GitGroup - A Web Based Software Development Management System

Runbo ZHAO
Jingyu BAO
Xuying CAO
Chengxiang YIN

Advisor: Edmund S. Yu

Course: CSE 682 Software Engineering

October 27, 2018

GitGroup

Contents

Chapter 1

Introduction

1.1 Purpose

The document is an official statement of what the system developers should implement. It includes both the user requirements for a system and a detailed specification of the system requirements. The document is essential for outside contractor who developing the software system. It is also useful to write a short supporting document that defines the business and dependability requirement for the system. The document is for system customers, managers, system engineers, system test engineers, system maintenance engineers.

1.2 What are GitHub APPs?

GitHub Apps are first-class actors within GitHub. A GitHub App acts on its own behalf, taking actions via the API directly using its own identity, which means you don't need to maintain a bot or service account as a separate user.
GitHub Apps can be installed directly on organizations and user accounts and granted access to specific repositories. They come with built-in webhooks and narrow, specific permissions. When you set up for your GitHub App, you can select the repositories you want it to access.
Some key ideas when creating GitHub Apps

1. A GitHub App should take actions independent of a user (unless the app is using a user-to-server token).
2. Make sure the GitHub App integrates with specific repositories.
3. The GitHub App should connect to a personal account or an organization.
4. Don't expect the GitHub App to know and do everything a user can.
5. Don't use a GitHub App if you just need a "Login with GitHub" service. But a GitHub App can use a user identification flow to log users in and do other things.
6. Don't build a GitHub App if you only want to act as a GitHub user and do everything that user can do.

1.3 Product Scope

GitGroup is a web-based GitHub Application. It is a project management platform that allows everyone on your team to communicate with the developers. Therefore, GitGroup is not only designed for programmer but also for people who has less development experience such as project

manager, project members and people who are related to the project. On GitGroup, it is easy to create an Agile workflow for your team with a kanban board. GitGroup?s seamless integration with GitHub keeps all of your GitHub Issue data in sync across both platforms in real time. It is an easy access web-based application that served for product management team based on Github API.
GitGroup is built by several software development tools. From the perspective of programming language, we use both TypeScript and JavaScript. TypeScript is an open-source programming language.It is a strict syntactical superset of JavaScript, and adds optional static typing to the language. And its strong Object-Oriented Programming features make it as a best developing language for building GitGroup. When the developing is done, TypeScript compiler can compile the source code to JavaScript program in order to let it run on the NodeJS. Javascript is a language which is also characterized as dynamic, weakly typed, prototype-based and multi-paradigm.
From the perspective of framework. ReactJS(Front end framework), Bulma(CSS framework), Frontawesome(Icon set and toolkit), ExpressJS(Back end framework) is chose. In the test process, our tool is Jest. It is a Javascript testing software. Jest is used by Facebook to test all JavaScript code including React applications. One of Jest's philosophies is to provide an integrated "zero-configuration" experience. This feature of Jest make the duration of developing shorter. Also we regard MongoDB as our host DBMS, which is one of the most popular database management tool. Other tools such as Git, Heroku, Visual Studio Code, Postman, are also included in developing process.

1.4 Definitions, Acronyms and Abbreviations

Owner	Who set up the project or mainly charge of the project
Collaborator	Who assists owner or project participants
Project	An element in GitGroup, including multiple repository
Repository	A folder which contains all project's files and stores each file's revision history
Issue	Information used to track ideas, enhancements, tasks, or bugs on GitHub
KanBan	A scheduling system for lean and just-in-time manufacturing
Cards	A message that signals depletion of product, parts, or inventory on KanBan

Table 1.1: Definitions, Acronyms and Abbreviations

1.5 Overview of The Document

The rest of the document will include overall description of Gitgroup in following perspectives. Product perspective, product functions, user characteristics, operating environment, design and implementation constraints, assumptions and dependencies. For interface requirement, we put efforts on user interfaces, hardware interfaces, software interfaces and communications interfaces. we will also include requirements like user requirements, system requirements, functional requirements, non-functional requirements and system models like use case diagram, activity diagram, sequence diagram, class diagram, system architecture, web design.

Chapter 2

Overall Description

This section of the SRS will include general factors that affect the product and its requirements. This section does not state specific requirements. Instead, it provides a background for those requirements, which are defined in detail in Section 3 of the SRS, and makes them easier to understand.

2.1 Product Perspective

The background of the Gitgroup coming from project managers are often busy to managing multiple projects at the same time. Sometimes, they are not familiar with details with products especially in technology perspective. However, Git and Github have a high entrance standard which project manager hard to manipulate it. For example, when you want to create an issue to suggest a new idea or track a bug, you need to learn markdown syntax, emoji code. And it is even more complex when you do pull request and merge the project. Comparing with Git and Github, GitGroup is an easier access tool . People who has little development experience, such as product manager. They also can handle GitGroup easily. The only thing they need to do is sign up an account and access to the GitHub api. The user can create online group chat, schedual and resord conference. After finding project idea, our online app also provide the task management service. In each project, every user will have their own grade. Furthermore, the online web can have a recommendation system which help the product manager and programmer match each other. In conclusion, it is a online service based on the GitHub API, and improve the user experience.

2.2 Product Functions

Our product focus on easy access, clearly, convenience, efficiency. Fuctions are surrounded our goals. Mainly function lists in the following.

1. Online meeting: Online meeting group chat. Record the conference. Make conference scheduling form and alarming the coming meeting.

2. Task management: The function is Like GitHub task management, but Gitgroup has better UI design. Make TODO list also included in task management, users create an issue to suggest a new idea or track a bug by means of setting up a TODO list. Then user can organize and assign tasks to their team members.

3. Developer grading system: Record the working of any team member. According the performance of users in all team which he joined, get users a grade.

4. Developer finding system: If you are a team leader, you can find a perfect developer who has related skill to your project. If you are a software developer, you can post your information to find suitable project for you.

2.3 User Characteristic

A high-level diagrammatic representation of the user activity is depicted below

2.3.1 For Owner

2.3.2 For Collaborator

2.4 Operating Environment

2.5 Design And Implementation Constraints

2.6 Assumptions And Dependencies

2.6.1 Assumptions

2.6.2 Dependencies

Chapter 3

Interface Requirements

3.1 User Interfaces

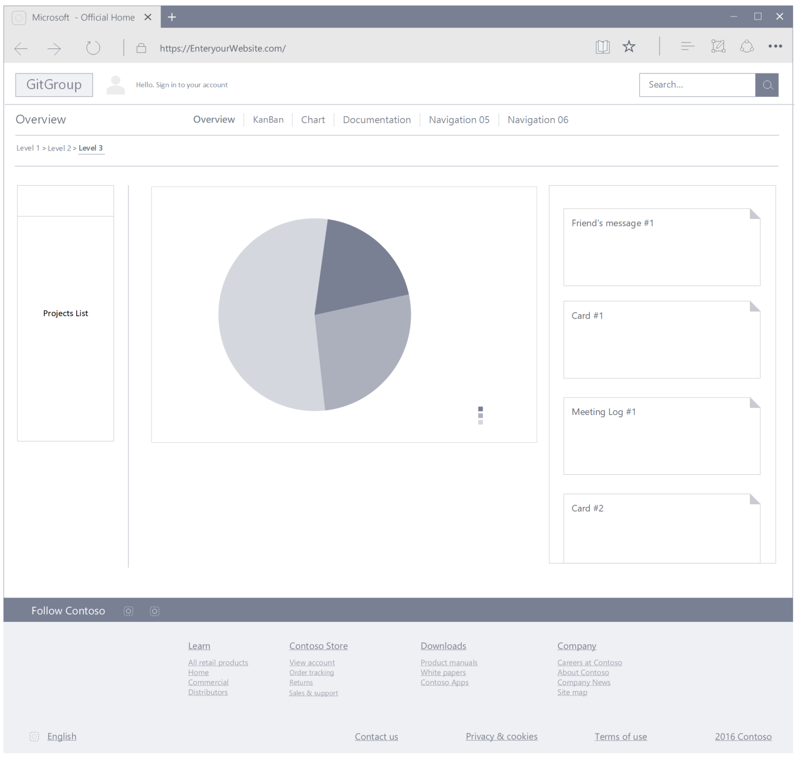


Figure 3.1: Overview User Interface (Details in Appendix A)

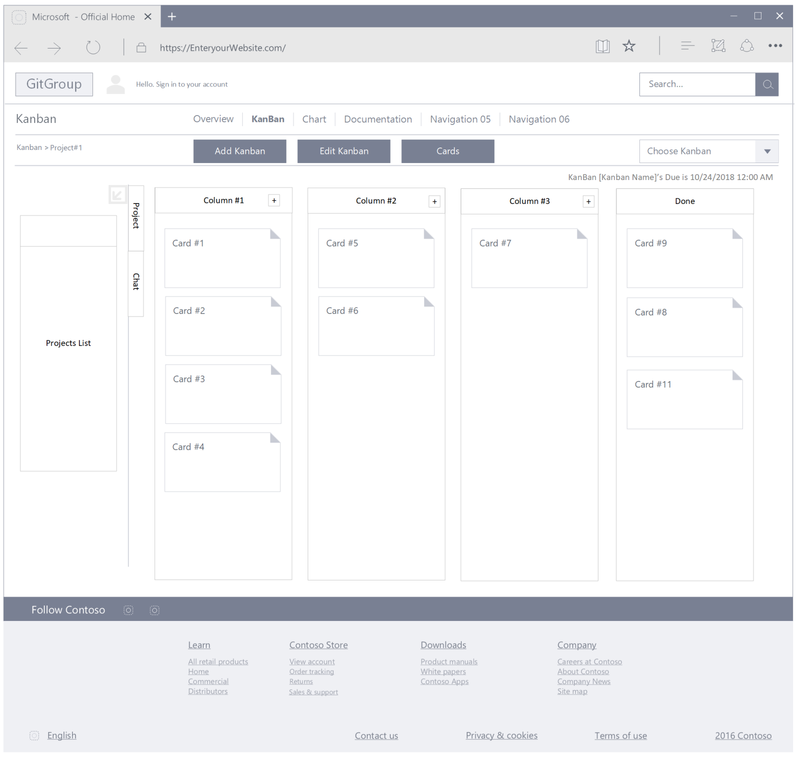


Figure 3.2: KanBan User Interface (Details in Appendix A)

