March 2017

WorkHero

Ryan J. Leaf
Worcester Polytechnic Institute

Tim Pascal Marschall
Worcester Polytechnic Institute

Follow this and additional works at: https://digitalcommons.wpi.edu/mqp-all

Repository Citation

This Unrestricted is brought to you for free and open access by the Major Qualifying Projects at Digital WPI. It has been accepted for inclusion in Major Qualifying Projects (All Years) by an authorized administrator of Digital WPI. For more information, please contact digitalwpi@wpi.edu.
WorkHero

A Major Qualifying Project Report:

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Bachelor of Science

By

_______________________________

Ryan Leaf

_______________________________

Tim Marschall

Date: March 1st, 2017

_______________________________

Professor Soussan Djamasbi
Project Advisor
Abstract

WorkHero is a web application designed to help students and professionals connect with tools and resources that help them to achieve their goals. Users of WorkHero can share and discover how to combine tools to accomplish their endeavors in an efficient and effective manner. This paper explores our process of iteratively analyzing market and user needs; designing rapid prototypes of prospective solutions and testing them; architecting our application; planning and managing the entire Software Development Lifecycle; developing WorkHero using modern web technologies; testing our developed solution; and revising our product roadmap to better meet the needs of our target markets.
Acknowledgements

We would like to extend our gratitude to the many people who have helped enable this project to be a reality. Without their generous help this project would not have been possible.

First and foremost, we would like to express our sincere appreciation for our project advisor, Professor Soussan Djamasbi. Without her support, encouragement, and drive, our project would not be nearly as successful as it is. We thank you for all of the work you have put into helping us develop this project and all of the time that you have invested in getting the project off the ground.

We would also like to recognize the work of the WorkMyWay project team who helped us in the original ideation phase of this project. Their insights into the need for a student-focused resource repository proved invaluable in helping us get started with the ideation phase for WorkHero.

Michael Hartl, you may not know of your role in this project, but your *Ruby on Rails Tutorial* laid the foundation of our entire backend. Ryan, in particular, wants to thank you for bringing him from zero knowledge in web development to, as you’d say, knowing “enough to be dangerous.” Your clear, concise, and accessible book on Rails development made getting started with web development enjoyable.
Executive Summary

The WorkHero project started with a desire to help students achieve their goals by connecting them with strategies and approaches for using available tools and resources. Given the number of tools on the market that already solve many business and social needs, a major challenge with user adoption is awareness of existing solutions and education about their usage. WorkHero aims to provide students and professionals with a platform for finding new tools, discovering how to use them in tandem with other resources, and sharing expertise on how to accomplish goals with existing resources.

Our team began building WorkHero by transforming the initial idea into an actionable design plan. We used the principles of System Analysis and Design, as well as user interface design to create rapid non-functional prototypes and interface mockups. After revising our initial designs based on conversations with prospective users and project stakeholders, we constructed a product roadmap outlining several future releases of the product. We also constructed a project plan based on the roadmap to help guide progress on WorkHero.

The process then pivoted towards product development. At the onset of development, we went through a process of deciding on appropriate underlying technologies to help us achieve our business and technical goals. We spent some time in this phase learning new technologies and experimenting with our development approach. Next, we began the process of iteratively developing the product, based on 2-week Scrum cycles.

In November, a few months after starting development, we publicly launched WorkHero version 1.0. We performed the launch by performing basic code review, testing the production state of our application, configuring the production server to run our application, and finally getting our
project into a working state for users to test. After launching, we demonstrated our project to a few student groups and collected their anonymous feedback. We used the feedback gained in this stage to revise our product roadmap in order to be better aligned to the needs of our end users.

With our new roadmap, we returned to the development process and continued to iteratively fix existing issues, make architectural improvements, address our technical debt and implement new functionality. We continued to engage with students around campus, as well as external contacts to test our new product iterations and improve WorkHero. The development of this project concluded with a final release of version 1.1.2 in the middle of February.
# Table of Contents

**ABSTRACT** 2

**ACKNOWLEDGEMENTS** 3

**EXECUTIVE SUMMARY** 4

**TABLE OF CONTENTS** 6

**AUTHORSHIP** 9

**CHAPTER 1: INTRODUCTION** 11

Glossary 12

**CHAPTER 2: TECHNICAL FEASIBILITY & DESIGN** 13

Team Considerations 13

Design Considerations 13

**Technology Considerations** 15

Frontend 15

Backend 17

- Deciding on a Datastore 17
- Choosing the Backend Framework 18
- Search 19
- Analytics Provider 20

**CHAPTER 3: BUSINESS FEASIBILITY & JUSTIFICATION** 21

Monetization Options 22

**CHAPTER 4: DEVELOPMENT** 23

Development Management 23

Version Control 23

Team Communication 26
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development Software</td>
<td>27</td>
</tr>
<tr>
<td>Containerized Development Environment</td>
<td>27</td>
</tr>
<tr>
<td>Vagrant</td>
<td>28</td>
</tr>
<tr>
<td>KVM-based Linux Production Server</td>
<td>29</td>
</tr>
<tr>
<td>Scaffolding</td>
<td>29</td>
</tr>
<tr>
<td>CHAPTER 5: DEPLOYMENT</td>
<td>31</td>
</tr>
<tr>
<td>Versioning</td>
<td>31</td>
</tr>
<tr>
<td>Deployment Process</td>
<td>32</td>
</tr>
<tr>
<td>SSL Certificates</td>
<td>33</td>
</tr>
<tr>
<td>CHAPTER 6: PRODUCT LAUNCH</td>
<td>34</td>
</tr>
<tr>
<td>CHAPTER 7: RECOMMENDATIONS</td>
<td>36</td>
</tr>
<tr>
<td>Development</td>
<td>36</td>
</tr>
<tr>
<td>Search</td>
<td>37</td>
</tr>
<tr>
<td>Authentication</td>
<td>38</td>
</tr>
<tr>
<td>Testing</td>
<td>40</td>
</tr>
<tr>
<td>Development Database and Environment</td>
<td>41</td>
</tr>
<tr>
<td>Guidelines for Creating a Development Database</td>
<td>41</td>
</tr>
<tr>
<td>Migrating to External Server Hosting</td>
<td>42</td>
</tr>
<tr>
<td>Hosting Requirements</td>
<td>43</td>
</tr>
<tr>
<td>Migrating the Database</td>
<td>44</td>
</tr>
<tr>
<td>Scaling WorkHero</td>
<td>45</td>
</tr>
<tr>
<td>A Further Discussion of Monetization</td>
<td>47</td>
</tr>
<tr>
<td>CHAPTER 8: CONCLUSION</td>
<td>48</td>
</tr>
<tr>
<td>Multi-Disciplinary Project Outcomes</td>
<td>48</td>
</tr>
<tr>
<td>Final Words</td>
<td>49</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td>51</td>
</tr>
<tr>
<td>APPENDIX A - LEAN CANVAS (FEBRUARY 2017)</td>
<td>52</td>
</tr>
<tr>
<td>APPENDIX B - FRAMEWORK MATRIX (FEBRUARY 2017)</td>
<td>53</td>
</tr>
</tbody>
</table>
APPENDIX C - MQP DEMO

