UNIVERSITY OF ALBERTA

ECE 493

SOFTWARE SYSTEMS DESIGN PROJECT

Memento

Test Plan and Suite

Owen Cooke

Levi Thomas

Mahmud Jamal

April 9, 2025

1. Software Analysis

1.1 Frontend Code Structure

The source code for Memento's mobile app client is located under the `/app/src` subdirectory within the monorepo and is divided into several logical sub-directories:

- `/api-client`: contains code automatically generated via Hey API, an OpenAPI to TypeScript codegen library, used for interacting with the backend from the client
- `/app`: contains the Expo file-based routing structure for the app, where each file corresponds to a singular screen in the UI
- `/components`: contains modular UI components used as building blocks (such as cards, modals, lists, etc.) within larger screens
- `/context`: contains React Context UI components and providers for passing commonly required data to lower descendants throughout the app component tree
- `/hooks`: contains custom React Hooks for modularizing common UI state or effects used by multiple components or pages
- `/libs`: contains modularized TypeScript functions and logic used across the app, independent of UI state.

From this analysis of the code organization, it is clear that `/libs` is the only directory containing files appropriate for white-box unit testing. This is because it contains TypeScript functions that do not rely on the state of the UI. Unit testing React Native components is possible but requires a component tree to be rendered, which can be hard to configure, mock, and test (in terms of relative time and effort). Given the time constraints for this software prototype, we have chosen to instead rely on black-box testing via automated end-to-end (E2E) UI testing to cover the code implemented within the other sub-directories, as they are primarily UI-based.

One last note: We will not be testing `supabase.ts` inside of `/libs` since it simply initializes a Supabase client used for managing authentication (no actual logic is executed).

1.2 Backend Code Structure

The source code for Memento's backend server is located under the `/backend/server` subdirectory within the monorepo and is divided into three logical sub-directories:

- `/api`: contains the REST-based FastAPI route functions necessary to expose backend logic to clients over the HTTP protocol
- `/config`: contains global configuration settings for the server, such as loading environment variables and configuring the logger levels
- `/services`: contains modularized Python functions and logic used across the server's API routes, including modules such as database queries, cloud storage functions, image processing routines, and recommendation algorithms.

The `/services` module is the most suitable for white-box unit testing as it encapsulates the core business logic and modular methods used throughout the backend. Since Memento's API routes were designed to follow a simple pattern—validating the client's request, invoking modular methods from `/services`, and returning the result as an HTTP response—white-box testing of `/services` will provide thorough coverage of the underlying functionality. API route behavior will be verified through black-box E2E testing, ensuring strong coverage across the backend's internal logic and external functionality.

2. Test Coverage Criteria

Based on the software analysis for Memento conducted above, white-box unit tests and suites will be designed to meet the following test coverage criteria:

- 100% branch coverage of the files under `/app/src/libs`
- 100% branch coverage of the files under `/backend/server/services`

For black-box testing, a suite of system-level acceptance tests will be developed to evaluate the application end-to-end through the mobile app's user interface. These tests will cover each of the functional requirements (FRs) outlined in the SRS document at least once to verify the software system's functionality from the end user perspective. The tests will also be required to list explicitly which FRs are being covered as part of the user flow.

It is important to note the emphasis on functional coverage through realistic user flows here. This is due to the resource constraint required to run automated E2E tests on a mobile device or emulator; the test development and test execution time would simply be too inefficient to consider all combinations of inputs.

3. Test Suite Design

3.1 White-Box Testing

3.1.1 Frontend Unit Tests

To white-box test the frontend logic, Memento leverages Jest, a JavaScript testing framework that provides support for various assertions and easy mocking methods. Additionally, it supports generating HTML code coverage reports without any additional configuration needed.

Test suites are defined under the `/app/tests/unit` directory, with each file mapping to a corresponding file from the source code using the name convention `[source_file].test.ts`. Each test is fully executable and labelled using Jest's declarative "describe" and "test" methods to summarize what is being tested. Where necessary, external dependencies and mock data are created under `/app/tests/unit/mocks` to facilitate isolated unit testing of each module. Tests generally follow the Arrange-Act-Assert (AAA) pattern for setting up mocks, calling the method under test, and performing assertions to ensure the expected result.