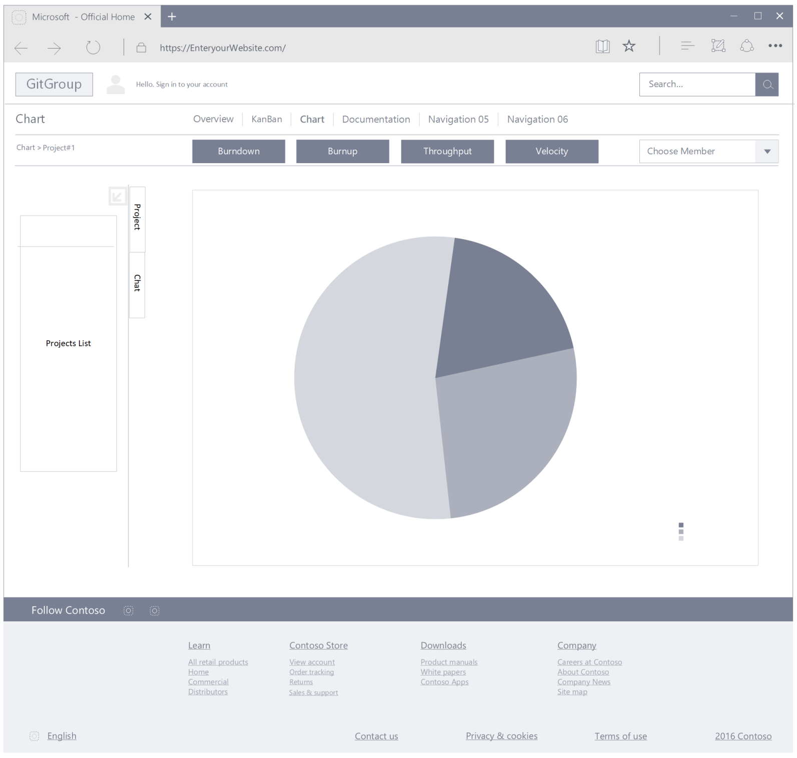


Figure 3.3: Chart User Interface (Details in Appendix A)

3.2 Hardware Interfaces

3.3 Software Interfaces

3.4 Communication Interfaces

Chapter 4

Requirements

4.1 User Requirements

4.1.1 Functional User Requirement

1. User shall be able to create a new project and new team in GitGroup.
2. User shall be able to manage GitHub repositories and team members like add or remove repositories and members.
3. User shall be able to use Kanban board to organize issues by dragging issue cards among several columns representing different stage of the development processes.
4. User shall be able to analysis the development duration, quality and the team work situation of the project by using a variety of agile charts and analytics. also manage tasks like suggesting a new idea or tracking a bug.
5. User shall be able to organize an online meeting group chat, schedule and record the conference within the team or the global scope.

4.1.2 Non-Functional User Requirement

1. The application shall be able to make GitHub easy to use, especially friendly for the person who has little development experience such as product manager.
2. The application shall be quickly restored to operational status after a failure occurs.
3. The app shall be reliable to uses with no downtime.
4. The application shall be able to provide maximum security against malicious attack.

4.2 System Requirements

4.2.1 Functional System Requirement

1.1 Developers can create a new project and make a team with other developers.

1.2 Developers can remove a project and collaborators of the project after checking if the project is empty.

2.1 Developers can manage their team by inviting to or removing collaborators from their projects.

2.2 Developers can manage their repositories by adding or removing repositories of their projects.

3.1 Developers can classify issues by different stages of development process via putting them in different columns of Kanban board.

3.2 Developers can add or remove a Kanban from their Kanban board.

3.3 Developers can customize their own stage by editing column name except Done column.

3.4 Developers can close issues by dragging it to the Done column, which will automatically close issues in the GitHub.

4.1 Project manager can track and communicate the progress of their projects by burn down and burn up charts.

4.2 Project manager can measure how much work a team can used in eXtreme Programming and Scrum from throughput chart.

4.3 Project manager can analysis the velocity of project going from velocity chart.

5.1 Team leader can organize an online meeting group chat within their team.

5.2 Developers can organize an online meeting group chat within the global.

5.3 Developers can receive a meeting notification from team leader.

5.4 Developers can set an alarm for notifying the coming events on a conference schedule form.

5.5 Developers shall be able to record content of meeting on a meeting notebook.

4.2.2 Non-Functional System Requirement

1.1 The user shall be able to use all the app functions without any kind of training. The average number of questions call about how to use app shall not exceed 10 per day.

1.2 The app shall be available for any kind of mobile device and PC.

2.1

3.1 The app shall be available to all users during whole day (Mon-Sun, 00:00 00:00)

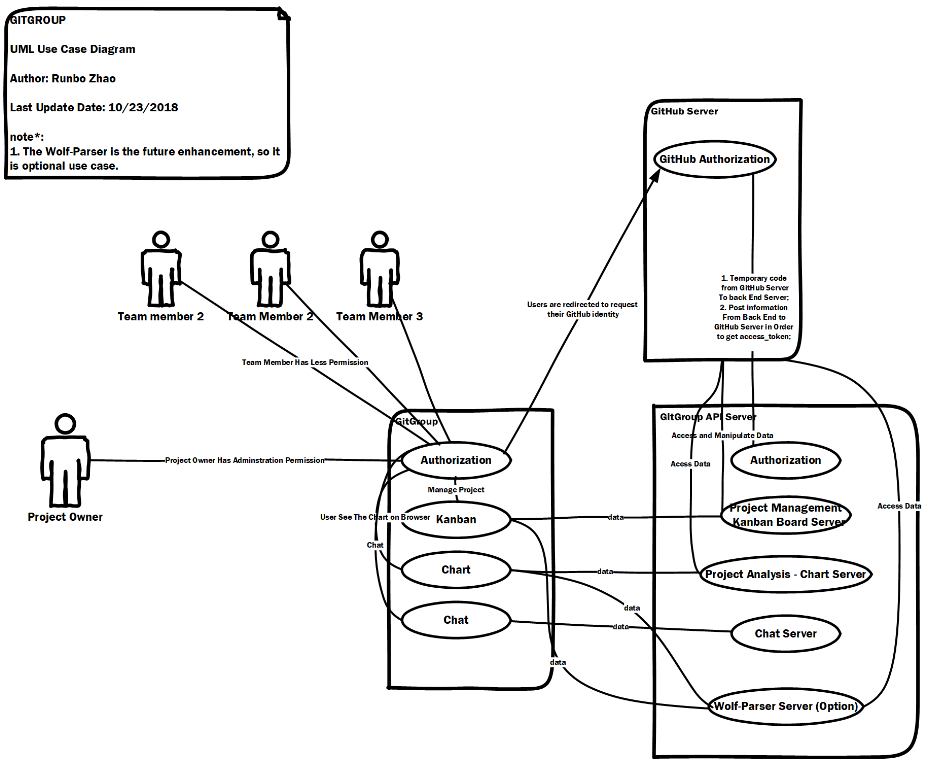
4.1 The app shall not expose contact information to other users.

4.2 The app should minimize the amount of personally identifying information (PII) that it collects.

Chapter 5

System Models

5.1 Use Case Diagram


GITGROUP

UML Use Case Diagram

Author: Runbo Zhao

Last Update Date: 10/23/2018

note*:
1. The Wolf-Parser is the future enhancement, so it is optional use case.

GitHub Server
GitHub Authorization
Users are redirected to request their GitHub identity
1. Temporary code from GitHub Server To back End Server; 2. Post information From Back End to GitHub Server in Order to get access_token;
Team member 2
Team Member 2
Team Member 3
Team Member Has Less Permission
Project Owner
Project Owner Has Adminstration Permission
GitGroup
Authorization
Manage Project
User See The Chart on Browser
Kanban
Chat
Chart
Chat
data
GitGroup API Server
Access and Manipulate Data
Acess Data
Authorization
Access Data
Project Management Kanban Board Server
data
Project Analysis - Chart Server
data
data
Chat Server
data
Wolf-Parser Server (Option)

Figure 5.1: Use Case Diagram (Details in Appendix A)

5.2 Activity Diagram

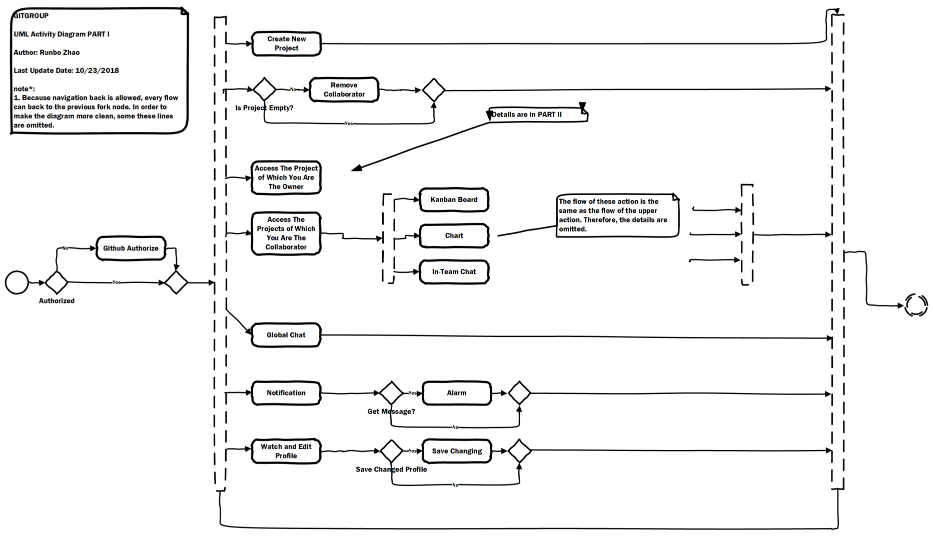

GITGROUP
UML Activity Diagram PART I
Author: Runbo Zhao
Last Update Date: 10/23/2018
note*:
1. Because navigation back is allowed, every flow can back to the previous fork node. In order to make the diagram more clean, some these lines are omitted.
Create New Project
Is Project Empty?
No
Remove Collaborator
Yes
Details are in PART II
Access The Project of Which You Are The Owner
Access The Projects of Which You Are The Collaborator
Kanban Board
Chart
In-Team Chat
The flow of these action is the same as the flow of the upper action. Therefore, the details are omitted.
No
Github Authorize
Yes
Authorized
Global Chat
Notification
Get Message?
Yes
Alarm
No
Watch and Edit Profile
Save Changed Profile
Yes
Save Changing
No

