment becomes visible after 2 seconds
testFurnitureControlsVisibility	Verify furniture manipulation UI appears when models are placed	Programmatically set placedModelsCount = 1 to trigger the control buttons display	Furniture controls panel appears with Move/Delete/Rotate buttons visible
testModelCounterVisibility	Test model counter functionality	Set placedModelsCount = 1, click "Clear All" button and click confirm	Counter shows "Models: 1" initially and after clicking “Clear All” and confirming, the placedModelsCount resets to 0
testSignoutButton	Verify logout functionality	Click logout button in navbar	Returns to login screen where "Welcome back" should be visible as it is on the login UI
testMeasurementControlsInitialState	Validate measurement tool initial state	Launch AR view without measurements	Clear measurement button is hidden
testMeasurementCreation	Test measurement line creation	Enter measure mode and simulate two screen touches (one of each point placement)	Measurement line created between points and the distance is calculated. The “Clear Measurement” button becomes visible
testMeasurementClear	Verify measurement clearing	Create measurement similar to the previous test and click the clear button	Measurement disappears after clicking and the “Clear Measurement” button is hidden again
navigateToSavedLayouts	Test saved layouts navigation	Click saved layouts icon in the navbar	"Saved Layouts" screen appears

testScreenshotButton	Verify layout saving workflow after placing furniture	Place a test furniture model and click the camera/screenshot button. Enter a layout name “”Test Layout”.	Save dialogue appears and the layout saves successfully
testScreenshotButtonNoModelsPlaced	Validate save button behavior with no models	Click save button with empty scene	Save button remains active and no save dialogue appears. No save occurs as there are no models placed.
testScreenshotDialogue	Verify save dialogue components	Place furniture and open save dialogue	Dialog shows "Save Layout" title, Name input field and the Save/Cancel buttons.

Coverage: These UI tests validate AR surface rendering to ensure proper visualisation of 3D models in the real world. Navigation controls are tested to validate smooth transitions between core functionalities like catalogue browsing, measurement tools and layout saving. The furniture manipulation interface (movement, rotation, and deletion) is covered to guarantee responsive user interactions. Additionally, the measurement tool’s accuracy and behavior are verified, along with the reliability of the layout saving system and the overall user authentication flow.
Dependencies: AndroidX Espresso, JUnit4, UI Automator, ARCore SDK, ARCore Sceneform, waitFor (Helpers folder), Firebase Authentication, Glide, ActivityRule, GrantPermissionRule, Glide Idling Resource.
Testing Achievements: These tests achieve full coverage of the essential UI components and user interactions, including navigation, furniture manipulation, measurements, and layout saving. They handle asynchronous operations and permission-dependent features using idling resources and rules for reliable execution. Edge cases such as no-model states, dialogue interactions and dynamic UI visibility are also validated.

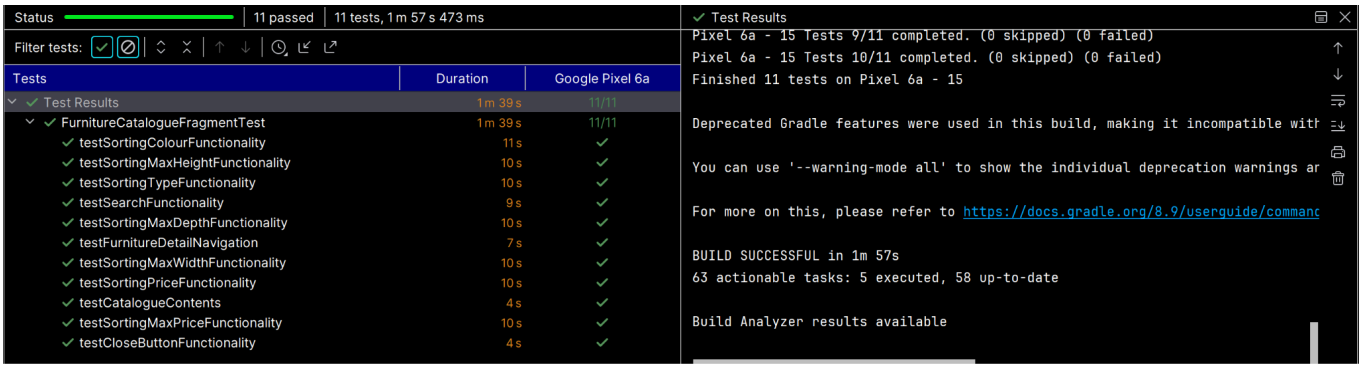


Furniture Catalogue User Interface Test

Test Class: FurnitureCatalogueFragmentTest
Package: com.example.arrangeit (Android test)
Purpose: Verify UI functionality and user interactions in the Furniture Catalogue screen
Pre-setup: A Glide setup integrating (a custom GlideIdlingResource and GlideTestRule) are used to ensure Espresso waits for Glide image loading to complete during UI tests by monitoring background jobs and disabling animations or transformations.

