Lifecycle Management For Docker UI

Final Report (Semester 2)

Stephen Coady

20064122

Supervisor: Dr. Brenda Mullally

BSc (Hons) in Applied Computing

Plagiarism Declaration

Unless otherwise stated all the work contained in this report is my own. I also state that I have not submitted this work for any other course of study leading to an academic award.

Contents

1 Introduction **5**
1.1 Problem Statement . . . 5
1.2 Aims and Objectives . . . 6

2 Technologies **8**
2.1 Docker . . . 8
2.2 Node JS . . . 8
2.3 Angular JS . . . 9
2.4 Travis CI . . . 9
2.5 Vagrant . . . 10

3 Design **11**
3.1 System Architecture Overview . . . 11
3.2 Front End Design . . . 12
3.3 Enterprise Level Considerations . . . 14

4 Methodology **15**
4.1 Agile . . . 15
4.2 Jira . . . 15
4.3 Gitflow . . . 15
4.4 Testing . . . 15
4.5 Continuous Integration/Deployment . . . 15

Appendices **16**
A Application Repository . . . 16
B Dockerode . . . 16
C Travis Repository . . . 16
D SonarQube Repository . . . 16
E Sprint Retrospectives . . . 16
F Formal System Models . . . 20
G Wireframes . . . 20

Bibliography **28**

Glossary

API Application programming interface. A set of endpoints which lead to functions and application logic. Allows developers to expose application functionality without directly exposing the code within the application. 5

Bootstrap A front end css framework designed to give developers more power with less work when creating html pages. Makes use of a standard 12-width grid system to provide an easy-to-use css styling method. 13

CLI Command Line Interface. A means of interacting with a computer program where a user can only enter commands in the form of successive lines of text. 5, 6

Continuos Integration The act of continuously merging newly created software with the current working version, validated by automatic build and testing tools allowing for early detection of faults. 9

CSS Cascading Style Sheet. Used for applying styles to markup text primarily displayed on web pages. 13

Docker An application which manages containers running on a host. 5, 8, 14

Docker container An isolated package which bundles everything required to run an application within a container. The application can then be run within this container without regard to the underlying architecture. 8, 11, 12

Docker daemon The server-side component of the Docker Engine. 6

Docker host The computer, server or virtual machine on which the Docker application is installed. One user may manage several Docker host's, depending upon how many computers they have installed it on. 6, 8, 13

Docker image A template from which many containers can be started. Can be pushed an pulled to/from a remote Docker registry. 6, 8, 14

HTML Hypertext Markup Language. A means of tagging text using "markup" to allow them to be positioned, styled and linked on web pages. 13

JWT JSON Web Tokens. A means to provide authentication over the internet using a simple string of characters. 14

REST Representational state transfer. A means of providing communication between computers across the internet. RESTful web services allow systems interact with each other using completely stateless operations. 6

Travis A website which allows for automatic builds of code and feedback of results. Can be used to automatically deploy code dependent on test results also. 9

UI User Interface. A graphical view towards an application which exposes functionality by using buttons or other components. Typically utilised using a computer mouse. 5

Vagrantfile A template file which provides Vagrant with all the instructions required to create the desired environment. Written in the Ruby programming language. 10

Virtual Machine An abstracted virtual version of a computer. Normally runs on another host with a dedicated piece of software to manage it. Appears to the user as a real computer, however all or most components are defined by software. 8, 10

1 Introduction

This report will aim to guide the reader through the planning and development of the Lifecycle Management for Docker UI application. After reading this report the reader should have a clear idea of why the application was built, what was used to build it and how the process was carried out.

This application will be built using open source principles and best practices, enabling it to be maintained and improved by any developer who wishes to contribute. For this reason many of the decisions made and processes employed were done so with an open source final product in mind.

1.1 Problem Statement

Currently the Docker application does not ship with any bundled UI. When installed, it is comprised of a client and a server side component (Docker, 2017*a*). The server side exposes itself through an API and is ultimately responsible for controlling all aspects of Docker on the host such as containers, images, networks and volumes etc. The API exposed by the server-side application of the Docker Engine is consumed by the Docker CLI which is the client side application. A graphical representation of the complete Docker engine can be seen in Figure 1.

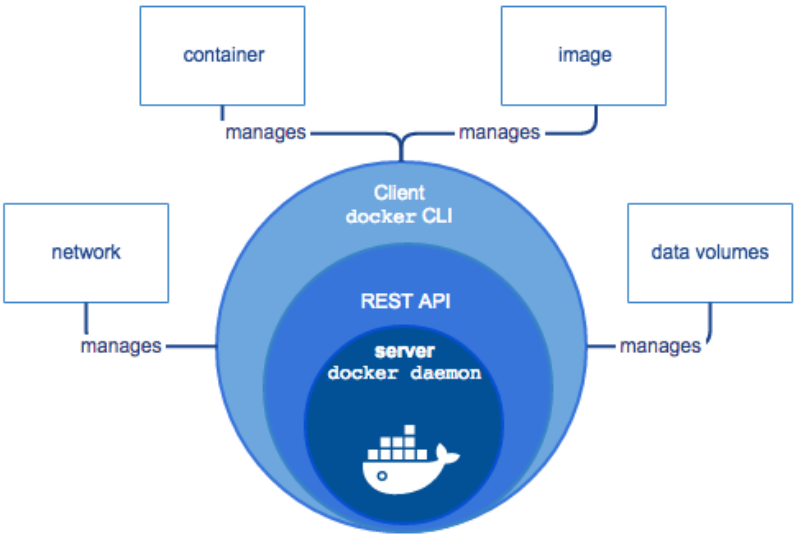

Figure 1: *Docker Engine Components. Credit: (Docker, 2017a)*

This model is extremely versatile as it allows the developer to control any Docker daemon (the server-side component of a Docker installation) once they have access to the command line of the host Docker is running on. In fact, if the API exposed by the Docker daemon is exposed remotely then the developer does not need access to the host's command line, instead they can directly access the API remotely.

While the CLI gives developers full control over the server-side component of a Docker host it also has its drawbacks.

- Learning curve - the person using the command line must be familiar with typical commands used to achieve certain tasks. This precludes anybody without these skills from using Docker.
- Vast set of Docker commands - there are a vast number of commands available to use from the Docker CLI. This is also a learning curve as even a developer who is familiar with a CLI must first learn the Docker commands to be able to use the client-side application.
- User friendliness - The command line does not produce content that is easily readable and can often format the data it is trying to present in an odd fashion depending on things like screen size etc.

1.2 Aims and Objectives

The aim of this project is to address all of these problems while also trying to increase the functionality available to anybody who wishes to use Docker.

At a high level the primary objectives of this project are:

- Fully functional server-side application
- Expose this application through a RESTful API.
- A fully functional front-end application
- A Docker image built to allow easy distribution of the application

These objectives will then provide the following functionality:

- A UI which will
  - allow users to manipulate images, containers etc on the host with the same capabilities as the command line
  - allow them to do this remotely
- A runnable container which has no other dependencies so that it can be run without installing the application

- An independent API which can be consumed by any front end application
  - This will provide flexibility if the front end framework needs to be changed further down the line

2 Technologies

In this section of the report the technologies used to create the application will be examined and explained in detail.

