GitGoing Code Reviewer

# Software Design Document

Names: Travis Lamb, Christina Nguyen, Michael Bloomquist, Sarah Ramazani

Date: (11/09/2020)

**TABLE OF CONTENTS**

1. [INTRODUCTION 2](#_bookmark0)
   1. [Purpose 2](#_bookmark0)
   2. [Scope 2](#_bookmark0)
   3. Definitions, Acronyms, and Abbreviations
   4. [References 2](#_bookmark0)
2. [SYSTEM OVERVIEW 2](#_bookmark0)
3. [SYSTEM Components 2](#_bookmark0)
   1. [Decomposition Description 3](#_bookmark1)
   2. [Dependency Description 3](#_bookmark1)
   3. Interface Description 3
      * Module Interfaces
      * User Interfaces (GUI)
4. [DETAILED DESIGN 3](#_bookmark1)
   1. [Module Detailed Design 3](#_bookmark1)
   2. [Data Detailed Design 3](#_bookmark1)
   3. RTM 3

### INTRODUCTION

## Purpose

This Software Design Document contains a comprehensive description of the structure of GitGoing, and its constituent components, including planned implementation. The expected audience is the Computer Science department of Bellevue College, including Professor Sara Farag, and other top-level Computer Science department members. Other Computer Science (CS) Department members may also find use of this document. Clients are not expected have access to this document, and thus the language is that expected of CS students/instructors.

## Scope

The basic architecture of GitGoing is an Amazon EC2 virtual machine and relational database backend, with an Amazon S3 hosted web application front-end for clients. The database is planned to be a relational database from MySQL, paired to a JavaScript (JS) web application that utilizes HTML, CSS, and React to implement UI systems. Planned features include, but are not limited to, the ability to publish code for review, reviewing the difference in code (or possibly many types of files), commenting on said reviews, direct messaging (DM) between users, and a notification system to alert users of new comments, reviews, etc.

## Definitions, Acronyms, and Abbreviations

## CS *Computer Science*

## JS *JavaScript*

## DM *Direct Message*

## VCS *Version Control System*

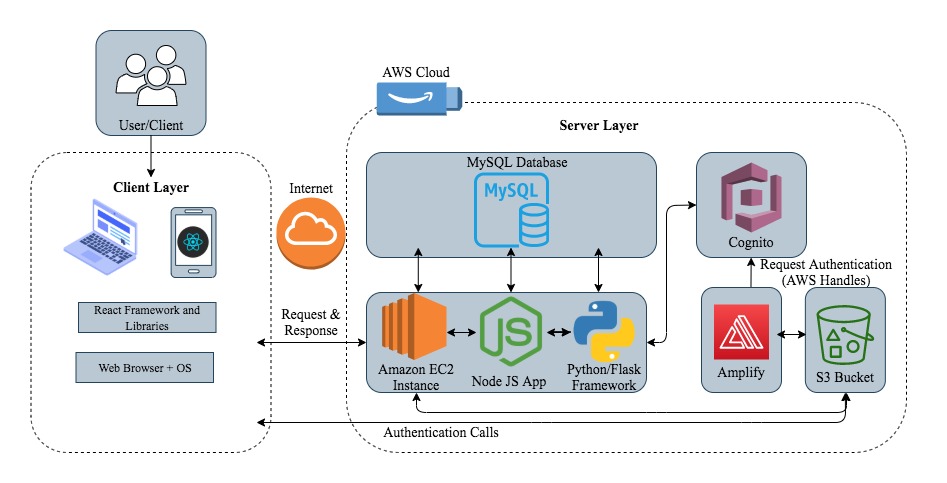
## Client A site user

## References

*This section is optional.*

List any documents, if any, which were used as sources of information for the test plan.

### SYSTEM OVERVIEW



GitGoing is implemented as a Client-Server pattern where the server layer will be hidden and solely used to listen to ongoing requests from the clients on the backend. Again, the Client layer will be a Web browser on the frontend built through React Frameworks and Libraries. To listen to ongoing requests, we have Amazon EC2 instance, Node JS App and Python/Flask frameworks which work together to communicate directly to the database. The site build sits on a S3 Bucket and Congito will process all request authentications through Amplify, which is AWS hosted. Altogether, two parties (client and server) work collectively to process, edit, store, and deliver an operable system to clients whether it be on a PC, smartphone, or laptop.

### SYSTEM Components

## Decomposition Description

## Git Going has both front and backend components. The frontend deals with all user interface and user features, as well as communicating with the backend. The backend orchestrates all database CRUD operations, as well as any of the work needed to attain diffs between file versions.

## The frontend will send HTTP requests to the backend in the form of a request message. The request is made up of the endpoint, which is the entry point at which the API connects to GitGoing. To get the meaning of the sent request, these HTTP methods will be used: GET, POST, PUT, DELETE.