Test Name	Objective	Test Cases and Input	Expected Result
testCatalogueContents	Verify basic catalogue UI components (close, search and filter icons)	Navigate to catalogue screen	The Close button, Search Bar and Filter Icon should be visible.
testSearchFunctionality	Test furniture item search	Enter "chair" into the search bar.	RecyclerView updates with matching items
testSortingPriceFunctionality	Validate price sorting	Open the filter panel, select "Price: Low to High" and click Apply	Items in the RecyclerView update and display in ascending price order
testSortingColourFunctionality	Tests colour filtering spinner	Open the filter panel, select “Blue” in the colour field and click Apply.	Only blue furniture items displayed
testSortingTypeFunctionality	Verify type filtering	Open the filter panel, select “Chair” in the type field and click Apply.	Only chair-type items displayed
testSortingMaxPriceFunctionality	Test maximum price filter	Open the filter panel, enter max price as 200 in the Max Price field and click Apply.	Only items ≤ €200 displayed
testSortingMaxDepthFunctionality	Test depth filtering	Open the filter panel, enter max depth as 79cm in the Max Depth field and click Apply.	Only items ≤ 79cm depth displayed
testSortingMaxHeightFunctionality	Test height filtering	Open the filter panel, enter max height as 90cm in the Max Height field and click Apply.	Only items ≤ 90cm height displayed
testSortingMaxWidthFunctionality	Test width filtering	Open the filter panel, enter max width as 200cm in the Max Width field and click Apply.	Only items ≤ 200cm width displayed
testCloseButtonFunctionality	Verify catalogue is closed	Click the close button at the top left corner.	Catalogue fragment closes
testFurnitureDetailNavigation	Test item detail view	Click "Modern Arm Chair" item	Detail screen opens for the “Modern Arm Chair” and the name and price are displayed

Coverage: Search functionality and Multi-criteria filtering tested thoroughly. Navigation between the ARCore, Catalogue and Furniture Details page tested.
Dependencies: AndroidX Espresso, SceneForm, UI Automator, Glide (image loading), Firebase Firestore (data), ARCore (navigation context), JUnit4, Glide Idling Resource, waitFor (Helpers folder), ActivityScenarioRule, GrantPermissionsRule,
Testing Achievements: 100% filter/sort combinations validated and realistic swipe gestures for scrollable filters implemented. Asynchronous data loading handled and cross-component navigation verified. Edge cases tested (empty results, keyboard states)

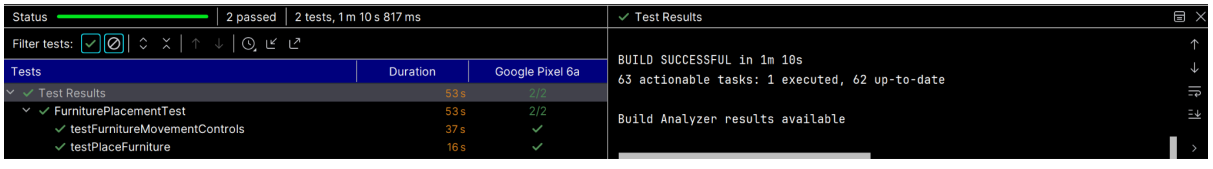


Furniture Placement User Interface Tests

Test Class: FurniturePlacementTest
Package: com.example.arrangeit (test)
Purpose: Verify furniture placement, movement and manipulation in the AR environment.
Pre-setup: A Glide setup integrating (a custom GlideIdlingResource and GlideTestRule) are used to ensure Espresso waits for Glide image loading to complete during UI tests by monitoring background jobs and disabling animations or transformations.

