Software Requirements Specification

Mirai

A web application for daily task organization

Derek Williamson
Tyler Carey
Anna Pikus
Anthony Nguyen

University of North Texas, CSCE 4901.070
Team Alpha

Monday, February 10, 2020

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this project is to develop and deliver “Mirai,” a web application that provides users an intuitive interface that enhances the process of daily planning and task organization. This application succeeds physical planners, calendar applications, and project management applications by combining the best of these three worlds: personalizable organization, straight-forward and familiar interfaces, and in-depth yet simplistic customization. The scope of this project includes the development of the entire system’s design, architecture, and implementation in order to provide a full web application.

1.2 Document Conventions

In this System Requirements Specification document, you will find a variety of typographical conventions that are consistent from page-to-page. Sections and subsections are denoted in the **Table of Contents** with their respective page number. Sections and subsections are also numbered in a hierarchy and bolded throughout the document. References to other sections from within a paragraph will be bolded and have their respective section/subsection number placed immediately afterwards (if available).

<+ Describe any standards around the format of figures and functional requirements.>

1.3 Intended Audience and Reading Suggestions

This document is intended for the following audiences:

1. The developers on the project’s team,
2. The professor of the course *(Dr. Jonathan Doran)*, and
3. The clients Kevin Carey and Rick Amstutz.

In the rest of this System Requirements Specification document, you will find more involved details on the product’s perspective, functionality, users, constraints, and dependencies. You will also find the interface requirements, both internal (within the program’s functionality) and external (hardware and software). Continue to read on from this Product Scope, keeping in mind any references to future or past sections. Section 2 will be most pertinent for developers while Section 4 will be most pertinent for both developers and non-developer readers.

1.4 Product Scope

The software product of our project is titled “Mirai,” a web application for daily task organization.

These features include implementations of both physical and digital organization processes, such as writing in a daily planner, designating tasks on sticky notes, recording upcoming events and reminders on a calendar application, and tracking work on a project management application. The objective of this project is to (1) provide the application’s source under the MIT license, (2) develop a web application for these existing tools and processes the target users already require or use, and (3) provide an interface that is primarily compatible with desktop PCs, laptops, and medium-to-large tablets. Although smartphones are not a part of the scope, compatibility for such devices has been recorded as a stretch goal for the developers to work on if time permits.

1.5 References

Sass - https://sass-lang.com/

Node.js - https://nodejs.org/en/

Express - https://expressjs.com/

MIT License - https://opensource.org/licenses/MIT

AWS - https://aws.amazon.com/

Project Plan -
https://drive.google.com/file/d/1D1R5ESuSsaQq_-kL11Pn_sCkzmGylzzO/view?usp=sharing

2. Overall Description

2.1 Product Perspective

The product of this project is a web application designed to run within the Amazon Web Services ecosystem and there does not require a set of on-site (within UNT's domain) resources. The application is designed for use on desktop, laptop, and tablet devices and requires that the user has both a browser and an Internet connection to ensure the ability to use the software. Although support for smartphone devices is out-of-scope for this application, it is a stretch goal for the developers. The application integrates the following AWS, GCP, and CI/CD resources: S3, ElasticBeanstalk, Travis CI, and GCP Compute (running MongoDB on Linux). Through the browser, the user will interact with the application by accessing a publicly accessible domain. A flow diagram is demonstrated in Figure 1.