To execute the white-box unit tests for Memento's frontend:

1. Clone the GitHub repository and cd into the `/app` subdirectory
2. Ensure your host system has all the necessary software package managers and frameworks installed, as outlined in Memento's frontend prerequisites.
3. Install the dependencies necessary for the app using `pnpm i`
4. Execute the automated Jest test suites using `pnpm test`

This will execute the automated test suites and the corresponding results will be printed to the terminal. Additionally, the HTML code coverage report will be generated and can be viewed by opening `/app/coverage/index.html` in your browser.

3.1.2 Backend Unit Tests

To white-box test the backend logic, Memento uses a combination of two Python testing frameworks:

- unittest: included in the standard library, it provides simple and useful methods for mocking such as MagicMock, AsyncMock, and patch.
- pytest: enhances test execution through "fixtures" (easily re-usable mocks, injected at runtime by the framework into test methods) and automatic generation of HTML code coverage reports

Test suites are defined under the `/backend/tests` directory, with each file mapping to a corresponding file or module from the source code using the name convention `test_[source_module].py`. External dependencies such as Supabase and test data are mocked under `/backend/tests/fixtures`, leveraging Pytest's fixtures to create modular mocks that can be easily reused across multiple tests.

Each test is labelled using a Python docstring to summarize what is being tested and is fully executable. Tests generally follow the Given-When-Then pattern, where we first set up mock data and methods, then execute the method under test, and finally, perform assertions on the result to ensure it's as expected.

To execute the white-box unit tests for Memento's backend:

1. Clone the GitHub repository and cd into the `/backend` subdirectory
2. Ensure your host system has all the necessary software package managers and frameworks installed, as outlined in Memento's backend prerequisites.
3. Install the Python dependencies necessary for the backend using `poetry install`
4. Execute the automated test suites using `poetry run test`

This will execute the automated test suites, the results will be printed in the terminal, and the HTML code coverage report will be generated for viewing (by opening `/backend/htmlcov/index.html` in your browser).

3.2 Black-Box Testing

To facilitate black-box testing, Memento uses Maestro, a simple and effective UI testing framework for Mobile and Web applications.

Automated test suites are defined under the `/app/tests/e2e` directory, with each file defining a flow of how the user would interact with the application with the naming convention `[user-flow].yml`. Flows that are used across many other flows are defined under `/app/tests/e2e/util`. These tests require running instances of both the back-end and front-end applications and do not mock the responses received from the back-end, thus, they rely on the robusticity and accuracy of the API routes.

To execute the black-box UI tests:

1. Clone the GitHub repository
2. Ensure your host system has all the necessary software package managers and frameworks installed, as outlined in Memento's backend prerequisites and Memento's frontend prerequisites.
3. cd into the `/backend` subdirectory

4. Install the Python dependencies using poetry install
5. Start the back-end server using `poetry run start`
6. cd into the `/app` subdirectory
7. Install the dependencies necessary for the app using `pnpm i`
8. Connect the physical device or Android emulator and install the app using `pnpm android`
9. Execute all test flows using `pnpm test:e2e` or a specific flow with `maestro test <path-to-flow.yml>`

This will execute the automated test suites by simulating presses and inputs on the physical or emulated device and can be very slow.

3.2.1 Automated UI Test Cases

Many UI test cases require specific setup data. Before running each set of test cases please ensure that the data matches the required setup detailed here. Required setup is also stated at the top on each flow file.

