Project Management Application

High-Level Design (HLD) Document

Version 1.2

July 22, 2020

Prepared By Group 2

Tulin Akbulut

Alex Alibrandi

Michael Jaouhari

Matt Merle

Brandon Miracle

Melissa Ross

Revision Tables

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
1.0	M. Ross B. Miracle A. Alibrandi	Outline and initial descriptions	June 22, 2020
1.1	M. Ross A. Alibrandi T. Akbulut M. Jaouhari M. Merle	Detailed descriptions, diagrams	June 30, 2020
1.2	A. Alibrandi M. Ross M. Jaouhari	Final Draft	July 22, 2020

Review and Approval

Approving Party	Version Approved	Signature	Date
Change Review Board	1.0	RBM	June 23, 2020
Change Review Board	1.1	MMR	July 1, 2020
Change Review Board	1.2	RBM	July 22, 2020

High-Level Design Document Review History

Reviewer	Version Reviewed	Signature	Date
B. Miracle	1.0	RBM	June 28, 2020
M. Ross	1.1	MMR	July 1, 2020
B. Miracle	1.2	RBM	July 22, 2020

Table of Contents

1. Introduction

Saga is a project management application that will streamline the development process of any product. The application will allow teams to collaborate and keep track of their projects.

1.1 Purpose

The purpose of this high-level design document is to add to the current project description the necessary detail and structure to present a suitable model for development. This document is also intended to help detect contradictions, misunderstandings, and design flaws before beginning development. This document should be used as a reference point and source of context regarding how the application's systems and modules interact at a high level.

1.2 Scope of this Document

This document presents the structure of the application's systems, with an emphasis on such topics as:

- Database architecture
- Application architecture layers
- Data flow
- Interfaces (External and Internal)
- Security systems

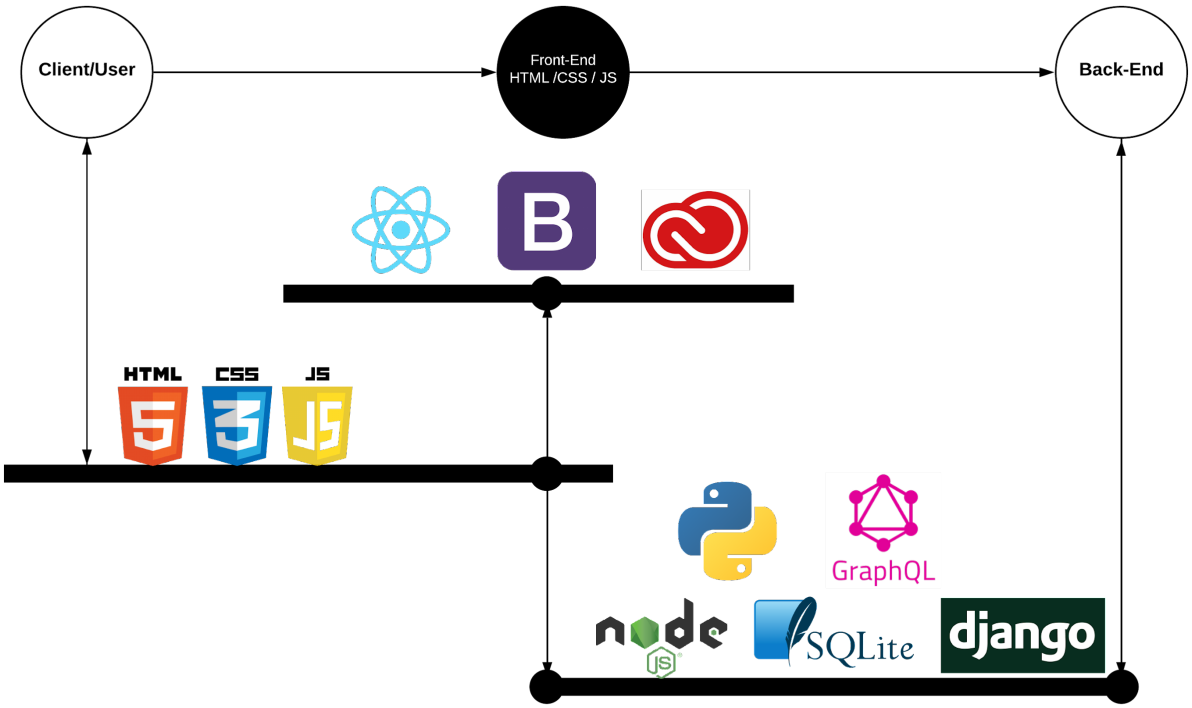
This document uses terms which range from non-technical to mildly-technical and should be understandable to both the developers and administrators of the application's systems. It will discuss all aspects of the application's design and define those aspects in detail. This includes, but is not limited to, describing the application's:

- User interface
- Hardware and software interfaces
- Performance requirements
- Design features
- Architecture
- Non-functional attributes

1.3 Definitions, Acronyms and Abbreviations

- API: application programming interface
- Adobe CC: Adobe Creative Cloud
- CRB: change review board
- CSRF: cross-site request forgery
- CSS: cascading style sheets
- DDoS: distributed denial-of-service attack
- Django: Python-based free and open-source web framework
- GraphQL: an open-source query language for APIs
- HLD: high-level design
- HTML: hypertext markup language
- HTTP: hypertext transfer protocol
- IDE: integrated development environment
- JSON: JavaScript object notation
- Node: an open-source, cross-platform, JavaScript runtime environment that executes JavaScript code outside a web browser
- PBKDF2: password-based key derivation function 2
- Python: a high level programming language
- QA: quality assurance
- RDBMS: relational database management system
- REST: representational state transfer
- Selenium: an open-source automation tool that is commonly used for testing
- SMTP: simple mail transfer protocol
- SPA: single page application
- SQL: structured query language
- TCP: transmission control protocol
- TLS: transport layer security
- UI: user interface
- URL: uniform resource locator

2. Design Summary



2.1 Overview of Design

This application will consist of several different components, using appropriate programming languages to meet each component's requirements and purpose. JavaScript will be primarily used to display and implement the application's user interface. SQL will be used to retrieve from, insert to, delete from, and update the application's database. Python will be used to submit SQL commands and facilitate back-end interactions with the Django framework, which will assist with the automated portions of the application's functionality as well as its administrative interface.

There will be three types of users with differing levels of access to the application: Site Administrators, Organization Admins, and Organization Members.

The application will be built on a client-server architecture, with the user's web browser being responsible for client-side interactions and the web server being responsible for back-end activities such as file hosting, communications with the database, and interactions with other web services. Site Administrators will interact with the application via the built-in Django admin interface, as well as the application's public user interface.

The core functions of the application will be organization management, project task management, and project progress reporting/measurement.

2.2 Design Issues

During the design and development of Saga, many design decisions, conflicts, and other issues have risen to the team's awareness and have needed to be addressed. The team's priority in making decisions on these issues is first to enhance the user experience where possible and second to choose options that work best with the developers available on the team.

2.2.1 Software Localization

As development progresses, we understand the need for our application to be accessible to users all over the globe. We realize not everyone speaks English.

One way we worked around this issue was factoring in language enabling into our user interface. We would program each corresponding letter of the alphabet to the related character in the user's designated language.

Another localization issue that we have come across in the development process is currency conversion. Since Saga is based within the United States our preferred payment currency will be the American dollar. We acknowledge, however, that not everyone who uses our application will use this currency, so we will also accept payments in all major global currencies.