APPENDIX D - ENTITY RELATIONSHIP DIAGRAM FOR WORKHERO
Table of Figures

FIGURE 1 - WORKHERO LANDING PAGE (FEB 2017) ........................................ 15
FIGURE 2 - INITIAL GIT STRATEGY .................................................. 25
FIGURE 3 - REVISED GIT STRATEGY .............................................. 26
FIGURE 4 - SAMPLE HA & SHARDING SCHEME ............................... 47
FIGURE 5 - LEAN CANVAS .............................................................. 52
FIGURE 6 - ENTITY RELATIONSHIP DIAGRAM ................................. 60
Authorship

- Abstract – Leaf
- Acknowledgements – Leaf
- Executive Summary – Leaf
- Introduction – Leaf, Marschall
- Technical Feasibility – Leaf, Marschall
  - Team Considerations – Marschall
  - Design Considerations – Marschall
  - Technology Considerations
    - Frontend – Marschall
    - Backend – Leaf
- Business Feasibility & Justification – Leaf, Marschall
- Development – Leaf, Marschall
  - Development Management – Marschall
  - Version Control – Marschall
  - Team Communication – Leaf, Marschall
  - Development Software – Leaf
  - Scaffolding – Marschall
- Deployment – Leaf, Marschall
  - Versioning – Leaf, Marschall
  - Deployment Process – Leaf
- Product Launch – Leaf, Marschall
- Recommendations – Leaf, Marschall
  - Development
    - Search – Leaf
    - Authentication – Leaf
    - Testing – Leaf, Marschall
  - Development Database and Environment – Leaf
  - Migrating to External Server Hosting – Leaf
  - Scaling WorkHero – Leaf
- Conclusion – Leaf, Marschall
- Appendix A – Lean Canvas – Leaf
- Appendix B – Framework Matrix – Marschall
- Appendix C – MQP Demo – Leaf, Marschall
- Appendix D – Entity Relationship Diagram for WorkHero – Leaf
Chapter 1: Introduction

Motivated people all have one thing in common: they are always searching for the next goal to achieve. Whether these goals are personal or professional in nature, the individuals accomplishing them often are not the first to do so. Even in the case where a goal is less common, it is not unusual for another’s solution to be at least tangentially related to the objective at hand.

Despite the likelihood that one motivated person’s goals are related to another’s, there currently is no viable central repository for sharing and discovering information related to completing goals. At the same time, many fantastic tools exist that solve many real world needs. The challenge of adopting these tools is often in finding the right tool(s) given the number of tools available. WorkHero is designed to bridge the gap between the tools available and the ways to use them in conjunction with each other for people who are looking to achieve their goals.

One intended outcome is to reduce the amount of time and effort different people spend on solving the same problems. By providing users with a go-to source for discovering how to start and succeed with their objectives, we hope to enable people to spend more time achieving their goals and less time encountering roadblocks.

In educational settings, WorkHero can help students improve their efficiency and effectiveness by helping them learn from their peers experiences in solving similar problems. Similarly, in corporate settings, WorkHero is applicable to the need of retaining solutions to previously solved problems, especially as teams change over time. The degree of effort that teams and companies spend on rediscovering solutions to previously solved problems adds cost, complexity and needless waste to the process of producing value.
## Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity</td>
<td>The methodology for using a tool or series of tools to accomplish a goal. Activities can include, but are not limited to, “kayak for the first time”, “learning to play the guitar”, or “building a web application.”</td>
</tr>
<tr>
<td>Tool</td>
<td>Software applications, processes, approaches, books, resources, or other atomic entities that can facilitate an activity. Specificity is only required where it is relevant - sometimes a specific make and model of a tool is required, while other times any similar tool can be used for the task. For example, tools include, but are not limited to: “#2 Phillips screwdriver”, “Google Docs”, “guided meditation”, “Kanban”, and “email.”</td>
</tr>
<tr>
<td>Tag</td>
<td>User-generated descriptors that are used to classify and organize tools and activities on the platform. They typically are one or two word phrases that help to describe a tool or activity. Tags can also be acronyms that are commonly used. Tags include, but are not limited to, “collaboration”, “first year experience”, and “NFL”.</td>
</tr>
</tbody>
</table>
Chapter 2: Technical Feasibility & Design

Achieving the business and people goals of WorkHero requires a platform that enables users to interact with our platform, share content with minimal friction, and access existing content. These goals also require technical solutions that enable the functional and non-functional requirements dictated by these goals. This process requires effective design and development in a skilled team.

Team Considerations

In order to build a successful online platform, it is crucial to have a team with understanding of two significant components: technical development and business. The former requires understanding of web development, user interface design, security and development operations (DevOps). Competency in business strategy, marketing and entrepreneurship are requisite for the latter.

Team dynamics and compatibility is another significant factor to consider. In this case the team consisted of only two people, which poses atypical challenges for the significant task of developing a full web application. Due to the small team size, timely delivery of development deliverables, fast response time and support of team members are essential to successfully realizing the potential of this type of project.

Design Considerations

One of the underlying goals of our design is to have a platform that is easily understood by new users by offering a clean and well-implemented user experience. Preliminary designs followed well-established design guidelines and principles and were refined with the goal of simplicity in
mind. In order to adequately position the platform, inspiration was drawn from well-respected platforms on the web, such as Twitter and Facebook. Inspiration also stemmed from the design guidelines laid out in Google’s Material Design (Google, n.d.).

While mobile usage of the web is expected to grow (Ericsson, 2016; Cisco, 2017), WorkHero was designed as “mobile supported” rather than “mobile first”, meaning that the site is accessible from mobile devices, but not the user interface’s focus. Mobile support is enabled using Twitter’s Bootstrap Cascading Style Sheets (CSS) framework, which implements responsive design principles that allow websites built on the framework to be viewed on all size screens.

From a marketing perspective, it is important to be consistent with brand image and messaging throughout products, social media and other official channels. A series of short brainstorming sessions led to the decision to name the project “WorkHero”. This name made the development of a narrative around users helping other users and thereby becoming “WorkHeroes” possible. This story-driven structure is presented throughout the application.

Brand consistency was established through a reusable library of widgets and theme palette within WorkHero’s codebase. The color choices were driven by the name and two of the typical
colors associated with heroes: red and blue.

Technology Considerations

Frontend

In order to choose the frontend framework for the application, a decision matrix laying out several of the most popular options was developed. It allowed for the exploration of the feasibility of researched choices in four dimensions: flexibility, simplicity, familiarity, and support.

Flexibility is crucial considering the scale and aim of the project. Being able to pivot and change the application quickly based on the limitations software typically faces, such as financial, hardware and time constraints is essential to any development team focusing on user driven design.
WorkHero, as became apparent through the early stages of planning, could become a large and complex piece of software. In order to make future project maintainability and development possible, emphasis was placed on the simplicity of the framework. Simplicity was defined as having both code readability as well as the ease of learning the framework by new project members.