Test Flow	Description	FRs Covered	Expected Result
`create-memento.yml`	Create a new memento using both image upload methods and typed form input. Also verifies metadata extraction and background removal of images.	FR-4, FR-5, FR-6, FR-7, FR-9, FR-10, FR-17, FR-19, FR-20	1. Navigate to Create Memento page 2. Open the camera and take 1 photo 3. Decline background removal 4. Select 1 image from device library 5. Accept background removal 6. Metadata extracted; autofills date/location form field 7. Can fill in form fields by typing 8. Submit the create form 9. New memento appears in grid view
`edit-memento.yml`	Edit an existing memento by replacing the image, updating form fields, and verifying that changes are saved or discarded based on user actions.	FR-30, FR-31, FR-33	1. Navigate to Edit Memento page 2. Cancel edits without saving, ensuring changes are discarded 3. Remove all old images, replacing them with a new image 4. Edit the form fields (caption and location) 5. Submit the edit form 6. Verify that the new caption and location are displayed on View page
`view-memento.yml`			
`delete-memento.yml`	Delete an existing memento.	FR-34	1. Navigate to the View Memento page for an existing memento 2. Click on the Trash icon 3. Click on Confirm button 4. Verify the memento is removed from the Mementos tab
`share-memento.yml`	Verify that the share option appears for mementos.	FR-54	1. Navigate to the View Memento page for an existing memento 2. Click on the Share icon 3. Sharing screen appears
`bulk-create.yml`	Create multiple	FR-22, FR-23,	1. Navigate to Bulk Create page 2. Select 2 photos from device library

	mementos in bulk by selecting multiple photos from the gallery, filling out a form for each, and adding them to a new collection.	FR-24	3. Verify 2 groups made (1 per photo) 4. Accept/reject background removals 5. Fill out details for each group (caption and location) and save 6. Submit the bulk create form 7. Add the mementos to a new collection via popup modal 8. Verify a new collection created, with correct title and mementos displayed
`generate-collage.yml`	Generate a collage for an existing collection.	FR-53, FR-55	1. Navigate to the Collections tab 2. Navigate to the View Collection page for an existing collection 3. Generate a collage for the collection 4. Verify the collection image and Export button appears
`view-map.yml`	Verify that existing collections appear on the map view.	FR-51, FR-52	1. Navigate to the Collections tab 2. Verify that there is an existing collection 3. Navigate to the Map View 4. Verify the collection marker appears on the map 5. Click on the collection marker 6. Verify collection fields appear on the View Collection page

The **Filter Memento** and **Create Collection** flows are orchestrated by top-level `filter.yml` and `collections.yml` flows. The individual flows are outlined here because they require initial setup data to be provided and to improve readability.

Required Setup					
	Caption	**Date**	**Location**	**Detected Text**	**Image Label**
Memento 1	Hello caption surfer	04-03-2025	Edmonton, AB, Canada	N/A	Envelope
Memento 2	Lorem	04-05-2025	Toronto, ON, Canada	Hello detected text surfer	N/A

Collection Flows			
Test Flow	**Description**	**FRs Covered**	**Expected Result**

Collection Flows			
`create-collection.yml`	Create a new collection with memento and typed form input. Also verifies metadata extraction.	FR-35, FR-36, FR-37, FR-41, FR-47, FR-48, FR-49, FR-50	1. Navigate to collections tab 2. Open collection create screen 3. Select memento 4. Accept derived metadata 5. Verify that metadata was filled 6. Fill form fields 7. Submit create form 8. New collection in grid view
`edit-collection.yml`	Edit a collection, remove and add memento, modify form details.	FR-42, FR-43, FR-44, FR-45	1. Navigate to collections tab 2. Press on collection card 3. Press edit button 4. Input garbage text 5. Exit without saving 6. Verify that changes were not saved 7. Press edit button 8. Remove selected memento 9. Add other memento 10. Accept derived metadata 11. Verify that metadata was filled 12. Edit form fields 13. Save changes 14. Verify that fields were changed on view collection screen
`view-collection.yml`	View a collection and it's details	FR-3	1. Navigate to collections tab 2. Press on collection card 3. Verify that collection details are visible
`delete-collection.yml`	Delete a collection and verify that it is removed from collection tab	FR-46	1. Navigate to collections tab 2. Press on collection card 3. Press on delete button 4. Confirm delete 5. Verify that collection was removed from collections tab