2.2.2 Model-View-Controller vs. Single Page Application

Python and Django will be driving the back-end of Saga, and were chosen for their feature richness, strong data libraries, and robust developer communities. They tend to be used in Model-View-Controller web applications, especially Django in its most common use case out-of-the-box. Saga, however, will also be developed as a Single Page Application; this design choice was made to ensure exceptional performance and a high-quality user experience. This will present development challenges, but should still be achievable by dissociating Python/Django code from controlling any of the front-end functionality. There are three main options for addressing this issue:

1. Run separate servers for Django back-end and Node back-end. This is not ideal for the reasons of server costs and maintenance of the application.

2. Utilize a serverless architecture and only pay for functions that are triggered by the client-side, querying the database from that point.
3. Run a Django back-end server that responds to the application's front-end requests.

Our approach will be the third of these options. The front-end application will run on users' web browsers. All front-end requests and logic will be handled in the browser with a React JavaScript application.

We will need an API that will be used for communication between the Python back-end and JavaScript front-end. There are two options for that: REST and GraphQL. We will use REST due to familiarity among the development team. Additional reasons for choosing REST can be found in **3.2.2**.

2.2.3 Conversion to Mobile Application

During the development process we made the decision to expand Saga from a web application to both a web and mobile application. We understand the conversion could provide an enhancement of functionality, improve on our existing capabilities, and increase our lead on the competition.

As we discussed the future of Saga we came to the realization that the future is only a few steps ahead. We have decided to focus on the creation of a flawless web application that runs properly. During a future update to Saga's features we will implement mobile application development. Converting to mobile use would be a great impact in the future so that we can have our clients take their work on the go anywhere they would like. This would open the market for business in general as more people have access to smartphones and tablets than computers. With the help of React's cross-platform development options, fragmented modules and code overlays the transition from a web page to a mobile app will be much easier.

2.3 Tools

2.3.1 Code Editors

Javascript and HTML code needed for this application will be written using Visual Studio Code, in combination with text editors such as Atom and Sublime Text.

2.3.2 IDEs

The Python code (including Django implementation) for this application will be developed within the PyCharm IDE, in combination with text editors such as Atom and Sublime Text. Various aspects of development may require use of the Command Line Prompt, especially in testing and validation for Django implementation.

2.3.3 RDBMS

The application will make use of SQLite as its relational database management system, due to its lightweight nature and straightforward integration with Python and Django.

2.3.3 Graphic Design Tools

Adobe Creative Cloud (specifically Photoshop CC) will be utilized for both design mockups and all graphic design required for the development of this application. Other Adobe CC applications may need to be used for smaller, more niche tasks, such as InDesign for project documentation needs or Illustrator for vector graphics or Dreamweaver for quick UI element mockups.

2.3.4 Collaboration

A variety of team collaboration tools will be used in the development of this application. Slack, Trello, Jira, Google G Suite, Lucidchart, and Github will be the primary collaborative tools used. For a detailed examination of collaboration tools and how they will be implemented, see Appendix 4.1.

2.4 Frameworks

2.4.1 Django

Since a portion of the application will be written with Python, Saga will be using Django as a framework for Saga's back-end development. Django will respond to Saga's front-end application requests which we feel will create a smooth interaction between the server and the users machine.

2.4.2 React

When it comes to Saga's front-end development we have decided to use React, the ideal JavaScript library for building our user interface. Our React code will interact with the Django server via our REST API. With this combination of Django and React frameworks we believe we have created a flawless user interface experience.

2.5 Libraries

2.5.1 SQLite

SQLite is a RDBMS contained within a C-language library; it implements a lightweight, fast, self-contained, high-reliability, full-featured, SQL database engine. This application will use SQLite 3, the most recent and supported version of SQLite. It will need to be included in the development environment within PyCharm.

2.5.2 Bootstrap

Bootstrap is a library of web interface options for HTML, CSS, and JavaScript. This library will be used to efficiently design the user interface of our application by providing different options for layout components, buttons, fonts, and color schemes.

3. Design Details

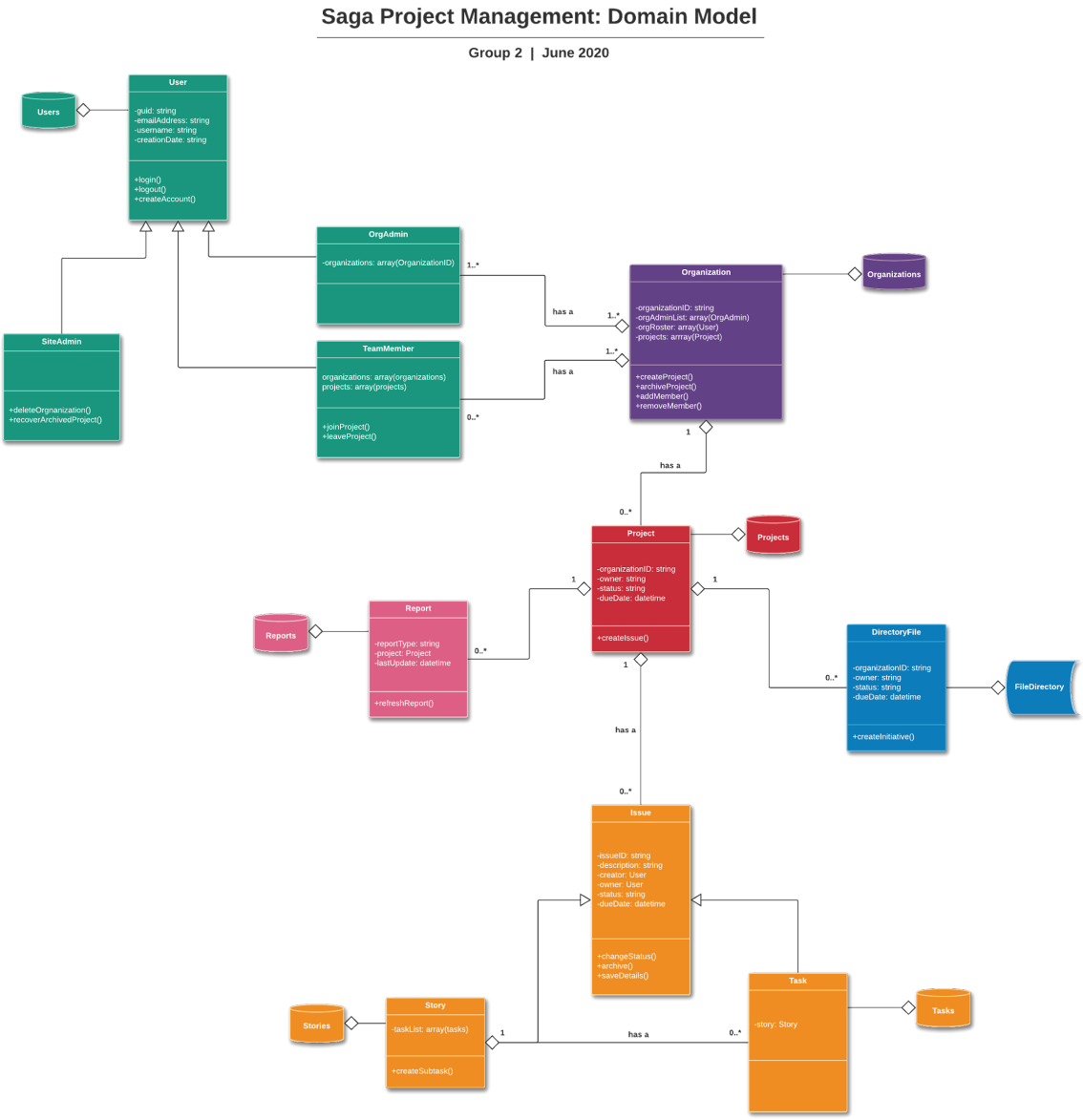
3.1 Top-Level

3.1.1 Domain Model

The core functional components of the application are broken into a few major databases that help to define how Saga's system communicates and stores data. These databases include Saga's Users database table which will hold the User data for users of type User, SiteAdmin, and OrgAdmin. Additionally, the Organization table will store all of Saga's Organization-related data. The Projects database table will contain all data about Projects. The Issues database will contain all project issues (Tasks and Stories) linked to projects.