## The query parameters will include admin, user, repo, git diff (to display changes in files or commit), commit, review (for comments, requests, approvals), etc.

## For example, get diff of a file might look like this:

## GET /repos/:user/:repo/git/diff

## REST Server: GET request from API for GET /repos/:user/:repo/git/diff

## REST server hands control to REST API, which receives the URL path parameter git diff

## API Gateway: takes the request and send it to lambda, which will get in touch with the database to load data for git diff

## REST API converts that data to json and POST (sends) it to the server.

## REST server sends the json response to client.

## Dependency Description

## Diagram Description automatically generated

## Like the system overview, the dependency diagram showcases a general overshot of the code review system and how it works. It should be noted that all components within the diagram depicted above requires internet for the web browser to function. More importantly, we must also note that GitGoing largely relies on the Git Diff command to track such changes within different files that are uploaded into the system. This is a vital aspect that allows the Code review to work. GitGoing implements this command using the Python/Flask framework, along with many other common Git command in order to process and track changes to review code.

## Interface Description

## Module Interfaces Graphical user interface, application Description automatically generated

## Listed above are the module interface components in GitGoing. A majority of the user interfaces used will require information to be processed through a user authentication system and stored in the database. For example, a client’s login information, notifications, editing history, files, and chat history is required to be stored in the database for the code review to function. Retrieving this data will require a user authentication system to allow access to these services.

## User Interfaces (GUI) Login

## Review Creator

## 

## Review Lister

## 

## Diff Window

## File View

### DETAILED DESIGN

## Module Detailed Design

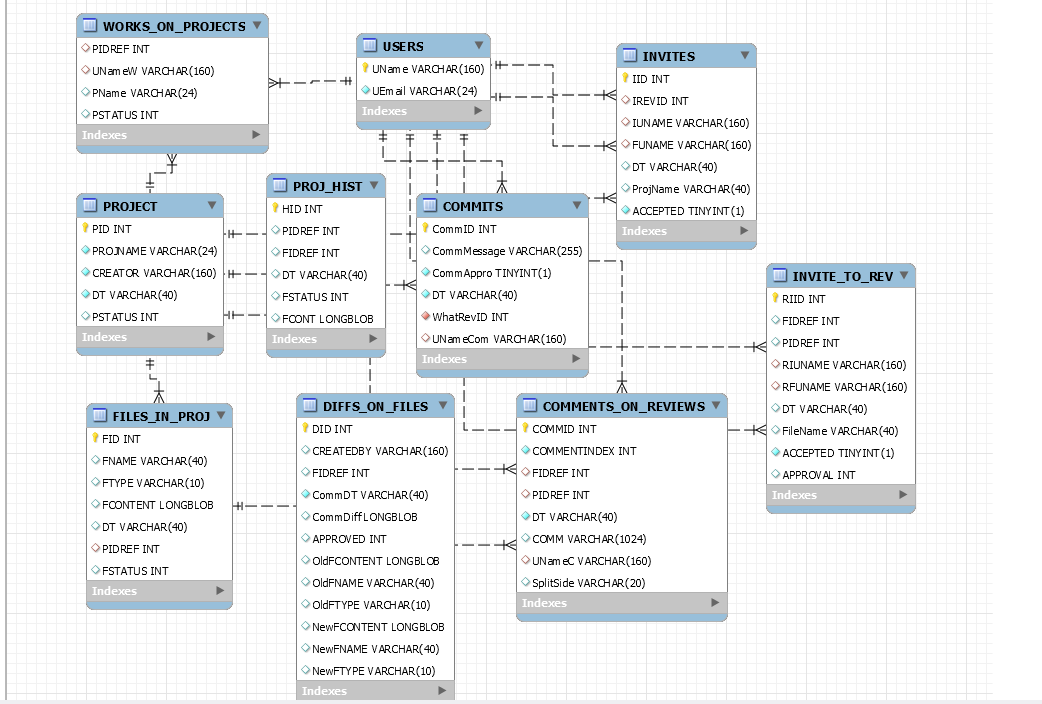
## Diagram Description automatically generated