Filter Memento Flows			
Test Flow	**Description**	**FRs Covered**	**Expected Result**
`filter-date.yml`	Filter mementos on date	FR-12	1. Set start date 3rd, end date 4th, apply filter 2. Verify Memento 1 visible 3. Set start date 4th, end date 5th, apply filter 4. Verify Memento 2 visible
`filter-location.yml`	Filter mementos on location	FR-13	1. Set location filter to “Alberta, Canada” 2. Verify Memento 1 visible

Filter Memento Flows			
			3. Verify Memento 2 not visible
`filter-detected-text.yml`	Filter mementos on the detected text of their associated images	FR-14	1. Add “detect” to search bar 2. Verify Memento 1 no visible 3. Verify Memento 2 visible
`filter-caption.yml`	Filter mementos on keywords in their caption	FR-15	1. Add “caption” to search bar 2. Verify Memento 1 visible 3. Verify Memento 2 not visible
`filter-image-label.yml`	Filter mementos on the classification of their associated images	FR-16	1. Set image label filter to “Envelope” 2. Verify Memento 1 visible 3. Verify Memento 2 not visible

3.2.2 Manual UI Test Cases

A few manual test cases are necessary to verify certain functionalities outlined in the SRS due to limitations imposed by the E2E testing framework (Maestro) or other technical constraints:

- **Drag-and-Drop Features:** Maestro does not support “hold-and-drag” actions necessary for changing the thumbnail for a memento (FR-21 and FR-32) and for dragging images between groups on the Bulk Create page (FR-22). This feature is currently open (to-do) as an enhancement issue on their GitHub.
- **Google Authentication:** Automating Google login (FR-1 and FR-2) for E2E testing is unreliable due to potential login denial or CAPTCHA challenges. Also, it becomes more difficult to maintain E2E test consistency across devices and securely manage test credentials. It was simpler to build a login backdoor into the app for an E2E test user that bypasses Google login (done by setting the `EXPO_PUBLIC_E2E_TESTING` environment variable to true and reloading the app server).

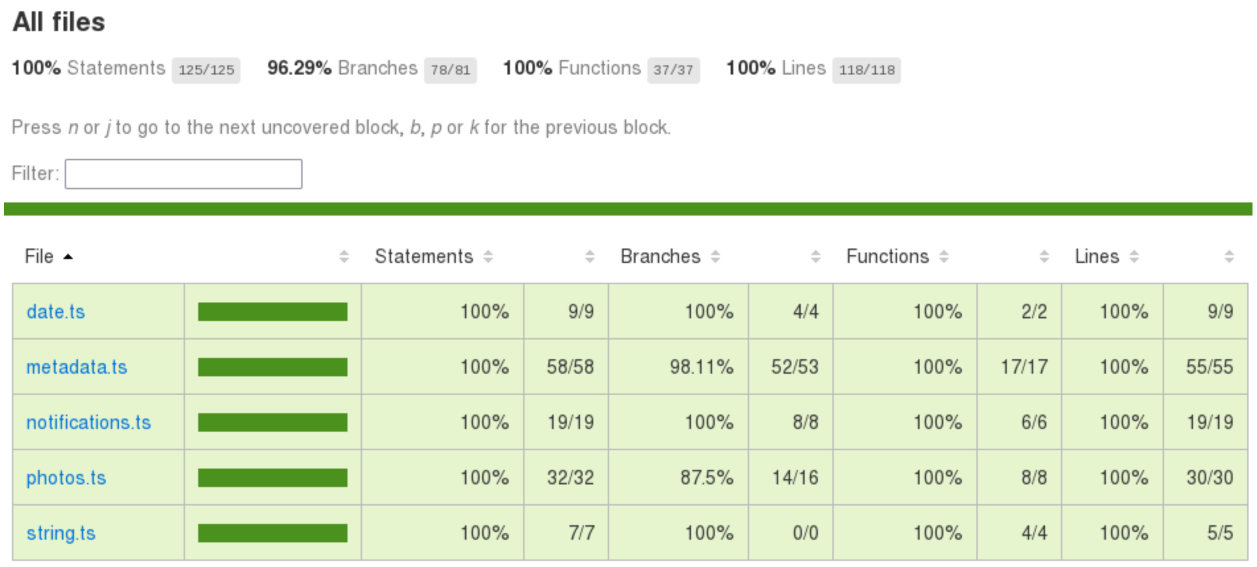
The test cases below should be executed manually on the device by following the test steps outlined and verifying the expected results at each step.