Test Name	Objective	Test Cases	Expected Result
testPlaceFurniture	Validate furniture item placement in AR view	Navigate to the catalogue and select a random furniture item from the list. Place the item in AR space by clicking the “Place in AR” button	Catalogue items appear on the screen and after choosing an item, the item details appear. After clicking “Place in AR” the model will be placed at the screen center. Model counter updates to "Models: 1"
testFurnitureMovementControls	Test all furniture manipulation controls (move, rotate, delete)	Place furniture (as previous test), activate move mode and perform a drag gesture For rotation, activate rotate mode and perform rotation gesture. Compare quaternion values. For deletion click delete button	Movement: Item position changes Rotation: Quaternion values change (≥0.0001f delta) Deletion: Model counter hides, controls disappear

Coverage: Core placement workflow and 3D manipulation controls (Translation, Rotation and Deletion) tested. Model counter updates and gesture recognition is also tested.
Dependencies: ARCore Sceneform, AndroidX Espresso, UI Automator, Motion event simulation, Quaternion math operations, Glide, Glide Idling Resource, waitFor (helpers folder), ActivityScenarioRule, GrantPermissionsRule, JUnit4, Firebase.
Testing Achievement: Verified movement of models and validated quaternion-based rotation with realistic gesture interactions. Full AR manipulation workflow coverage.



Saved Layouts User Interface Tests

Test Class: SavedLayoutsTest
Package: com.example.arrangeit (test)
Purpose: Verify the complete workflow of saving and viewing AR furniture layouts
Pre-setup: A Glide setup integrating (a custom GlideIdlingResource and GlideTestRule) are used to ensure Espresso waits for Glide image loading to complete during UI tests by monitoring background jobs and disabling animations or transformations.

Test Name	Objective	Test Cases	Expected Result
testCompleteFlowFrom PlacementToSavedLayouts	Validate end-to-end layout saving and retrieval	Place test furniture in AR space and save the layout with the name "Test Living Room". Navigate to Saved Layouts screen and verify the layout appears in gallery	Save dialogue appears with name input after inputting the name, the layout saves successfully after processing the action. After clicking into saved layouts the screen displays and a non-empty grid view appears with a layout card with correct name “Test Living Room”

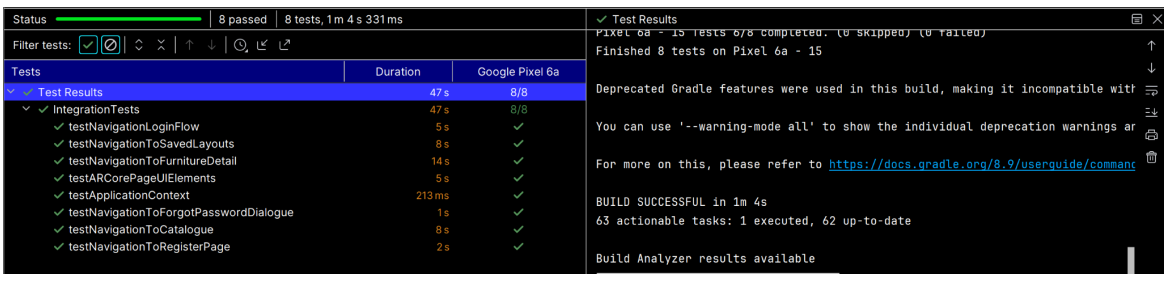
Coverage: AR scene capture with layout naming workflow, Firebase saving operation and gallery retrieval tested.
Dependencies: Firebase Storage/Realtime DB, ARCore SDK and SceneForm, Glide (image loading), AndroidX Navigation Component, UI Automator, Glide, Glide Idling Resource, waitFor (helpers folder), ActivityScenarioRule, GrantPermissionsRule, JUnit4.
Testing Achievement: Verified full user journey from AR → Firebase → Gallery, Validated persistent storage retrieval and tested authenticated user flow

Integration Tests

Test Class: IntegrationTests
Package: com.example.arrangeit (Android test)
Purpose: Verify end-to-end user flows and cross-component interactions across the application
Pre setup: Signs out any existing Firebase user to ensure a clean test environment and Espresso Intents used to verify activity navigation. TEST_EMAIL set as "test@example.com"" and TEST_PASSWORD set as "Password123!".