## Diagram Description automatically generated Diagram Description automatically generated

## Data Detailed Design

Through a MYSQL relational database, the basic design will look something like:

|  |
| --- |
| create table USERS ( |
| UName VARCHAR(160) NOT NULL, |
| UEmail VARCHAR(24) NOT NULL, |
| PRIMARY KEY (UName) |
| ); |
|  |
| create table PROJECT ( |
| PID INT(12) NOT NULL AUTO\_INCREMENT, |
| PROJNAME VARCHAR(24) NOT NULL, |
| CREATOR VARCHAR(160) NOT NULL, |
| DT VARCHAR(40) NOT NULL, |
| PSTATUS INT(2), |
| PRIMARY KEY (PID) |
| ); |
|  |
| CREATE TABLE FILES\_IN\_PROJ ( |
| FID INT(12) NOT NULL AUTO\_INCREMENT, |
| FNAME VARCHAR(40), |
| FTYPE VARCHAR(10), |
| FCONTENT LONGBLOB, |
| DT VARCHAR(40), |
| PIDREF INT(12), |
| FSTATUS INT(2), |
| FOREIGN KEY(PIDREF) REFERENCES PROJECT(PID), |
| primary key(FID) |
| ); |
|  |
| CREATE TABLE WORKS\_ON\_PROJECTS ( |
| PIDREF INT(12), |
| UNameW VARCHAR(160), |
| PName VARCHAR(24), |
| PSTATUS INT(2), |
| FOREIGN KEY(UNameW) REFERENCES USERS(UName), |
| FOREIGN KEY(PIDREF) REFERENCES PROJECT(PID) |
| ); |
|  |
| CREATE TABLE COMMENTS\_ON\_REVIEWS ( |
| COMMID INT(12) NOT NULL AUTO\_INCREMENT, |
| COMMENTINDEX INT(12) NOT NULL, |
| FIDREF INT(12), |
| PIDREF INT(12), |
| DT VARCHAR(40) NOT NULL, |
| COMM VARCHAR(1024), |
| UNameC VARCHAR(160), |
| PRIMARY KEY(COMMID), |
| FOREIGN KEY(FIDREF) REFERENCES FILES\_IN\_PROJ(FID), |
| FOREIGN KEY(PIDREF) REFERENCES PROJECT(PID), |
| FOREIGN KEY(UNameC) REFERENCES USERS(UName) |
| ); |
|  |
| CREATE TABLE COMMITS ( |
| CommID INT(12) NOT NULL AUTO\_INCREMENT, |
| CommMessage VARCHAR(255), |
| CommAppro BOOLEAN NOT NULL DEFAULT '0', |
| DT VARCHAR(40) NOT NULL, |
| WhatRevID INT(12) NOT NULL, |
| UNameCom VARCHAR(160), |
| FOREIGN KEY(UNameCom) REFERENCES USERS(UName), |
| PRIMARY KEY(CommID), |
| FOREIGN KEY(WhatRevID) REFERENCES PROJECT(PID) |
| ); |
|  |
| CREATE TABLE INVITE\_TO\_REV ( |
| RIID INT(12) NOT NULL AUTO\_INCREMENT, |
| FIDREF INT(12), |
| PIDREF INT(12), |
| RIUNAME VARCHAR(160), |
| RFUNAME VARCHAR(160), |
| DT VARCHAR(40), |
| FileName VARCHAR(40), |
| ACCEPTED BOOLEAN NOT NULL DEFAULT '0', |
| APPROVAL INT(2), |
| FOREIGN KEY(RIUNAME) REFERENCES USERS(UName), |
| FOREIGN KEY(RFUNAME) REFERENCES USERS(UName), |
| PRIMARY KEY(RIID) |
| ); |
|  |
| CREATE TABLE DIFFS\_ON\_FILES ( |
| DID INT(12) NOT NULL AUTO\_INCREMENT, |
| CREATEDBY VARCHAR(160), |
| FIDREF INT(12), |
| CommDT VARCHAR(40) NOT NULL, |
| CommDiff LONGBLOB, |
| APPROVED INT(2), |
| OldFCONTENT LONGBLOB, |
| OldFNAME VARCHAR(40), |
| OldFTYPE VARCHAR(10), |
| NewFCONTENT LONGBLOB, |
| NewFNAME VARCHAR(40), |
| NewFTYPE VARCHAR(10), |
| PRIMARY KEY(DID) |
| ); |
|  |
| CREATE TABLE PROJ\_HIST ( |
| HID INT(12) NOT NULL AUTO\_INCREMENT, |
| PIDREF INT(12), |
| FIDREF INT(12), |
| DT VARCHAR(40), |
| FSTATUS INT(2), |
| FCONT LONGBLOB, |
| PRIMARY KEY(HID) |
| ); |
|  |
| CREATE TABLE INVITES ( |
| IID INT(12) NOT NULL AUTO\_INCREMENT, |
| IREVID INT(12), |
| IUNAME VARCHAR(160), |
| FUNAME VARCHAR(160), |
| DT VARCHAR(40), |
| ProjName VARCHAR(40), |
| ACCEPTED BOOLEAN NOT NULL DEFAULT '0', |
| FOREIGN KEY(IUNAME) REFERENCES USERS(UName), |
| FOREIGN KEY(FUNAME) REFERENCES USERS(UName), |
| FOREIGN KEY(IREVID) REFERENCES PROJECT(PID), |
| PRIMARY KEY(IID) |
| ); |