Test ID	**Description**	**FRs Covered**	**Test Steps**	**Expected Result**
0	User can register a new account in Memento by signing in with a Google account they haven’t used before and complete the onboarding flow.	FR-1	1. If the environment variable `EXPO_PUBLIC_E2E_TESTING` is set to true, remove it, restart the app server, and re-open the app. 2. If logged in, log out by navigating back to the Mementos or Collections tab, click your avatar in the header in the top right corner, and press the logout button. 3. Verify you are brought back to the	Current user should be able to log out. New users should be able to register by clicking the Sign In With Google button. New users should be

			Memento login screen. 4. Tap the Google Sign-in button and select a never-before-used Google account (this may require signing out of your Android device's Google account beforehand). 5. After Google authenticates, verify you are navigated to the "Welcome to Memento" / new user screen. 6. Enter your birthday and tap on the continue button. 7. Verify you are navigated to the Mementos tab, with your Google profile picture displayed in the header.	redirected to an onboarding screen to input their birthday (for special day reminders) before being navigated to the typical Mementos tab/home screen.
1	User can log into Memento by signing in with a Google account they previously registered with, and can see their previous app data.	FR-2	1. If the environment variable `EXPO_PUBLIC_E2E_TESTING` is set to true, remove it, restart the app server, and re-open the app. 2. If logged in, log out by navigating back to the Mementos or Collections tab, click your avatar in the header in the top right corner, and press the logout button. 3. Verify you are brought back to the Memento login screen. 4. Tap the Google Sign-in button and select a previously used Google account (this may automatically use your Android device's main Google account). 5. After Google authenticates, verify you are navigated to the Mementos tab, with your Google profile picture displayed in the header. 6. If previous mementos or collections were created for this account, verify that they are displayed on the respective tabs.	User can log out of their account, without losing any of their data. User can sign back into their account using the same Google account used previously. User is directed to the proper pages when logging in or out.
2	User can specify the thumbnail image when creating a new memento by dragging to re-order.	FR-21	7. Navigate to Create Memento page 8. Select 2 images from device library 9. Ignore background removals 10. Verify a star icon is shown for first image in the grid (thumbnail) 11. Change the thumbnail by long pressing on the second image until it expands, then drag it over top of the first image and release 12. Verify the images swapped places 13. Submit the create form 14. Verify that the new memento appears in the grid view, and that the image showing matches the image set as thumbnail	The user should be able to drag images into different positions within the photo grid used on the Create Memento page. The image that was first when the form was submitted should match the memento's thumbnail on subsequent screens.
3	User can change the thumbnail for a memento when editing,	FR-32	1. Click on an existing memento to navigate to View page 2. Click the "Edit" icon on bottom bar 3. Ensure at least 2 images are present	The user should be able to drag images into different positions

	using “drag-to-reorder” and the selected thumbnail should save.		(by adding one more if necessary) 4. Verify a star icon is shown for first image in the grid (thumbnail) 5. Change the thumbnail by long pressing on the second image until it expands, then drag it over top of the first image and release 6. Verify the images swapped places 7. Submit the edit form, which navigates back to the View page 8. Verify that image ordering on the View page updates (the image selected as thumbnail should now be first)	within the photo grid used on the Edit Memento page, allowing them to change its thumbnail. When the edit form is submitted (changes saved), the new thumbnail should be reflected on subsequent screens (such as the View page)
4	System notifies user on special days	FR-56	1. Remove user_info record for user from database 2. Login as user and set birthday as current day 3. Verify that birthday reminder is received	The user should be reminded through notification to add mementos on their birthday.
5	System notifies user of recommended collection	FR-38 FR-39 FR-40}	1. Create new memento with Edmonton location 2. Create new memento with Sherwood Park location 3. Create new memento with Nisku location 4. Press notification 5. Verify that those mementos are added to that collection	The user should be notified of a possible collection, recommended based on location density. Minimum recommendation size is three. When user presses notification, they are brought to pre-filled create collection screen