Test Name	Objective	Test Cases and Input	Expected Result
testApplicationContext	Validate correct application package context	Launch application context check	Package name matches "com.example.arrangeit"
testNavigationToRegisterPage	Test sign-up flow initiation	Click sign-up button from login screen	RegisterPage activity launches and email and password fields visible. Intent verification from swapping activity passes
testNavigationToForgotPasswordDialogue	Verify password recovery flow	Click forgot password link and interact with dialogue	Password reset dialogue appears and contains email field, reset and cancel buttons. Dialogue is dismissed on cancel
testNavigationLoginFlow	Validate authenticated user journey	Enter test credentials and click login	Successful navigation to ARCorePage where the Intent matches the ARCorePage component.
testARCorePageUIElements	Verify AR interface components	Launch AR view as authenticated user (login first and then access ARCore as a registered user)	AR fragment is visible after login and the navigation bar is present.
testNavigationToCatalogue	Test catalogue access flow	From AR view, open the catalogue and close the catalogue	RecyclerView displays furniture items and the close button functions properly in displaying no catalogue.
testNavigationToFurnitureDetail	Validate item detail to AR placement flow	Select "Modern Arm Chair" and initiate AR placement feature.	Detail screen shows the item name and the "Place in AR" button is visible after swipe gesture. Returns to AR view with navigation bar.
testNavigationToSavedLayouts	Test saved layouts gallery access	Click saved layouts/screenshots icon from AR view	"Saved Layouts" title appears to confirm the activity has changed.

Coverage: Authentication flows (login/register/recovery) tested and core navigation paths confirmed (AR interface components, Catalogue system integration, Saved layouts access)
Dependencies: Firebase Authentication, AndroidX Navigation Component, Espresso and Espresso Intents, UI Automator, ARCore SceneForm, JUnit4, Helpers (folder), ActivityScenarioRule, Glide, Glide Idling Resource
Testing Achievements: This validates all critical user journeys while maintaining clean test isolation through proper setup/teardown of authentication state and activity scenarios. The tests combine Espresso for UI validation with Intents for component verification, ensuring both visual and architectural correctness.

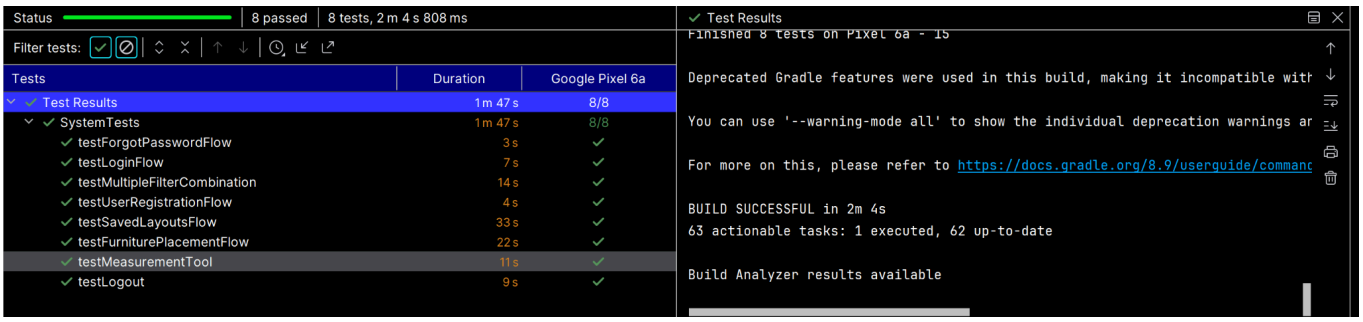


System Tests

Test Class: SystemTests
Package: com.example.arrangeit (Android test)
Purpose: Validate complete end-to-end user workflows and complex system interactions
Pre setup: Signs out any existing Firebase user to ensure a clean test environment and Espresso Intents used to verify activity navigation. TEST_EMAIL set as "test@example.com"" and TEST_PASSWORD set as "Password123!".