Figure 5.2: Activity Diagram PART I (Details in Appendix A)

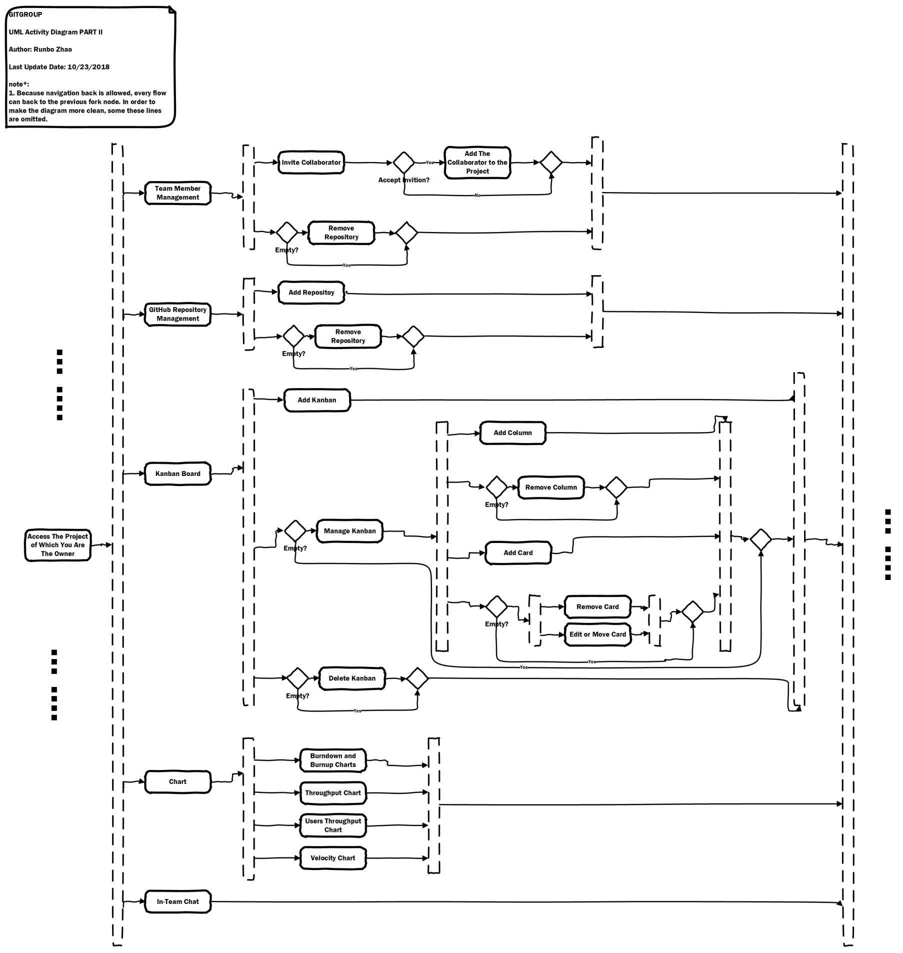


Figure 5.3: Activity Diagram PART II (Details in Appendix A)

5.3 Sequence Diagram

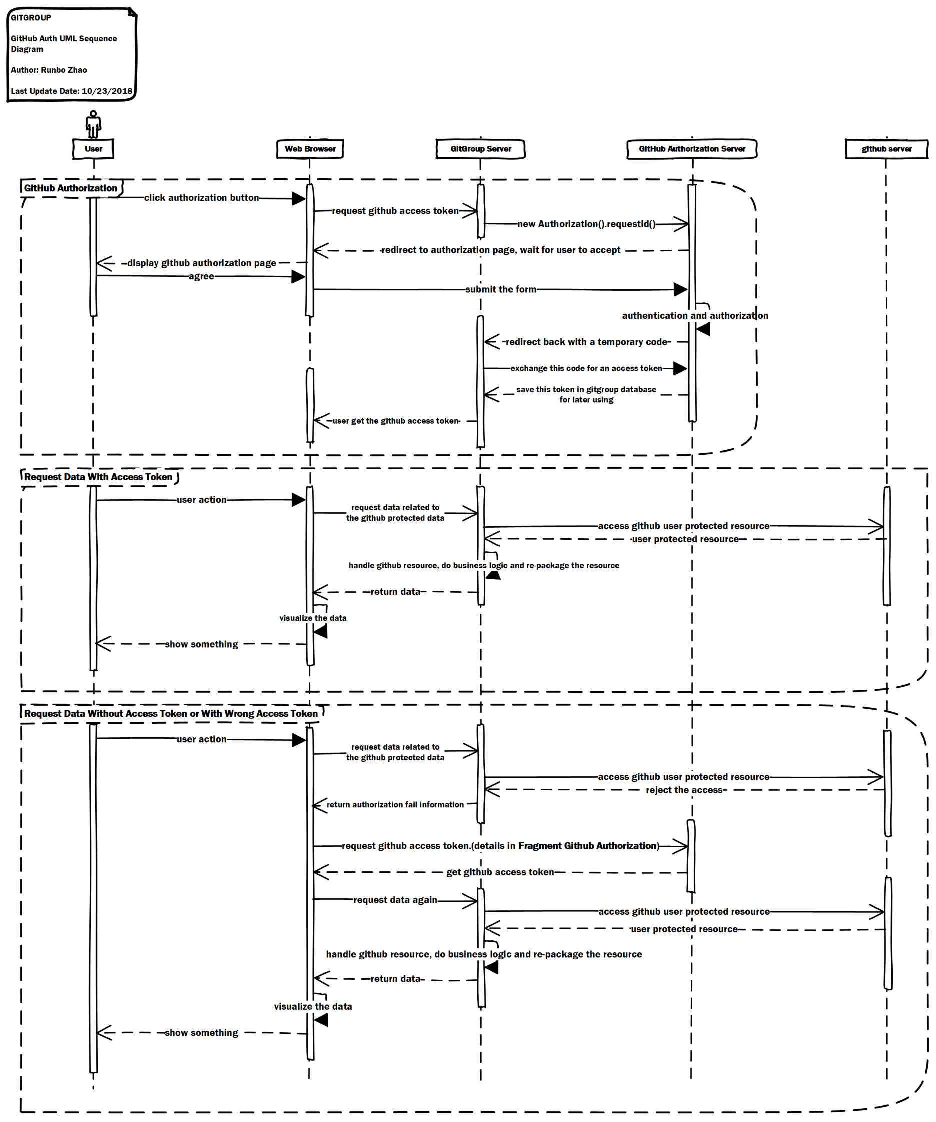

GITGROUP
GitHub Auth UML Sequence Diagram
Author: Runbo Zhao
Last Update Date: 10/23/2018
User
Web Browser
GitGroup Server
GitHub Authorization Server
github server
GitHub Authorization
click authorization button
request github access token
new Authorization().requestId()
redirect to authorization page, wait for user to accept
display github authorization page
agree
submit the form
authentication and authorization
redirect back with a temporary code
exchange this code for an access token
save this token in gitgroup database for later using
user get the github access token
Request Data With Access Token
user action
request data related to the github protected data
access github user protected resource
user protected resource
handle github resource, do business logic and re-package the resource
return data
visualize the data
show something
Request Data Without Access Token or With Wrong Access Token
user action
request data related to the github protected data
access github user protected resource
reject the access
return authorization fail information
request github access token.(details in Fragment Github Authorization)
get github access token
request data again
access github user protected resource
user protected resource
handle github resource, do business logic and re-package the resource
return data
visualize the data
show something

Figure 5.4: Sequence Diagram For Authorization (Details in Appendix A)