4. Test Results



All files

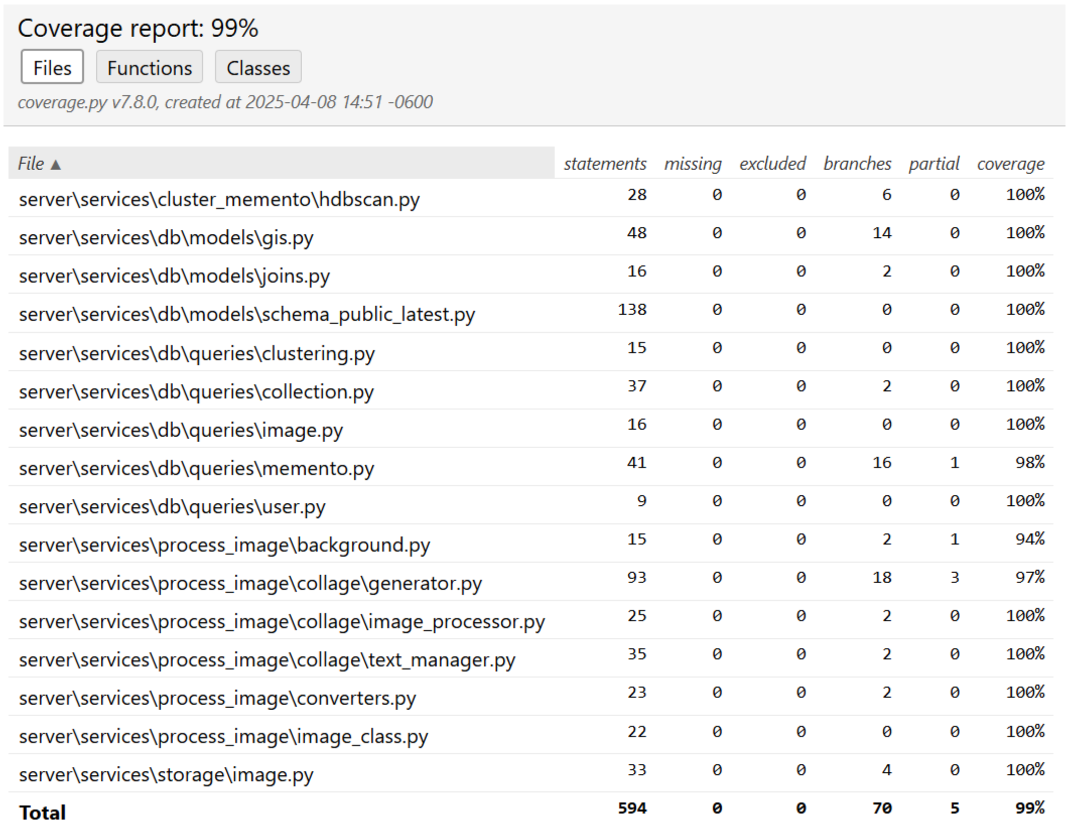
100% Statements 125/125 **96.29%** Branches 78/81 **100%** Functions 37/37 **100%** Lines 118/118

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

Filter:

File ▲		Statements		Branches		Functions		Lines	
date.ts		100%	9/9	100%	4/4	100%	2/2	100%	9/9
metadata.ts		100%	58/58	98.11%	52/53	100%	17/17	100%	55/55
notifications.ts		100%	19/19	100%	8/8	100%	6/6	100%	19/19
photos.ts		100%	32/32	87.5%	14/16	100%	8/8	100%	30/30
string.ts		100%	7/7	100%	0/0	100%	4/4	100%	5/5