Test Name	Objective	Test Cases and Input	Expected Result
testUserRegistrationFlow	Verify new user account creation	Enter a unique email "test@example.com" and System.currentTimeMillis() used to input a unique account and valid password "Password123!"	Registration form displays correctly and redirects to the login screen after submission. Timestamp ensures unique test emails
testLoginFlow	Test authenticated session establishment	Enter valid test credentials "test@example.com" and "Password123!". Submit login form	Successful navigation to ARCorePage with intent verification to confirm correct activity launch. 1-second loading buffer to allow for complete transition.
testForgotPasswordFlow	Validate password recovery system	Initiate password reset and enter registered email "test@example.com". Submit reset request	Reset dialogue fields appear when entering email. Dialogue box hidden after submitting reset request and login screen visible (Login email box present)
testFurniturePlacementFlow	Verify complete AR furniture placement	Navigate from Login → Catalogue → Select "Modern Arm Chair" Place the item in the AR environment by clicking "Place in AR". Click to place the model.	Model counter updates to "Models: 1" and Item responds to center-screen placement. 5-second AR scene stabilisation period
testSavedLayoutsFlow	Test layout saving system	Place test furniture (as performed in previous test), Save as "Test Layout 1" and navigate to saved layouts	Layout appears in the grid with the correct name displayed. There is a minimum of 1 item in the gallery
testMeasurementTool	Validate AR measurement functionality	Enter measure mode and create measurement line with random coordinates	Clear button appears after measurement with a 3 second stabilisation between points. Coordinates avoid navigation bar area
testLogout	Verify session termination	Enter an authenticated session and click logout	Returns to login screen with the login form visible after 2 seconds
testMultipleFilterCombination	Test complex catalogue filtering	Apply filters with Type: Chair, Color: Grey, Max price: €200 and press Apply	RecyclerView shows ≥1 matching item. 2 second results loading time to allow for glide to load in the images

Coverage: Authentication lifecycle (register/login/recover/logout) tested fully and AR core functionality (placement/measurement) validated. Data persistence (layout saving) and Complex UI interactions (multi-filter combinations verified. Cross-component navigation, Error handling (timing/timeouts) were covered.
Dependencies: Firebase Authentication, ARCore SDK and Sceneform, Firebase Realtime Database, UI Automator (for complex gestures), Espresso Intents (navigation verification), Helpers (folder), System Clock, ActivityScenarioRule, GrantPermissionsRule.
Testing Achievements: This validates all critical user journeys while maintaining realistic timing expectations and handling complex interaction scenarios. The tests combine precise UI validation with architectural verification through intent matching, ensuring both surface-level functionality and deeper system integration work as designed.



AdHocTests

Real-World Environment Validation

During adhoc testing we evaluated furniture placement in diverse real-world conditions to assess AR tracking reliability. We used the app in environments with varying lighting such as brightly lit rooms and dimly lit rooms. From this, we implemented a toast message “Too dark to detect surfaces. Please move to a well-lit area.” when the scene was too dark for plane detection. We tested on different surfaces (wooden floors, carpets, table tops) to ensure the AR anchors remained stable. Room sizes were also a factor and we tested in small room spaces versus large open areas to verify that the app could place furniture appropriately.

Feature Switching & Measurement Cleanup

A key discovery during testing was that switching between app features mid-placement (such as jumping from furniture mode to measurement tool) sometimes left residual measurement lines or UI inconsistencies. To fix this, we introduced a “Clear” button that only appears when measurements are active. Similarly, we enforced dynamic UI rules where model manipulation buttons, model selected and the counter only appear when a model is placed, preventing confusion in an empty AR space.

Gesture Optimisation for Usability

Early testing revealed that two-finger rotation gestures were difficult, especially on smaller screens. Based on feedback we replaced it with a one-finger circular motion for rotation which proved more intuitive. Drag-to-move was retained but fine-tuned for smoother responsiveness.

Cloud Sync & Data Persistence Verification

To validate upload reliability, tests were done on saving layouts, logging out and relaunching the app to confirm that designs persisted. We also manually checked Firebase Storage to verify that screenshot thumbnails and layout metadata were properly stored and remained accessible after reloads. No sync failures were observed during testing.

Performance & Stability Observations

During prolonged AR sessions (20+ minutes), we monitored battery drain, overheating, and memory usage. Fortunately, no significant performance degradation or crashes were detected.

User Testing

To evaluate the usability and user friendliness of the application we conducted user testing with five participants across varying age groups, ensuring diverse perspectives on functionality and design. Participants were asked to complete consent forms and to complete three core tasks:

1. **Room Scanning:** Use the room scanning feature to detect walls and floor space for better furniture placement.
2. **Placing & Arranging Furniture:** Browse the furniture catalogue, select any furniture item of your choice, and place it in a suitable location in your room. After placing the first item, choose one or two more furniture pieces and arrange them to fit well within the space.
3. **Removing & Replacing Furniture:** Remove one of the previously placed furniture items and replace them with new one from the catalogue. Ensure they are arranged appropriately within the space.

After these tasks were complete, the user also filled out a general feedback question on their overall satisfaction with the application. The data collected was mainly through quantitative ratings (1-5) and qualitative feedback. The questionnaires and supplementary data can be found in the appendix section.