The classes for SiteAdmin and OrgAdmin each inherit from the parent class User. Story and Task classes inherit from the parent class Issue.

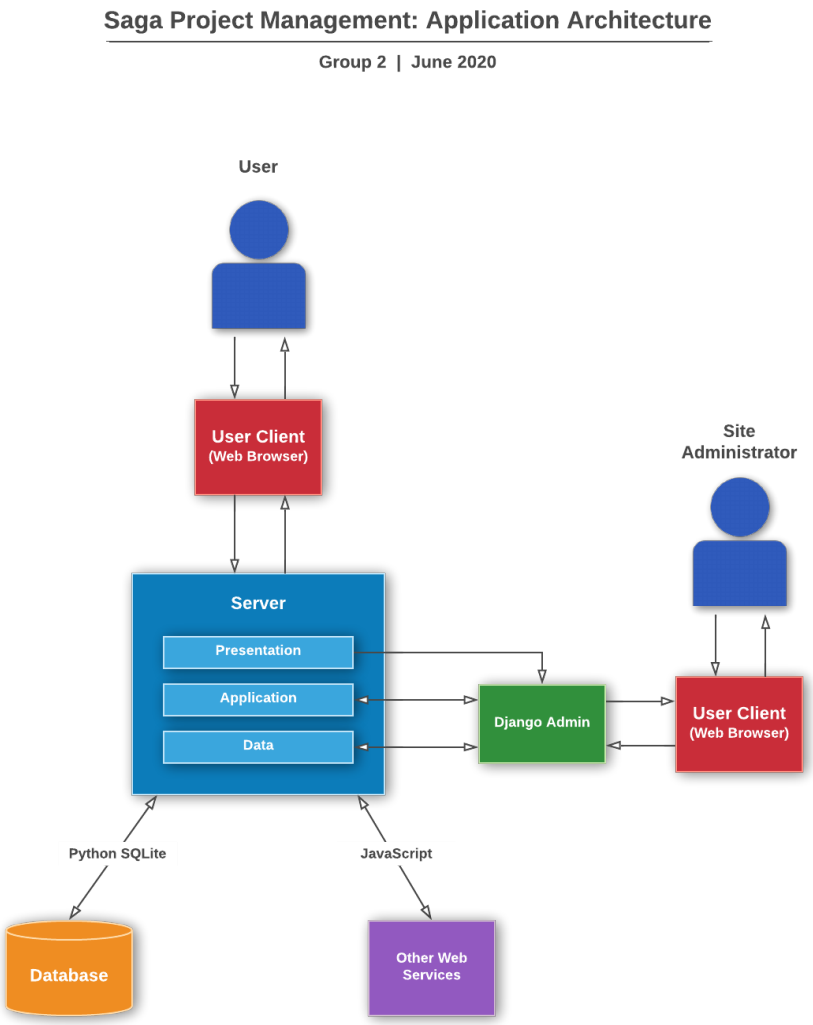
Shared files will be stored on the server and accessed from that point.


Saga Project Management: Domain Model
Group 2 | June 2020
User
-guid: string
-emailAddress: string
-username: string
-creationDate: string
+login()
+logout()
+createAccount()
Users
OrgAdmin
-organizations: array(OrganizationID)
SiteAdmin
+deleteOrgnanization()
+recoverArchivedProject()
TeamMember
organizations: array(organizations)
projects: array(projects)
+joinProject()
+leaveProject()
Organization
-organizationID: string
-orgAdminList: array(OrgAdmin)
-orgRoster: array(User)
-projects: array(Project)
+createProject()
+archiveProject()
+addMember()
+removeMember()
Organizations
has a
1..*
0..*
1
Project
-organizationID: string
-owner: string
-status: string
-dueDate: datetime
+createIssue()
Projects
Report
-reportType: string
-project: Project
-lastUpdate: datetime
+refreshReport()
Reports
DirectoryFile
-organizationID: string
-owner: string
-status: string
-dueDate: datetime
+createInitiative()
FileDirectory
Issue
-issueID: string
-description: string
-creator: User
-owner: User
-status: string
-dueDate: datetime
+changeStatus()
+archive()
+saveDetails()
Story
taskList: array(tasks)
+createSubtask()
Stories
Task
-story: Story
Tasks

3.2 Architecture

3.2.1 Client-Server

The application will be built on a client-server architecture. Users will interact with the application via their web browser, which will send and receive information from the web server. This web server will be designed on a standard Presentation-Application-Data layer architecture. The server will handle communications with the database and any other web services needed. Simultaneously, the server will handle interactions with the Django Admin interface, which allows site administrators to view, manage, and update site data.



3.2.2 REST

When it comes to Saga's architecture we chose REST API for a few reasons. REST API'S layered systems will help make Saga a more scalable and modular application. Also, REST's stateless client server protocol allows each HTTP to contain all the necessary information to run on it.This makes it so neither the client nor the server need to remember any previous state to satisfy it.

3.2.3 Single Page Application

Saga will work off of a Single Page Application architecture. Using this architecture will benefit our application by speeding up load times and improving the overall quality of interacting with the application. Considering Saga is an app to allow for effective collaboration between team members, integrating SPA will be crucial in giving teams the ability to communicate and interact with the application in real-time. Also, using SPA architecture with Saga will give the business a competitive edge on other applications. The project management tool Trello uses SPA architecture as well, so using it will give us the opportunity to provide an equal or better user experience.

3.2.4 User Types

There will be three main types of users who interact with Saga, each with different levels of permissions and access.

3.2.4.1 Users

Users are the base users of the application who can manage their user account and update their profile. They are team members who will only have access to manage tasks within the projects to which they're assigned by their Organization Admin(s).

3.2.4.2 Organization Admins

Organization Admins are team/project leaders who can fully manage their organization, projects, rosters, reports, and other related aspects. The Organization Admin will also inherit all the permissions associated with Users.

3.2.4.3 Site Administrators

Site Administrators (Site Admins) are application developers and operational administrators who have nearly-full access to the application's database through use of the Django admin

interface; they can directly view and alter organization data, user data, organization roster data, project data, tasks, and other objects associated with the application. Site Admins have similar permissions as Organization Admins, and they also inherit all permissions from Users.

3.2.5 Subscription Types

The application will feature both a free-to-use model and a paid model, both dictated at the Organization level.

Members and Admins of free-to-use subscribing organizations will have a limit to the number of projects that can belong to their organization and will also have advertisements displayed within the application.

Members and Admins of paid subscription organizations will have the ability to create and manage an unlimited number of projects and have no ads displayed.

3.2.6 System Integration

The system integration that best fits Saga is star integration. It interconnects each separate system to its designated subsystems. We believe with star integration we will be able to quickly implement new features to the application while still keeping a flexible deployment schedule with minimal effects to existing features and core functionality.

3.3 Hardware

3.3.1 Server Structure

For Saga's server structure we have decided to utilize cloud-based servers. The scalability of cloud-based servers will help ensure Saga is always in good hands regarding the flexibility of upgrades and advancements. Cloud-based servers will also always insure Saga's business continuity by ensuring that all information is backed up and protected in a safe location. Also the flexibility of cloud-based servers will allow the ability to access automatic updates, so Saga's IT requirements will always be up-to-date.