2.1 Docker

Docker is a platform which allows developers to package their applications into isolated containers which contain only the software dependencies required by the application. A container is different to a virtual machine in that a container does not contain a full operating system (Docker, 2016).

Since a primary objective of this application is to manage a Docker host it makes sense to leverage the capabilities of Docker and run this application within a Docker container. This provides several benefits over distributing source code, such as:

- Portability - If a Docker image can be built and uploaded to a public repository then it makes it easier for other developers to pull and run the application.
- Ease of use - As the application will be running in a container a developer does not need to install any third party components on their system to use the application. They do not need to worry about their environment at all, once their system has Docker installed it will run the application.
- Ephemeral - Docker containers are designed to be 'throw-away'. This means that if this application needs to be quickly stopped and restarted then containers are the perfect vehicle to do this.

2.2 Node JS

Node JS is a server-side JavaScript runtime, it is built on the same V8 engine that powers the popular Chrome browser. It uses an event-driven, non-blocking I/O model that makes it lightweight, efficient and very fast. Node JS' package ecosystem, NPM, is the largest ecosystem of open source libraries in the world. (Nodejs.org, 2016). Node JS provides an excellent way to build highly scalable network application which are non-blocking and extremely performant (Griffin et al., 2011). This means that if the application was adapted in the future to deal with large numbers of servers then the technology choice will be able to deal with that.

Node JS was also deemed a good fit for this project as it has a large and extremely active online community. Since this is an open source project this will increase the likelihood of other developers taking part in the project and contributing. We can see in Figure 2 that there are currently (as of

November 2016) more node modules available through the node package manager (NPM) than any other of the large package managers such as the ones used by Go, PHP, Python and Ruby.

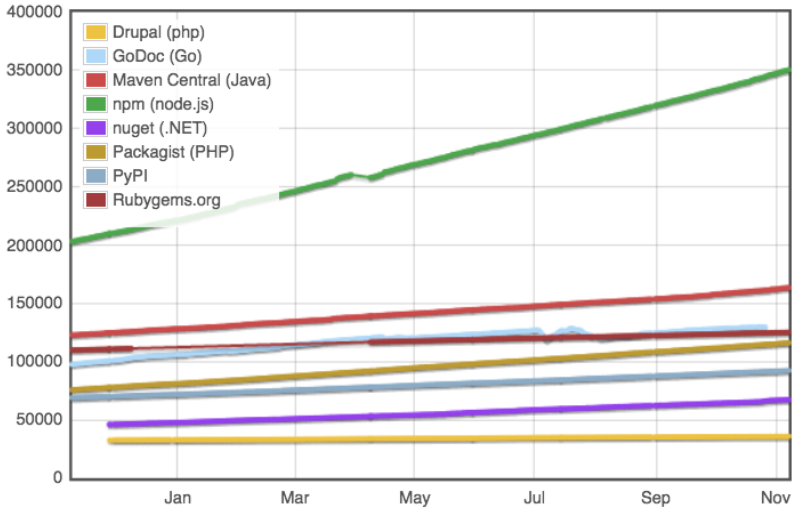

Figure 2: *Various Module Counts (Credit: modulecounts.com)*

2.3 Angular JS

Angular JS is a web framework for building dynamic web applications using Javascript as a controller. It uses HTML as its template language to display the information passed to it from the application controller (AngularJs, 2017).

For this application the front end requirements were relatively low. Once the framework provided a mechanism to show dynamic content then it was a candidate. For this reason several were evaluated and Angular was chosen as it had the lowest barrier to entry. The front-end application will be discussed further in Section 3.

2.4 Travis CI

Since this project was developed as an open source project it made sense to have a transparent and fully automated build process integrated into the project. For this reason continuos integration in this project is handled by Travis CI. It is a feature-rich service free to use for open source projects and has a large user base. The set up used in this project for testing and continuous integration will be explored further in Section 4.

2.5 Vagrant

Vagrant is a technology to create configurable, reproducible and portable environments by using a set of programmable steps to produce a virtual machine which the developer can use to develop in (Vagrant, 2017). While this is just one use-case of Vagrant it is the main reason it was used in this project.

Since this project is open source it is useful to have one standardised VM within which all development can take place. This virtual machine can then be shared (either as its own separate repository or included in the main application repository). This is useful as it enables any developer who wishes to contribute to the project to instantly have the required environment. For instance, a developer can contribute to this project by using the Vagrantfile supplied without requiring them to first install Docker or Node JS.

3 Design

The formal information modelling of the system remains unchanged from Report 1 and is available in Appendix F. However in this section we will examine the system overview and all aspects of design will be discussed.

3.1 System Architecture Overview

To gain a better understanding of the system we will now look at it from a high-level architectural view. This will give the reader an idea of how all major components fit together.

A diagram showing each major component of the application can be seen in Figure 3 below.

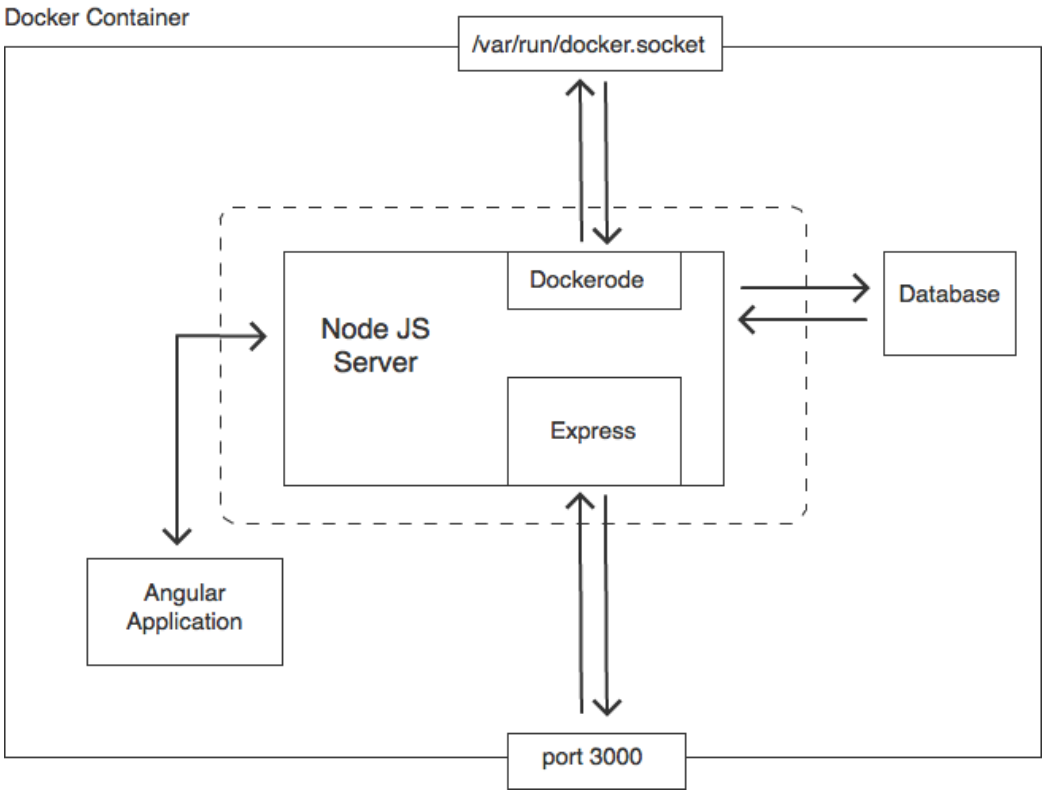

Figure 3: *System Architecture*

The whole application is bundled in a Docker container. This is represented in Figure 3 and means the architecture will never vary across different systems.