Reported Findings

Task 1:

The room scanning feature received positive feedback with 60% of our users rating it as "Very Effective" 5/5 and the remaining 40% as "Effective" 4/5. All participants confirmed that the application correctly aligned furniture with the scanned room dimensions, demonstrating strong spatial accuracy. Notably, no technical issues were reported during scanning, indicating a reliable experience. These results suggest that the room scanning functionality performs well in real-world use cases.

Task 2:

The furniture placement and arrangement feature also received mostly positive feedback with 80% finding the catalogue access "Very Intuitive" 5/5 with the remaining 20% rating it 4/5. 60% rated the visual appearance of items as "Very Appealing" (5/5). The AR placement process was also smooth, with 60% describing it as "Very Easy" (5/5) and 100% confirming realistic, accurate placement with proper snapping/alignment. Users also found it easy to add and arrange multiple items, with another 60% rating this task as "Very Easy" (5/5). For this test, some feedback issues were noted

“Some of the 3D model boundary area would cause difficulties in placing other models beside it.”
“There is a delay waiting for the images to load for the models”
“Some of the models looked smaller when placed than in their details page”

These are areas we will be aiming to improve in further development. Much of the feedback pertains to the 3D models that were used which were open-source and can vary in quality. In the future we aim to partner with furniture retailers as a tool that they can integrate into their existing business and therefore use professionally optimised models, ensuring better accuracy. The image loading delays stem from our current backend (Firebase) and image loading library (Glide). We will prioritise optimising this workflow either through caching improvements or alternative solutions to create a smoother browsing experience.

Task 3:

All five users found removing existing furniture very easy, giving it 5/5. Similarly, adding and arranging furniture was rated as easy or very easy by all users, with 80% selecting the highest rating. Additionally, all participants agreed that the furniture placement and movement felt smooth and realistic. No issues were reported when replacing furniture, indicating a seamless and intuitive user experience.

General Feedback:

The general feedback from users was overwhelmingly positive and all five participants indicated they were highly likely to use the application for home decor planning in the future and would recommend it to others. Satisfaction levels were also high, with 80% rating their overall experience as very satisfying and the remaining 20% as satisfied. There were also suggestions for future features that could be implemented to enhance the application such as

“AI model suggestions based on the data given about what you are looking for”
“A purchasing section for the models”
These suggestions will be taken into consideration in further developing the application as possible new features.

Appendices

User Testing Question Form

Task 1

Task 1: Room Scanning
Instruction: Use the room scanning feature to detect walls and floor space for better furniture placement.

Survey for Task 1:

1. How effective was the room scanning feature in detecting your space? (Scale: 1-5, 1 = Not Effective, 5 = Very Effective) *

0 1 2 3 4 5

2. Did the application correctly align the furniture with the scanned room dimensions? *

- Yes
- No
- Other: _____

3. Did you experience any technical issues with room scanning? If so, please describe.

Your answer

Task 2

Task 2: Placing and Arranging Furniture

Instruction: Browse the furniture catalogue, select any furniture item of your choice, and place it in a suitable location in your room. After placing the first item, choose one or two more furniture pieces and arrange them to fit well within the space.

Survey for Task 2:

1. How easy was accessing and selecting items from the catalogue? (Scale: 1-5, 1 = Not Intuitive, 5 = Very Intuitive) *

1 2 3 4 5

2. How visually appealing was the display of furniture items in the catalogue? (Scale: 1-5, 1 = Not Appealing, 5 = Very Appealing) *

1 2 3 4 5

3. How easy was it to place the furniture in the AR environment? (Scale: 1-5, 1 = Very Difficult, 5 = Very Easy) *

1 2 3 4 5

4. Did the furniture placement feel accurate and realistic? *

- Yes
- No
- Other...

5. How easy was it to add and arrange multiple furniture items? (Scale: 1-5, 1 = Very Difficult, 5 = Very Easy) *

1 2 3 4 5

6. Did the furniture snap or align correctly with the environment when moved? *

- Yes
- No
- Other...

7. Did you experience any difficulties browsing the catalogue, placing, or arranging the furniture? If so, please describe.

Long-answer text

Task 3

Task 3: Removing and Replacing Furniture

Instruction: Remove one of the previously placed furniture items and replace them with new one from the catalogue. Ensure they are arranged appropriately within the space.

Survey for Task 3:

1. How easy was it to remove an existing furniture item? (Scale: 1-5, 1 = Very Difficult, 5 = Very Easy) *

1 2 3 4 5

2. How easy was it to add and arrange another furniture? (Scale: 1-5, 1 = Very Difficult, 5 = Very Easy) *

1 2 3 4 5

3. Did the furniture placement and movement feel smooth and realistic? *

- Yes
- No
- Other...