Considering the development team behind WorkHero was small and the project scope relatively large, a certain level of familiarity with the used technologies was required to allow an initial release. A poorly selected frontend stack has the potential to slow development enough to force the delivery of an incomplete product. The ability to deploy, run user tests and to push WorkHero in a potential marketable direction were major goals of the project. Rather than narrowing the scope, importance was placed on the team’s ability to achieve these goals by using suitable and familiar frontend technologies. The last dimension considered was support, defined as the amount of community engagement and usage of a framework. In order to quantify this dimension, usage across the Internet was compared using Libscore and the number of forks on GitHub. All four dimensions were rated on a scale from low to high. The matrix can be found in Appendix B.

The conclusion of the framework matrix led to the choice of Google’s AngularJS 1.x. Angular has very high community support, has vast tutorials available, was familiar to the development team and is flexible. Furthermore, it is simple to use Angular with various stacks and offers many built-in capabilities designed to speed up development. Due to the backwards compatibility of Angular 2, it is possible to move towards this more modern successor of the framework when the community support for it has grown.
Backend

Similar to the frontend, a decision matrix was used to decide on backend technologies. Since the backend is a combination of a datastore and a logic server, backend technologies had to integrate with the frontend as well as with each other.

The most basic requirement when deciding on a backend is the ability to easily create a fully function JSON-encoded REST API, since the WorkHero frontend relies on API calls to retrieve data, post new content and modify existing content. One of the advantages of using a RESTful API is the simplicity in application design; all actions within the application are based on the four CRUD functions - create, retrieve, update and delete. This not only makes it simple to integrate the frontend and backend frameworks, it makes understanding the behavior of the application far simpler, since every action in the app must fit within one of these four operations.

Deciding on a Datastore

Prior to deciding on a backend framework, the team evaluated several datastore options. Based on the developed interface the fields and datatypes required were sketched out. The initial design was developed by first planning out the fields and data expected to be required by the application, based on our interface mockups. The exposure of most data through the web interface simplified the process. The team opted to not place these fields into models in the initial pass as to not prematurely define data as relational.

Once all fields were defined, the modelling process began. By sketching a preliminary UML the team began to search for relationships between data. The UML diagram and UI mockups allowed for the quick identification of several many-to-many relationships between data. While comments and authorship could be modeled just as easily by a document-oriented model (and
using a document-oriented datastore, like MongoDB), user votes, ratings, tags, and activity-to-tool pairings proved to be better represented by a relational model.

Performance was a consideration with this choice of datastore as well. For example, in order to quickly query the database for activities that a specific tool is used for, the relationship is established when a user creates a new activity and assigns tools to that activity (through the toolings model). It is faster to query the toolings table for either the ID of an activity (and find all tools that belong to that activity) or the ID of a tool (and find all of the activities that belong to that tool) than to search through n-documents for matches.

Another aspect of the design is the existence of distinct entities, such as activities and tools. While it is possible to create a type field in each document to represent the different entities, the more explicit option of creating separate tables for each entity type made the application simpler, easier to work with, and more performant for the types of queries it needs to handle.

Finally, we considered the business environment the application would be built in. Since WPI already offers the infrastructure for MySQL, opting for a datastore that was already well-supported by the institution in lieu of an unsupported datastore is a logical decision.

Choosing the Backend Framework

After deciding on the datastore for the application the backend was chosen. Using the same matrix used for our frontend and adding backend options allowed for an evaluation based on the criteria in Appendix B.

Upon careful consideration, Ruby on Rails came out as the best option. One of the major advantages of Rails is ActiveRecord, a built-in component of Rails that makes working with
relational databases in a pragmatic and well-structured way simple. It eliminates the need to write SQL queries to perform almost all tasks, while offering the ability to construct manual SQL queries (*Active Record Basics*, n.d.). Furthermore, ActiveRecord is well supported by most Rubygems, allowing more functionality to be added without needing to write custom gems to support non-relational databases or use a less well-documented path for getting gems that support NoSQL databases to work properly.

For example, while the popular authentication gem Devise does support MongoDB (Kehoe, 2012) as a datastore through Mongoid, the support for ActiveRecord is far better documented and popular. As a result, the opportunity to consult a variety of tutorials, documentation sources, and answers on StackOverflow for help with getting Devise to work is greater with ActiveRecord than with Mongoid. Esoteric gems are also more likely to support ActiveRecord than non-traditional database interfaces.

Regardless of technological constraints, the objective in the technology selection was to facilitate the creation, deployment and maintenance of a valuable product. In the entire decision-making process, and indeed the whole project, emphasis was placed on keeping the context of making a useful application that solves problems real people have in mind. Moreover, WorkHero is designed with future maintainability of the project as a consideration, since value has to be sustained over time.

Search

Search in WorkHero initially was implemented by using simple SQL queries that allowed us to search the database for titles and tags that matched the keywords provided by the user. In v1.1.0, we investigated and implemented a solution for handling full text search, so we could better scale the search aspect of our application - since SQL queries are slow for searching -
and improve the quality of search results by also searching through the descriptions of activities and tools.

We selected Apache Solr as our search provider since the corresponding gem Sunspot is well-documented and well-supported (“GitHub – sunspot/sunspot”, n.d.), allowing for agile implementation and deployment. Solr is a search engine that is built on top of Apache Lucene and is designed to handle deployments that range from megabytes of data to several terabytes of data (and beyond) without compromising usability or performance.

Analytics Provider

Most modern web applications use analytics to better understand how their application is used. WorkHero implements behavioral and demographics-based analytics for this reason. The choice for the platform in this respect is Google Analytics, a comprehensive and extensive analytics platform. The choice was made based on the support the community extends towards it as well as simplicity of implementation.
Chapter 3: Business Feasibility & Justification

WorkHero is built with real customers in mind. While some projects exist for the purpose of furthering technology, WorkHero exists to solve the needs of real people. In particular, WorkHero solves the challenge that people face when sorting through tools, resources, and strategies for achieving their goals.

Through the interactions with students, staff, faculty and professionals, the need for an online repository for tools was established. WorkHero is not unique in terms of online tool repositories; several competing products exist, such as:

- AlternativeTo
- G2 Crowd
- Stackshare.io

Most competing solutions are focused on software tools, business needs, and technology applications. While these solutions are great for IT professionals, developers, and computer enthusiasts, they are not as useful for broader audiences. These solutions are also heavily focused on the tools themselves, rather than on the goals of the tools' users. As a result, the aforementioned platforms are inadequate for applications where people have a goal in mind rather than a specific set of tools that they wish to compare.

Although we later consulted several stakeholders in better understanding the need for WorkHero, the initial idea originated from one of the team member’s (Ryan’s) involvement with the WorkMyWay IQP, which sought to build an online repository for academic tools. WorkHero makes the solution more generic and universally applicable, allowing professionals to use the service. Additionally, it offers the ability for users to share the way they accomplish their goals, instead of just sharing tools that help them do so.
Monetization Options

As WorkHero gains traction, the need for financial self-sufficiency is fundamental to scaling the project and continuing to offer the service. Today’s implementation of WorkHero is able to use a single low-cost server to run the service. However, the application cannot be indefinitely served by a single server; as user acquisition grows, the need for several servers to handle the increased number of requests, as well as the need for increased site uptime, dictates additional cost for the service. Furthermore, as the site becomes more valuable, the need for full-time staff to manage the site, make improvements, and further the WorkHero vision will entail additional cost to the project.