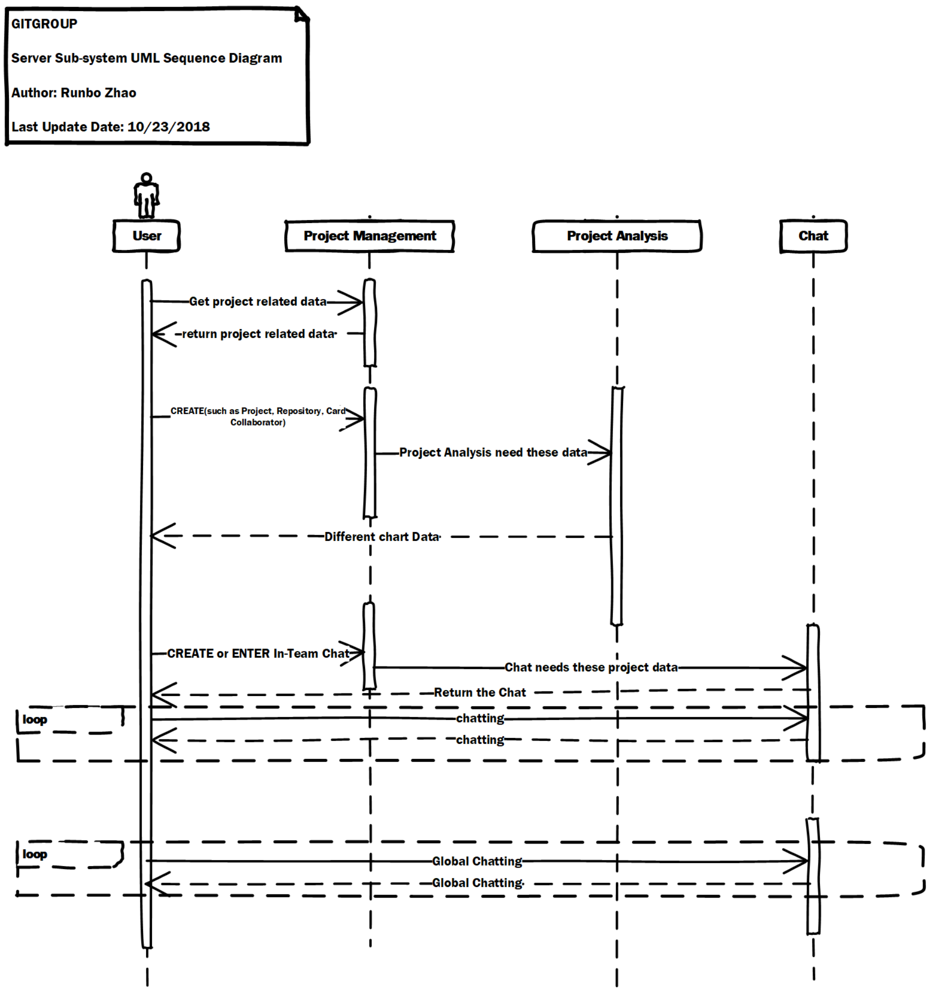

GITGROUP
Server Sub-system UML Sequence Diagram
Author: Runbo Zhao
Last Update Date: 10/23/2018
User
Project Management
Project Analysis
Chat
Get project related data
return project related data
CREATE(such as Project, Repository, Card Collaborator)
Project Analysis need these data
Different chart Data
CREATE or ENTER In-Team Chat
Chat needs these project data
Return the Chat
loop
chatting
chatting
loop
Global Chatting
Global Chatting

Figure 5.5: Sequence Diagram For Sub-Systems (Details in Appendix A)

5.4 Class Diagram

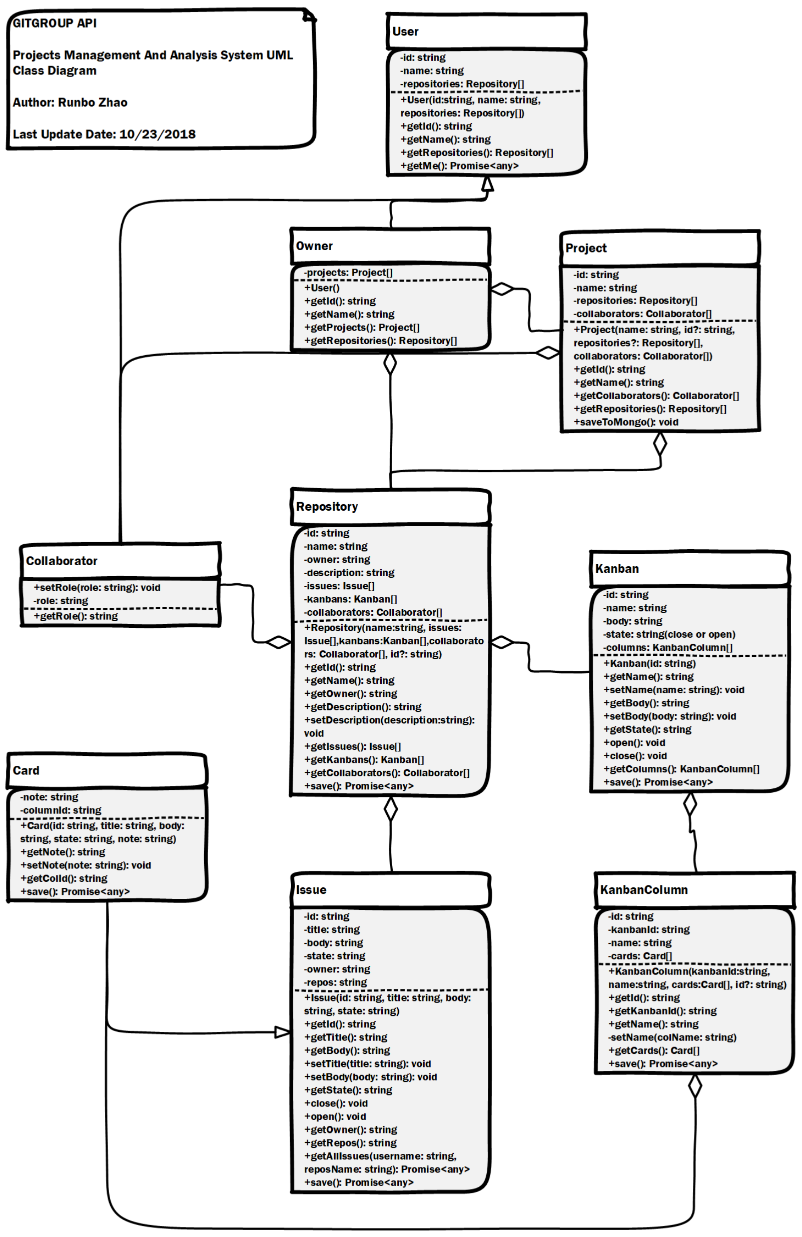

GITGROUP API
Projects Management And Analysis System UML Class Diagram
Author: Runbo Zhao
Last Update Date: 10/23/2018
User
-id: string
-name: string
-repositories: Repository[]
+User(id:string, name: string, repositories: Repository[])
+getId(): string
+getName(): string
+getRepositories(): Repository[]
+getMe(): Promise<any>
Owner
-projects: Project[]
+User()
+getId(): string
+getName(): string
+getProjects(): Project[]
+getRepositories(): Repository[]
Project
-id: string
-name: string
-repositories: Repository[]
-collaborators: Collaborator[]
+Project(name: string, id?: string, repositories?: Repository[], collaborators: Collaborator[])
+getId(): string
+getName(): string
+getCollaborators(): Collaborator[]
+getRepositories(): Repository[]
+saveToMongo(): void
Repository
-id: string
-name: string
-owner: string
-description: string
-issues: Issue[]
-kanbans: Kanban[]
-collaborators: Collaborator[]
+Repository(name:string, issues: Issue[],kanbans:Kanban[],collaborators: Collaborator[], id?: string)
+getId(): string
+getName(): string
+getOwner(): string
+getDescription(): string
+setDescription(description:string): void
+getIssues(): Issue[]
+getKanbans(): Kanban[]
+getCollaborators(): Collaborator[]
+save(): Promise<any>
Collaborator
+setRole(role: string): void
-role: string
+getRole(): string
Kanban
-id: string
-name: string
-body: string
-state: string(close or open)
-columns: KanbanColumn[]
+Kanban(id: string)
+getName(): string
+setName(name: string): void
+getBody(): string
+setBody(body: string): void
+getState(): string
+open(): void
+close(): void
+getColumns(): KanbanColumn[]
+save(): Promise<any>
Card
-note: string
-columnId: string
+Card(id: string, title: string, body: string, state: string, note: string)
+getNote(): string
+setNote(note: string): void
+getColId(): string
+save(): Promise<any>
Issue
-id: string
-title: string
-body: string
-state: string
-owner: string
-repos: string
+Issue(id: string, title: string, body: string, state: string)
+getId(): string
+getTitle(): string
+getBody(): string
+setTitle(title: string): void
+setBody(body: string): void
+getState(): string
+close(): void
+open(): void
+getOwner(): string
+getRepos(): string
+getAllIssues(username: string, reposName: string): Promise<any>
+save(): Promise<any>
KanbanColumn
-id: string
-kanbanId: string
-name: string
-cards: Card[]
+KanbanColumn(kanbanId:string, name:string, cards:Card[], id?: string)
+getId(): string
+getKanbanId(): string
+getName(): string
-setName(colName: string)
+getCards(): Card[]
+save(): Promise<any>

Figure 5.6: UML Class Diagram (Details in Appendix A)

Chapter 6

System Architecture

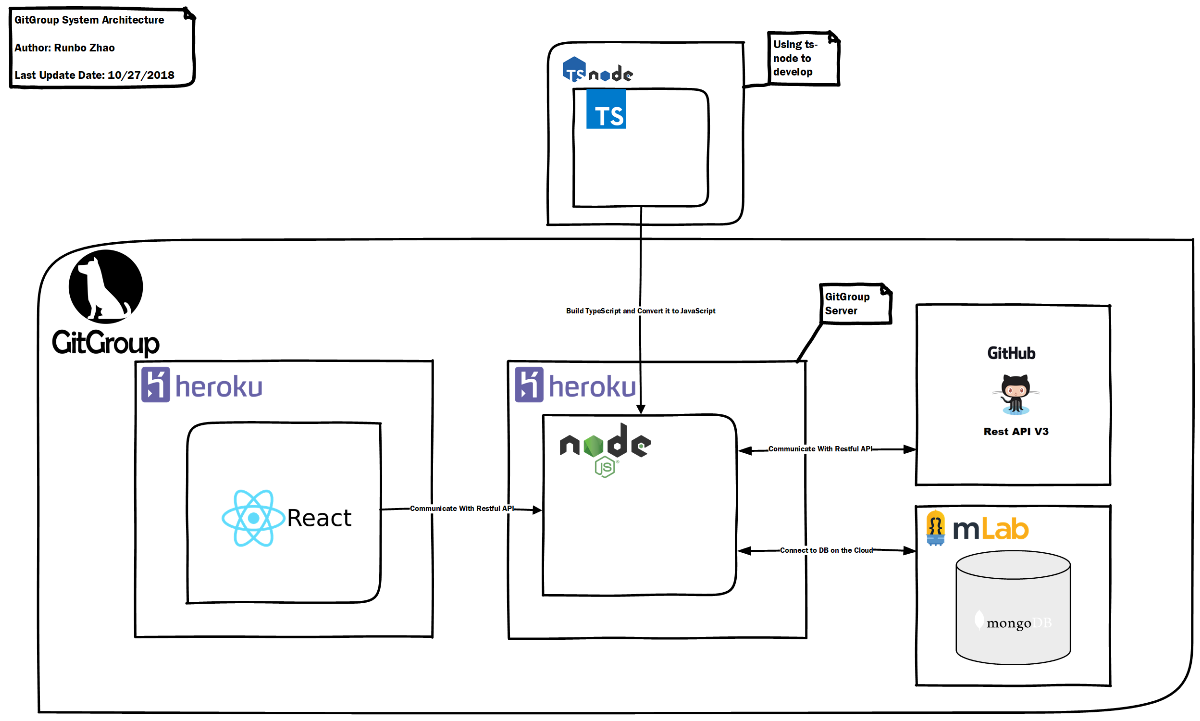

GitGroup System Architecture
Author: Runbo Zhao
Last Update Date: 10/27/2018
TS node
TS
Using ts-node to develop
GitGroup
Build TypeScript and Convert it to JavaScript
GitGroup Server
heroku
React
Communicate With Restful API
heroku
node JS
Communicate With Restful API
Connect to DB on the Cloud
GitHub
Rest API V3
mLab
mongoDB