3.4 External Interfaces

3.4.1 Communication Interfaces

The main external communication interface of this application will be e-mail, used for sending notifications, account recovery, and account confirmation messages to users, among other potential messages.

The application will use Simple Mail Transfer Protocol (SMTP) as its e-mail protocol, implemented primarily on the server through Python code. SMTP specifies the rules, procedures, and components of e-mail messages, as well as how to send one in a standard format that will be interpreted by any other email server in a platform-agnostic manner. Because of SMTP's ubiquity, we can rely on even legacy devices and software being able to handle SMTP.

3.4.2 Web Sockets for Client Communication

The application will utilize Web Sockets to communicate notifications between the client and the server. Web Sockets is a bi-directional protocol with full-duplex communication that requires only one TCP connection. Once a connection is established through an initial handshake between the client and the server, the connection remains open. This will allow users to view and make real-time updates to projects and communicate with other users.

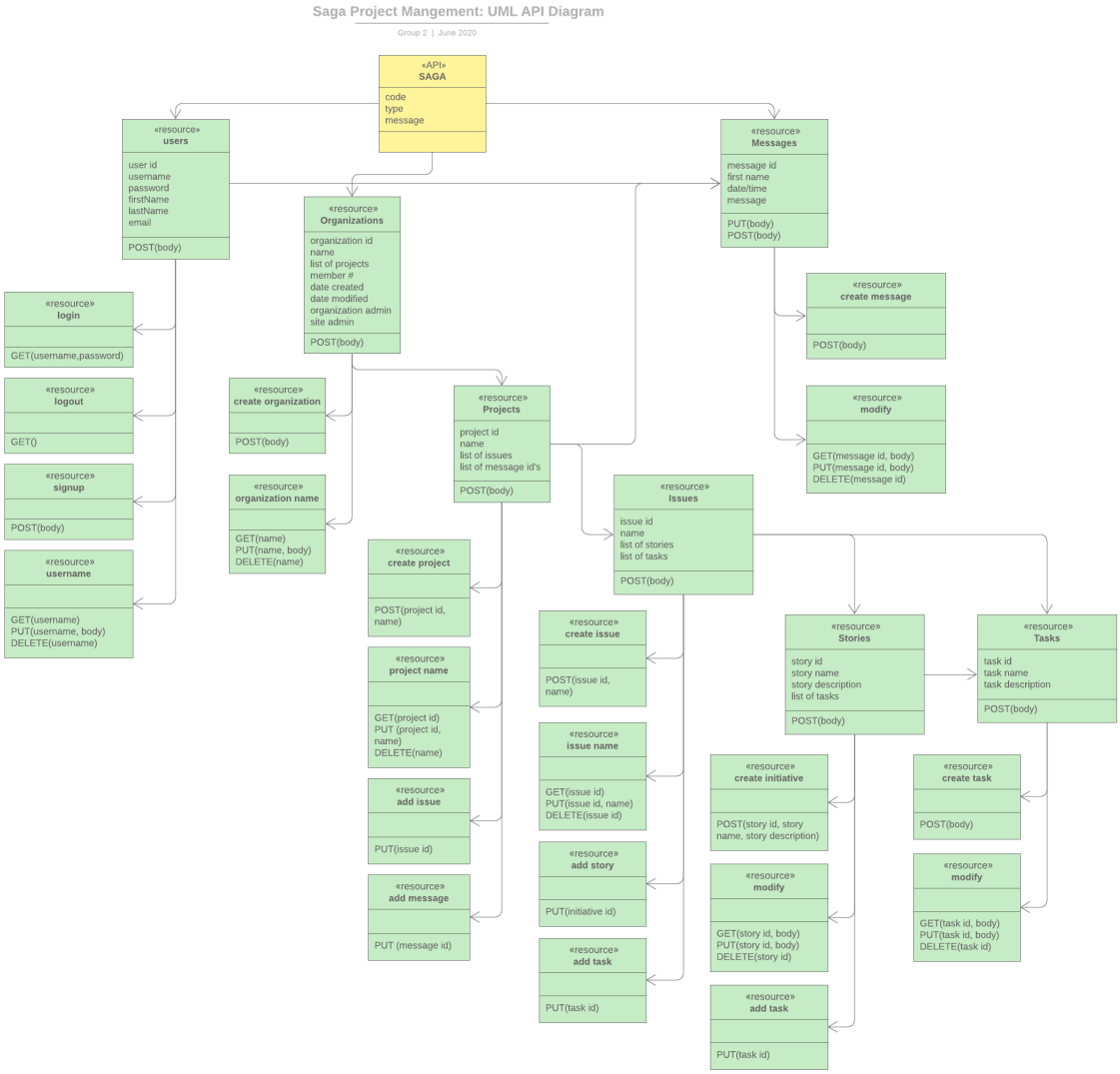
3.4.3 Network Communication Interfaces

Saga's network communication between the application and servers will be encrypted by TLS encryption. We chose this method due to its additional protection against data breaches and DDoS attacks.

We will facilitate our encrypted connection between the web browser and Saga servers through the use of Organization Validation TLS certificates.

3.4.4 APIs

We will utilize a REST API to facilitate internal communication between the front-end and back-end server/database portions of the application. The communication from the REST operations will result in developing the correct endpoint or route that the URL is requesting. This request would be sent from the user or admin. REST API will use HTTP requests to retrieve resources, change the state of or update resources, create resources, and finally delete resources. It will work well with Saga due to its exceptional relationship with cloud-based applications.



Saga's development team also realizes that more advanced API's may come to light in time. With that being said, we are open to the idea of change and the possibility of additional API's joining the Saga application.

3.4.5 Third-Party Integrations

Saga understands the necessity and importance of third-party API integrations. These integrations attract and retain customers while also streamlining users' work and lives. As of right now, Saga is focused on making sure the core functions of the application work flawlessly within each other, but plan to add third-party integrations as a feature at a later point in time post-deployment.

3.5 Internal Interfaces

3.5.1 Site Admin Interface

This application will utilize Django's built-in site administration tools; this will allow us to save development time instead of designing and developing proprietary tools and ensure that we're using industry-standard, well-supported administrative tooling.

The site admin interface will allow administrative users to manage and operate on the application's stored data, as well as assist users with problems they may encounter.

3.6 User Interfaces

The application will feature a top-navigation bar with important button links to other parts of the interface and overall site. This bar will also have buttons for sign-in/sign-out and settings. It will also contain a search bar within it.

The application will also have a footer at the bottom of the site at all times. This footer will contain important links to other parts of the application that are not directly part of its core functionality (user support, user knowledge base, contact us, ect), as well as all relevant legal information regarding copyright and trademark.

All user interfaces will adhere strictly to the Saga Style Guide in terms of functionality, styles, and layouts. The Saga Style Guide will be fully written before development of the application begins.

3.6.1 User Login and Account Creation

This prompt will first ask users if they would like to sign into an existing Saga account or create a new account. The left half of the prompt will contain the form for creating a new account, while the right half will contain the form for logging into an existing account.

Signing into an existing account will require a username/e-mail address and password. There will be a clickable button for “Sign In” and a link to login assistance for forgotten usernames and passwords.

Choosing to create a new account will ask the user for an e-mail address, unique username, password, and password confirmation. There will be a clickable button for “Create Account”.

Clicking either of those buttons to submit this form for signing in or creating an account will validate the user’s information to ensure that it is correct and meets requirements, then sign the user into their account.