Given all of the potential ways that WorkHero can grow, several options for monetization were carefully considered. Most importantly, the value of WorkHero lies in the ability for diverse and disparate groups of students, professionals, and experts to share their knowledge. Thus, while reasonable for the launch phase of WorkHero, keeping the site limited to users within WPI reduces the long-term value of the service. In a way, WorkHero is similar to a university library, which has books on a plethora of topics; a university library would be far less valuable if it limited its access to academic papers, books, and resources related solely to the university’s degree programs.

Specifically, the following monetization options were explored:

Section intentionally left blank.
Chapter 4: Development

When implementing the product, we encountered a major challenge. The core product had to be launched in a timely manner to be able to gather feedback and better address the need the platform was trying to fulfill. At the same time, the project team was limited to the commitment of two part-time developers. Hence, the team was faces with a managerial challenge to efficiently use its resources and effort. Our development process kept this reality in mind and worked to ensure a solution to manage development and push the product ahead was implemented.

Development Management

The team was used Agile development to fulfill the need for structured development. Trello, a Kanban-style project management software to organize, proved to be useful for this purpose. Each card (issue) on the Trello board marked a single, contained development goal. All issues were sorted by their priority at the time, from “core”, which marked the highest priority and was to be done by our initial release day, to “wishlist”, which marked ideas thought to be useful in future releases. We organized efforts into two week sprints, in which a predetermined number of achievable tickets had to be completed. Before each sprint, the team sat together and determined which issues should be prioritized next. Through this methodology, the WorkHero team was able to see at all times how far along the product was in its development and quickly address roadblocks. We also used Trello to maintain a centralized place for sharing important documents related to each card.

Version Control

Rapid software development requires the ability for developers to work on different features at the same time without the risk of losing progress or fostering file conflicts. The most common
solution to the problem is version control software and due to its popularity and familiarity within the team, Git was chosen as the solution for WorkHero. However, to fully leverage the potential of version control system, a secure central server to store the codebase was needed. We chose to run Gitlab on a private server, since it provides many similar convenience and collaboration features of GitHub without the same concern of public disclosure of proprietary information.

Git is a flexible solution, which allows teams to employ different strategies dependent on their particular use case. We used a simple branching strategy until our initial deployment, and then moved to an adjusted branching strategy to reflect the possibility of future production releases.

During our initial development milestone, a single main branch ("master"), from which all feature branches were derived was used. Once a feature was considered complete - meaning it exhibited the desired behavior - the branch was merged with master and all conflicts were resolved while merging. One of the major benefits of branching for the team is that it allows for parallel development, without the need to worry about conflicting commits while working asynchronously.
After deploying WorkHero to the production server for the first time, the strategy had to be adjusted. Changing to a new default branch - “Development” - allowed to continue making improvements in the development environment, while only periodically merging changes to the master branch. This also allowed for the use of the master branch for production deployments since it made accidental merges or commits unlikely. During each deployment, a new tag was created for the commit, including a changelog and a description of the release.
Team Communication

Effective communication was a major contributor to the agility and efficiency of the WorkHero team. Most of it was actionable, clear, and relevant, minimizing inefficiencies in making decisions and pushing the project forward. Weekly advisor meetings were conducted and served as an opportunity to demonstrate work completed in the previous week. At the end of every two-week sprint, new features were demoed, followed by a discussion of any roadblocks. We then finalized plans for the next sprint at the end of the meeting. On some occasions, we had to adjust our plan, especially when a different logical next step was needed.
Despite having meetings each week, most of the team communication took place on Slack, a team communication service which allows sorting of messages into different channels as well as vast integration possibilities. We initially used Webhooks from our Gitlab server to monitor updates to our source repository, but later added Trello as an integration to Slack to reflect the current project management board in Trello.

Besides Gitlab and Trello, Google Drive was used to collaborate on documents throughout all phases of the project. It was chosen not only because of the real time collaborative editing nature of Google Docs, Sheets, and Slides, but also because it integrates with design tools already used, such as Draw.io. Drive proved much simpler than using a SMB share in this use case, since it does not require a VPN connection in order to access documents when away from campus, as well as enabling for parallel collaborative editing of the same document.

Development Software

While the team did find it necessary to agree on common team communication and management software, there was no need for a common development environment (text editors, IDEs, etc.) Our only goal with this aspect of the decision-making process was that each developer was as comfortable and familiar as possible with their development environment, so they could work as efficiently as possible.

Containerized Development Environment

One of the requirements for the project was the ability to work on WorkHero without requiring a developer to utilize a central server. This requirement was chosen because:

- It allows the team to work anywhere, even without an active Internet connection.
- Working in parallel without concern about whether the development server will be affected by one developer's actions is possible.
- Portability of the application is already necessary when moving from development to production.

Like many other web applications, WorkHero comprises of several different server components. Each component has several dependencies, which sometimes can conflict with each other. Docker was chosen to solve the dependency conflict and portability issue, since we can isolate each server role, such as our Ruby on Rails backend and our Nginx-based frontend.

Docker is currently the most popular container format and distribution system (DataDog, 2015; Business Cloud News, 2016) which has a vibrant community and many prebuilt images available for use. For WorkHero, the team utilized Docker along with Docker Compose to define and run entire server environments both on our local machine and in production.

Docker Compose is an essential component to the project’s containerization strategy. While other tools exist for creating and orchestrating entire multi-container environments, few tools are simpler to configure or utilize for development environments than Docker Compose. It is possible to also use Docker directly to achieve the same outcomes as Docker Compose, but this approach is neither simple nor efficient.

Vagrant

One caveat of using Docker is that it requires Linux (or Windows Server 2016) to run. Since one of our design goals was to allow our developers to use the development environment they’re already familiar with, and one of the developers primarily utilized Windows, a need arose for the use of some form of virtual machine to run a Linux system that can then run Docker and Docker Compose. Vagrant was the choice for this task, since it is designed specifically for defining and
creating development virtual machines. Vagrant also simplifies several common tasks that developers need, such as sharing folders between the host and the guest machine and remotely connecting to the virtual machine over SSH.

KVM-based Linux Production Server

The other developer overcame the challenge of needing Linux to run Docker by running a virtual machine on a dedicated virtualization server. While the primary reason for choosing this option was the urge for a persistent development environment that could be remotely accessed on any machine that has an SSH client, where one can return to the last used state, this option also made it possible to use any client operating system such as MacOS or Windows. Persistence was made possible by the use of a terminal multiplexer - such as TMux - since it allows for multiple sessions that can be detached (and reattached), without requiring the client to stay connected to the session.

Another benefit of using a dedicated virtual machine for each project is that it creates separation between several projects. Maintaining separate environments allows for better organization, greater reliability and more flexibility with respect to the development environment.

