Dana Huget, V00860786

Rocket.Chat
The All Important Community and their Modifiability of Rocket.Chat

Table of Contents:

Part 1: Introduction

The project I have chosen from the team mapping is Rocket.Chat, with sixteen thousand stars on GitHub and almost four hundred contributors making it an extensively documented project, which I believe to be a promising start for this assignment. Rocket.Chat is a free open-sourced Slack alternative team chat web service that runs privately on your own infrastructure (self-hosted which enables privacy for those concerned with the security of their data) in addition to their free desktop, iOS and Android app platforms. It combines tools for collaborating and tools for live chat and was launched on GitHub as a global community project. Rocket.Chat features real-time translation in 37 different languages, supports a wide variety of different plugins and APIs, and also offers cloud-based hosting for your team community needs for a fee. One of the main advantages of this open-source project over its competitors is the ability for developers to build their own chat platforms and create their own white label versions of Rocket.Chat. Rocket.Chat works with developers of all levels, such as state or university entities, to help build their own systems and in return developers contribute applicable code back to the Rocket.Chat project in a sort of symbiotic relationship (Engel, 2016). The prevalent importance of the community highlights an important business goal, which directs the quality attribute scenario in this report - the system needs to be able to easily adapt to integrate community feedback and improve functionality so that Rocket.Chat can stay ahead of its leading competitors. This use case modifiability scenario will be followed by a discussion on how the current architecture of Rocket.Chat meets the scenario. This report will also discuss the expansion of the use case quality attribute scenario for growth and how the system would handle changes from users on a larger scale for seamless modification.

Part 2: Quality Attribute Scenarios (QAS)

The chosen quality attribute, modifiability, demonstrates time required to make change, test it, and commit it. The specified UI Change scenario involves a change by a community contributor, or developer, to the user interface within the Rocket.Chat UI package and subsequent Rocket.Chat end-to-end UI testing of the modified functionality with the resulting passed change merged within one hour. The developer locates the desired UI module within the RocketChat ui package (see Figure 1) and performs their desired changes, then following the GitHub pull request template (see Figure 2). These template requirements include various style guidelines for the Rocket.Chat community contributors such as checking for no errors on ESLint prior to submitting a pull request. The developer would then submit a pull request for their changes and the changes would be merged within an hour. This is an anticipated average time to merge a pull request during the overall development process of Rocket.Chat (codescene.io CI/CD Readiness report). Figure 1 highlights the important modules a developer would access for any modification to the UI and the subsequent testing. The client modules within the UI package (components, lib, or views) have an inbound dependency from the packages file in Figure 1 and many outbound dependencies to external and public entities which are not included for these purposes. The UI tests can be found within the end-to-end module of the RocketChat tests (again, Figure 1), these can be used by the developer to ensure the proper functionality of their UI changes. Meteor provides developers a simple command line interface to seamlessly run the Rocket.Chat tests.

The growth of Rocket.Chat would most likely lead to an increase in users so that the contributor portion no longer outweighed the user base (Engel, 2016). This growth in user base with the continued goal of having a community feedback loop for development leads to the potential growth scenario in which users are able to make changes to their UIs. Customization of the user's personal interface would require internal system testing, which currently Rocket.Chat uses BrowserStack. The following tables provide a detailed look at both the modifiability scenario and the growth scenario previously mentioned.

QAS Scenario:

Aspect	Details
Scenario Name	User Interface (UI) Change
Business Goals	Be able to adapt the system to integrate community feedback and improve functionality so as to stay ahead of competitors
Quality Attributes	Modifiability
Stimulus	A change to UI via raising an issue on GitHub
Stimulus Source	Community developer
Response	Change is made to UI and there is a Git pull request (with no errors on eslint prior to PR - i.e. follows style guideline specifications)
Response Measure	The desired change passes end-to-end/ui tests and newly revised change is merged < 1hr

QAS Growth Scenario:

Aspect	Details
Scenario Name	Growth
Business Goals	User base increases and users are able to modify the system as well as developers
Quality Attributes	Modifiability
Stimulus	A change to UI via UI instead of GitHub
Stimulus Source	User (or community developer)
Response	A change to UI request is submitted to the system
Response measure	The change passes internal testing and user sees UI change < 30min

Part 3: QAS Documentation

Investigative approach outline:

- Initial research of GitHub files and online news articles to determine business goals and choose relevant QAS
    - i. frameworks research: Meteor, NodeJS, Mongo DB, Electron, Cordova
    - ii. tools research: Hubot, WebRTC, Docker
- Download and run Rocket.Chat
    - i. browse through code, focus on core components (relative size indicating importance)
    - ii. walk through tests
- Fork repository into personal Git repository to run code through various reverse code engineering softwares (such as codescene.io, softagram, and understand)
    - i. focus on modules pertaining to UI and testing
        - a. look at architectural structure
        - b. look at dependencies
        - c. determine average deployment on changes
- Illustrate relevant code files in a diagram
- Create QAS and growth scenario tables