4. Did you encounter any issues while replacing furniture? If so, please describe.

Long-answer text

General Feedback

General Feedback:

Description (optional)

How likely are you to use this application for home decor planning in the future? (Scale: 1-5)

1 2 3 4 5

Would you recommend this application to others interested in home decor planning?

- Yes
- No
- Other...

Was there any feature you expected but did not find in the application? If so, please describe.

Long-answer text

How satisfied were you with the overall experience? (Scale: 1-5, 1 = Not Satisfied, 5 = Very Satisfied)

1 2 3 4 5

User Testing Consent Form

Clarification of the purpose of the research

This study aims to evaluate the usability and effectiveness of an Augmented Reality (AR)-Based Virtual Home Decor Application. The research will involve participants interacting with the app to test key features such as AR object placement, furniture catalogue browsing, room scanning and saved room layouts. The goal is to collect feedback to improve the app's functionality and user experience.

As part of this study, data will be collected on participants' interactions with the app, including responses to surveys and questionnaires. The data will be processed solely for research purposes and will remain confidential.

Confirmation of particular requirements as highlighted in the Participant Information Sheet:
Participant – please complete the following (Circle Yes or No for each question)

I have read the Participant Information Sheet (or had it read to me) *

- ◯ Yes
- ◯ No

I understand the information provided *

- ◯ Yes
- ◯ No

I have had an opportunity to ask questions and discuss this study *

- ◯ Yes
- ◯ No

I have received satisfactory answers to all my questions *

- ◯ Yes
- ◯ No

I am aware that my interview will be audiotaped (if necessary) *

- ◯ Yes
- ◯ No

- I understand that my participation is voluntary and that I can withdraw at any time without any consequences. *

- I understand that my data will be collected, anonymized, and securely stored, and I consent to its use for research purposes.

- I have read and understood the information in this form. My questions and concerns have been answered by the researchers, and I have a copy of this consent form. Therefore, I consent to:

- ☐ Participating in this study
- ☐ Being recorded (if applicable)

Submit Clear form

User Testing Results

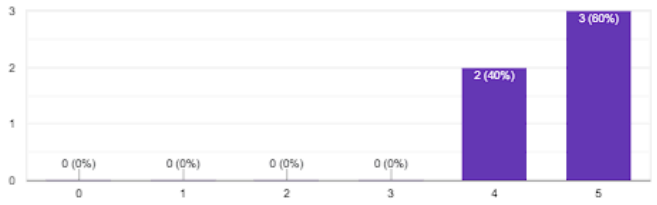
Task 1

Task 1: Room Scanning

1. How effective was the room scanning feature in detecting your space? (Scale: 1-5, 1 = Not Effective, 5 = Very Effective)

Copy chart

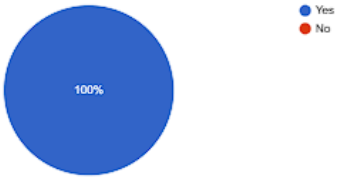
5 responses



2. Did the application correctly align the furniture with the scanned room dimensions?

Copy chart

5 responses



3. Did you experience any technical issues with room scanning? If so, please describe.

0 responses

No responses yet for this question.

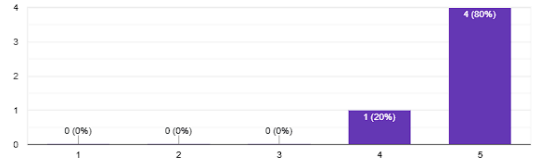
Task 2

Task 2: Placing and Arranging Furniture

1. How easy was accessing and selecting items from the catalogue? (Scale: 1-5, 1 = Not Intuitive, 5 = Very Intuitive)

Copy chart

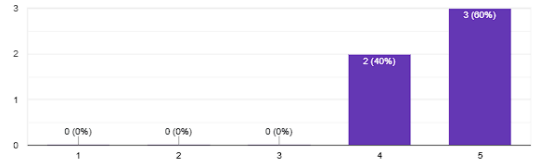
5 responses



2. How visually appealing was the display of furniture items in the catalogue? (Scale: 1-5, 1 = Not Appealing, 5 = Very Appealing)