Fig. 1. The coverage report displaying results for our frontend white-box unit tests



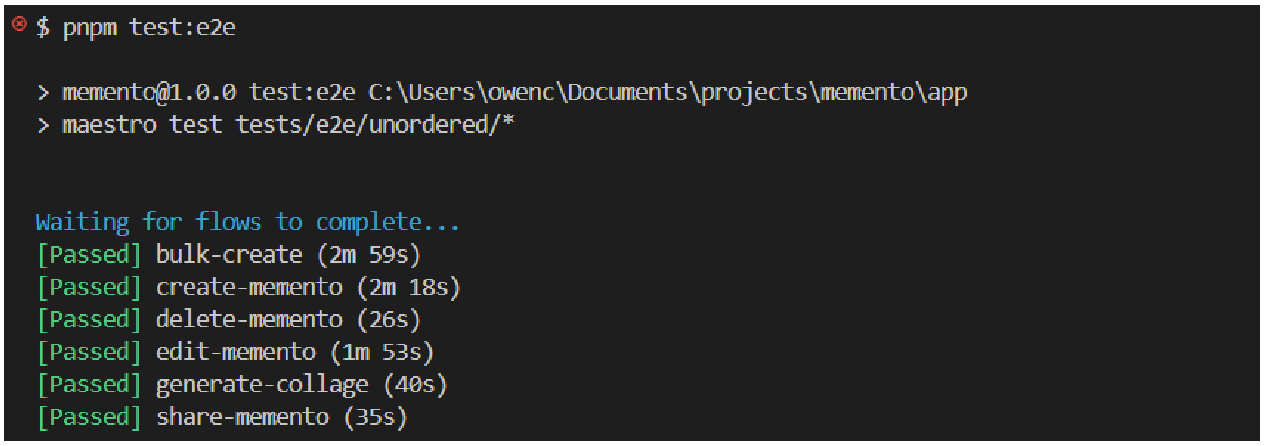
Coverage report: 99%

Files Functions Classes

coverage.py v7.8.0, created at 2025-04-08 14:51 -0600

File ▲	*statements*	*missing*	*excluded*	*branches*	*partial*	*coverage*
server\services\cluster_memento\hdbscan.py	28	0	0	6	0	100%
server\services\db\models\gis.py	48	0	0	14	0	100%
server\services\db\models\joins.py	16	0	0	2	0	100%
server\services\db\models\schema_public_latest.py	138	0	0	0	0	100%
server\services\db\queries\clustering.py	15	0	0	0	0	100%
server\services\db\queries\collection.py	37	0	0	2	0	100%
server\services\db\queries\image.py	16	0	0	0	0	100%
server\services\db\queries\memento.py	41	0	0	16	1	98%
server\services\db\queries\user.py	9	0	0	0	0	100%
server\services\process_image\background.py	15	0	0	2	1	94%
server\services\process_image\collage\generator.py	93	0	0	18	3	97%
server\services\process_image\collage\image_processor.py	25	0	0	2	0	100%
server\services\process_image\collage\text_manager.py	35	0	0	2	0	100%
server\services\process_image\converters.py	23	0	0	2	0	100%
server\services\process_image\image_class.py	22	0	0	0	0	100%
server\services\storage\image.py	33	0	0	4	0	100%
Total	**594**	**0**	**0**	**70**	**5**	**99%**

Fig. 2. The coverage report displaying results for our backend white-box unit tests



```
$ pnpm test:e2e

> memento@1.0.0 test:e2e C:\Users\owenc\Documents\projects\memento\app
> maestro test tests/e2e/unordered/*


Waiting for flows to complete...
[Passed] bulk-create (2m 59s)
[Passed] create-memento (2m 18s)
[Passed] delete-memento (26s)
[Passed] edit-memento (1m 53s)
[Passed] generate-collage (40s)
[Passed] share-memento (35s)
```

Fig. 3. The execution results of the automated black-box E2E tests