Figure 6.1: GitGroup System Architecture

6.1 TypeScript

TypeScript is an open-source programming language developed and maintained by Microsoft. It is a strict syntactical superset of JavaScript, and adds optional static typing to the language.
TypeScript is used to develop GitGroup because its strong Object-Oriented-Programming features.
TypeScript is designed for development of large applications and transcompiles to JavaScript.As TypeScript is a superset of JavaScript, existing JavaScript programs are also valid TypeScript programs. TypeScript may be used to develop JavaScript applications for both client-side and server-side (Node.js) execution.

6.2 mLab

mLab is a fully managed cloud database service that hosts MongoDB databases. mLab runs on cloud providers Amazon, Google, and Microsoft Azure, and has partnered with platform-as-a-service providers.

6.3 MongoDB

MongoDB is a free and open-source cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with schemata. MongoDB is developed by MongoDB Inc., and is published under a combination of the Server Side Public License and the Apache License.
MongoDB is used via mLab in this project. And Mongoose is chosen as the ORM(Object-relational mapping).

6.4 Express

Express, a web server built on Node.js, forms the middle tier or the web server. It defines routes, specifications of what to do when a HTTP request matching a certain pattern arrives. The matching specification is regular expression (regex) based and is very flexible. The what-to-do part is just a function that is given the parsed HTTP request. Express parses request URL, headers, and parameters for us. On the response side, it has, as expected, all of the functionality required by web applications including setting response codes, setting cookies, sending custom headers, etc.

6.5 React

React is a front-end technology from Facebook. It?s not a full-fledged MVC framework, but it?s a JavaScript library for building user interfaces, so in some sense it?s the view part of MVC. We use React to render a view because it tries to solve the problem from the View layer by creating abstract representations of views. It breaks down parts of the view in the Components. These components encompass both the logic to handle the display of view and the view itself. It can contain data that it uses to render the state of the app.

6.6 NodeJs

NodeJs, a server-side JavaScript runtime environment, has an asynchronous eventdriven, non-blocking input/output (I/O) model, as opposed to using threads to achieve multitasking. It relies heavily on callbacks and promises to let you know that a pending task is completed. Node.js achieves multitasking using an event-loop. This is nothing but a queue of events that need to be processed and callbacks to be run on those events. An event-based approach also makes Node.js applications fast and lets the programmer be blissfully oblivious of the semaphores and locks that are utilized to synchronize multithreaded events.

6.7 Heroku

Heroku is a cloud platform as a service (PaaS) supporting several programming languages. Heroku, one of the first cloud platforms, has been in development since June 2007, when it supported only

the Ruby programming language, but now supports Java, Node.js, Scala, Clojure, Python, PHP, and Go.
In this project, Heroku is the platform where GitGroup is deployed to.

