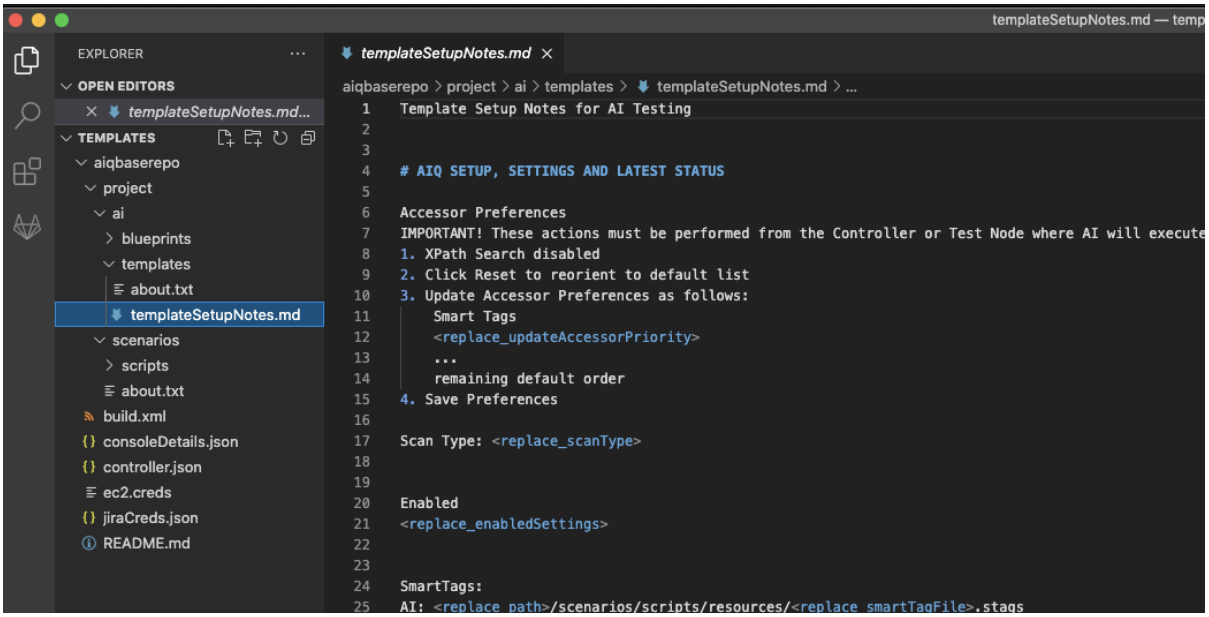
AIQ AI Implementation Strategy

Strategy, Thought Process and Driving Toward Success

1. Reset Assessor Preferences on Controller/TestNode where AIQ AI will run
    1. If Accessor Preferences for the AUT are already known, update and save those on Controller/TestNode where AIQ AI will run
2. Determine domains to include & which to avoid
    1. Instant Replay > Record, then
    2. Test Designer IDE > Record
        1. Navigate around the app, not worrying about what IDE is recording
        2. Stop Recording
    3. Return to Instant Replay, dump results into CSV, and determine Valid domains and if observed any ignore URLs
3. Begin tracking the settings in ai/templates/templateSetupNotes.md
    1. Adjust on Controller and TestNode(s) where AI will run
    2. (sample base repo structure)