Creating a new account will then prompt the user again to choose whether they would like to join an existing organization or create a new one. Choosing to join an existing organization will allow the user to search for organizations on Saga and request to join them. Choosing to create a new organization will take the user to the Organization Creation interface.

3.6.2 Organization Creation

When a user selects to create a new Organization (either at the time they first create their account or at a later point in time), they will be presented with the Organization Creation interface.

This interface will feature a web form that asks the user to fill in the following basic details of the Organization they wish to create (required fields marked by *):

- Organization Name *
- Organization Description
- Organization Admin(s) *
- Organization Contact Info
    - Contact Name(s)
    - Street Address
    - Phone Number *
    - E-mail Address *

The interface will also allow the user to add team members via e-mail invite or searching for existing users’ usernames directly.

Last, the interface will prompt the user to select a subscription type (free or paid) for their Organization.

At the bottom of this interface will be a clickable “Create Organization” button. Clicking it will cause the application to validate the information that the user input, then submit the form. If the user selected a paid subscription for their organization above, this interface will close and open the Payment Module interface. Otherwise, the user will be directed to their Organization Hub interface to begin managing their Organization and its projects.

3.6.3 User Profile

Each individual user will have their own unique user profile page. This page will feature details about the user including name, organization membership, projects they’re working on, and other details that the user may choose to provide.

The user profile will feature a default avatar picture which a user can choose to override with their own uploaded image.

The profile will have fields for a variety of information about the user, including:

- Public Name
- Occupation
- Organization(s)
- Project(s)
- Location

Each of these fields can be disabled to display nothing if the user prefers by pushing the designated disable field button.

3.6.4 User Account Settings

Each user can access an account settings page that offers a number of options for account-level configuration. The settings will be broken down into groups of related settings, including:

- User Profile
    - Edit Profile
    - Public/Private
- Account Security
    - Password Change
    - Recovery Mobile Number
- Personal Details
    - Name

    - E-Mail Address
    - Address
    - Phone Number
- Account Management
    - Delete/Disable Account
    - Change Username

3.6.5 Organization Hub

Each organization will have its own hub interface which acts as a central location for all of its projects, high-level reporting, and organization management. Organization management includes managing various projects' rosters, creating and updating projects, and project planning resources.

The main body of the organization hub will be focused primarily on reporting, including updated statuses on individual projects and metrics about the organization's projects as a whole. There will be links to the individual projects that allow for quick navigation into their Project Home interfaces.

3.6.6 Organization Settings

Organization Admins can access their Organization's Settings interface which offers a number of options for organization-level configuration and saved data. The settings will be broken down into groups of related settings, including:

- Subscription Management
    - Subscription Type (Free, Paid)
    - Payment Method
    - Subscription Schedule (Monthly, Yearly)
- Organization Admins
    - Admin Roster
- Organization Information
    - Organization Name
    - Organization Address
    - Organization Contact(s)
- Organization Management
    - Delete/Disable Organization

3.6.7 Payment Module

The payment module interface will be displayed anytime an Organization Admin creates a new organization and selects a paid subscription or chooses to alter their organization's payment method.

The user can choose their payment method (credit card, debit card, PayPal).

If choosing a credit or debit card payment method, the form will contain fields for card number, cardholder name, card expiration date, card security code, and the zip code in which the card is registered.

If choosing PayPal payment method, the form will contain fields that prompt the user for their PayPal login information (username, password).

The prompt will have a "Continue" button which will cause the form to submit. This will first validate the input information, then save it to the user's Organization. If there are any fields that do not pass validation, the form will reject the submission and prompt the user to correct the information.

If the user is making a payment via this page, the application will then begin the billing process. If the user is updating their existing payment method, it will not bill them.

3.6.8 Project Home

Every project will have its own Project Home interface, which displays key metrics and basic information about the project as a whole.

Project metrics will be displayed above the other columns in a section that spans the width of the page.

Any issues that belong to this project which have a high priority will be displayed on the left-hand side of this interface in a list containing the issue ID, issue title, and priority. Clicking on an issue from this section will take you to the Project Board interface and open the Issue Detail View for that issue.

Any issues that belong to this project which are past their assigned due date will be displayed in the center column of this interface in a list containing issue ID, issue title, and due date. Clicking on an issue from this section will take you to the Project Board interface and open the Issue Detail View for that issue.

The roster of Organization Admins and Members who are assigned to a project will be displayed on the right-hand side of this interface in a list. Clicking on a user's name from this roster will open their User Profile.

3.6.9 Project Board

A project's Project Board interface will be the primary method of task management for that project. On the Project Board, users can create stories and tasks, then manipulate those objects through the interface in multiple ways. This interface will feature a Kanban-style task board, a clickable "New Issue" button for creating new issues, and the ability to double-click on an existing issue to view/edit its details.

3.6.9.1 Project Kanban

The Project Board's Kanban-style board is displayed in a four-column layout. The columns, from left to right, will be labeled "Unassigned", "Assigned for Development", "In Progress", and "Completed".

Issues (Stories and Tasks) will be displayed as cards in these four columns. Newly-created issues will always be filed under the "Unassigned" column if a team member is not assigned to the issue at time of creation. If a team member is assigned to an issue when it's created, the issue will be filed under "Assigned for Development".

Issues can be clicked-and-dragged between the four columns. Doing so automatically updates an issue's status to match the column to which it is dropped.

Issues can be double-clicked to open their Detail View.

Organization Admins will have an additional button displayed on an issue's card that allows them to archive an issue, removing it from the board.

3.6.9.2 Issue Creation

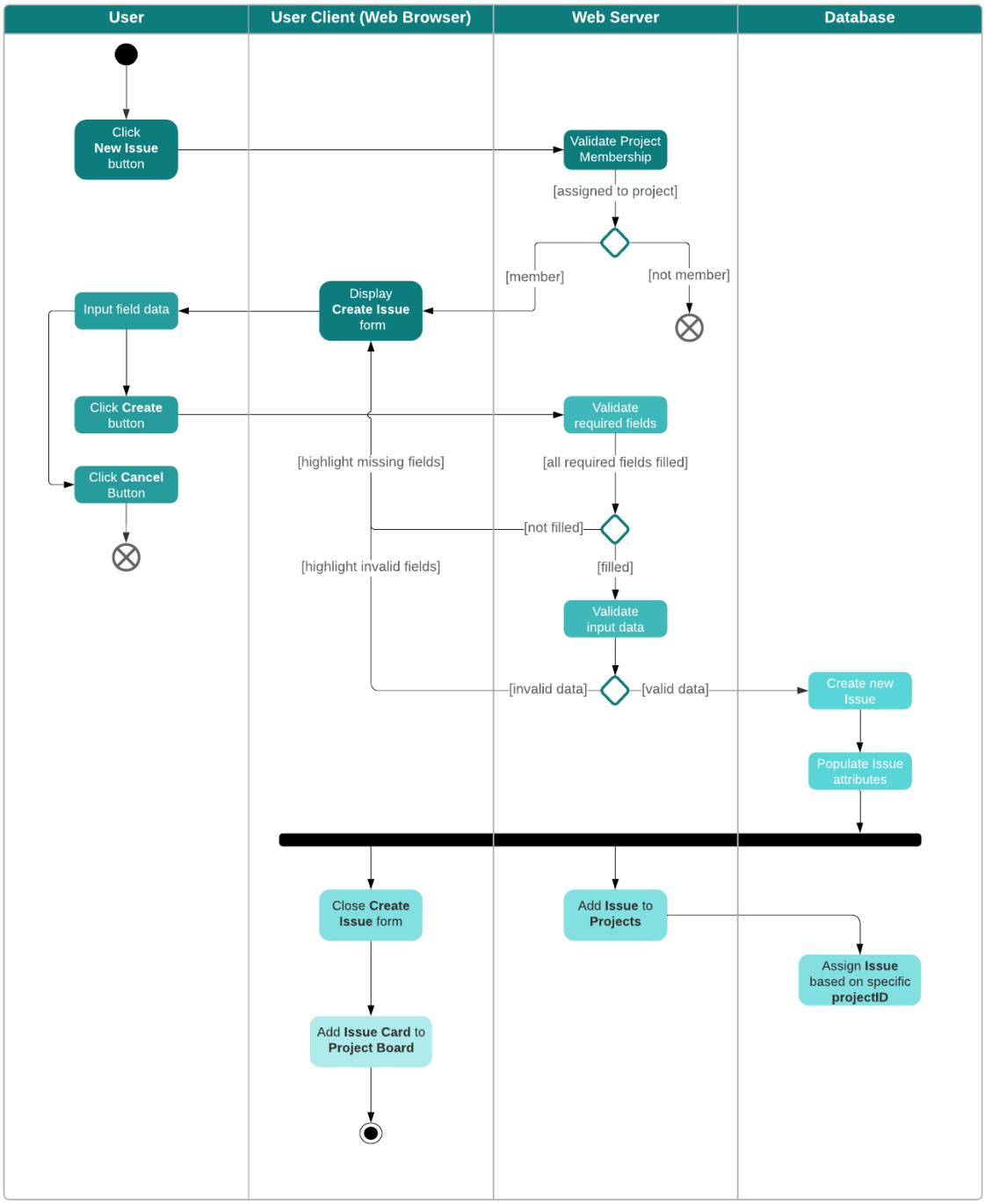
Clicking on the "New Issue" button within the Project Board page will open an Issue Creation form, which is similar to the Issue Detail View, but blank and ready to accept user input in all fields. The Issue Creation form will have a clickable "Create" button at the bottom of the form to save the issue and its details; this will first validate for required fields (marked by * below)