Appendix A

Figures

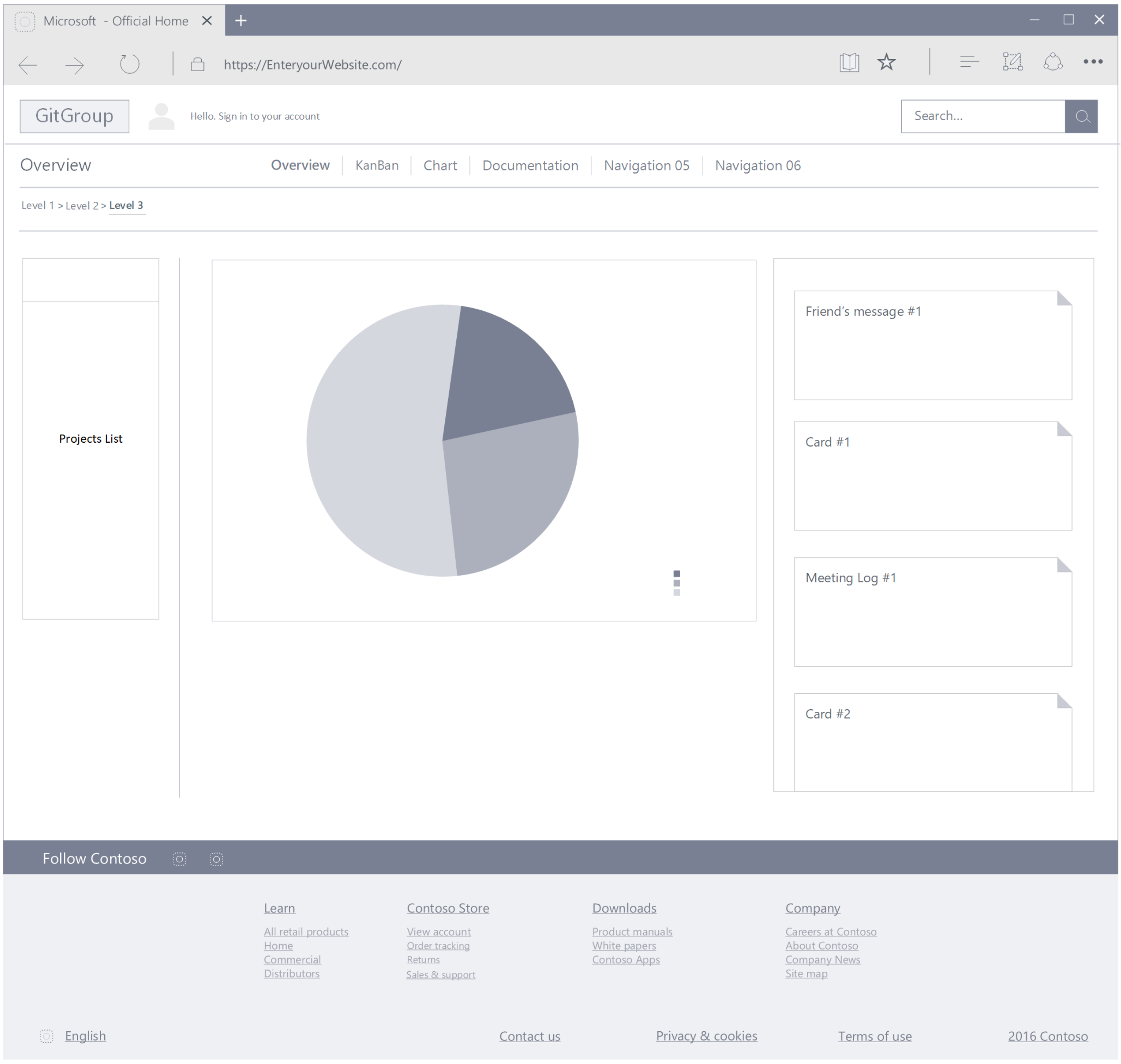


Figure A.1: **DETAIL** Overview User Interface

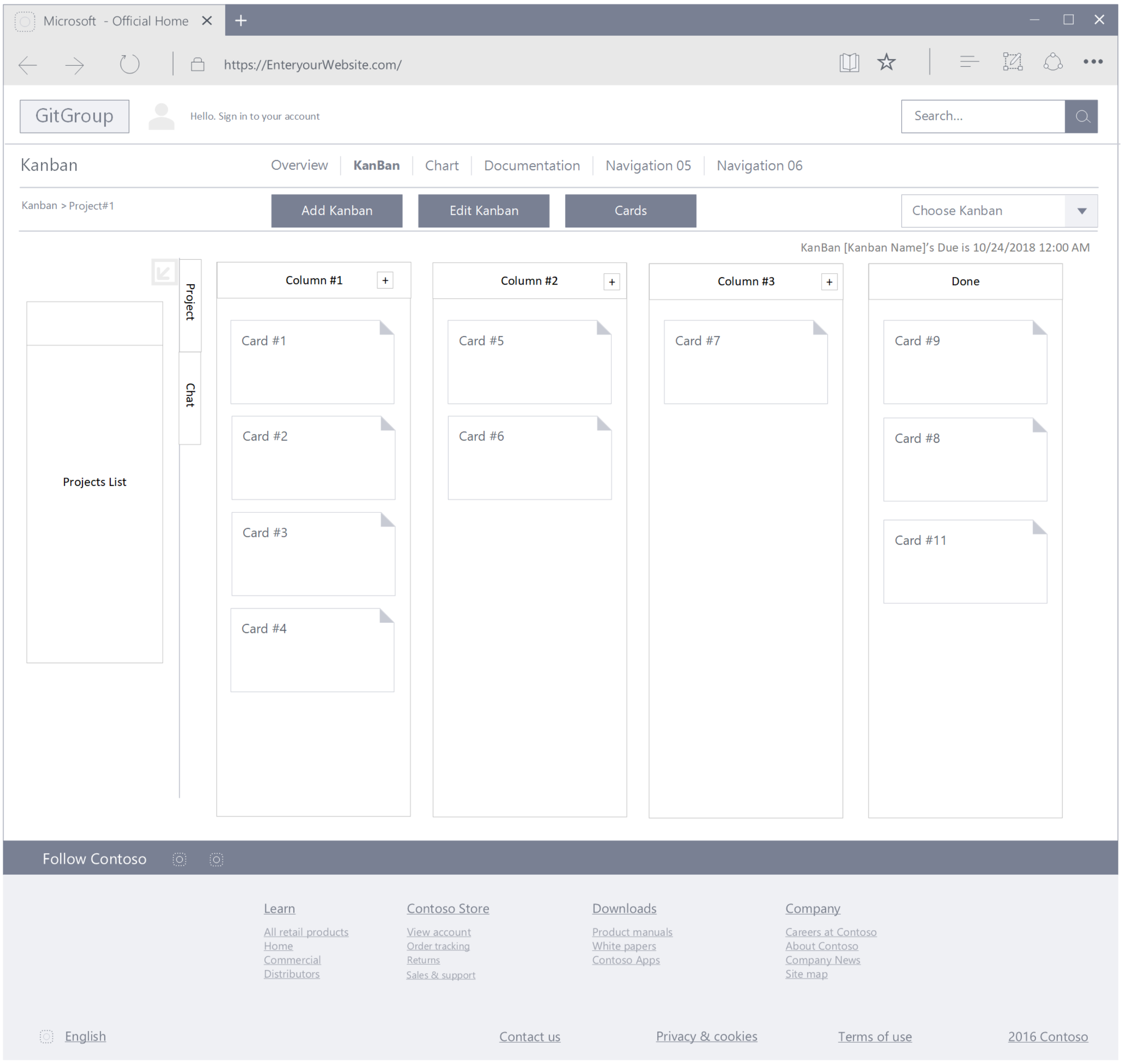


Figure A.2: **DETAIL** KanBan User Interface

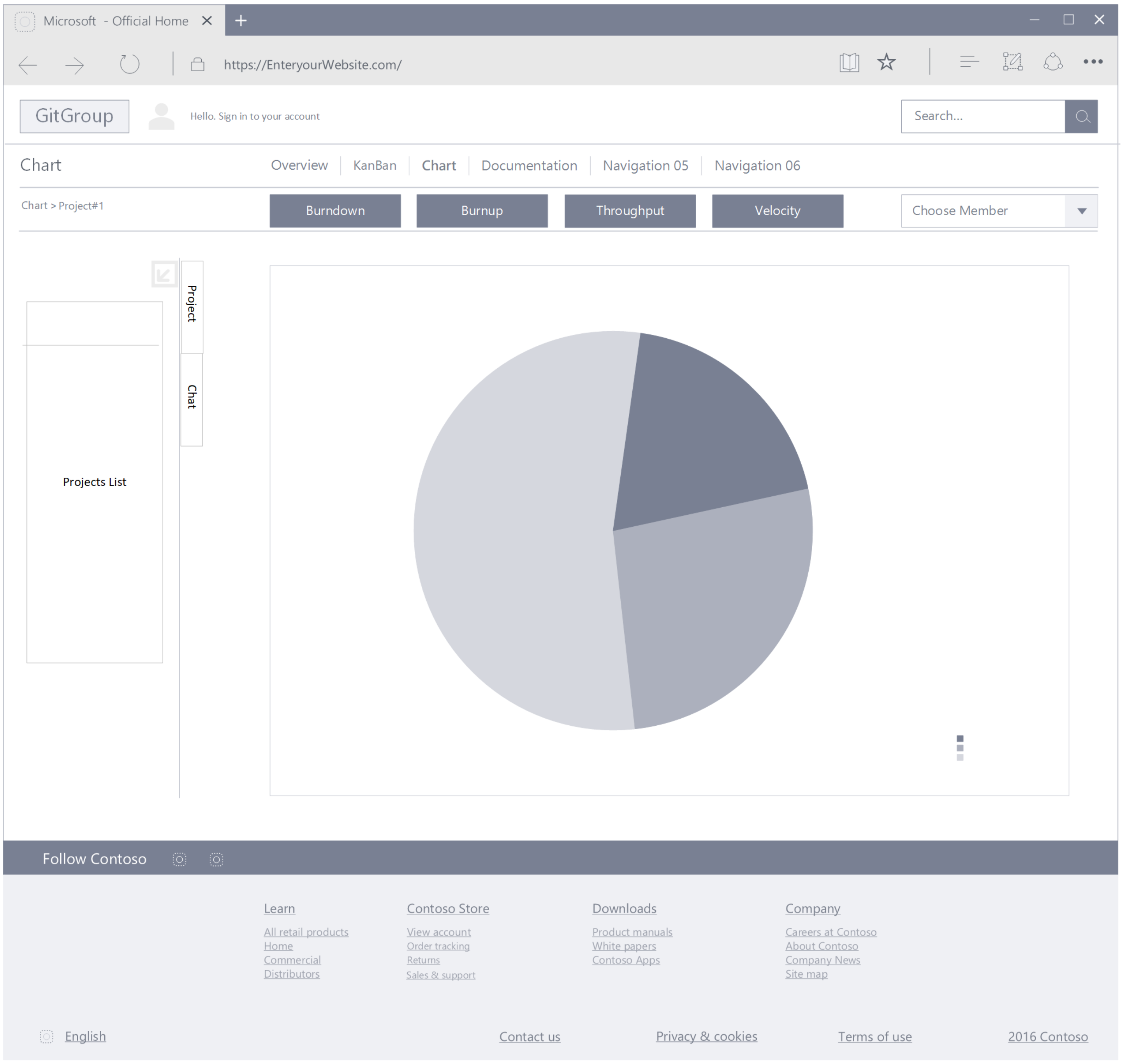
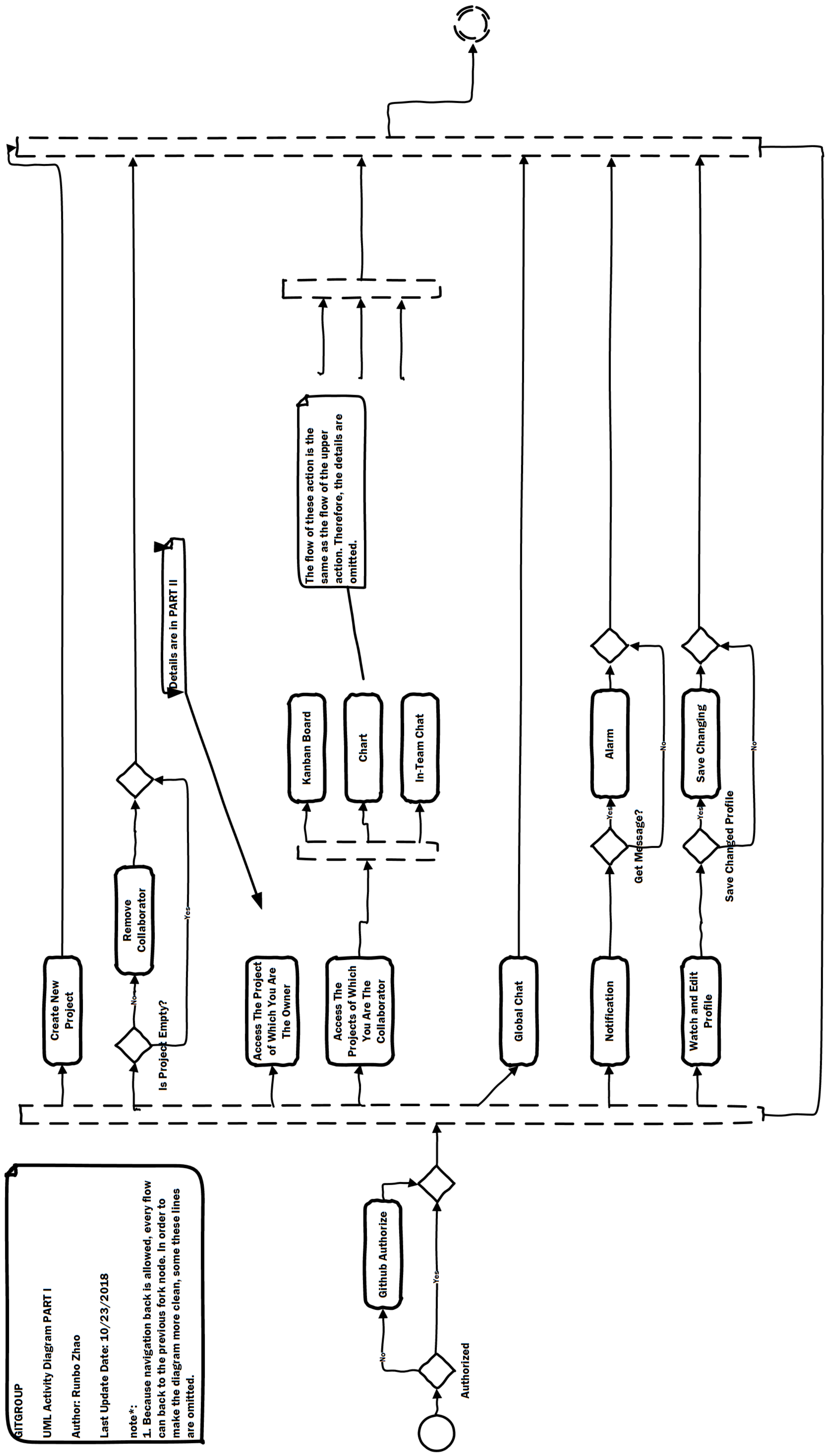