Most of the application's logic is in the Node JS server. It is the most important component as all

others are built around it. It provides a mechanism to access the application through an Express API while also communicating with the Docker API using Dockerode as an external library. It also performs all communication with the database and front end application.

It is important to note that the design of this application is deliberately modular. This means a low cohesion between each component of the application which in turn means that any one component can be swapped for any another which provides the same functionality. For instance, if in the future it was decided that a different front end framework was required then Angular could easily be replaced. Everything in Figure 3 outside of the dashed rectangle can be easily swapped for a different technology in this same fashion.

The Node JS application 'listens' on port 3000 within the Docker container which in turn maps to port 3000 on the host it is running on. It also maps the Docker unix socket '/var/run/docker.socket' as a volume on the Docker container.

3.2 Front End Design

As previously discussed the requirements for the front-end application are relatively low. They are:

- The application should allow for dynamic content.
- It should allow for re-use of code to keep the codebase small and manageable, i.e. 'templating'.
- As this project is open source it should be a relatively well known framework, meaning low barrier to entry for potential contributors.
- The application would have a clean, minimalistic interface using a sidebar to navigate all available pages.
- It would be responsive - meaning it would scale well on smaller devices such as mobile phones and tablets.

Therefore it was decided Angular was the best choice as it satisfied all of these conditions.

3.2.1 Wireframes

With these design decisions in mind an initial batch of wireframes were created which would be used when writing the front end code. One of these, the containers view, can be seen below in Figure 4 while the rest are available in Appendix G.

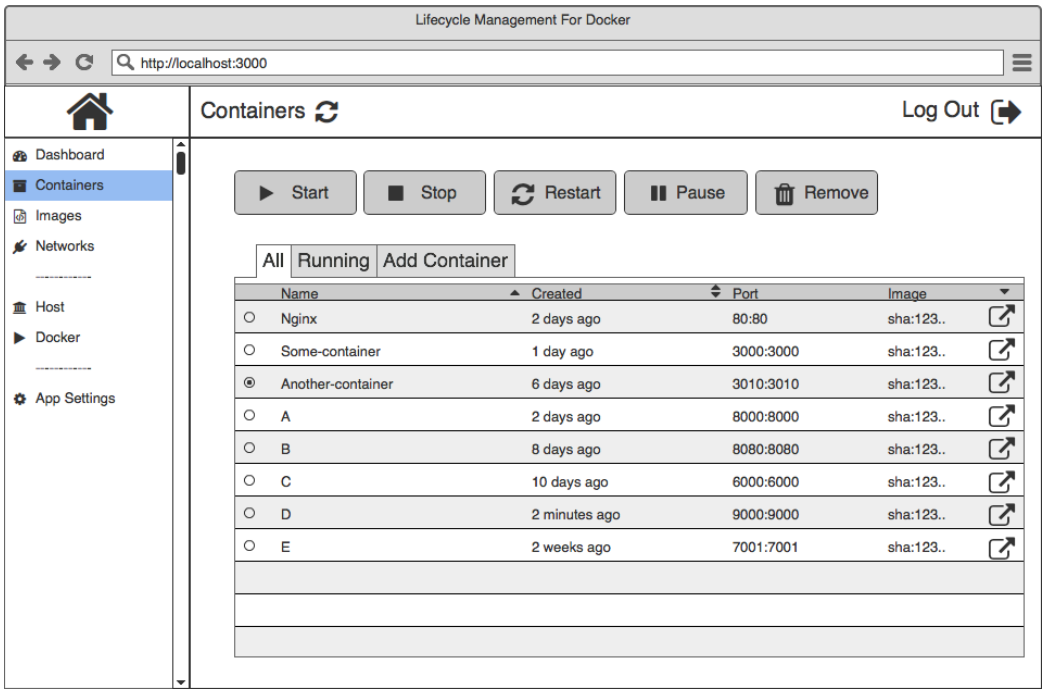

Figure 4: *Containers Mockup*

3.2.2 Mobile Web Application

Instead of creating a dedicated mobile application on a proprietary platform it was decided that a web application would be better. This is mainly because:

- Since the application would also need to be installed it would not make sense to require the user to install another application just to manage the first one.
- Not restricting users to certain devices was a major factor in deciding to create a web application as any user with a web browser can manage their Docker host.

Bootstrap was therefore decided as the best method of delivering a fully functional mobile web application. Bootstrap gives the developer full flexibility in terms of choosing the arrangement of content on different sized devices. For example, on a large device the developer can decide to have a 2 column 2 row grid of items. If the size of the screen then shrinks to mobile-size it can rescale to a 1 column 4 row grid. To do this simply means adding CSS classes to the HTML code and then Bootstrap will handle the layout.

3.3 Enterprise Level Considerations

When developing any application that will potentially be used in a production environment it is important to consider this in the design phase. There are several headings under which careful consideration must be given to ensure the application performs as it should while also being usable.

Scalability On a system which will potentially be used my many different users at the same time, how it scales and adapts to an increase in usage is important. To ensure that any user has complete flexibility in this regard this application is distributed using a Docker image, and therfore it can easily be used in a system which natively supports scaling such as Docker swarm or Kubernetes (Kubernetes, 2017; Docker, 2017*b*).

Performance A number of previously mentioned technology choices significantly increase the performance of the application.

- Node JS - Node JS has been shown to have significant performance benefits due to its asynchronous non-blocking event-driven model (Tilkov and Vinoski, 2010).
- Docker - Since this application will run in a container it is designed to be lightweight. It will only contain the third party modules it absolutely needs and it should minimise the risk of memory issues on the host it is running on.

Security To secure an application the Express endpoints will be secured using JSON web tokens (JWT). The front end application will authenticate itself with the Node JS server and will then use the returned JWT token to access all Express endpoints.

Distribution To make distribution a simple process it has already been described how Docker will be the vehicle of choice. It makes downloading and running the application extremely easy for any developer who wishes to do so. Once the developer has Docker installed on their system all they need to do is run one command to download and begin using the application.