Part 4: Created Diagrams

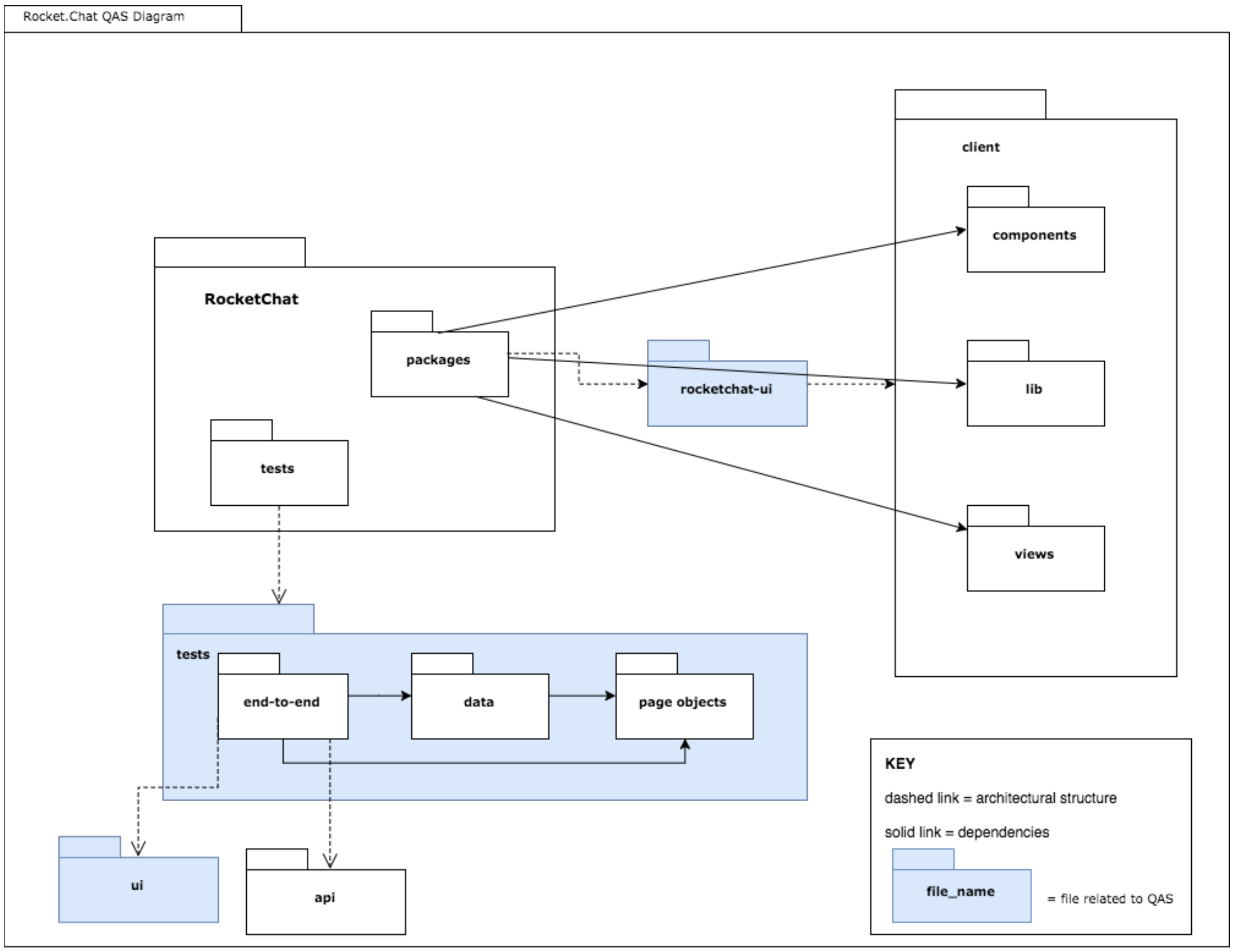

FIGURE 1 ROCKET.CHAT QAS DIAGRAM

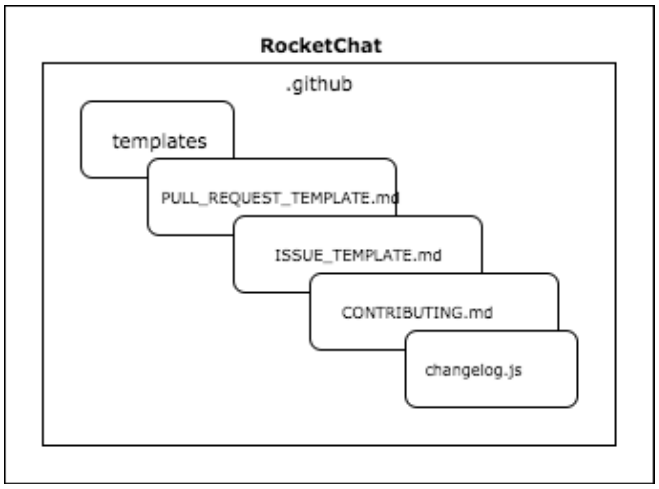

FIGURE 2 ROCKET.CHAT GITHUB TEMPLATE FILES

References

Engel, Gabriel. (2016, June 14). Rocket.Chat: Enabling Privately Hosted Chat Services [Web log post]. Retrieved Feb 8, 2018, from https://blog.blackducksoftware.com/rocket-chat-enabling-privately-hosted-chat-services