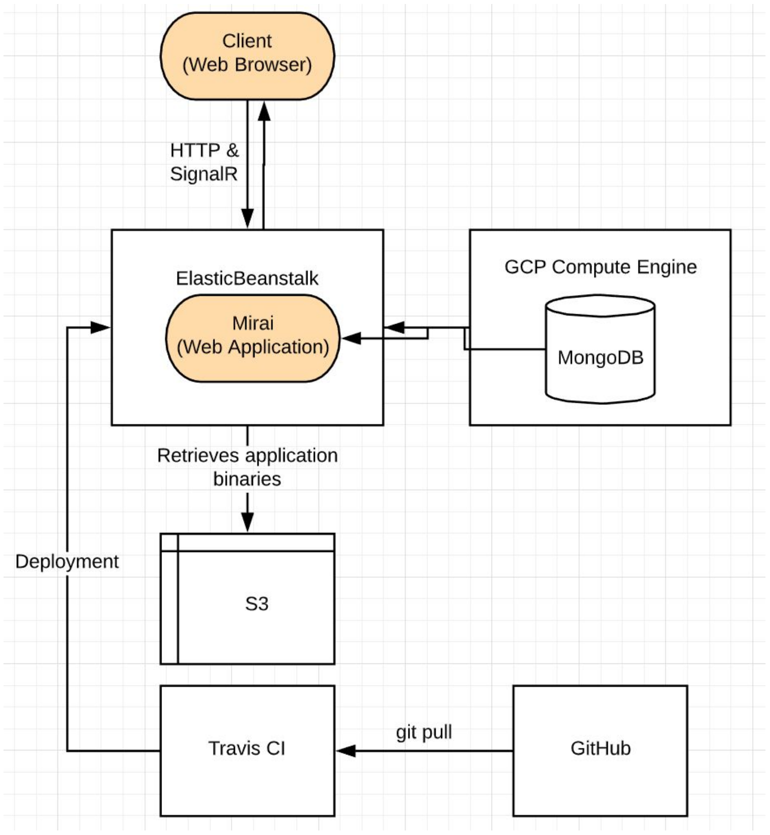

Figure 1 - Interaction between the major components

2.2 Product Functions

In order for Mirai to be considered version 1.0 complete, the software must let users perform the following functions:

- Organize events and tasks as "cards" on their dashboard
- Customize their dashboard's theme
- Personalize their own profile
- Attach documents, URLs, and pictures to their cards

2.3 User Characteristics

Mirai is primarily intended for use by two classes of users: those who currently have a written or digital method of organization that requires improvement and those who don't currently have such methods or applications. Mirai is also intended for frequent, almost daily use with little need for technical expertise. The only experience required is that at the level of device familiarity. If the user is familiar with the usage of their own Internet-capable device, they should be capable of navigating and interacting with Mirai's interface. Mirai will not have an admin panel in v1.0, therefore administrative user classes are not specified. The most important user class to satisfy are those who currently have a method of organization that require improvements.

2.4 Operating Environment

The software application for Mirai will operate on an AWS ElasticBeanstalk environment instance, connected to an AWS S3 bucket for file storage and a GCP Compute Engine instance running MongoDB for data persistence. No other external components or applications are required for the operation of this program.

2.5 Design and Implementation Constraints

Mirai front-end will be based on HTML, Javascript, and Bootstrap. The styling will be based on CSS and Sass. Mirai backend will be based on Express and NodeJS. MongoDB will be used as data storage. Additionally, Mirai has the following constraints:

- Compatibility with mobile browsers may not be supported
- Limited budget
- Limited time
- Limited to frameworks that are compatible with the above languages and frameworks
- Dependent upon the status of third-party web services

2.6 User Documentation

Mirai will provide a tutorial page on the application to inform users on how to navigate and use the application. Documentation will be minimal as Mirai will include a user-friendly and simplistic interface to allow users to easily perform functions. Users may be able to contact the administrator for any questions (method of contact is to-be-determined).

2.7 Assumptions and Dependencies

It is assumed that:

- Users will have a working computer or tablet that has access to the internet
- The database will accommodate all requests
- Third party web services will be consistently working
- The application will handle a reasonable amount of users

See the Project Plan for additional assumptions and dependencies.

3. External Interface Requirements

3.1 User Interfaces

<Describe the logical characteristics of each interface between the software product and the users. This may include sample screen images, any GUI standards or product family style guides that are to be followed, screen layout constraints, standard buttons and functions (e.g., help) that will appear on every screen, keyboard shortcuts, error message display standards, and so on. Define the software components for which a user interface is needed. Details of the user interface design should be documented in a separate user interface specification.>

Mirai will be designed to be an easy-to-use, customizable, and free-flowing organizer. Rather than just a calendar, Mirai will have a "cards" system where the events and tasks are easily displayed with their statuses on a dashboard. Mirai will have only a few different screens the user can change between. First will be the landing page, here all important info of what Mirai is and how it can help will be displayed. Secondly the user, after logging in through a popup widget in the top right-hand corner, will see the dashboard page. Here will be a combination of the sneak peak view and the full view. Throughout the entire interface, notifications should popup in the bottom left-hand corner as to keep up the flow and design standards of how Mirai should interact and look. The logo for Mirai should always appear in the top left-hand corner and should provide the user with the ability to click to return to the dashboard page. A login/logout widget icon should always be present in the top right-hand corner and will allow the user to alter any settings in a easier flowing manner than having an entirely separate page.