The fields of this form include:

- Issue Title (Short Text) *
- Issue Type (Drop-Down Selection, Task or Story) *
    - If type is Task, then additional field for which Story it's associated with
- Issue Due Date (Datetime Selection)
- Issue Description (Long Text)
- Issue Tags (Comma-Separated Text)
- Issue Comments (Long Text)
- Issue Priority (Drop-Down Selection, 1-5) *
- Issue Status (Drop-Down Selection, Issue Statuses, Default "Unassigned")
- Issue Reporter (Drop-Down Selection, Organization Members Assigned To Project, Default User Creating Issue) *

Activity Diagram: Issue Creation

Group 2 | July 2020



3.6.9.3 Issue Detail View

Double-clicking on an issue's card on the Project Board will open its Issue Detail View, a pop-up form with the same fields that are available in the Issue Creation form. These fields are all in the same layout as the Issue Creation form in the above section, but are input-disabled by default. A pencil icon button is displayed on the right side of each field, which will enable input on that specific field and allow a user to edit it.

At the bottom of this interface is a clickable "Save" button, which will allow the user to save the changes they have made to the issue.

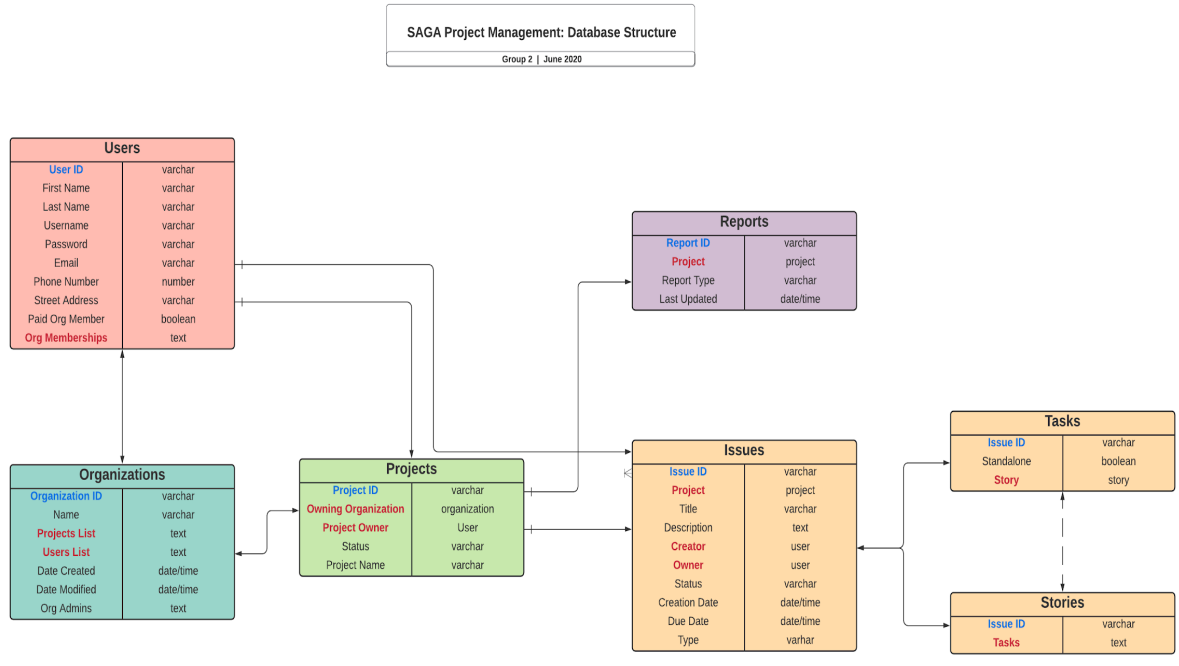
3.7 Databases

3.7.1 Database Implementation Description

This application's database will be implemented on the back-end to ensure that all data is consistently stored and can be easily retrieved when needed. It will do this with the use of its API, which will utilize REST operations to make data manipulation possible. The different entities within the application's database are connected based on the way they interact and rely on each other.

3.7.2 Entity Relationship Model

Saga's database will contain the entities Users, Organizations, Projects, Reports, Issues, Stories, and Tasks. The flow will begin with the Users table which is connected to the Organizations table. The Organizations table is connected to the Projects table which will connect to both the Reports and Issues table. The Issues table will then be connected to the Stories and Tasks tables. The Stories and Tasks tables are also connected, however, they only need that connection in certain cases. All tables are connected together based on how they will properly interact with one another within the application.