4 Methodology

4.1 Agile

4.2 Jira

4.3 Gitflow

4.4 Testing

4.5 Continuous Integration/Deployment

Appendices

A Application Repository

https://github.com/StephenCoady/lifecycle-management-for-docker

B Dockerode

https://github.com/apocas/dockerode

C Travis Repository

https://travis-ci.org/StephenCoady/lifecycle-management-for-docker

D SonarQube Repository

https://sonarqube.com/dashboard/index?id=lifecyle-management-for-docker

E Sprint Retrospectives

2017-02-01 Docker Project Sprint 1 Retrospective

Date 01 Feb 2017
Participants Leigh Griffin Stephen Coady

Retrospective

What did we do well?

- Already had prototype in place to accelerate development
- 3rd party module knowledge accelerated development
- Guidance from Red Hat really helped focus the sprint
- Scope on tickets well understood from Red Hats perspective

What should we have done better?

- Story points were inflated because domain knowledge was higher than anticipated
- Testing strategy needs to be revised, very time consuming

Actions

- Stephen Coady review backlog for story point accuracy based off of current domain knowledge
- Stephen Coady add a ticket to review / spike testing strategies, feel free to consult David Martin and Leigh Griffin on specifics
- Stephen Coady add a ticket for UI frameworks investigations and spikes, end result should be an Epic that we can triage and prioritise

2017-02-15 Docker Sprint 2 Retrospective

Date 15 Feb 2017
Participants Leigh Griffin Stephen Coady David Martin

Retrospective

What did we do well?

- We performed a backlog review and set the priority for the remaining 3 sprints
- Progression of the sprint was excellent, good pace to it and good story pointing
- Comms was pretty good
- Smooth sprint, story points and tickets were well scoped
- Sprint Planning revisited the story points so very little surprises
- Team (Stephen) came to the Stakeholders (Dave & Leigh) with the plan for the next Sprint

What should we have done better?

- Sprint started at a bad time college wise, with other assignments due
- Not keeping in touch with the sprint day to day (Dave & Leigh)

Actions

- Stephen Coady to share wireframes as a mid sprint review asynchronously. Would recommend emailing that to us both.
- Stephen Coady metrics spike insight when you get to it (this sprint possibly)

2017-02-28 Docker Sprint 3Retrospective

Date 28 Feb 2017
Participants Leigh Griffin Stephen Coady

Retrospective

What did we do well?

- The UI is very usable, lots of nice feedback and functionality visible now
- Consistency in the velocity at 20 points
- Got the wireframe relationship, it really helped with my front end skills which I was not confident in
- Wireframe feedback was excellent, really helped scope the work
- Got to demo to my supervisor which gave her a lot of insight
- Overall work pace was judged well for the most part, consistent delivery

What should we have done better?

- The story pointing on the skeleton was completely off, it could have derailed the entire sprint
- Velocity last sprint was off
    - you should have descoped when you realised how big the UI was
    - you should have re story pointed the UI mid sprint to allow a controlled descope
- Tickets are not descriptive enough, need to add more metadata
- Didn't descope the testing in a container ticket, should have done that when the UI became so big

Actions

- Stephen Coady to review the backlog with a view to WHAT and WHY being evolved in the tickets as well as story points
- Stephen Coady to define the critical path through the project, ~80 story points left with a ~60 story point burn predicted
- Stephen Coady to add some investigative tasks around KeyCloak SSO for future work i.e. out of scope of this project

