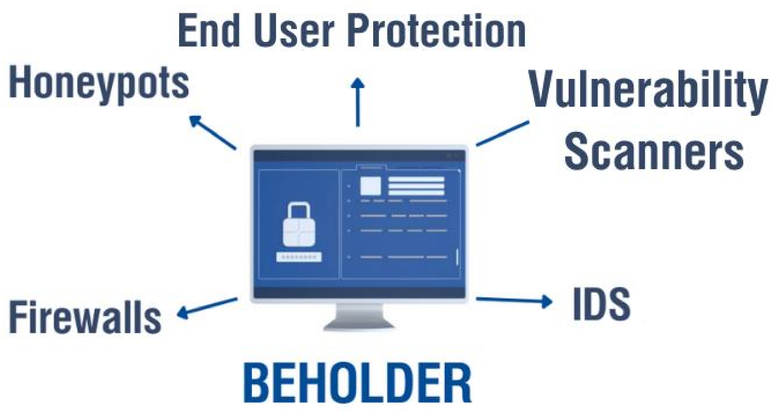
FERMILAB-POSTER-23-254-STUDENT

RSpec Testing in Beholder

Saud Ahmed, *Lewis University*, mentored by Jeny Teheran and Jason E. Ormes

Security and Emergency Management Division, Cybersecurity Team, Fermi National Accelerator Laboratory, Batavia, Illinois 60510



Introduction

In the realm of cybersecurity, safeguarding sensitive data and protecting critical infrastructure are of utmost importance. Beholder, a custom Ruby on Rails system, plays a pivotal role in fortifying network security at Fermilab. With Beholder as its central hub, security events from a range of detection systems can be efficiently and effectively collected and processed. This includes Nessus, IDS, honeypots, and firewalls. Seamlessly coordinating the flow of information, it ensures the swift and accurate transfer of these events to our blocking systems, encompassing firewalls, proxies, DNS systems, and other indispensable defense mechanisms. Taking note of the sophistication of Beholder's approach, it plays a crucial role in enhancing cybersecurity

This work was produced by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the U.S. Department of Energy. Publisher acknowledges the U.S. Government license to provide public access under the DOE Public Access Plan DOE Public Access Plan

This research was supported in part by the U.S. Department of Energy (DOE), Omni Technology Alliance Internship Program. The program is championed by the DOE's Office of Chief Information Officer (OCIO) and represents a partnership with the leadership of the Office of Economic Impact and Diversity, the Office of Science, the Office of Nuclear Energy, and the National Nuclear Security Agency. The program is administered by the Oak Ridge Institute for Science and Education.

Role of RSpec Testing in Beholder

- RSpec testing in Beholder is crucial for ensuring the application's reliability, data integrity, and security.
- It validates the core functionalities of the system, including the processing of security events and data transfer between components.
- RSpec tests help detect and prevent security vulnerabilities by simulating various scenarios and edge cases.
- Maintaining code quality is facilitated by RSpec, encouraging clean, modular, and maintainable code through Test-Driven Development (TDD) principles.
- RSpec tests serve as living documentation of the application, aiding communication and debugging among team members.

How does RSpec work ?

RSpec works as a testing framework that allows developers to verify and validate the functionality of their code. It follows a behavior-driven development (BDD) approach, where tests are written in a human-readable format. RSpec enables developers to define test cases, known as examples, that describe the expected behavior of the code. These examples are organized into groups, called contexts, to provide better readability and structure.

```
.....................................................
.....................................................

Finished in 10 seconds (files took 5.32 seconds to load)
300 examples, 0 failures

Coverage report generated for RSpec to /home/ahmeds/beho
```

The sample report displays various scenarios, models, and test cases, presenting a comprehensive number of examples.

RSpec also provides various matchers that allow developers to define specific expectations for their code. Matchers are used to compare actual values to expected values, enabling developers to check for conditions such as equality, containment, or truthiness.

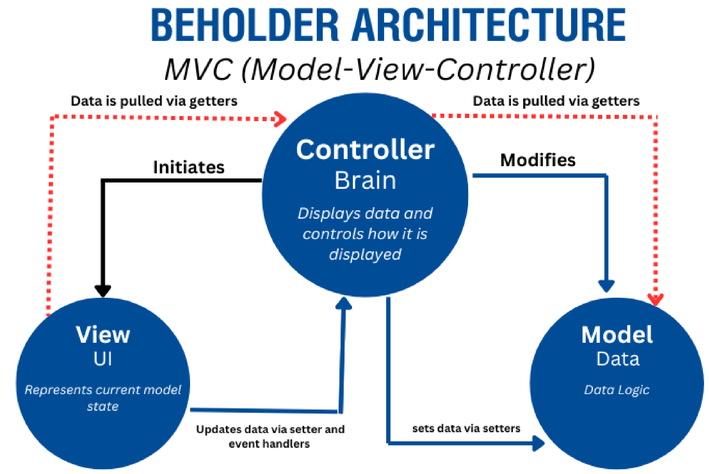
```
Failure/Error: expect(result).to eq("asdlfsjdkf8*W#NVSSDF")

  expected: "asdlfsjdkf8*W#NVSSDF"
       got: "=UTF-8?B?SGVsbG8gd29ybGQh?=!"
```

```
.............
......F.....
```

When the RSpec test suite is executed, it runs through each example, evaluating whether the actual behavior matches the expected behavior. If the expectations are met, the example passes, indicating that the code works as intended. However, if the expectations are not met, the example fails, indicating a potential issue in the code that needs to be addressed.

Implement RSpec in Beholder



Model Testing:
Create separate spec files for each model under the spec/model's directory.Use RSpec syntax (describe, context, it) and matchers to test model methods and validations.

View and Controller Testing:
Create spec/views and spec/controllers' directories for view and controller tests.For view testing, create a spec file for each view under spec/views and utilize the render_template matcher.
For controller testing, create a spec file for each controller under spec/controllers. Utilize RSpec's methods (get, post, patch, put, delete) to simulate HTTP requests and test controller actions..
.