## RTM

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Req # | Requirement | Design Specification | Program Module | Expected Result | Test specification | Test Case Numbers | Successful test verification | Modification of requirement | Remarks |
| 1 | The user can register an account | 3.2.1 – User Registration and login system | User Client, User Authentication System | Account info is registered and saved |  |  |  |  |  |
| 2 | The user can verify account through email | 3.2.1 – User Registration and login system | User Client, User Authentication System | User receives registration email |  |  |  |  |  |
| 3 | The user can login to their account | 3.2.1 – User Registration and login system | User Client, User Authentication System | User is logged in |  |  |  |  |  |
| 4 | The user can reset their password | 3.2.1 – User Registration and login system | User Client, User Authentication System | User’s password is reset |  |  |  |  |  |
| 5 | The user can logout of their account | 3.2.1 – User Registration and login system | User Client, User Authentication System | User is logged out |  |  |  |  |  |
| 6 | The user’s account information is secure | 3.2.1 – User Registration and login system | User Authentication System | User’s account info is securely saved |  |  |  |  |  |
| 7 | The user is notified when assigned to a review | 3.2.2 – Notification System | User Client, Code Review System, Notification System | User receives a notification on review assignment |  |  |  |  |  |
| 8 | The user is notified when a review is approved | 3.2.2 – Notification System | User Client, Code Review System, Notification System | User receives a notification on review approval |  |  |  |  |  |
| 9 | The user is notified when a review is rejected | 3.2.2 – Notification System | User Client, Code Review System, Notification System | User receives a notification on review rejection |  |  |  |  |  |
| 10 | The user is notified of review inquiries | 3.2.2 – Notification System | User Client, Code Review System, Notification System | User receives a notification on review inquiries |  |  |  |  |  |
| 11 | The user is notified of DMs | 3.2.2 – Notification System | User Client, Chat System, Notification System | User receives a notification on incoming DMs |  |  |  |  |  |
| 12 | The user is notified of merge. | 3.2.2 – Notification System | User Client, Chat System, Notification System | User receives a notification if a review they are assigned to / managing is merged |  |  |  |  |  |
| 13 | The user can message other users | 3.2.3 – Chat System | User Client, Chat System | User sends DMs to other users |  |  |  |  |  |
| 14 | The user can add other users to a collaborator’s list | 3.2.3 – Chat System | User Client, Chat System | A collaborator is added |  |  |  |  |  |
| 15 | The user can remove a collaborator to a project | 3.2.3 – Chat System | User Client, Chat System | A collaborator is removed |  |  |  |  |  |
| 16 | The user can block another user | 3.2.3 – Chat System | User Client, Chat System | A user is blocker |  |  |  |  |  |
| 17 | The user can initiate a code review | 3.2.4 – File Change System | User Client, Code Review System | A review is created/ initiated |  |  |  |  |  |
| 18 | The user can assign a reviewer | 3.2.4 – File Change System | User Client, Code Review System | User is assigned to review |  |  |  |  |  |
| 19 | The user is shown a diff between two file versions | 3.2.4 – File Change System | User Client, Code Review System | User is shown a diff for the given file name |  |  |  |  |  |
| 20 | The user can leave inline comments | 3.2.4 – File Change System | User Client, Code Review System | An inline comment is left and visible on the review |  |  |  |  |  |
| 21 | The user can leave comments on the review page (not inline) | 3.2.4 – File Change System | User Client, Code Review System | A comment is left and visible on the review |  |  |  |  |  |
| 22 | Users assigned to reviews can approve changes | 3.2.4 – File Change System | User Client, Code Review System | An approval is left and visible on the review |  |  |  |  |  |
| 23 | Users assigned to reviews can reject changes | 3.2.4 – File Change System | User Client, Code Review System | A rejection is left and visible on the review |  |  |  |  |  |
| 24 | Users assigned to a review can leave inquiries | 3.2.4 – File Change System | User Client, Code Review System | An inquiry is left and visible on the review |  |  |  |  |  |
| 25 | The app can handle a minimum of 10k users | 3.3 – Performance requirements | Server | More than 10k users can register with valid email addresses |  |  |  |  |  |
| 26 | Servers have enough storage space for user information and files | 3.3 – Performance requirements | Database | All users’ info and files are saved in database |  |  |  |  |  |
| 27 | Chat function, notifications and comments should all perform close to real-time | 3.3 – Performance requirements | User Client, Notification System, Chat System, Code Review System | Functions perform close to real time |  |  |  |  |  |

### APPENDICES

*This section is optional.*

Appendices may be included, either directly or by reference, to provide supporting details that could aid in the understanding of the Software Design Document.