4. Tags Only Execution with a twist: SmartTags + Tags Execution (with 1 dummy SmartTag)

1. *Note: You can run with SmartTags + Tags with one 'dummy' SmartTag to avoid having to create a new AISG in step 6 (below)*
2. Run first AISG using SmartTags + Tags with one 'dummy' SmartTag - only at ~15 - ~25 browsers (not too much)
3. Note observations such as:
    1. Accessors which need altering
        1. Use to determine optimal accessor preferences
        2. Use to determine SmartTags required for proper assessor usage
    2. Repetitive links?
        1. Determine classes and or characteristics of links which are hit multiple times, but should only be triggered once, or once per page, etc.
        2. Note ideas behind potential SmartTags needed
    3. Missing elements which visible in Visual Hints' capture but not picking up in page state element lists
        1. Note as potential SmartTags needed
    4. Data needed to get any type of depth and breadth?
        1. Create js (or synth, just not hash) csv with some sample data
5. Adjustments based on SmartTag + Tags (w/ 1 dummy SmartTag) Execution
    1. Adjust Assessor Preferences based on observations on Controller/TestNode where AIQ AI will run
    2. Create at least one SmartTag based on observations above
    3. Update templateSetupNotes.md to keep it in sync with findings
6. SmartTags + Tags Execution
    1. Update AISG with changes to SmartTags file (using File > Update SmartTags Library), and allow it to run again
    2. Again, note observations such as:
        1. How have accessors improved? Still need changes?
        2. How are SmartTags behaving? Make any necessary adjustments
        3. Remember: do not create SmartTags unless necessary
        4. Remember: Try to keep as much automatic as possible, as minimize "prescriptive" actions
        5. What other SmartTags are needed?
            1. SmartTags needed due to accessors?
            2. SmartTags needed due to ai hinting?
    3. Repeat until:
        1. AI not recursively actioning items it shouldn't
        2. AI able to find and interact properly with all necessary objects (varies by scope of AI / POC)
        3. SmartTags are helping drive areas automatically as much as possible
    4. NOTE: ***IF*** too many SmartTags are required to avoid recursive erroneous clicking, consider shifting to SmartTags + Inputs. Do not do so too soon, or you may create more SmartTag work than needed..
7. Run template from ant (or pipeline) to generate dashboard
    1. While the AISG is not 'complete', it's often interesting and beneficial to start generating dashboard results now, allowing you to observe the growth and differences more easily.

8. Data-driving the AISG
    1. Some applications require data to get almost anywhere, in such cases, the data driving likely needs to come earlier.
    2. Using the template created above, begin incorporating data into the AISG
    3. Careful not to overly prescribe too many actions
    4. But instead, map actions in page states to help exercise portions of the application which:
        1. Cannot be automatically exercised using SmartTags and auto-navigations
        2. Upon performing mapped actions in page states, reveal new areas of the application to AI
        3. Consider using Test Designer “snippets” with AI Hint feature to speed up (or replace) mapping data and actions within page states
    5. Avoid:
        1. Duplicating use cases already implemented in Test Designer which are very specific and narrow scoped flows, which have little area to branch out from when navigated.
        2. Expanding AI by only adding mapped actions on page states, and where I even with those additions is finding little to nothing else beyond the prescriptive steps
9. Ready for Validations When:
    1. If need validations sooner, keep in mind, and watch for the impact of adding validations, for example, if resulting pages or page states vary once the validation is added, if resulting data from one action must be validated and used for subsequent actions.
    2. Element accessors and SmartTags have stabilized
    3. AI is covering both breadth and depth of the application, with as much automatic as possible
    4. Page state and/or AI Hint script snippets provide the nudges needed to reveal additional areas of the application which AI then covers automatically
    5. Failed and Inconsistent actions are minimal if at all present. Note: Some failed actions will almost always exist. It is for the user to judge if it’s acceptable or not.
10. Validations:
    1. Create Validation Workbench trigger-based validations
    2. Create SmartTag Workbench based validations (adding additional SmartTags as needed).
    3. Add each validation as created (or small collections) to the existing blueprint template and run to validate the expected result. Repeat.
    4. Autonomous auto-validations
        1. Create
        2. Export to CSV (don’t forget to commit this CSV, before it is converted to JSON format upon import)
        3. Clean up validations, and provide more meaningful names (can be done quickly with find replace and formulas in excel)
        4. Retain only those applicable for your application
        5. Save and retain the CSV

6. Import the autonomous auto-validations into Validations Workbench and add to template
7. Run and test the auto-validations, and repeat cleanup until working as expected

11. Continue building and expanding based on observations
    1. A AISG is never complete. There is always more to explore, and many times additional discoveries within the same or new builds.
    2. Save the resulting AISG of each execution, analyze and compare prior executions, and expand or enhance based on your observations.

Above is an ideal world. Not all implementations entirely mesh with the ideal world but following this approach will ensure you implement AI wisely!

Last updated: September 06, 2023

If you have any feedback on this topic, please contact us at documentation@appvance.ai

---

Revision #7
Created 2026-07-31 15:38:39 UTC by Admin
Updated 2026-11-17 21:58:07 UTC by AI_Mananger