Figure A.3: **DETAIL** Chart User Interface


GITGROUP

UML Activity Diagram PART I

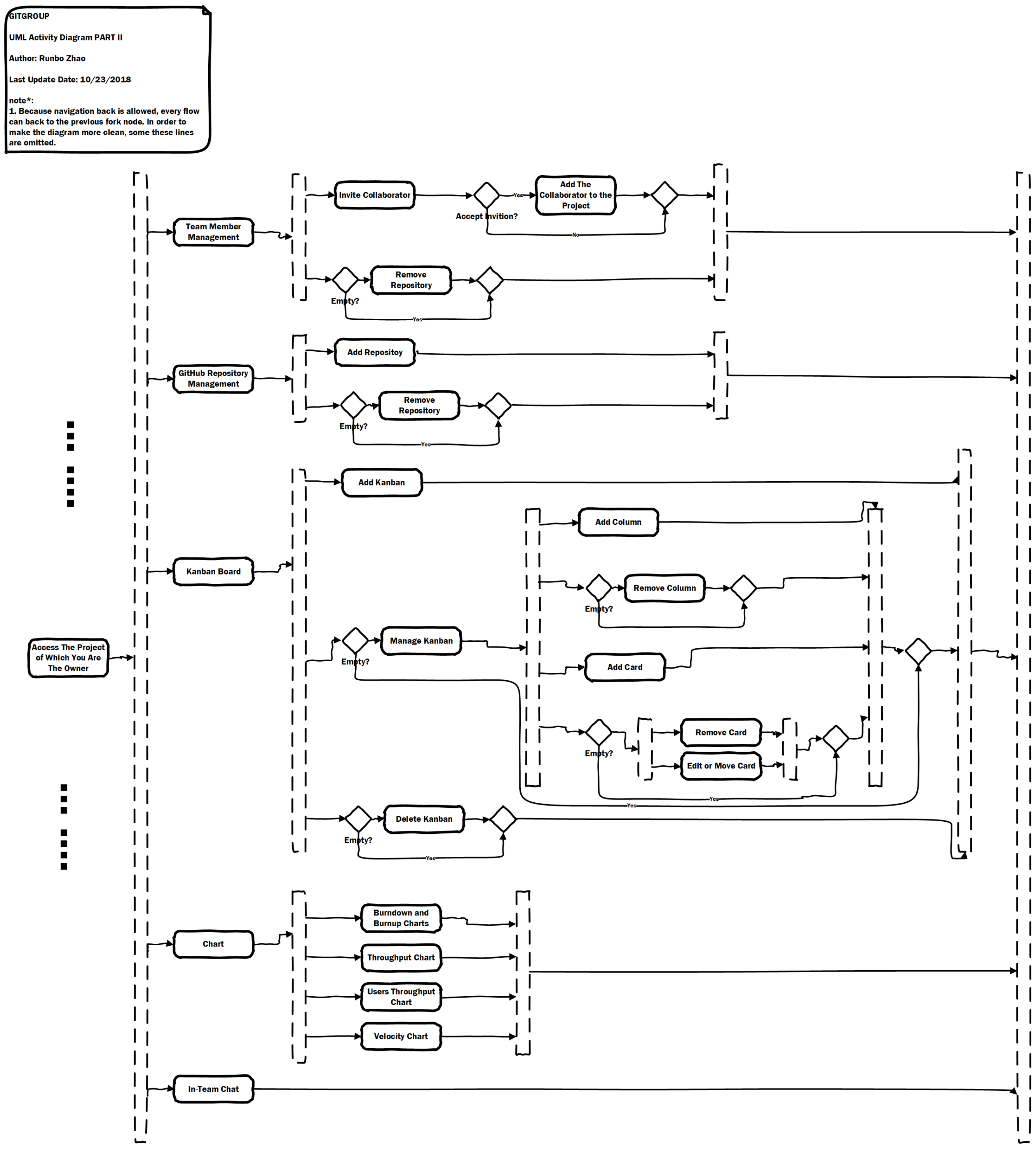
Author: Runbo Zhao

Last Update Date: 10/23/2018

note*:
1. Because navigation back is allowed, every flow can back to the previous fork node. In order to make the diagram more clean, some these lines are omitted.

The flow of these action is the same as the flow of the upper action. Therefore, the details are omitted.

Figure A.4: **DETAIL** Activity Diagram PART I


GITGROUP

UML Activity Diagram PART II

Author: Runbo Zhao

Last Update Date: 10/23/2018

note*:
1. Because navigation back is allowed, every flow can back to the previous fork node. In order to make the diagram more clean, some these lines are omitted.

Figure A.5: **DETAIL** Activity Diagram PART II

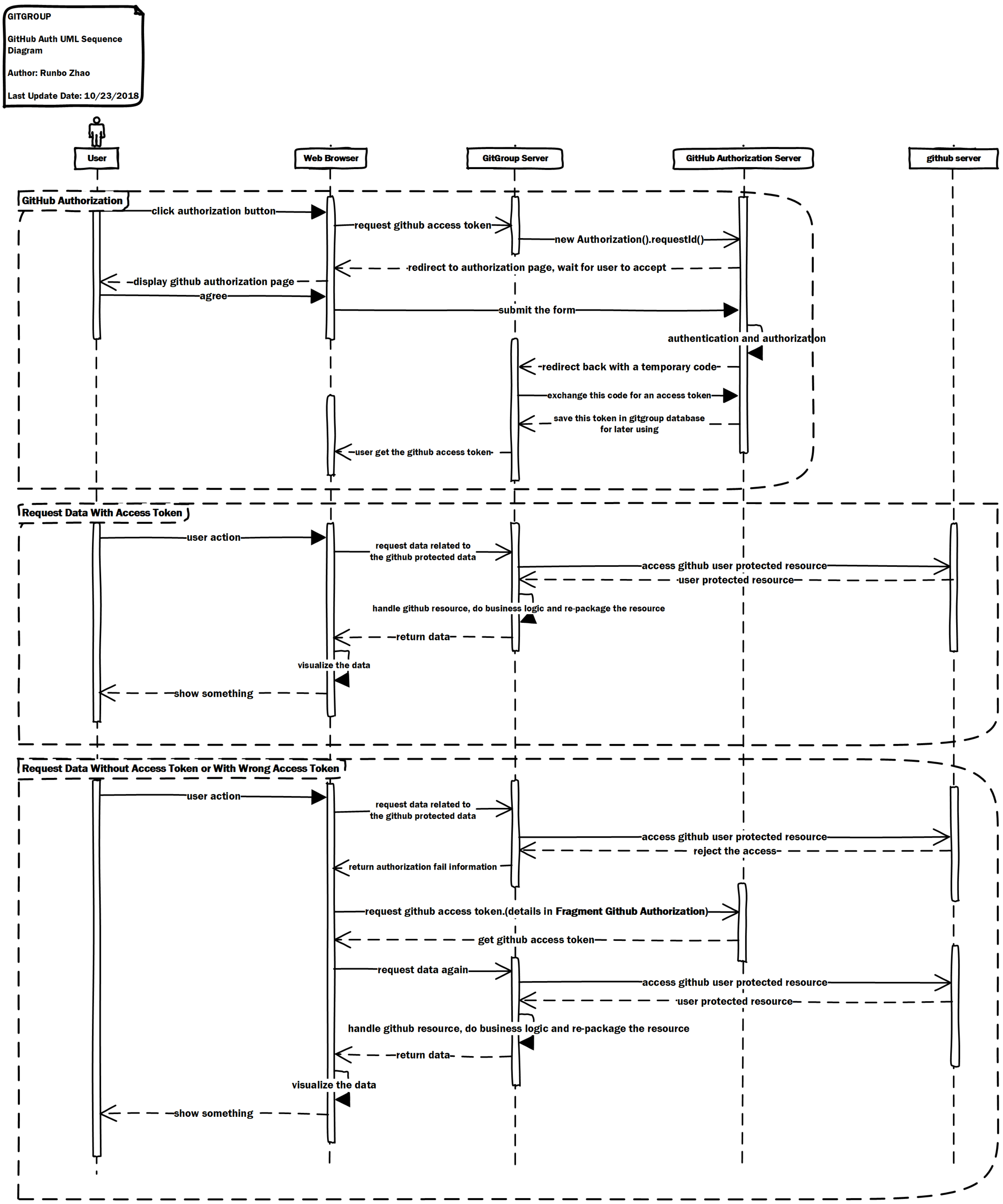

GITGROUP
GitHub Auth UML Sequence Diagram
Author: Runbo Zhao
Last Update Date: 10/23/2018
User
Web Browser
GitGroup Server
GitHub Authorization Server
github server
GitHub Authorization
click authorization button
request github access token
new Authorization().requestId()
redirect to authorization page, wait for user to accept
display github authorization page
agree
submit the form
authentication and authorization
redirect back with a temporary code
exchange this code for an access token
save this token in gitgroup database for later using
user get the github access token
Request Data With Access Token
user action
request data related to the github protected data
access github user protected resource
user protected resource
handle github resource, do business logic and re-package the resource
return data
visualize the data
show something
Request Data Without Access Token or With Wrong Access Token
user action
request data related to the github protected data
access github user protected resource
reject the access
return authorization fail information
request github access token.(details in Fragment Github Authorization)
get github access token
request data again
access github user protected resource
user protected resource
handle github resource, do business logic and re-package the resource
return data
visualize the data
show something