Scaffolding

Modern web applications are complex to setup. In order to expedite development and ensure a robust initial structure, a Yeoman generator was used. Yeoman is a tool that allows users to generate all files for general structured web applications. In particular, WorkHero used cg-angular-generator, a generator designed for large scale AngularJS applications that includes automation tools for building, testing and running a web app as well as subgenerators to add
specific components. When adding new features to the codebase, the appropriate subgenerator should be used to keep consistency and ensure all necessary references exist.
Chapter 5: Deployment

One of the business requirements for WorkHero is frequent iterations that improve product quality, fix bugs, and providing more relevant functionality to our users. Along with the development process, the process for deploying our application plays a significant role in the ability for our team to be agile in rolling out changes to our product. As a result, our deployment process is designed to be simple to manage, quick, and reliable.

Two types of deployment processes exist for WorkHero: minor and major changes. Minor changes include immediate bug fixes or simple product improvements that need to be rolled out immediately. Major changes typically happen at the end of a milestone and include a significant volume of code changes. Over time, we expect major deployments to become less frequent, as the volume of highly complex and significant (in terms of quantity of code changes) will subside in favor of many small changes. In essence, the distinction exists solely because many early features had a significant number of dependencies that are related to the core functionality of the application.

Versioning

In order to keep releases easy to understand, WorkHero follows a specific versioning convention. Releases encompassing extensive rework of the application is denoted in the first number, while major releases are noted through the second number. Lastly minor, frequent fixes and changes are denoted via the last number. As of February 24, 2017, the version of WorkHero is 1.1.1, denoting the first major release, with major features added over 1.0.0 and several bugfixes since version 1.1.0.
Deployment Process

Besides versioning, our deployment process for major releases differs only slightly from minor releases. The need to rerun scripts to setup the production environment in Docker, for instance, is one difference between a major and minor release. Other than that difference, our deployment process goes as such:

1. Merge “Development” into “master”. Only “master” should be used as a branch during deployment.
2. Create a tag on the commit, with the appropriate version name, such as “v1.1.2” for the second minor release of “v1.1”.
3. Connect to the production server(s) and run `git pull` to pull the code changes down to the server(s). This process can be automated using a DevOps automation tool like Ansible.
4. Checkout the tag on the production server(s) using `git checkout v1.1.2` where v1.1.2 is the name of the version tag created in Step #2.
5. OPTIONAL - if you changed the dependencies for the application, including the Gemfile, then run `./tools/setup_prod_env.sh` to rebuild the appropriate Docker images. Only changed images will go through the entire process, so this process should be relatively quick.
6. On each production server(s), run `./tools/build.sh` to execute the build process.
7. OPTIONAL - if you changed the database schema, you’ll need to migrate the changes to the database by running `docker-compose --file production/docker-compose-production.yml run backend rake db:migrate`. This only needs to be done on a single server, since the database should be common on all servers.
SSL Certificates

Since our application involves sending and receiving potentially sensitive data, especially including plaintext passwords, our application should be served via an SSL connection. SSL provides transport-level security, by incorporating an key exchange process that uses asymmetric cryptography to allow the client and server to “handshake” and exchange a symmetric key that is used to encrypt the data in transit (robertckl, 2014). As a result, connections to an SSL-enabled website are protected from interception and allow for sensitive data transmission; SSL is the same technology used by banks to protect connections between account holders and the bank.

Currently, we use Let’s Encrypt to generate SSL certificates, as it provides for a free-of-cost and simple solution to generating and installing SSL certificates. One of the downsides, however, of using Let’s Encrypt are the short certificate lifespans; certificates on Let’s Encrypt only last 90 days from issuance (Aas, 2015). While this process can be automated, the added hassle of monitoring Let’s Encrypt-generated certificates may not be optimal for long-term deployments.

In addition, if WorkHero begins to offer a whitelabel solution to other institutions, Let’s Encrypt cannot issue wildcard certificates (Let’s Encrypt, 2017), which are a common way to provide SSL for multitenant environments that are switched using subdomains (such as acmecorporation.workheroapp.com, wpi.workheroapp.com, or umass.workheroapp.com). While it may be possible to script the creation of new certificates using Let’s Encrypt, Let’s Encrypt enforces strict rate limiting, making it difficult for the application to handle the creation of a very large number of new tenants in short period of time. As of the time of this publication, Let’s Encrypt limits requests to 2,000 new subdomains per week (Let’s Encrypt, 2017).
Chapter 6: Product Launch

In order to collect more information about user reactions and needs, the team decided to release a core version of the platform on November 4th, 2017 at https://workheroapp.com. While the initial features were selected by the team, the platform was designed to include only core functionality in the initial release, in order to gather user feedback to see which measures would bring most value and to possibly pivot the platform if necessary. Several qualitative user tests in which users navigated through the platform and described their experience and thoughts as they moved through the interface yielded the first data and increased the amount of development tickets as several previously unknown bugs and user experience issues were encountered. Examples of this include, but are not limited to, the user interface not scaling properly on different resolutions and bugs in the voting system.

After addressing the most pressing problems discovered during this phase, we shifted focus to scalability and usability of the platform. Later, we conducted another user session by inviting 35 undergraduate WPI students to use WorkHero and provide us with feedback.

This user session, which lasted about an hour, was successful and yielded some interesting results. For example, the team realized that WorkHero does not work on Microsoft Internet Explorer 9 and lower. Other feedback pointed towards more engagement, which we had previously considered but deemed less important than search and data entry features.

Based on the feedback received, we developed a list of issues that we added into the Trello board. These issues were subsequently categorized by importance depending on how much the issue came up, our ability to develop the necessary components in time and which direction
would take WorkHero closer towards its mission. We then incorporated this new feedback in our revised product roadmap.

While the platform was technically open to anyone who went to the website, no active marketing was undertaken. We postponed public marketing of the platform until we could ensure the platform was better designed to retain users, provide value, and scale with a growing user base.

The next major release, apart from bug fixes, focused on limiting the amount of loaded data and thus enhanced both load time and scalability and was released as version 1.1.0 on February 3rd, 2017.
Chapter 7: Recommendations

WorkHero is an ongoing software project which requires support as well as improvements to garner and sustain success. However, since the current project team is moving to an advisory role as this MQP is coming to a close, the need for specific recommendations regarding the future of WorkHero is provided by this chapter. We encourage all future teams to read this section as it contains crucial information about next steps for the project.

Development

Many of the upcoming development tasks are focused on improving community engagement. In order to effectively monetize, it is crucial to build a set of active users and strive for growth in sign-ups. The team thinks it probable that adding to the engagement aspects of the platform will increase use time and thus make the product more valuable.

Adding the ability to directly contact the author of a post and engage with them via a chat feature could offer users the ability to get direct help from people with experience and therefore move the product a step closer to its intended mission. By leaving the chat exclusive to the platform instead of opting for other methods, such as contact via email, we could increase the number of recurring users. More time and engagement on the platform translates to more opportunities for monetization.

Other opportunities to increase engagement include building on the model of social media. Some relevant social media functionality includes the ability to follow people, comment on activities and add more content beyond text, such as videos and photos.
Improving search and ordering capabilities could further improve the user experience within WorkHero. Filtering of activities provides the user with more control over their search results, which in conjunction with enabling activities to be trending could offer a strong argument for users to check the platform more in order to see the latest content or check on the performance of their own submissions.