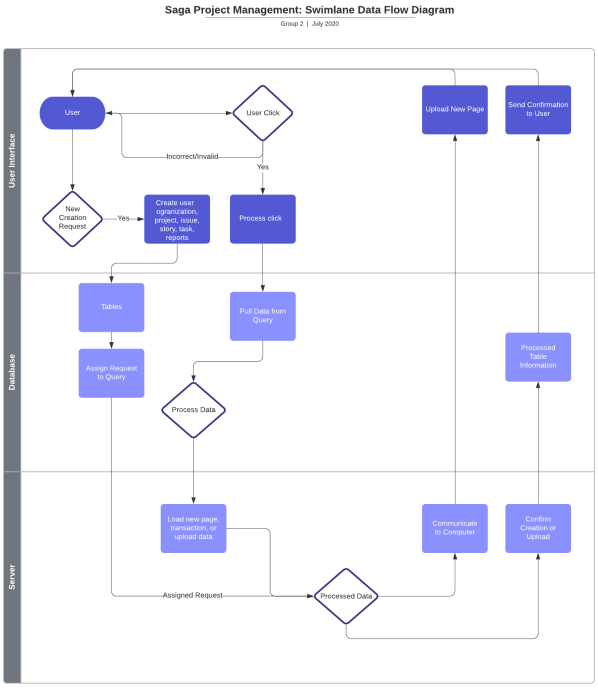


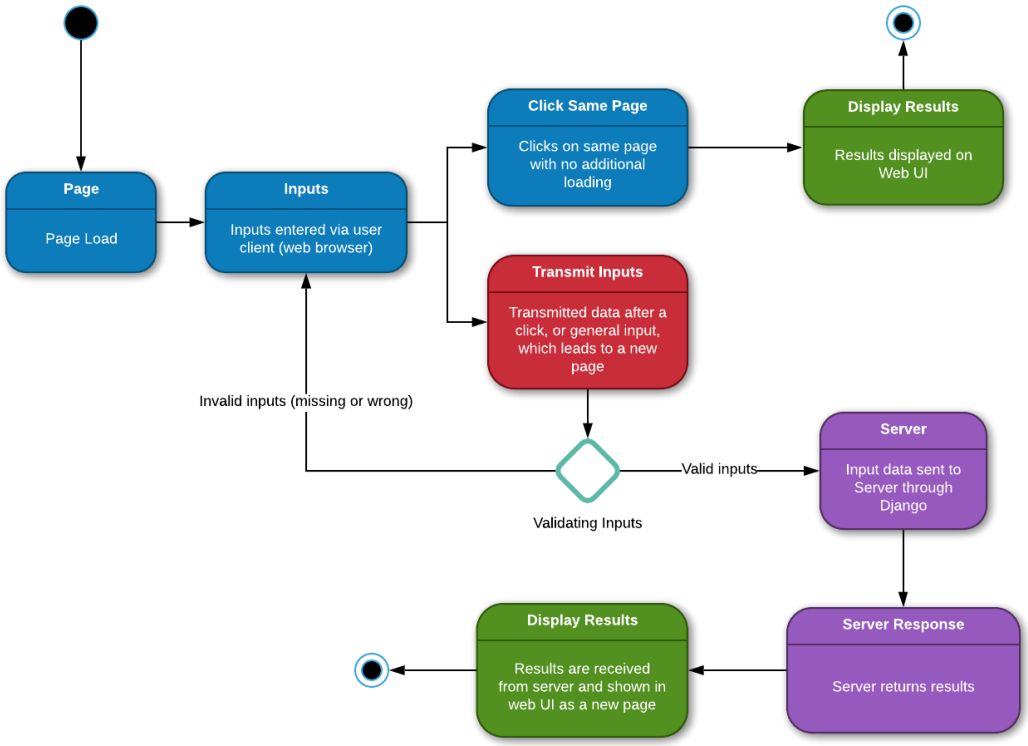
3.8 Data Flows and States

3.8.1 Description

The diagram below represents the flow of data between both user and server. With new creation, or possibly change requests, the data will be sent from the user, passed through the database for necessary modifications or data retrieval, and then processed through the server. The server will then send confirmation of processing back to the user, again passing through the database for validation and, potentially, modification. With any user click requests, the request will be processed and sent to the server, first going through the database to retrieve any necessary data. After processing through the server, the server will send back a new or updated page to the user.

Below the data flow diagram is a high-level diagram, broadly depicting the interaction between a user input and server. We show a start and end point in this diagram to demonstrate the interactions between the user and server, in order to generate an uploaded or new page. If there are invalid inputs from the user side the program will not upload the page, and it will loop back around to the user to awate a valid input.




Page
Page Load
Inputs
Inputs entered via user client (web browser)
Click Same Page
Clicks on same page with no additional loading
Display Results
Results displayed on Web UI
Transmit Inputs
Transmitted data after a click, or general input, which leads to a new page
Invalid inputs (missing or wrong)
Validating Inputs
Valid inputs
Server
Input data sent to Server through Django
Server Response
Server returns results
Display Results
Results are received from server and shown in web UI as a new page

3.9 Security

Security is an important aspect of developing a successful application. User information should be private and we strive to ensure this privacy.

3.9.1 TLS

All user passwords shall be encrypted over the TLS when connecting server to client.

3.9.2 Password Hashing

All passwords stored into the user database will be hashed using the PBKDF2 hashing function. Whenever users type in their password to login, the application will hash their input and see if it matches the hashed password within the database. From there, the application will either provide access if the hashed password matches or display a statement informing the user of their incorrect input.

3.9.3 Account Recovery Procedures

Upon creation the user will provide an email address to recover lost or forgotten passwords. The user will click the forgotten password link below the email and password text fields located at sign in. The user will then receive an email regarding a change of password. The user will then follow recovery instructions to reset their password.

3.9.4 Authentication Methods

Saga will offer a multiple factor authentication option where the user can provide a valid cellular number which is then stored within the User database as an integer. At login the user will type in their proper credentials and click the sign in button. Once the button is clicked the user will be prompted to insert a 6 digit pin number, the user will then receive a text message containing the number. The user will then enter it into the Pin code text field to gain access to Saga.

3.9.5 Device Logging

At Saga account creation the user will be asked to remember the device used to create their account. This device will be set as their default device. Once a new device is used to log into their account Saga will send the user an email notification stating a new device has been used to login to their account. The user will then open the email to confirm that the new device is indeed them or an unwanted user. Once the device has been logged, the web browser will save the information and store it as a cookie for future application use.

3.9.6 Attack Prevention

Saga developers will continuously test its database's strength by administering SQL injection vulnerability tests. There will also be in-depth black-box, white-box, dynamic, static, manual, and automated testing to insure safety from all attacks that may compromise the users information and Saga's system.

3.9.7 Account Inactivity Procedures

Saga's system will administer checks on user account inactivity. If a user's account has been inactive for a year or more, Saga will render this account inactive until the user follows the proper account re-enabling steps.

3.10 Configuration Data

Configuration data for Saga will be formatted using JSON and stored in a source configuration file. This file will include connection strings, preferences, and other information that are used to run the application. Some of the configuration settings, such as the connection strings to the database and other resources, are more application-based while others focus on user settings. Configuration pertaining to the user's screen size, date format, and time zone will be retrieved from the user's browser. Most configuration settings, such as pricing, reporting, and display details, can only be modified by Site Admins.

The application will also make use of browser cookies to store certain, limited pieces of user information that will enhance the overall user experience.

3.11 Reports

3.11.1 User-Facing Reporting

Saga offers different reporting options for users to track their organizations and projects. The organization reports will allow Organization Admins to view information about their organization, such as the number of users on their roster and the number of projects the organization is currently working on. The reports on projects will be useful for users to see how their projects are progressing. These reports will include information like the estimated task completion time, the actual task completion time, and the overall completion percentage. Project progress reports will also include standard Agile-style reporting mechanisms that measure a project's progress, such as burndown charts, key performance indicators, risk reports, variance reports, status reports, and resource reports.

3.11.2 Internal Reporting

This application will generate internal reports containing operational and financial information. Usage reports will be used to track the amount of time users spend on the application, what times they use the application, payment information, and organizations that use the application. These reports will allow us to identify usage trends, performance issues, and any unauthorized access to the application. The data collected in these reports will assist with further development and maintenance of the application.

3.12 Other Outputs

Saga will not currently generate any additional outputs during initial development, but if necessary, other outputs might be added in future versions of the application.

4. Verification and Validation Testing

4.1 Testing Overview

The Saga Test Plan describes the software testing strategy, goals, timing, forecasting and outcomes, and the resources required for testing.The Test Plan also helps us determine the effort required to verify the quality of the application being tested.

Saga will be tested during development and after deployment to ensure functionality, usability, interface performance, overall application performance, and security concerns are all met and working according to standards.