Figure A.6: **DETAIL** Sequence Diagram For Authorization

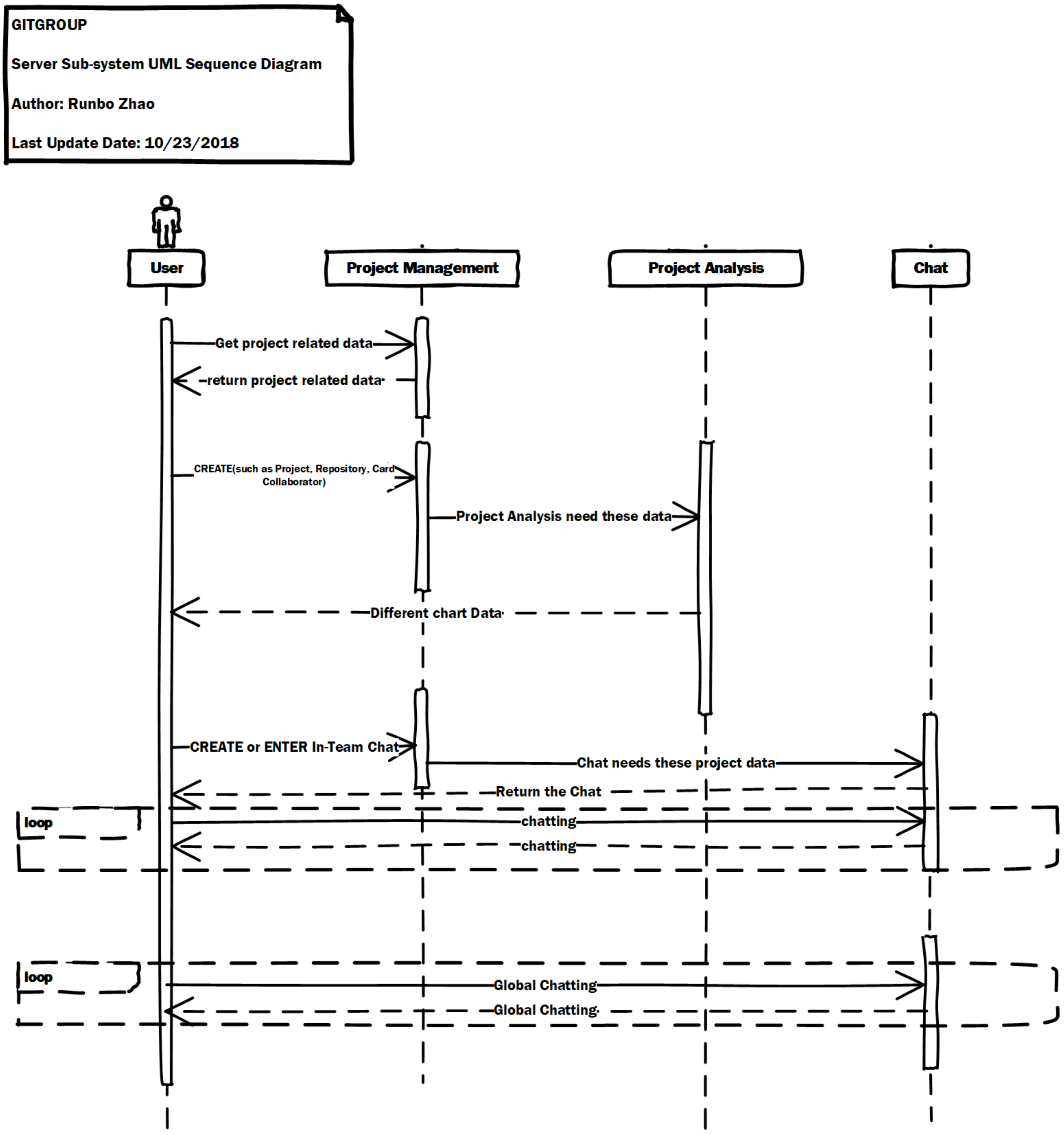

GITGROUP
Server Sub-system UML Sequence Diagram
Author: Runbo Zhao
Last Update Date: 10/23/2018
User
Project Management
Project Analysis
Chat
Get project related data
return project related data
CREATE(such as Project, Repository, Card Collaborator)
Project Analysis need these data
Different chart Data
CREATE or ENTER In-Team Chat
Chat needs these project data
Return the Chat
loop
chatting
chatting
loop
Global Chatting
Global Chatting

Figure A.7: **DETAIL** Sequence Diagram For Sub-Systems

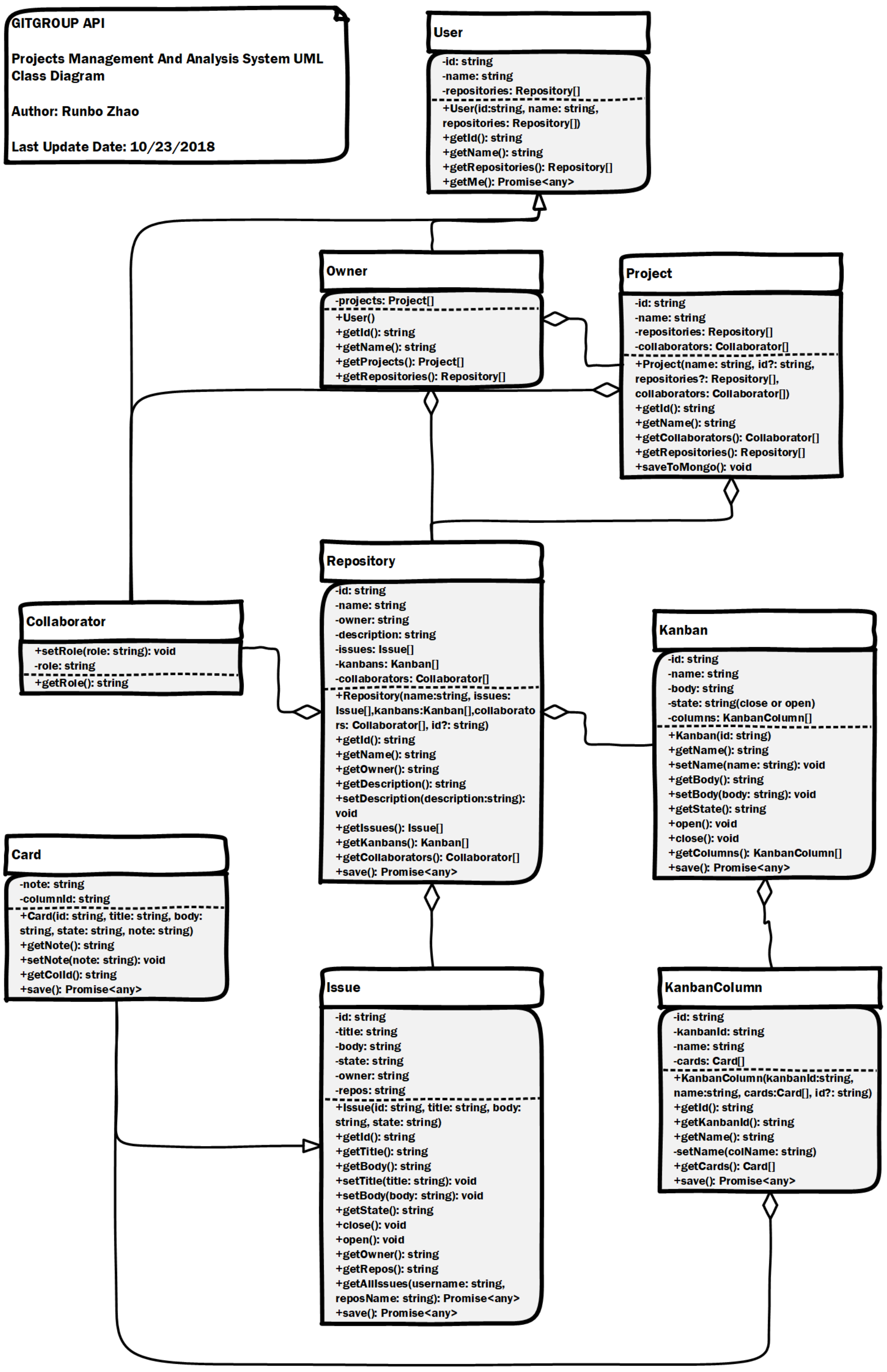

GITGROUP API
Projects Management And Analysis System UML Class Diagram
Author: Runbo Zhao
Last Update Date: 10/23/2018
User
-id: string
-name: string
-repositories: Repository[]
+User(id:string, name: string, repositories: Repository[])
+getId(): string
+getName(): string
+getRepositories(): Repository[]
+getMe(): Promise<any>
Owner
-projects: Project[]
+User()
+getId(): string
+getName(): string
+getProjects(): Project[]
+getRepositories(): Repository[]
Project
-id: string
-name: string
-repositories: Repository[]
-collaborators: Collaborator[]
+Project(name: string, id?: string, repositories?: Repository[], collaborators: Collaborator[])
+getId(): string
+getName(): string
+getCollaborators(): Collaborator[]
+getRepositories(): Repository[]
+saveToMongo(): void
Repository
-id: string
-name: string
-owner: string
-description: string
-issues: Issue[]
-kanbans: Kanban[]
-collaborators: Collaborator[]
+Repository(name:string, issues: Issue[],kanbans:Kanban[],collaborators: Collaborator[], id?: string)
+getId(): string
+getName(): string
+getOwner(): string
+getDescription(): string
+setDescription(description:string): void
+getIssues(): Issue[]
+getKanbans(): Kanban[]
+getCollaborators(): Collaborator[]
+save(): Promise<any>
Collaborator
+setRole(role: string): void
-role: string
+getRole(): string
Kanban
-id: string
-name: string
-body: string
-state: string(close or open)
-columns: KanbanColumn[]
+Kanban(id: string)
+getName(): string
+setName(name: string): void
+getBody(): string
+setBody(body: string): void
+getState(): string
+open(): void
+close(): void
+getColumns(): KanbanColumn[]
+save(): Promise<any>
Card
-note: string
-columnId: string
+Card(id: string, title: string, body: string, state: string, note: string)
+getNote(): string
+setNote(note: string): void
+getColId(): string
+save(): Promise<any>
Issue
-id: string
-title: string
-body: string
-state: string
-owner: string
-repos: string
+Issue(id: string, title: string, body: string, state: string)
+getId(): string
+getTitle(): string
+getBody(): string
+setTitle(title: string): void
+setBody(body: string): void
+getState(): string
+close(): void
+open(): void
+getOwner(): string
+getRepos(): string
+getAllIssues(username: string, reposName: string): Promise<any>
+save(): Promise<any>
KanbanColumn
-id: string
-kanbanId: string
-name: string
-cards: Card[]
+KanbanColumn(kanbanId:string, name:string, cards:Card[], id?: string)
+getId(): string
+getKanbanId(): string
+getName(): string
-setName(colName: string)
+getCards(): Card[]
+save(): Promise<any>

Figure A.8: **DETAIL** UML Class Diagram

List of Figures

List of Tables