Another means to foster community engagement is an achievement system. In this case users would get certain achievements based on their contributions. This could be particularly useful if the platform enables them to build a profile which can be linked to existing platforms such as LinkedIn or Xing to demonstrate skill to recruiters.

While we decided to build WorkHero as a web platform, its API is clearly separated from the frontend. This means there is the potential to develop a mobile app that merely taps into the API without requiring its own backend. The frontend can also be migrated to a semi-native mobile app using Ionic, which allows us to preserve and share much of the existing frontend codebase.

Search

One of the major improvements made in v1.1.0 is the addition of a robust search solution using Apache Solr. In previous releases, WorkHero was only able to search the titles of activities and tools, as well as tags, since full text search is not scalable using direct SQL queries. Apache Solr is a drop-in full text search engine that can index a wide variety of datastores, including many relational databases, and can quickly (on the order of only a few milliseconds) return results that even include long descriptions for activities and tools. Solr is designed to scale in large enterprise environments and can easily handle terabytes or more of data without greatly slowing query performance.
Our deployment of Solr relies on Sunspot, which serves as both a driver for Solr and a domain-specific language (DSL). Sunspot’s DSL makes it possible to quickly define common search query operands, such as facets - parameters that help to narrow down the search, like a date range or a boolean checks; facets are what make narrowing down by the number of stars on popular online shopping websites possible. Additional configuration of search functionality using Solr is possible through the `schema.xml` configuration file.

We recommend future teams work to improve search quality and usefulness. For example, search currently is limited to textual queries only; there are no filters or facets enabled. Another improvement we suggest is the creation of a portal for users to filter tools and activities based on categories, so users can look at only categories that interest them. This feature can be combined with user-specific preferences on categories that should be displayed. Search can be configured through the DSL to return results based on the user’s category preferences, as well as allowing for portal-driven navigation of categories.

Authentication

WorkHero currently uses a combination of Devise and AngularDevise to support secure user logins on our platform. Devise is a popular and well-supported authentication module (gem) for Ruby on Rails, which offers out-of-the-box support for email and password authentication, as well as Facebook, Twitter, and Google login. Currently, we only rely on email and password-based authentication, since it offers adequate flexibility for users.

Future teams may want to enable support for social media logins, as this provides users with less friction during the registration process, since they do not need to fill out a form to create a new account. Another login option that is particularly important for white-label service offerings is the integration of Shibboleth-based single-sign on (“SSO”) providers. Shibboleth is the
underlying authentication technology for WPI’s single sign-on solution called Central Authentication System (“CAS”). Many other institutions also use Shibboleth for their own SSO needs, so it is a logical addition to authentication options for WorkHero.

Adding support for Shibboleth requires some additional work, since its functionality is not integrated directly into Devise. However, one of the advantages of using Devise is the ability to add additional authentication functionality - including Shibboleth support - through the installation of Devise-compatible Rubygems. Another possibility is the addition of multi-factor authentication. Again, like Shibboleth, multifactor authentication can be added to WorkHero through the installation of Devise-compatible Rubygems.

In all cases, the frontend will need to be modified to support the additional providers. For example, adding time-based OTP multifactor authentication, requires the addition of another login field on a separate page, unless multifactor authentication is required for every user. These additions also need to accommodate the possibility of different authentication types on a per-user basis. Nevertheless, one of the most useful and well-implemented aspects of Devise is the ability to simultaneously support many different authentication providers at the same time.

Another area for improvement that should be addressed by future teams is the improvement of the mailer templates when sending emails via Devise. Currently, the default mailer template is used, which lacks branding entirely. We encourage future teams to create mailer templates that are consistent with the project’s brand image and the design language used by the application. This not only makes the mailer more pleasant to work with, but also it helps to foster trust with users, since many fraudulent emails are generic in nature and lack any branding (or have poorly implemented branding).
Testing

To speed up development, the team relied on manual testing. This enabled development to be faster while the platform’s features were changing rapidly. However, as the platform grows, adequate testing standards reflecting the new state of the product will have to be established.

The frontend of WorkHero already possesses most of the necessary infrastructure for unit tests. Code quality is ensured using JSHint, a broadly used static tool for code analysis. JSHint checks every JavaScript file within the frontend and finds potential syntax issues. Currently JSHint is run during the build process and while serving files in development.

The second layer of front end testing are unit tests written with the Jasmine framework and executed using the Karma test runner. Each JavaScript file that is intended to be tested within WorkHero has a corresponding test file with a name extension “-spec”. These files are designed to contain all tests for their respective associated file. Karma executes these functional tests in two browser environments (specifically Google Chrome, Mozilla Firefox) and a headless WebKit scriptable (via PhantomJS).

A further recommendation the team makes is to add a code coverage tool (such as Istanbul) to the frontend test infrastructure. This enables detailed reporting on code coverage and allows developers to employ minimal coverage standards before deployment.

In terms of backend testing, we suggest using the built-in testing capabilities of Ruby on Rails. Rails provides support for model tests, unit tests, and integration tests. These tests can be run during the build process, as is described in the next paragraph about automated testing, and include the ability to test the behavior of models, actions, controllers, views, and integrations. In
particular, unit tests provide the ability to test specific features, while integration tests allow for broader testing of the interaction between features and major program features.

To simplify and automate testing and deployment the WorkHero team recommends the usage of continuous integration software (CI), which can be configured to automate several consecutive crucial tasks, monitor their progress and abort in case a standard is not fulfilled. For example, a CI system could be setup to run both backend and frontend tests whenever a branch is merged or even if a commit is pushed. If an error in testing occurs the developers can be notified automatically and be assigned to fix issues, making for a simple and more qualitative production pipeline.

Development Database and Environment

During the course of this project, the team has relied on local database instances for testing during development. While this has served well in the early stages of the project, as the application becomes more dependent on the availability of data it will require a more careful management.

As a result, we recommend future project members to configure and utilize a common development database and search appliance. This helps to eliminate the discrepancies between environments and helps to ensure testing is done in a consistent and reliable manner.

Guidelines for Creating a Development Database

Teams wishing to set up a development database should carefully read this section, as it contains important security, performance, and reliability guidelines. Since database servers often contain sensitive data, or at the very least, need to be kept operational, it is imperative that
teams setting up a new development database be configured on a limited access network. It is recommended to use a VPN that limits access to the database server only to people who are connected to the network. We also recommend using separate user accounts for each developer on the database. Regular automated backups should be established and periodically checked, in order to protect the long-term integrity of data in the development database.

Although ActiveRecord attempts to provide a generic interface to relational databases, we recommend future teams use the same database product and version in both development and production. For example, if the production server is running on MySQL 5.7, then we suggest the development database to be MySQL 5.7 as well. We even suggest copying non-sensitive data from the production database to the limited-access development database, in order to aid in the testing process.