4.1.1 Automation

Saga will use Selenium, an open-source automation testing tool, for automation. We chose Selenium because of its exceptional framework and language support, its support across multiple operating systems and browser types, and its ease of implementation.

The functionality test will help the development team check database connectivity, all links on web pages, and forms used to send and/or receive information from the user.

The browser compatibility test will help to check whether the application is displayed correctly in different browsers.

These tests will be primarily written in Python, using the Selenium library.

4.1.2 Dedicated Testers

Once the application nears its deployment phase, the development team will require no fewer than four dedicated testers who will be led by the Quality Assurance Lead. These testers will execute manual test suites, as well as oversee all automated testing.

4.2 Test Schedule

The application will be tested regularly throughout the development life cycle, including tests of new components and thorough regression testing for existing components.

Each new feature, component, or release will have its test suite written by the Quality Assurance team in conjunction with the developer stakeholders that worked on it. The test

suite will be written and compiled no later than five days before a scheduled release is set to go live.

Development branches will undergo ongoing and regular testing by the Quality Assurance team. The production application will also undergo regular testing by the Quality Assurance team post-deployment to ensure the continuing quality of use for customers.

4.3 Test Tracking

Test cases are written in a common spreadsheet using Google Sheets. The owner of this spreadsheet will be the Quality Assurance Lead and it will be submitted to the entire Quality Assurance team and change review board.

These test cases will be organized into test suites that focus on a particular version, component, or piece of functionality.

Once approved by the QA Lead and CRB, all test packages will be converted into test plans in Jira, which will help fine-tune and monitor the repeated execution of the tests.

4.4 Defect Management

Defects found by members of the development team will be submitted immediately within Jira as issues that can be collaborated on and tracked by the team.

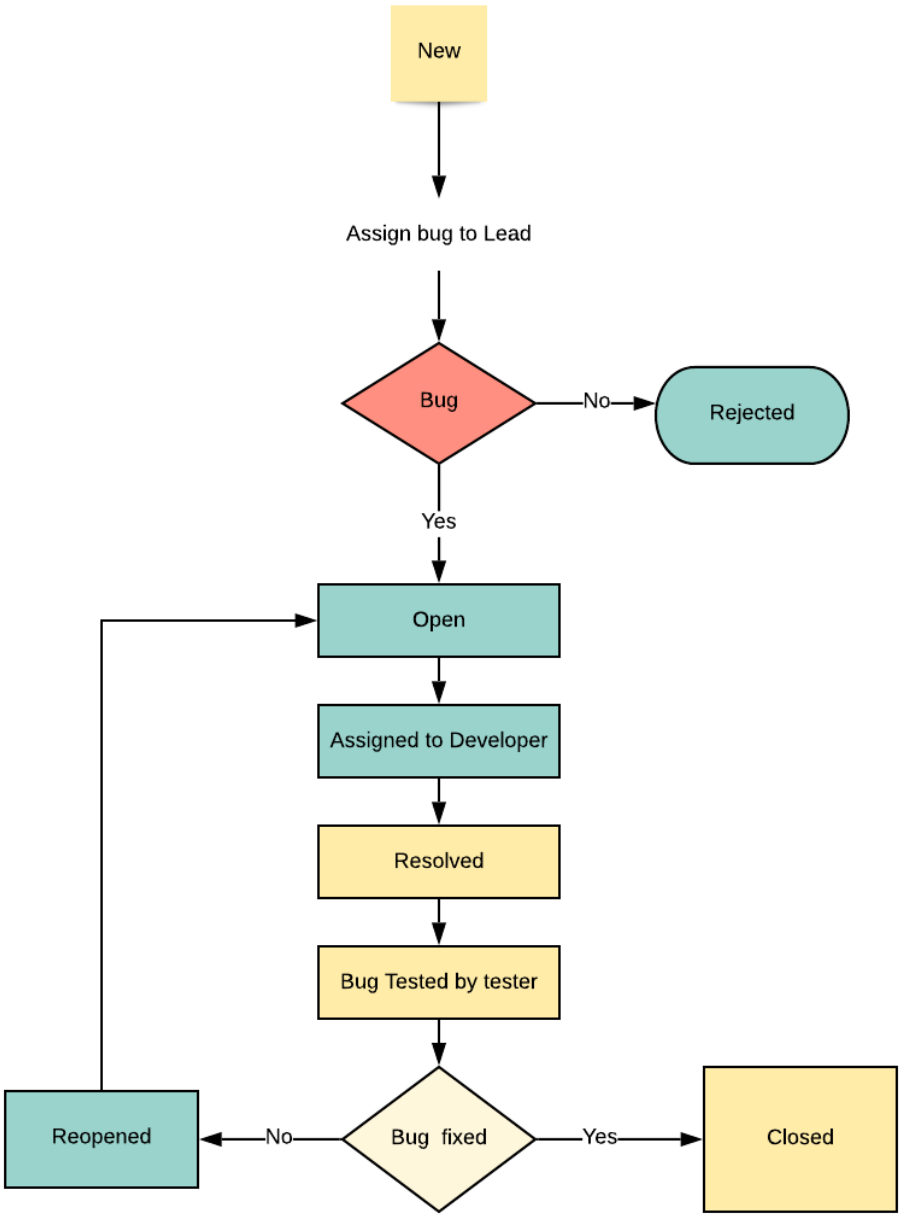
There will be a page for public users to submit defect reports via an online form.

The Quality Assurance team will evaluate and validate the existence of all reported defects, including logging all repeatable steps to reproduce the defect. They will assign each defect with a priority based on severity and impact to user experience, as well as provide all relevant details possible.

Defects with a high priority (1 or 2) will be brought to the attention of all team Leads, who will discuss the issue and ensure that its investigation and repairs are prioritized as immediately necessary. They will designate a Lead or other senior developer as the owner of that issue and delegate its progress to them, expecting regular reporting on it and a timely fix if possible. Any resources that need shifted to fix a high-priority defect will be determined by the issue owner and the Project Manager.

Defects with a low priority (3 or 4) will be logged in a persistent collaborative spreadsheet; any that remain unfixed until the following Weekly Triage Meeting will be discussed and prioritized for investigation and repair during that Triage Meeting.

Known issues that affect users will be posted to a publicly-available roadmap, along with relevant status updates on those defects.



5. Appendix

5.1 Collaboration Tools

5.1.1 Communication Collaboration

- Team communication will be facilitated primarily by **Slack**, which allows for discussions and simple file sharing. It should not be used to share project files, but rather to share quick mock-ups and links to other collaborative tooling where the project's actual files are being stored.
- Meetings will be conducted using **Google Meet** (and its interface with Google Calendar) for teleconferencing. All developers should have a Google account which is known to their lead for scheduling and conferencing purposes.

5.1.2 Documentation Collaboration

- Google's **G Suite** will be used as the primary hub for all documentation. Google Docs will be used to write documents. Google Slides will be used to prepare needed slide decks and basic diagrams for internal use. Google Sheets will be utilized for creating necessary spreadsheets. Google Forms will be lightly used to facilitate communication, such as allowing developers to submit document change requests.
- **Lucidchart** will be used for more finalized diagram creation, such as those within this document and any outward-facing documents.

5.1.3 Version Control

- The project's development files - primarily code and art - will be stored within a **Github** repository for version control purposes. This will allow for controlled pull requests, development branches, and control over the master branch.

5.1.4 Project Management

- Project tasks and issues will be tracked through **Jira**, making use of its Kanban board functionality to maintain status updates on a variety of issues.
- Project scheduling will largely be maintained through a common team **Trello** board, as well as other boards for individual development teams.