Copy chart

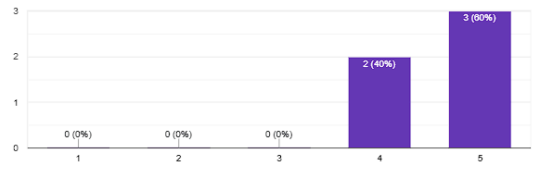
5 responses



3. How easy was it to place the furniture in the AR environment? (Scale: 1-5, 1 = Very Difficult, 5 = Very Easy)

Copy chart

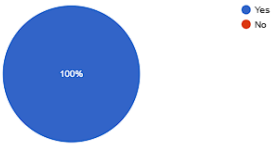
5 responses



4. Did the furniture placement feel accurate and realistic?

Copy chart

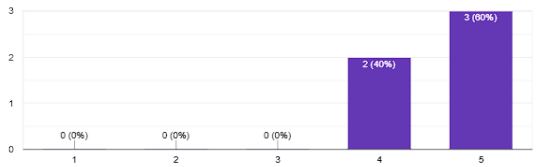
5 responses



5. How easy was it to add and arrange multiple furniture items? (Scale: 1-5, 1 = Very Difficult, 5 = Very Easy)

Copy chart

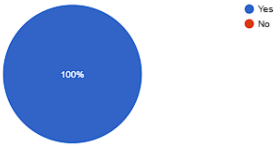
5 responses



6. Did the furniture snap or align correctly with the environment when moved?

Copy chart

5 responses



7. Did you experience any difficulties browsing the catalogue, placing, or arranging the furniture? If so, please describe.

3 responses

- Some of the 3D models boundary area would cause difficulties in placing other models beside it
- There is a delay waiting for the images to load for the models
- Some of the models looked smaller when placed than in their details page

Task 3

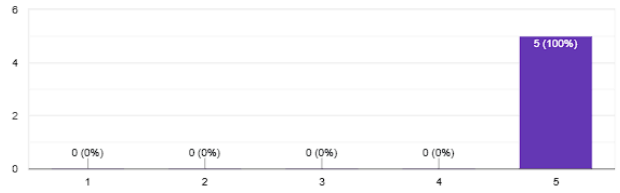
General Feedback

Task 3: Removing and Replacing Furniture

1. How easy was it to remove an existing furniture item? (Scale: 1-5, 1 = Very Difficult, 5 = Very Easy)

Copy chart

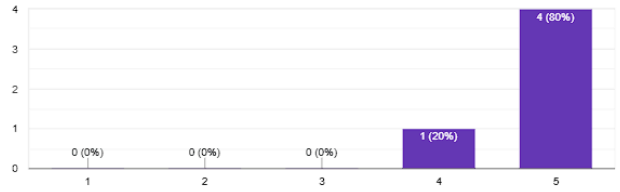
5 responses



2. How easy was it to add and arrange another furniture? (Scale: 1-5, 1 = Very Difficult, 5 = Very Easy)

Copy chart

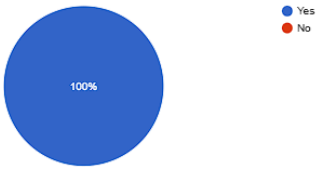
5 responses



3. Did the furniture placement and movement feel smooth and realistic?

Copy chart

5 responses



4. Did you encounter any issues while replacing furniture? If so, please describe.

0 responses

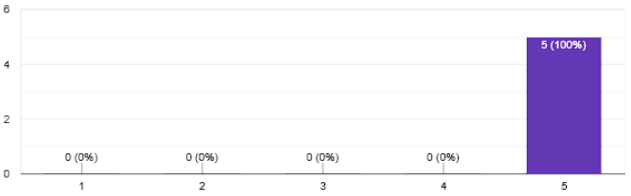
No responses yet for this question.

General Feedback:

How likely are you to use this application for home decor planning in the future? (Scale: 1-5)

Copy chart

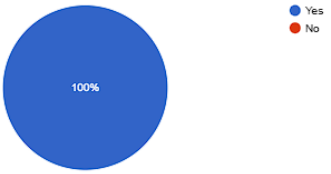
5 responses



Would you recommend this application to others interested in home decor planning?

Copy chart

5 responses



Was there any feature you expected but did not find in the application? If so, please describe.

2 responses

AI model suggestions based on the data given about what you are looking for

A purchasing section for the models

How satisfied were you with the overall experience? (Scale: 1-5, 1 = Not Satisfied, 5 = Very Satisfied)

Copy chart

5 responses