3.2 Hardware Interfaces

The application is compatible with any Windows or Linux environment capable of installing packages using npm and running Node.js applications. Since the production environment will support application scaling, specific hardware configurations will not be a direct concern for the developers. For the client, a modern internet browser will be required to access the application and use its features.

3.3 Software Interfaces

The software currently requires the following software components and packages to run:

- Node.js - used for the framework of the application
- Express - used to deliver the view to the client application
- Passport.js - used for user authentication
- npm - used for the installation of packages to be used with Node.js
- MongoDB - used for retrieval and persistence of data
- AWS ElasticBeanstalk - used for hosting the production application
- AWS S3 Bucket - used for storing data and uploads to the website

The application will use the MVC model for data sent between the client and the server. Data from the client will be passed through controllers which manipulate the MongoDB database. Data from the server will update the application seen by the client. AWS ElasticBeanstalk and an AWS S3 Bucket are used to run the server side application and will communicate with a MongoDB database hosted on a Google Cloud Platform Linux instance.

3.4 Communications Interfaces

The application will require users to connect to the web application's domain through a modern web browser on a desktop, laptop, or tablet device. The application will communicate between the server and client by using HTTP requests and the MVC model using Node.js and Express. The server will also use Node.js to communicate with the MongoDB database on the Google Cloud Linux instance.

4. System Features

4.1 Home Button

4.1.1 Description and Priority

The logo for Mirai will always be located in the top left-hand corner of every page in the header and will provide the user the ability to click on the logo to return to the dashboard page. This will allow the user to refresh the page easier or return to the main page when needed. This will be a medium priority

4.1.2 Functional Requirements

- Must be located in the top left-hand corner
- Must show the logo of Mirai as the clickable button
- Must be easily seen as a clickable item
- Must return the user to the dashboard page
- Must be visible on every page

4.2 Login Widget

4.2.1 Description and Priority

An icon will be displayed in the top right-hand corner of every page to illustrate the login and logout feature. Clicking on the icon will open a small pop-up window in the top right-hand corner allowing the user to alter the handful of settings Mirai will have as well as either login or logout. Such settings will be things like "Allow notifications", "Change Labels" (See 4.6 Creating a Card), and "Setting up an Email". This pop-up will also display the users name and profile picture that is setup through their Google account. This will be a medium to high priority.

4.2.2 Functional Requirements

- Must be located in the top right-hand corner
- Must be seen as a clickable button with a standard google login image
- Must allow the user to login and logout
- Should allow the user to adjust the settings in the widget without having to leave the page
- Should appear in the top left-hand corner as a pop-up when opened

4.3 Header

4.3.1 Description and Priority

Both of the above features will be located in a standardized header, this header will keep throughout each page. Mirai will also have a standardized footer, the footer will simply include copyright information in the bottom right-hand corner. This will be a low to medium priority.

4.3.2 Functional Requirements

- Should appear at the top of the page
- Should include and house the Home button and the Login widget

4.4 Dashboard Page

4.4.1 Description and Priority

The dashboard page will be divided in half horizontally into two separate “windows”. The top portion of the page is going to be known as the “Sneak Peak” view, this will allow users to specify by a time requirement what is the most recent task they need to be working on. Easily viewable the card they created for the task will be displayed in a list view with the due date displayed next to the title. From there the user can click on the card and look over whatever task they need to be working on. Underneath the “Sneak Peak” section is the section that will be known as the “Full View” section. The “Full View” section will be the actual organizer and location for the cards and folders to be kept and held. For the Full View system feature please see (4.5 Full View Section). This will be a high priority.

4.4.2 Functional Requirements

- Should display the Sneak Peak view and Full view
- Should still include the header and footer
- Sneak Peak view must show the task closest to the due date first
- Sneak Peak view should be in a list format showing just the title of the task
- Titles in the Sneak Peak view should be clickable and in such should then display the task

4.5 Full View Section

4.5.1 Description and Priority

The Full View section will be where users can create new cards, organize them however they’d like, assign due dates and label them accordingly. For the organizational feature they may be able to do so however they’d like. Mirai is to be used in the best interest of the user. Whether it be displayed all around the page or organized in folders the Full View area should be able to be utilized and fully customizable as to where cards or folders may be placed, including on top of each other. This will be a high priority.