F Formal System Models

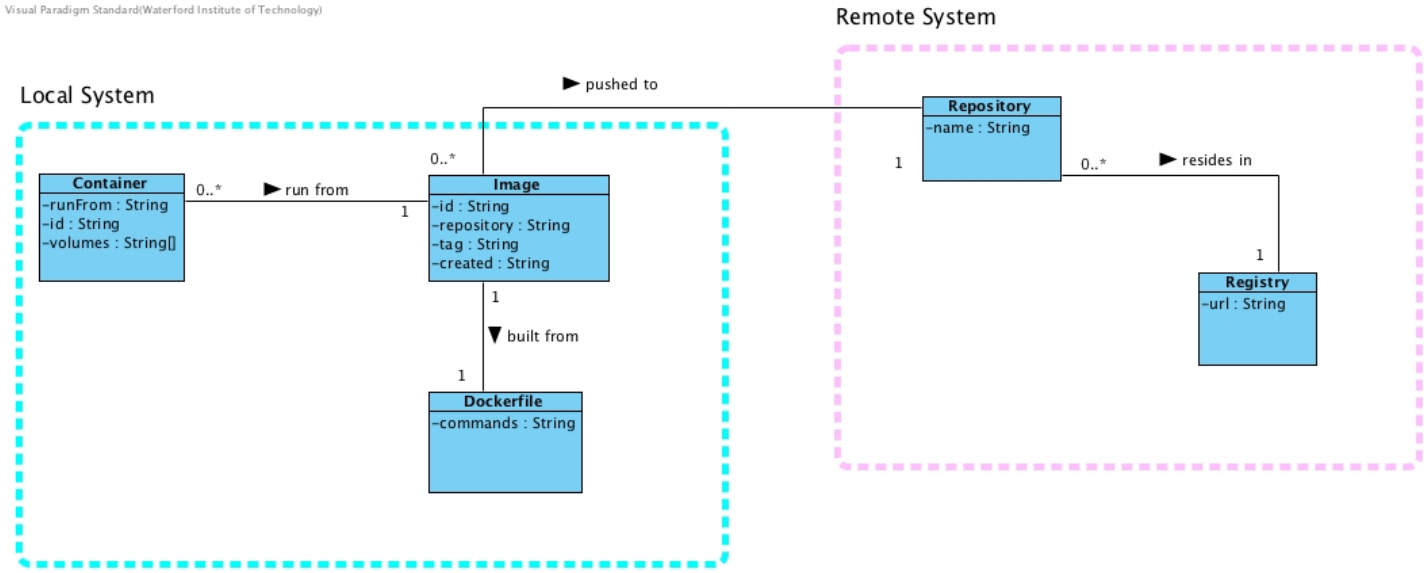

Figure 5: *Class Diagram*

G Wireframes

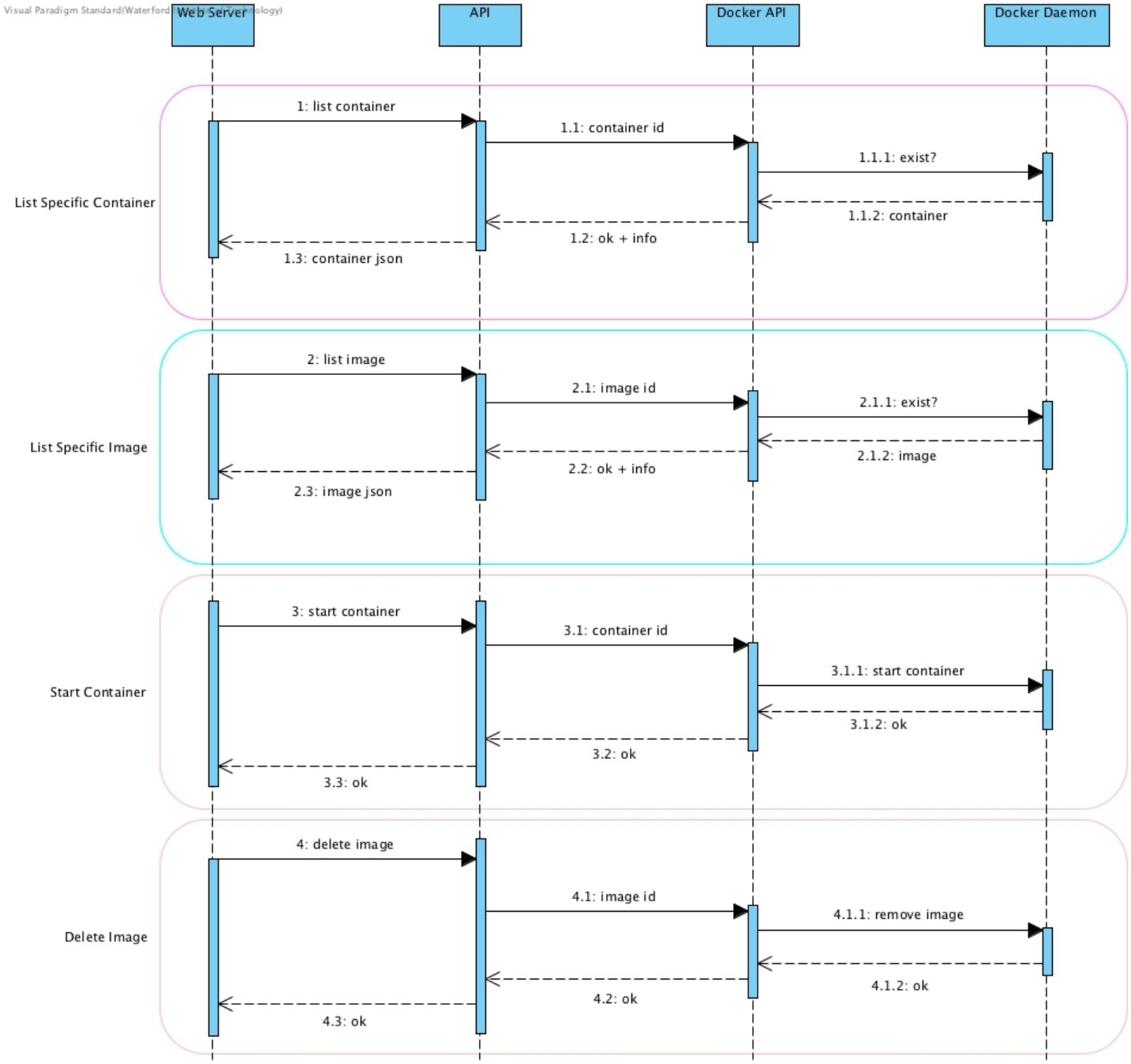

Figure 6: *Sequence Diagram*

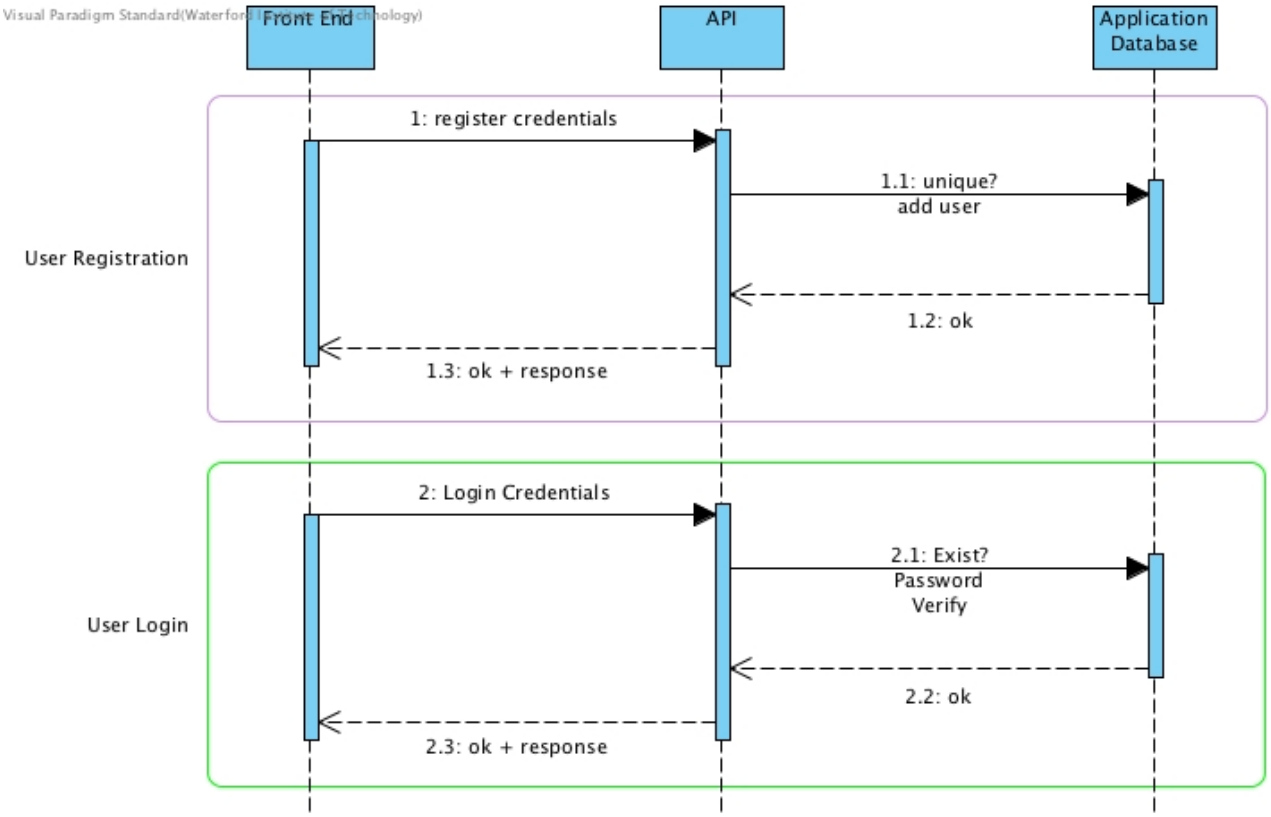

Figure 7: *User System Sequence Diagram*

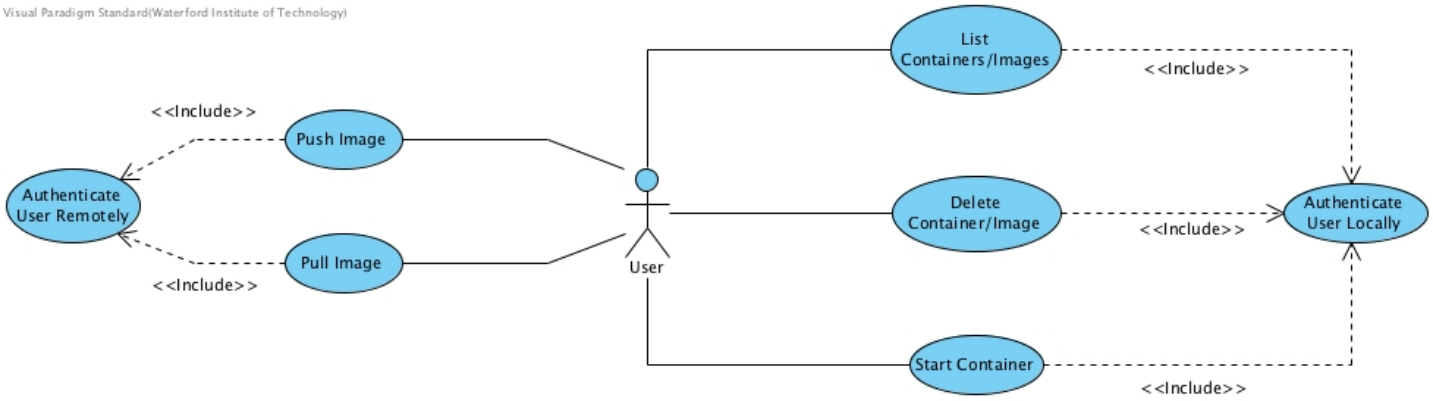

Figure 8: *Use Case Diagram*

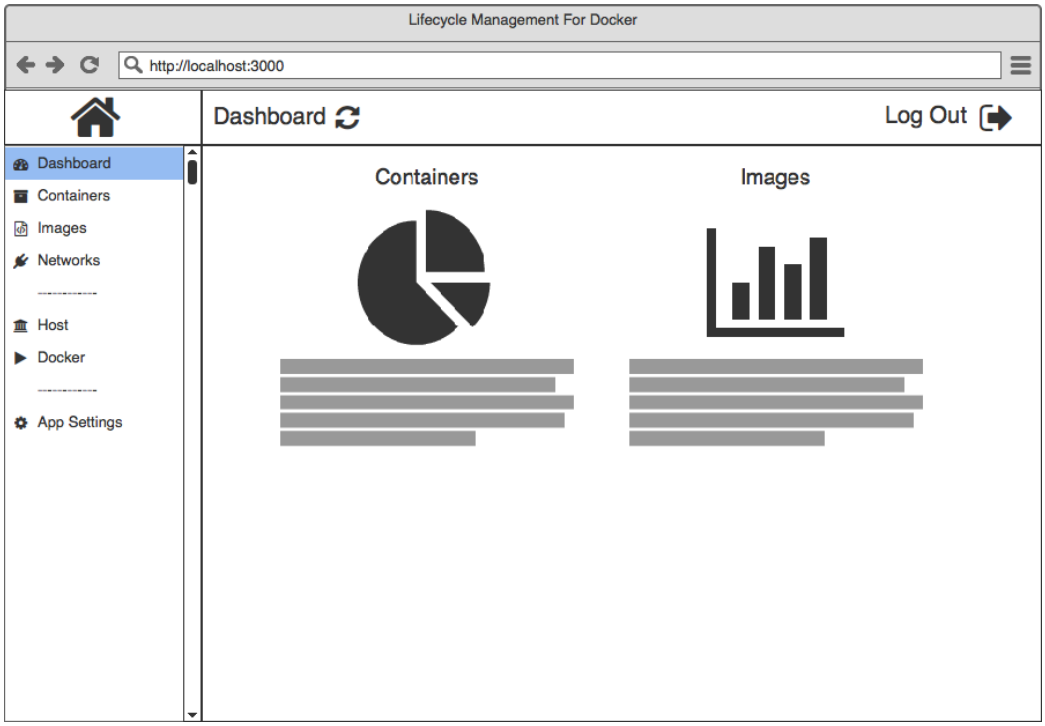

Figure 9: *Dashboard Mockup*

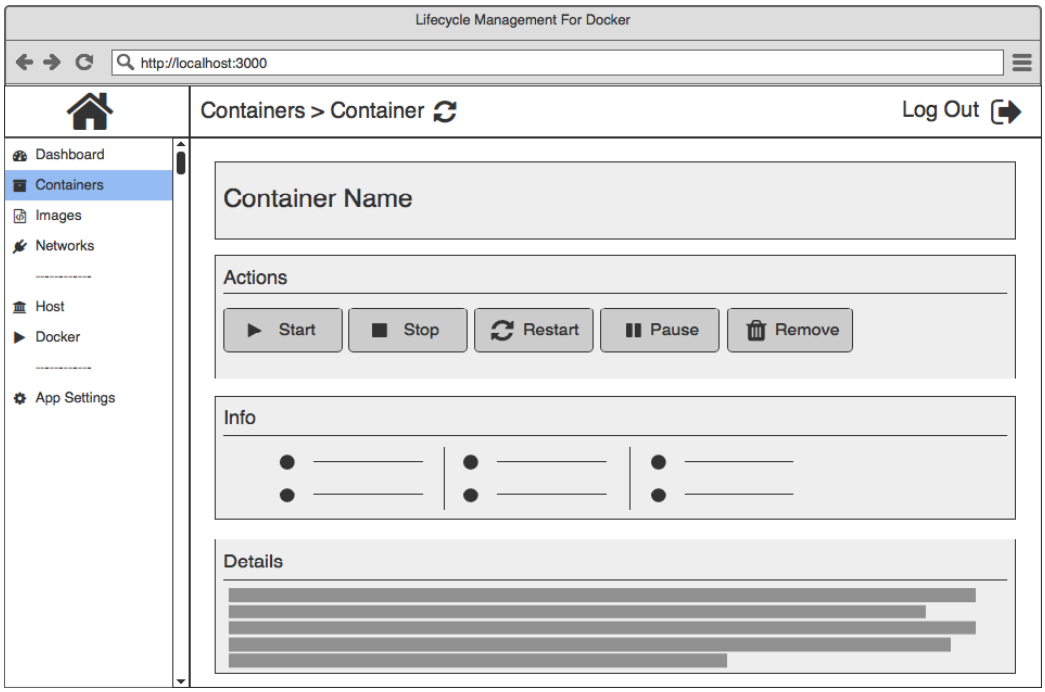

Figure 10: *Container Mockup*

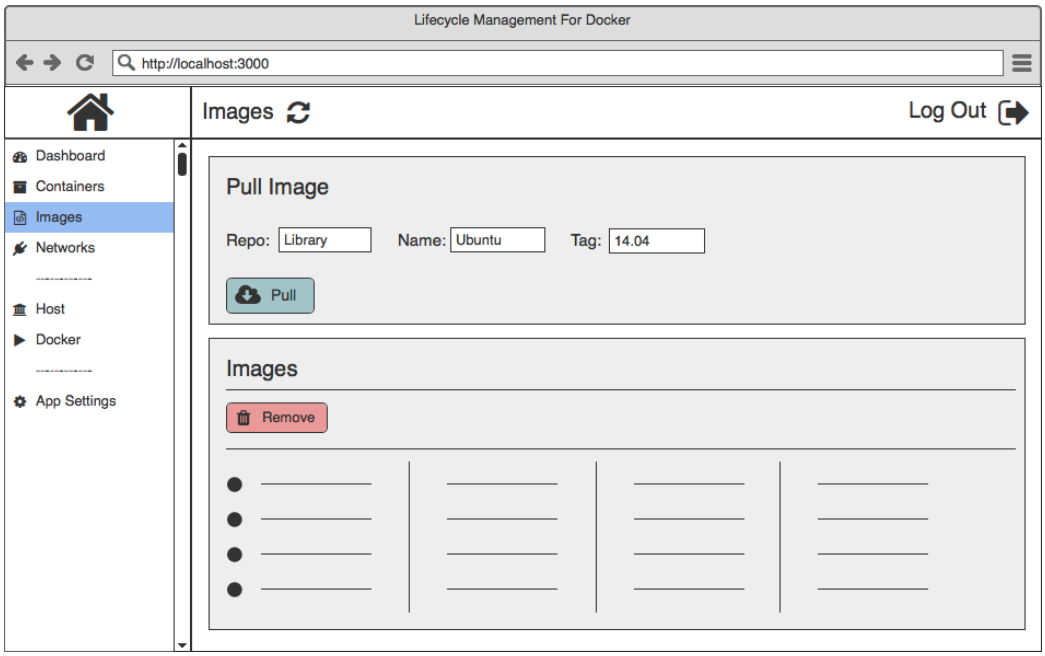

Figure 11: *Images Mockup*

