Improvements/Innovations Discovered:

1. Ovecome Limited Customer Base Limitations

The current product has the following two major limitations:

1. It can only support automated testing for JavaScript-based web applications, as far as I could infer from the documentation, operating the platform, output logs, and its implementation details.
2. Even in JS-based applications, the software must be a web-hosted or web-browser-accessible application, which implies that either the application is hosted on the public internet or it's hosted on a locally accessible host with a URL. Either way, this is a strict requirement, and it imposes a limit on the extent of software that could benefit from Gen-AI automated testing.

The implications of these two hard requirements are that the majority of software owners wouldn't be able to leverage Blinq.io's pioneering idea of automating testing via Gen-AI. So, one of the most important improvements, or to be more accurate, the next steps in the journey of perfecting Blinq.io, is to expand its capabilities to support alternate tech stacks and types of software systems, such as testing of remote scripts, High-Performance Computing (HPC) jobs, CLI applications, IoT systems, Blockchain infrastructures, and many more domains, all of which can significantly benefit from a tool like Blinq.io, in my opinion.

> While this is a relatively big task, it has great potential to benefit Blinq.io's clients and the software engineering community in general.

2. Enhance Coverage and Robustness of the Underlying Gen-AI Model in Gherkin Code

Make the Gherkin operations more focused and definitive. For example, the demo project includes the following operation in the Gherkin code:

```
Then verify "Thank you for your order" can be found in the page
```

While this works for a simple application, it's not an advisable way to verify if the outcome of the operation is the intended one. In more complex applications, if we vaguely signal to the automated tester to check if a particular string exists or not in the entire webpage or current viewport, it can open many possibilities for judgment errors in the model. For example:

1. The string may be dynamic and not a fixed value. It may depend on values that cannot be gathered and passed to the tester beforehand, values which are computed at runtime due to external agents.

2. Another example is if we don't have the actual implementation of the code, then we cannot simply state that if the output contains a particular string, then our intended behavior has occurred. The string could be set to be hidden or encoded in the output UI, and the tester may mistake some other occurrence for the intended occurrence, hence leading to an incorrect verdict.

> So in conclusion, an improvement would be to define strict unambiguous verification checks for the intended behavior, reducing the probability for interpretation errors.

> This improvement is moderately feasible and should be worth investigating further.

3. Increase Flexibility of the `Record Scenario` Feature

The `Record Scenario` feature seems to be fairly rigid in terms of the applications it can successfully record and understand to write the Gherkin code. It requires the application interface to have properly defined and labeled elements in the UI (like `Email_input_field`). However, this is highly unlikely even in most properly made, production-grade software systems, where every element in the UI is not perfectly and distinctly labeled. The conversion from the video to Gherkin code seems to lose information essential for complex scenarios, such as movement patterns of the mouse or unregistered changes in the UI state that result in specific outcomes.

Thus, according to me the Record Scenario feature has room for improvement in the following aspects:

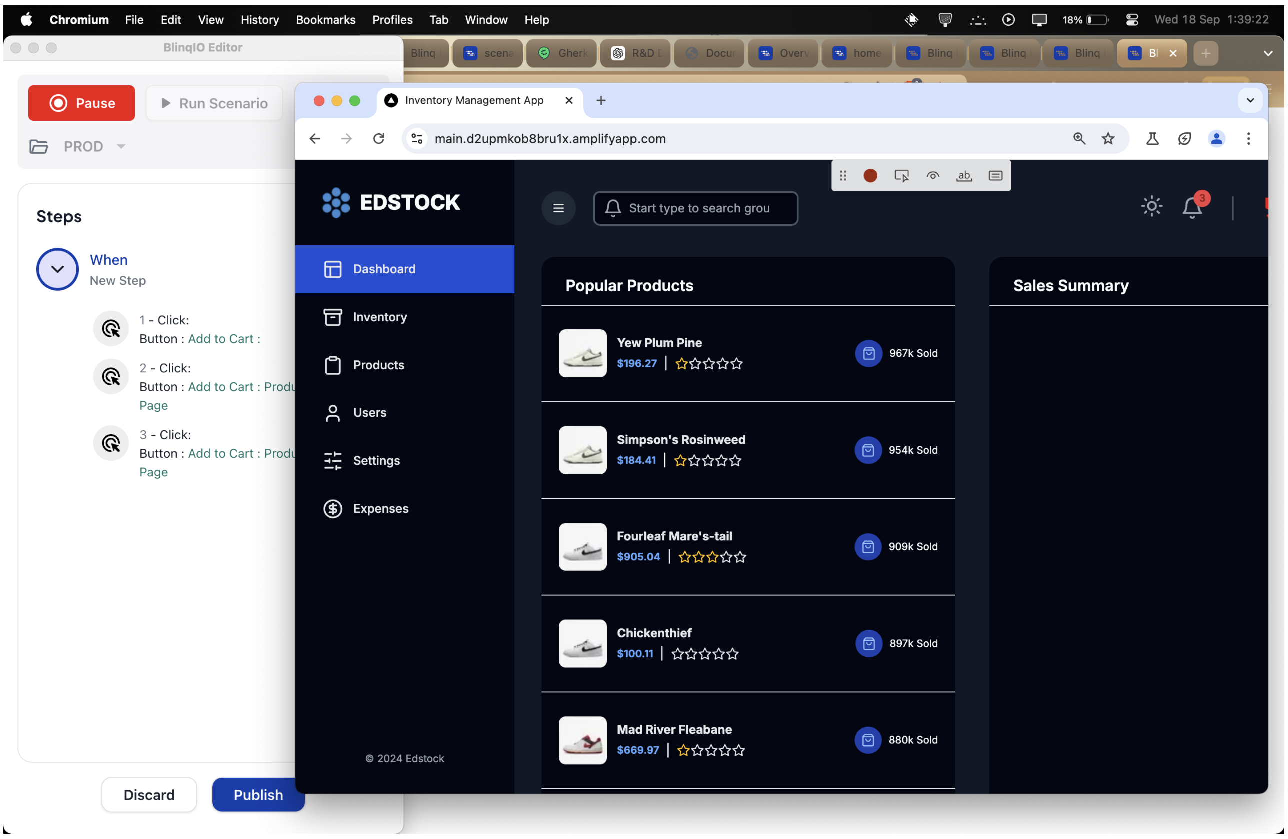
- Improve its ability to comprehend more complex scenarios.
- Improve its ability to predict and extrapolate by deepening its understanding of the user interface, instead of rigidly searching for exact matches of labels to elements' perceived names by the AI.
- Augment Record Scenario's functionality to also allow **speech input** in the feature's recording. The audio could be used as an additional input vector to better train the model to gain a deeper understanding of the user's motives and actions.

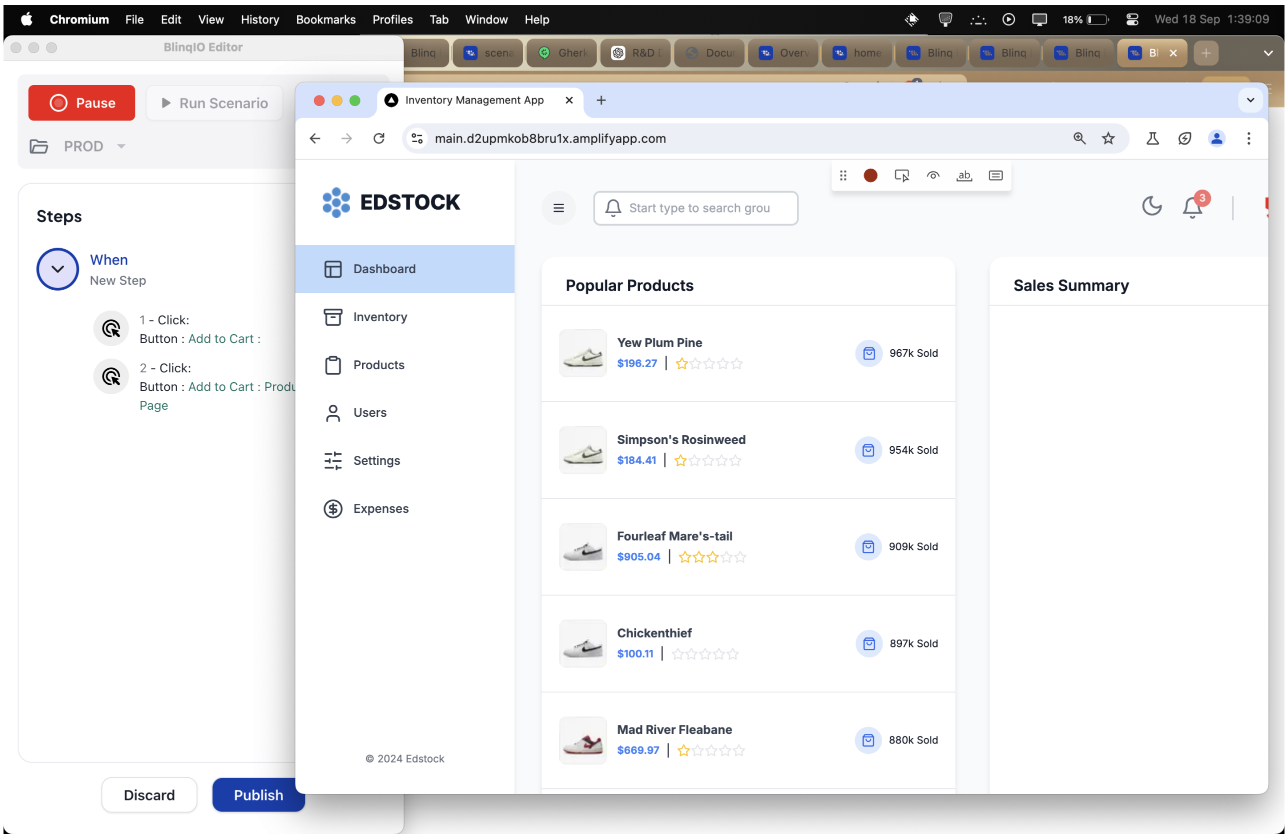
To test the flexibility or rigidity of the `Record Scenario` feature, I coded three simple web applications to test straightforward operational logic. Although the code was written following industry best practices and naming conventions, it was not built with automated testing as a requirement. The UI elements did not contain proper labels, as they were not part of the original requirements. As a result, the `Record Scenario` feature failed in several areas:

- It failed to detect all click events, keyboard input events, and alert events, despite these events being sparse.

- It failed to detect changes in the UI after detecting actions like clicks. For example, the task was to test if the change theme button on a website works properly to toggle between light and dark modes.

- It misidentified what the what the user is trying to imply in the UI in the many scenarios. In the screenshots below, I attempted to test an inventory dashboard I built, which has a button for toggling between light and dark mode themes. There is no mention of a `Add to Cart` button in the entire code base nor is it part of the utility a dashboard should provide, but still every time I clicked the toggle theme button, the AI mistakenly extrapolated it to be an `Add to Cart` button click event, perhaps due to biases while training the model, leading it to associate any unlabelled element in the top right part of the header of a commercial website containing products to have the highest probability of being related to a Cart.


BlinqIO Editor
Pause
Run Scenario
PROD
Steps
When
New Step
1 - Click:
Button : Add to Cart :
2 - Click:
Button : Add to Cart : Produ
Page
3 - Click:
Button : Add to Cart : Produ
Page
Discard
Publish
Inventory Management App
main.d2upmkob8bru1x.amplifyapp.com
EDSTOCK
Start type to search grou
Dashboard
Inventory
Products
Users
Settings
Expenses
© 2024 Edstock
Popular Products
Yew Plum Pine $196.27 967k Sold
Simpson's Rosinweed $184.41 954k Sold
Fourleaf Mare's-tail $905.04 909k Sold
Chickenthief $100.11 897k Sold
Mad River Fleabane $669.97 880k Sold
Sales Summary


BlinqIO Editor
Pause
Run Scenario
PROD
Steps
When
New Step
1 - Click:
Button : Add to Cart :
2 - Click:
Button : Add to Cart : Produ
Page
Discard
Publish
Inventory Management App
main.d2upmkob8bru1x.amplifyapp.com
EDSTOCK
Start type to search grou
Dashboard
Inventory
Products
Users
Settings
Expenses
© 2024 Edstock
Popular Products
Yew Plum Pine $196.27 967k Sold
Simpson's Rosinweed $184.41 954k Sold
Fourleaf Mare's-tail $905.04 909k Sold
Chickenthief $100.11 897k Sold
Mad River Fleabane $669.97 880k Sold
Sales Summary

Please note that since I only experimented with the free version of Blinq.io, its quite possible that the application's quality and performance in terms of the intelligence and capabilities of the Generative AI model were capped at a cost efficient threshold since you are providing a free of cost demo of the app so to save computational costs, a new potential user may get a much rudimentary version of the application, so I cannot comment on the actual accuracy and intelligence of the AI model with certainity, but there is certainly scope of innovation in the capabilities and accuracy of the current Gen-AI model.