Moreover, while developers can use the same database server for production and development by creating separate tables and users, we do not recommend this practice. In general, we suggest limiting access to the production database to only trusted administrators and the application itself; allowing broad access to the production database opens up an opportunity for abuse, increases the attack surface, and could compromise the integrity of stored data.

**Migrating to External Server Hosting**

WorkHero currently relies on WPI’s hosting services for providing a virtual machine and database server. However, as the project grows and the need to be able to scale the application becomes more evident, external hosting providers may need to be considered. Fortunately, there is a plethora of choice with regard to hosting providers at nearly any price point.
DigitalOcean is the world’s second largest hosting provider, only behind Amazon, and is growing rapidly. One of the major advantages to using selecting DigitalOcean is the price advantage in lower end tiers over Amazon, Azure, or Google, without compromising performance or service quality and reliability.

While DigitalOcean is the most popular developer-oriented hosting provider, they are not the only option. Many other companies have begun to offer competing services at a similar price point that promise more server resources (memory, storage, processing power, or bandwidth), but many are only able to achieve that price point by overselling hosting (sometimes making the entire service unusable), compromising customer service and support (some providers take days or weeks to respond), or by maintaining poor business practices (such as billing for services after cancellation, despite full payment already taking place).

Hosting Requirements

Since the project uses Docker, the process for transferring WorkHero to an external service provider is designed to be relatively simple, except for the database migration. It is possible to run WorkHero on a separate hosting provider while continuing to use the WPI MySQL server, but we do not recommend this migration path since the latency between the Rails server on an external hosting provider and WPI’s MySQL server is high enough to cause noticeable performance regressions for WorkHero.

Not all hosting solutions are suitable for WorkHero. Most importantly, the minimum requirements for running WorkHero are:

- 2 CPU cores / threads
- 2GB of RAM (with a swapfile of 4GB enabled)
- 10GB of disk space
- **CentOS 7 (or any Linux distribution with a minimum kernel version of 3.10, as required by Docker Engine)**

While it is possible to run the entire application on this type of environment, performance may not be optimal. We suggest creating multiple virtual servers to run WorkHero with at least one for the Rails server and the Nginx frontend server, one for the Apache Solr distribution, and one for the database server.

### Migrating the Database

The current implementation relies on the WPI MySQL server that is made available to student applications. This server is accessible from mysql.wpi.edu on port 3306 and can be connected to from any Internet facing client. While MySQL is used for convenience - mainly because the MySQL server is backed up regularly by WPI - we recommend using PostgreSQL because of its better support by some Rubygems, more advanced configuration capabilities, and better support for certain datatypes (such as indexable JSONb) (Tezer, 2014).

While it is possible to dump the entire database using the MySQL dump tool (mysqldump), as seen below:

```bash
> mysqldump --databases workmyway > workhero.sql
```

The team recommends using DBBeaver as a client for managing and migrating the database to a new provider. One of the major advantages of using DBBeaver is that it makes it simple to migrate between databases directly, without requiring reimportation between the two databases.
Warning: Rails typically handles creating a unique ID for each row in a table by incrementing the last ID by 1, it stores the current highest ID and uses that stored value when it creates a new record. Since importing data does not use Rails, the Rails server needs to have its counter reset to match the database. This can be done by running:

```
$ docker-compose --file run backend production/docker-compose-production.yml bash -c 'RAILS_ENV=production rails console'

> ActiveRecord::Base.connection.reset_pk_sequence!("table_name")
```

It needs to be ensured to replace ‘table_name’ with the correct table. This will reset the counter to match what is in the database.

Scaling WorkHero

Scalability is a concern with most modern web applications, including WorkHero. As more users begin to regularly access the platform, the ability to handle many concurrent connections without high latency or application crashes is essential to growing the application’s user base.

The frontend of WorkHero is static, since all of the logic and presentation-layer effort is handled by the client’s browser, rather than a central server. As a result, the AngularJS-based frontend for WorkHero can be easily scaled by adding more HTTP servers - and placing these servers behind a load balancing proxy - or by storing the site data on a Content Delivery Network (CDN). The primary constraint for the frontend is available network throughput (bandwidth) and CPU performance. Memory usage is fairly limited if Nginx is used, since it only requires a few kilobytes per connection (Ellingwood, 2015).
The backend of WorkHero, however, is more difficult to scale. The Rails server can be scaled by adding multiple Rails servers and placing a load balancing proxy in front of the servers. Nginx is designed for large-scale, high traffic web servers and can be used as a load-balancing proxy. Other load balancing options include haproxy and load balancers offered by hosting providers, like Amazon or DigitalOcean.

The difficulty with scaling the backend, however, lies with the database. Since WorkHero uses a transactional database, data consistency is guaranteed, meaning users only see one set of data. While consistency is essential to applications that require data to be the same for all users, such as bank transfers and ticket purchasing applications (you don’t want users to buy the same ticket twice), WorkHero is more tolerant to inconsistency. An eventually consistent write strategy is appropriate for WorkHero, with the only caveat of users potentially adding the same tool or activity. The likelihood of such an event will decrease as more data is added to the platform.

Nonetheless, both MySQL and PostgreSQL can be made highly available with some effort. Scaling on a relational database is typically handled through sharding, where data is partitioned between several servers and requests are handled through a broker that understands the sharding scheme and can properly route the request to the appropriate server. High availability is provided by maintaining redundant copies of the same database. Most high availability solutions rely on a master-slave configuration, where only one server typically handles requests unless the master becomes unavailable.

Master-slave replication schemes are helpful in maintaining consistency, since requests are less likely to create conflicting transactions than in a many-master scenario - since only one master
handles all requests while it is elected as the master. Master-slave replication can be used in conjunction with sharding, with each node in the sharding cluster having any number of slaves replicating them, as is shown in the diagram below.

![Figure 4 - Sample HA & Sharding Scheme](image)

A Further Discussion of Monetization

*Section intentionally left blank.*
Chapter 8: Conclusion

With the current team transitioning off the project, the decisions made concerning the development have to be reevaluated in the context of their success. Considering the ongoing nature of WorkHero it is necessary to reevaluate the decisions in light of their merit for future teams. While the team has worked well together, scoping issues have occurred. The agile development method has enabled WorkHero to become a reality and the team to become more realistic with goals over time, yet occasional pivoting was inevitable to address user feedback and therefore several features have not yet been developed.

Multi-Disciplinary Project Outcomes

Throughout its development, WorkHero was built with influences from a variety of disciplines. Overall, it models a typical technology start up and thus involves several fields of study and disciplines to make it possible. The most frequently employed discipline in this project is computer programming, which was integral to building WorkHero and was the main activity throughout most of the project. Systems analysis and design was another closely related field that was essential to our project and was heavily used during the initial phases. Systems analysis and design allowed us to transform business requirements into designs for product features, which were then developed using computer programming skills.

Since WorkHero attempts to provide a product that is helpful to students and professionals, the business viability was also fundamental to our process. We utilized our knowledge of the principles of marketing, branding, and human psychology to successfully tailor WorkHero to the needs of our customer base. We also worked on building a business model that helps our project maintain long-term viability as a product, by allowing it to be financially sustainable.
Product management was another core activity throughout the entirety of this project and was involved in nearly every process; product management includes building our product roadmap, collaborating with users to build solutions for their unique needs, evangelizing our product and presenting the product to key stakeholders in both technical and non-technical contexts. Moreover, we used product management skills to serve as the liaison between technical and business disciplines throughout our project.