4.5.2 Functional Requirements

- Must allow the user to create a card
- Should be easy to use and flow well
- Should allow the creation of folders and being able to add cards to them
- Should allow a user to place a task anywhere in the Full View area

4.6 Creating a Card

4.6.1 Description and Priority

Creating a card in Mirai should be easy and flow well with the overall architecture displayed. When wanting to create a card a pop-up window will open allowing the user with multiple options. Firstly it will ask the user for a title and description of the card. Afterwards the user may label it accordingly, labels will be displayed in the form of either colors, symbols, numbers or even a combination of those. This can be specified beforehand in the settings pop-up widget when logging in. After a label is assigned, other assignments may be made, such as due dates, people and even other labels. The due dates should be fully customizable in the sense that they may be set to remind the user at multiple specified times via their prefered notification settings. The reminder system should send an email to the user if an email has been provided, a pop-up should appear on the screen in the bottom right hand corner that must be clicked away for it to leave. This will be a high priority.

4.6.2 Functional Requirements

- Must be able to assign the task a name and description
- Should allow the user to assign a due date to the task
- Should allow the user to assign labels to the task
- Should allow the user to add other people to the task
- Should prompt the notification system when a due date is approaching

5. Other Nonfunctional Requirements

5.1 Performance Requirements

The following requirements define user performance requirements when accessing and interacting with the application:

- Every page of the application must load in under two seconds.
- The application must handle an undefined number of cards to fit the unbound needs of the user.
- The application must handle differences in time zones between the client and server.
- Server-client interactions and events must occur in under five seconds.
- The UI must maintain the same stylesheet theme across all pages and devices.

5.2 Availability Requirements

The following requirements define server availability requirements for application integrity:

- No more than 1 per 10000 database transactions shall result in a failure.
- The application shall meet or exceed 99.9% uptime.
- The defect rate shall be less than 1 failure per 1000 hours of operation.

5.3 Security Requirements

The following requirements define security requirements for the application:

- The application shall identify all client applications before allowing them to access application functions.
- Sign in requests must be authenticated by Passport.js.
- Account information shall not be stored on the application's database or file storage.
- All production credentials must be secured using a set of secret keys and shall not be shared publicly.
- The application must use HTTPS to encrypt communication between the server and client.

5.4 Software Quality Attributes

The following attributes define how users should be able to understand the site:

- Users should understand how to create a card within five minutes of using the site for the first time.
- Users should understand how to customize and organize cards within ten minutes of using the website for the first time.

5.5 Business Rules

As operating principles for the product, the following must be employed during the entire life of the application:

- User created data shall be achievable, not immediately deletable. Data that has been archived will be permanently deleted after 7 days.
- Administrators shall not have permissions to manually alter data in production.
- Any changes to production data must be scripted and reviewed before deployed to production.
- Developers must have their code peer-reviewed before merging and deploying.

6. Other Requirements

The MIT license and copyright shall be included in all copies of work.

Appendix A: Glossary

Term	Definition
AWS Elastic Beanstalk	An auto-scaling, load-balanced virtual server that executes and maintains the life of any deployed application without the need for developers to configure an operating system.
Amazon S3 (Amazon Simple Storage Service)	A web-based cloud storage service for backing up and archiving data
Bootstrap	A CSS framework which contains Javascript and CSS design templates
CSS (Cascading Style Sheets)	A style sheet language used for describing the presentation of a document written in a markup language like HTML.
Express	A NodeJS web application framework
GCP (Google Cloud Platform)	A suite of cloud computing services offered by Google that runs on the same infrastructure that Google uses internally for its end-user products.
HTML (Hypertext Markup Language)	The standard markup language for documents designed to be displayed in a web browser.
HTTPS (Hypertext Transfer Protocol Secure)	A secure version of HTTP which encrypts and authenticates messages using SSS/TLS protocol
Javascript	The object-orientated programming language that Mirai is programmed in.
NodeJS	An open-source, cross-platform, JavaScript runtime environment that executes JavaScript code outside of a browser.
MIT license	A short, permissive software license which enables the use of Google Sign in and Express
MongoDB	A cross-platform document-oriented database program.
MVC (Model View Controller)	A design pattern for the application which consists of a model, view, and a controller which acts on both the model and the view.
Sass (Syntactically Awesome Style Sheets)	A style sheet language that is an extension of CSS and enables you to employ variables, nested rules, inline imports and more.
UI (User interface)	Contains visual elements that provide human-computer interaction