> This is also a relatively difficult tasks but as the `Record Scenario` feature is one of the core features of the product, I believe it's worth a try. I would rate it moderately feasible.

4. Generating Realistic Test Data with Increased Coverage for Edge Cases

Currently, the platform relies on example-driven testing, or in other words, behavior-driven development, in the following two ways:

1. The AI infers the possible data values and uses simple and non-realistic examples (such as `test_user` for user name) to verify the intended behavior or it relies on randomly generated data (the example in the documentatin of a random string for a git repo init test).
2. Another way to input data is through CSV tables or User data features in the Test Data section, while this is a more pragmatic approach it still relies on the user to provide the clean and processed CSV files, which isn't always convenient for most users.

> So here I would like to present the following attempt to innovate for better test coverage using self generated datasets: Implement a functionality which is able to understand the correlation between various inputs and how they are processed, and generate on its own, a more diverse dataset with high variance, low correlation and realistic values to increase scenario coverage, it will more efficiently discover and test for edge cases, non-deterministic conditions and unexpected errors.

> This idea has the potential to further add significant value to the already existing revolutionary product, as testing quality is heavily dependent on the data used.

5. Other Miscellaneous Innovation/Improvement Ideas:

- Allow more granular control over the customization of tests to better deal with corner cases. This may be done by allowing the user to decide the characteristics and target functionality of the test.
- On top of the already existing self-healing of failed test scenarios, we can improve the Gen-AI model to better understand and report on the root cause of failing tests to better pin point and resolve dependency conflicts and bugs.
- The AI can be trained on it's own generated tests through a Reinforcement Learning from Human Feedback (RLHF) strategy by gathering user's feedback and satisfaction and feeding it back to the underlying model.
- Enhance the existing multi-language support to not only test translational accuracy but also cultural relevance, I/O formats and other nuances to be more welcoming of users from diverse cultures.

Improvements in Implementation and UX details

1. Issues with the `Open with VSCode` utility

The `Open with VSCode` button doesn't seem to work in the free version of the product. However, I followed the instructions on the `Debugging Test Scripts in Visual Studio Code` documentation page and ran the script as advised. The downloaded code bundle was missing the essential `.mjs` files in the features directory, which was the primary reason for downloading the project test files. The documentation does not mention this missing detail. The platform interface also differs from the current free version in key aspects. I suspect this may be because the documentation refers to an older version or the paid version of the product.

The improvements that I would incorporate are:

- Make the `Open with VSCode` button operational, as it is currently non-functional.
- Provide a downloadable code bundle in the form of a `.zip` file which consists of the scenario description and the `.mjs` files in the `features` directory,which are missing from the script's output. This will allow users to download the code bundle, work locally on test logic, and use the JavaScript Debug Terminal successfully in any code editor of their choice.

2. UI Performance Improvements

The website interface can be improved in terms of user experience by addressing performance issues caused by slow loading states between different sections of the application. For example, since the web interface is a Single Page Application, different routes on the page can be and should be cached on the client side for a better snappy user experience, especially the ones which render static or almost static data on past tests and runs, for instance when you are on the `Features` page and you navigate to `Reports` or `Dashboard` there is a significant wait time not just for the first time you visit them but for every subsequent time as well in the same session, but since these pages render mostly static data such as total number of features or information about `All Runs`, these can be easily cached for an instantaneous UX at the cost of minimal memory overhead, which is a very feasible tradeoff. There are more parts in the application which can benefit from client-side caching as well.

3. UI Appearance Improvements:

- The CSS styling is a bit off in a few places, for example the bottom left portion of the page where the exit button collides with the user name and email.
- Another instance is in the `Scenario Outline` text box where there is no inherent word wrapping and it just continues to the right limiting readability of the entire scenario at any given instant and if you want to see the entire scenario you need to manually enter line breaks which not only provide irregular formatting but also add unnecessary characters in the input data which could mislead the AI's interpretation and degrade the results.
- Another minor CSS flaw is the overlapping of the Save and Cancel buttons in this same text box which are opaque and block the user input beneath them. These UI issues can be fixed via minor CSS adjustments and attention to detail, which adds to the product's appeal.

4. Incomplete/Sparse/Outdated/Inconsistent documentation

I understand that this product is still undergoing development but I just wanted to mention this as a potential improvement.

Document submitted by: Yash Meena, ymsc98@gmail.com