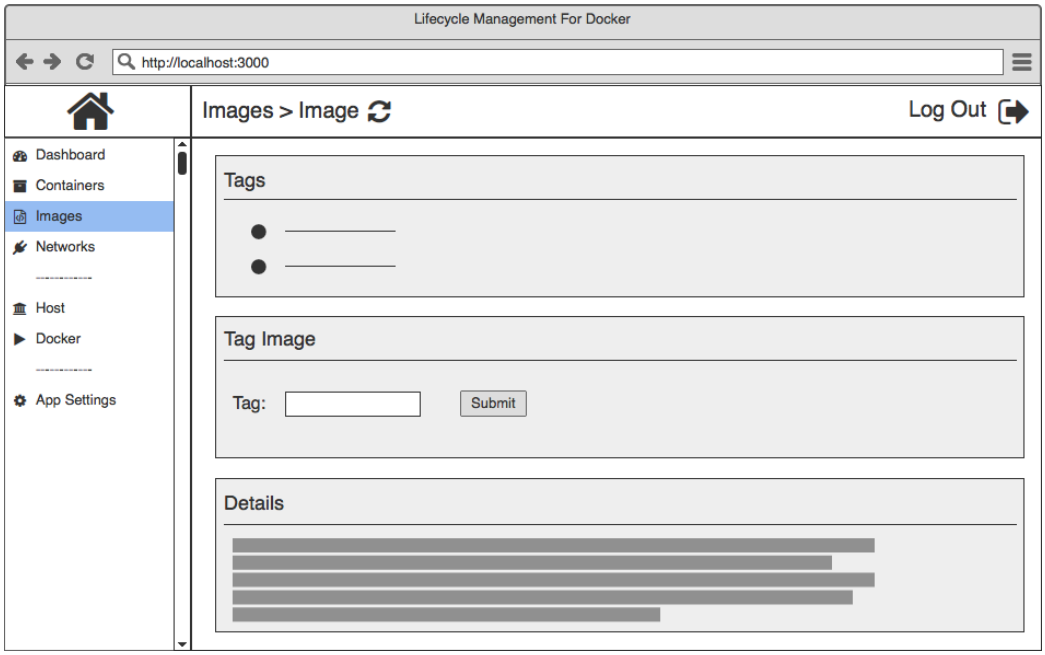

Figure 12: *Image Mockup*

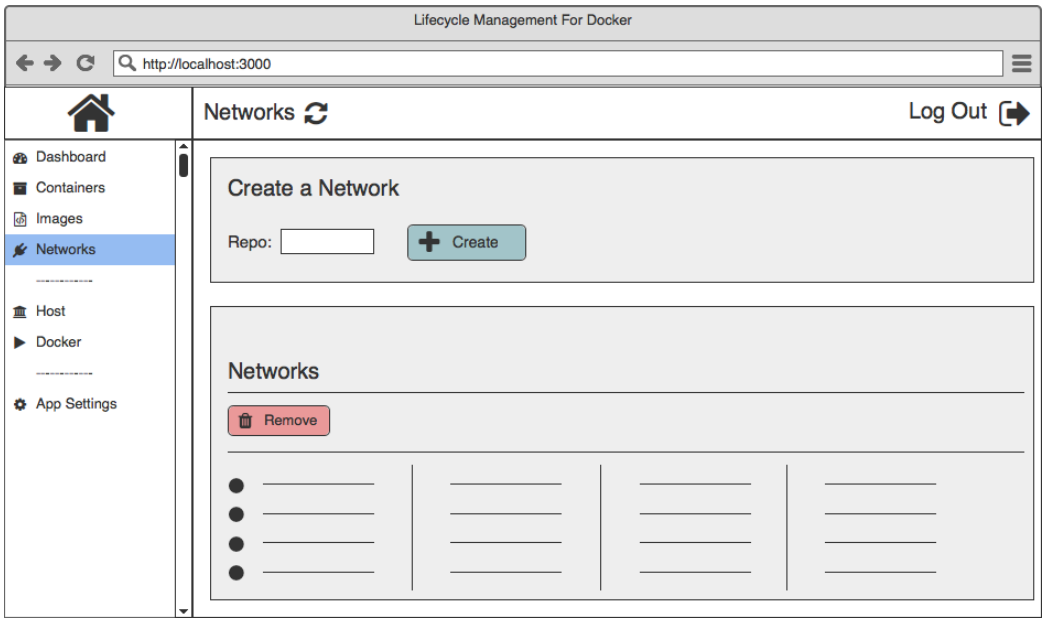

Figure 13: *Networks Mockup*

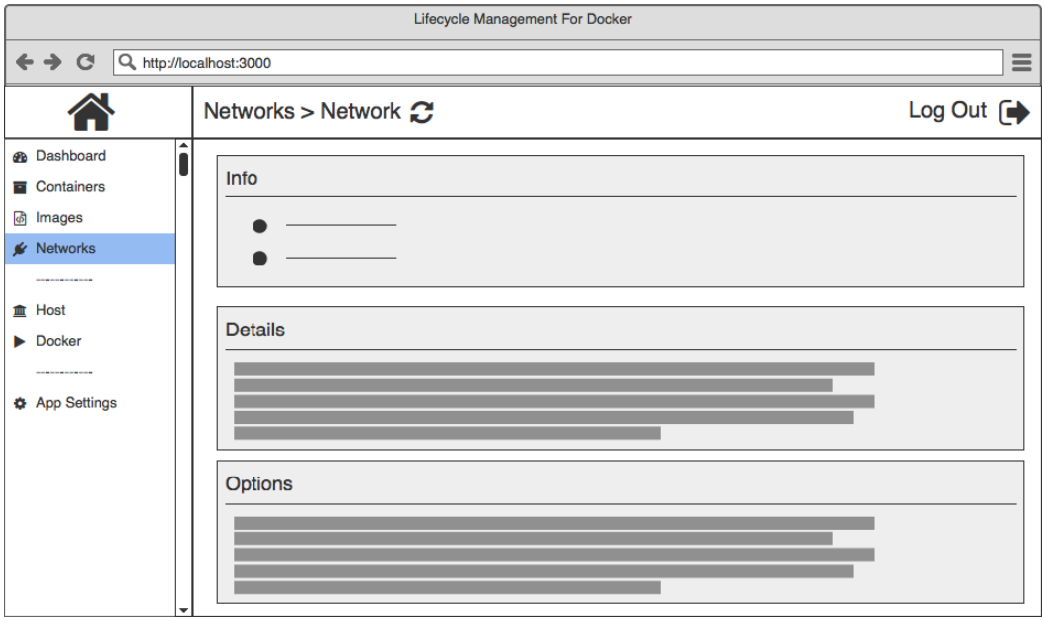

Figure 14: *Network Mockup*

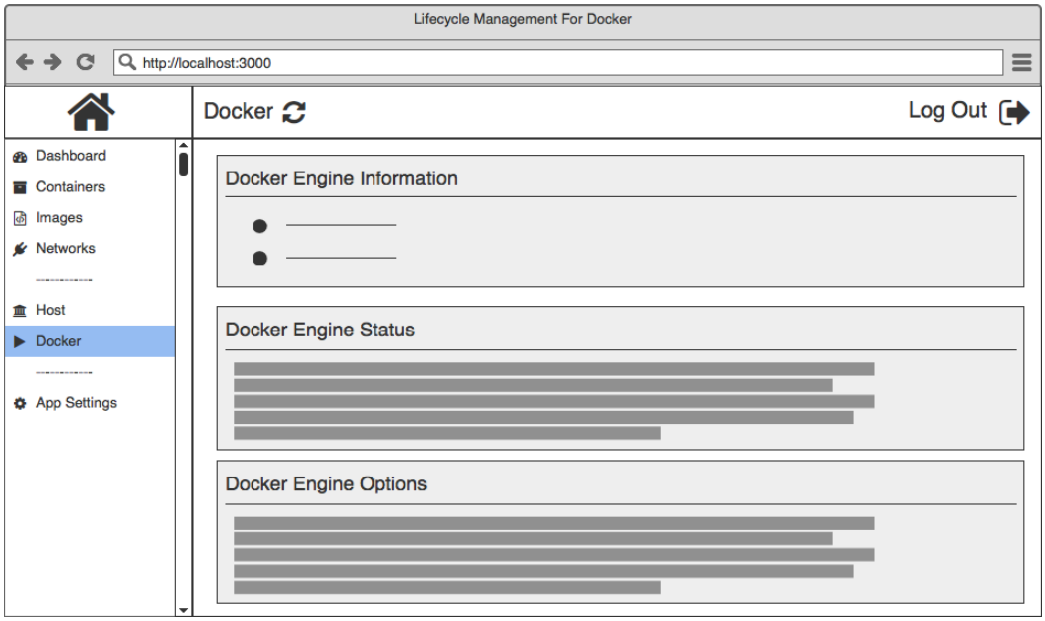

Figure 15: *Docker Mockup*

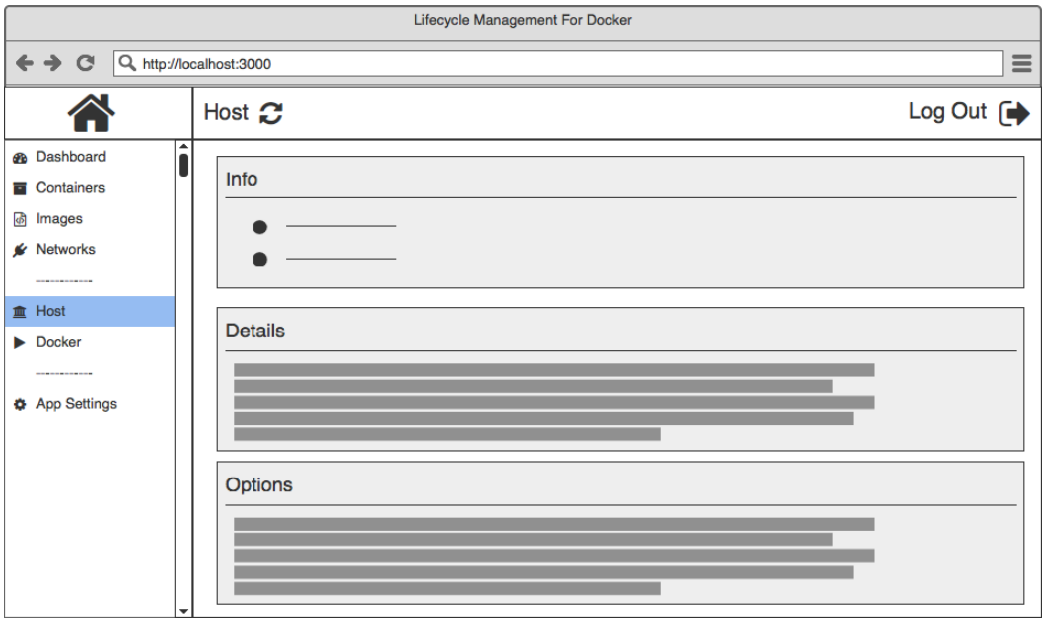

Figure 16: *Host Mockup*

Bibliography

AngularJs (2017), 'What is angularjs?'. [online] Accessed: 10-03-2017.
URL: *https://docs.angularjs.org/guide/introduction*

Docker (2016), 'What is Docker'. [online] Accessed: 27/11/2016.
URL: *https://www.docker.com/what-docker*

Docker (2017*a*), 'Docker overview'. [online Accessed: 08/03/2017].
URL: *https://docs.docker.com/engine/understanding-docker/*

Docker (2017*b*), 'Swarm mode key concepts'. [online] Accessed: 19-03-2017.
URL: *https://docs.docker.com/engine/swarm/key-concepts/*

Griffin, L., Ryan, K., de Leastar, E. and Botvich, D. (2011), 'Scaling instant messaging communication services: A comparison of blocking and non-blocking techniques'. [online] Accessed: 10-03-2017.
URL: *http://repository.wit.ie/1636/*

Kubernetes (2017), 'What is kubernetes?'. [online] Accessed: 19-03-2017.
URL: *https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/*

Nodejs.org (2016), 'Node.js'. [online] Accessed: 27/11/2016.
URL: *https://nodejs.org/en/*

Tilkov, S. and Vinoski, S. (2010), 'Node.js: Using javascript to build high-performance network programs', *IEEE Internet Computing* **14**(6), 80–83.

Vagrant (2017), 'Why vagrant?'. [online] Accessed: 10-03-2017.
URL: *https://www.vagrantup.com/docs/why-vagrant/*