Furthermore, project management was a key component of our overall process, since we worked to successfully plan our project using Agile and Scrum methodologies. Our project management process incorporated resource planning and allocation, time budget planning, task prioritization and task scheduling that is typical to the field of project management. We also successfully implemented aspects of the Agile and Scrum methodologies, such as short iterative work units called sprints, as well as story point-based progress tracking, and biweekly Sprint Review Meetings.

Since our overall goal with WorkHero was to build a product that was useful to our target market, we were required to be flexible throughout the course of the project and learn whatever was necessary to complete the project’s objectives. Both team members actively learned new skills to make this project a success. The most notable skill acquired during this project was backend web development, which was made possible by Michael Hartl’s *Ruby on Rails Tutorial*.

Final Words

WorkHero has the ability to fulfill its mission of bridging the experience gap between beginners and experts in a helpful manner as long as several conditions are true; most importantly the team has to ensure that the platform receives valuable content while pursuing continued growth. Therefore, a team with marketing and business expertise is desirable. Secondly, the team will
have to monetize the platform or find funding in another way to bring WorkHero towards full self-sustainability. Lastly, new features will have to be developed and tested with. Users are the single most important factor for success. Gathering, analyzing and implementing their vision of the platform while keeping the mission in mind could enable WorkHero to broaden its reach.
Bibliography


Active record basics


DataDog.8 surprising facts about real docker adoption. Retrieved from https://www.datadoghq.com/docker-adoption/


Appendix A - Lean Canvas (February 2017)

Section intentionally left blank.
## Appendix B - Framework Matrix (February 2017)

### Framework Matrix

<table>
<thead>
<tr>
<th>Category</th>
<th>AngularJS (Angular2)</th>
<th>Meteor.js</th>
<th>React</th>
<th>Ember.js</th>
<th>Django</th>
<th>Ruby on Rails</th>
<th>WordPress</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layer</td>
<td>Front-End</td>
<td>Front-End</td>
<td>Front-End</td>
<td>Front-End</td>
<td>Back-End (though it has full stack capabilities)</td>
<td>Back-End</td>
<td>Full-Stack</td>
</tr>
<tr>
<td>Pattern</td>
<td>MVVM</td>
<td>None enforced (MVC typical)</td>
<td>Only View</td>
<td>MVC</td>
<td>MVC</td>
<td>MVC (strict)</td>
<td>MVC (strict)</td>
</tr>
<tr>
<td>Type</td>
<td>Framework</td>
<td>Framework</td>
<td>Library</td>
<td>Framework</td>
<td>Framework</td>
<td>Framework</td>
<td>CMS</td>
</tr>
<tr>
<td>Flexibility*</td>
<td>Medium - High</td>
<td>Medium</td>
<td>High</td>
<td>Medium - High</td>
<td>Medium</td>
<td>Medium</td>
<td>Low</td>
</tr>
<tr>
<td>Simplicity**</td>
<td>Medium (High)</td>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
<td>High (Learn) / Low (Code)</td>
</tr>
<tr>
<td>Language</td>
<td>JS (TypeScript/ Dart/JS)</td>
<td>JS</td>
<td>JS + JSX</td>
<td>JS</td>
<td>Python</td>
<td>Ruby</td>
<td>php</td>
</tr>
<tr>
<td>Our Familiarity</td>
<td>High (Low)</td>
<td>Medium</td>
<td>Low</td>
<td>None</td>
<td>None</td>
<td>High</td>
<td>Medium (low in</td>
</tr>
</tbody>
</table>
*Flexibility: How easy is it to combine with libraries? Another Front/Back-End? Break out of typical pattern?

**Simplicity: How easy is it to learn? How easy is code to read?

***Support: Based on Libscore Site usage and Github Forks
Appendix C - MQP Demo

Shortly after our late October product launch, we gave a presentation to approximately 35 students in order to gain feedback about our product and look for input on how we can improve the service.

Feedback is essential to any new product. We need to know whether we’re on the right track and how we can better meet your needs.

Help us build something you find useful!
Go to workheroapp.com
1 point for each tool you post.
2 points for each activity you add.
3 points for every bug you find.
4 points for each unique bug or useful insight you offer.
A funny thing about feedback...

Just having feedback does not automatically lead to a better product.

A key aspect of a Product Manager’s job is to turn feedback into an action plan. Important, not all feedback should be acted on.

Sometimes, feedback is a result of a latent need that the customer doesn’t know they have.

“If I had listened to customers, I’d have given them a faster horse.”
- Henry Ford

workheroapp.com

How metrics are key in our process

1. Ideation - no key metrics
2. Planning + Design
   a. # of tasks and milestones
   b. Proposed Features
3. Development + On-The-Fly QA
   a. Overall % per sprint
4. Deployment (all metrics)
   a. Uptime
   b. Page load time (ms)
   c. Backend query latency (ms)
   d. Perceived + actual UI lag (ms)
5. User Testing + Feedback
   a. Bug reports (feedback / metric)
   b. Time spent per visit (metric)
   c. # of active monthly users (metric)
   d. User suggestions + product improvement engagements (feedback)
Structuring the Demo

One of the goals of our demonstration was to evaluate the usability of our platform for people who try to access our platform independently. We wanted to see how successful users were with using our platform without prior knowledge of the platform. As a result, we started our presentation with only a request for help, instead of describing the functionality of our platform.

During the feedback stage, users were asked to create content and provide oral and written feedback in the form of bug reports. After the hands-off testing and feedback stage, we asked students to provide general feedback. The feedback provided during this phase was not limited to just the platform itself: many contributions were focused on the business viability and future of WorkHero. We combined all the feedback received during this session into a combined document, which we later used to revise our roadmap.
Revising our Roadmap based on Feedback

Student feedback was a valuable component to our overall strategy for improving WorkHero. Since we utilized Trello for organizing and visualizing our product roadmap, we created cards for each unique piece of feedback (with an indicator for feedback that was popular). We then organized the feedback into lists representing the featureset of subsequent versions of WorkHero.
Appendix D - Entity Relationship Diagram for WorkHero

Diagram intentionally left blank.

Figure 5 - Entity Relationship Diagram

The above entity-relationship diagram (ERD) shows the current state of the data models and their relationships for WorkHero. The diagram was automatically generated using Rails-ERD, a package designed for automatically capturing and creating diagrams that reflect the application’s actual implementation. Each level, as delineated by each section from top-to-bottom, represents logical subgroups of models.

The top group consists solely of the user model, which is the base unit (since all other models stem from user interactions). Users typically interact with the second group of entities, such as “tools”, “tags”, and “activities.” Finally, the third group contains metadata about the tools, tags, and activities, including the associations between models, scoring information, and comments. Both scoring information and comments are user-associated models, while the associations between tags, tools, and comments are based on user-specified connections (such as when a user creates an activity with many tools associated with it).