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 a database back-end hosting a 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 better 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*

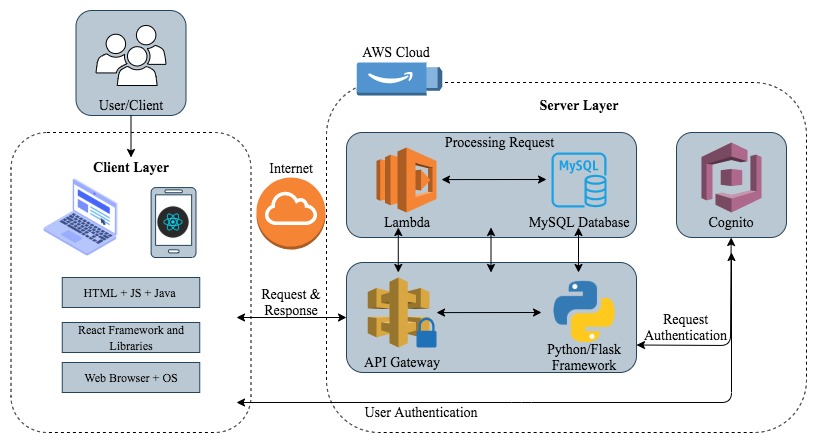
## Client A site user

## 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 HTML, JS, Java, React Frameworks etc. The Server layer will be built using AWS cloud, namely, the API gateway. Cognito, a User Authentication tool within AWS will be used to enable authentication to requests. To process these requests, GitGoing uses Lambda and a MySQL database to do so. 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

## GitGoing is the main component and it orchestrates all other modules whereas git diff the main command whose goal is to track and to compare changes made on a data source such as a file, commit, etc.

## The REST client will send an HTTP request to the REST server in the form of a request message (ours primarily comprises the git diff command). 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) Main Window Review Creator Review Lister Diff Window Account Dropdown Social Dropdown Notification Dropdown

### 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 ( |
| UID INT(12) NOT NULL AUTO\_INCREMENT, |
| UPW VARCHAR(16) NOT NULL, |
| UName VARCHAR(16) NOT NULL, |
| UEmail VARCHAR(24) NOT NULL, |
| URevCnt INT(3), |
| SentMessHist LONGBLOB, |
| RecvMessHist LONGBLOB, |
| PRIMARY KEY (UID) |
| ); |
|  |
| create table REVIEW ( |
| REVID INT(12) NOT NULL AUTO\_INCREMENT, |
| RevUserCnt INT(3) NOT NULL, |
| RevMessages LONGBLOB, |
| CurrRev LONGBLOB NOT NULL, |
| DT DATETIME NOT NULL, |
| PRIMARY KEY (REVID) |
| ); |
|  |
| CREATE TABLE WORKS\_ON ( |
| REVIDREF INT(12), |
| UIDREF INT(12), |
| FOREIGN KEY(REVIDREF) REFERENCES REVIEW(REVID), |
| FOREIGN KEY(UIDREF) REFERENCES USERS(UID) |
| ); |
|  |
| CREATE TABLE COMMITS ( |
| CommID INT(12) NOT NULL AUTO\_INCREMENT, |
| CommMessage VARCHAR(255), |
| CommAppro BOOLEAN NOT NULL DEFAULT '0', |
| CommDiff LONGBLOB, |
| DT DATETIME NOT NULL, |
| WhatRevID INT(12) NOT NULL, |
| PRIMARY KEY(CommID), |
| FOREIGN KEY(WhatRevID) REFERENCES REVIEW(REVID) |
| ); |
|  |
| CREATE TABLE COMMITS\_ON\_REVIEWS ( |
| CommID INT(12), |
| REVID INT(12), |
| CommDT DATETIME NOT NULL, |
| CommDiff LONGBLOB, |
| FOREIGN KEY(CommID) REFERENCES COMMITS(CommID), |
| FOREIGN KEY(REVID) REFERENCES REVIEW(REVID) |
| ); |
|  |
| CREATE TABLE ADMINS ( |
| AID INT(12) NOT NULL AUTO\_INCREMENT, |
| UserCount INT(6), |
| ReviewCount INT(6), |
| VisitCount INT(6), |
| MessageCount INT(6), |
| CommentCount INT(6), |
| PRIMARY KEY(AID) |
| ); |

## RTM

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Req # | Requirement | Design Specification | Program Module | 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 |  |  |  |  |  |
| 2 | The user can verify account through email | 3.2.1 – User Registration and login system | User Client, User Authentication System |  |  |  |  |  |
| 3 | The user can login to their account | 3.2.1 – User Registration and login system | User Client, User Authentication System |  |  |  |  |  |
| 4 | The user can reset their password | 3.2.1 – User Registration and login system | User Client, User Authentication System |  |  |  |  |  |
| 5 | The user can logout of their accoun | 3.2.1 – User Registration and login system | User Client, User Authentication System |  |  |  |  |  |
| 6 | The user’s account information is secure | 3.2.1 – User Registration and login system | User Authentication System |  |  |  |  |  |
| 7 | The user is notified when assigned to a review | 3.2.2 – Notification System | User Client, Code Review System, Notification System |  |  |  |  |  |
| 8 | The user is notified when a review is approved | 3.2.2 – Notification System | User Client, Code Review System, Notification System |  |  |  |  |  |
| 9 | The user is notified when a review is rejected | 3.2.2 – Notification System | User Client, Code Review System, Notification System |  |  |  |  |  |
| 10 | The user is notified of review inquries | 3.2.2 – Notification System | User Client, Code Review System, Notification System |  |  |  |  |  |
| 11 | The user is notified of DMs | 3.2.2 – Notification System | User Client, Chat System, Notification System |  |  |  |  |  |
| 12 | The user can message other users | 3.2.3 – Chat System | User Client, Chat System |  |  |  |  |  |
| 13 | The user can add other users to a friends list | 3.2.3 – Chat System | User Client, Chat System |  |  |  |  |  |
| 14 | The user can remove a friend | 3.2.3 – Chat System | User Client, Chat System |  |  |  |  |  |
| 15 | The user can block another user | 3.2.3 – Chat System | User Client, Chat System |  |  |  |  |  |
| 16 | The user can initiate a code review | 3.2.4 – File Change System | User Client, Code Review System |  |  |  |  |  |
| 17 | The user can assign a reviewer | 3.2.4 – File Change System | User Client, Code Review System |  |  |  |  |  |
| 18 | The user is shown a diff between two file versions | 3.2.4 – File Change System | User Client, Code Review System |  |  |  |  |  |
| 19 | The user can leave inline comments | 3.2.4 – File Change System | User Client, Code Review System |  |  |  |  |  |
| 20 | The user can leave comments on the review page (not inline) | 3.2.4 – File Change System | User Client, Code Review System |  |  |  |  |  |
| 21 | Users assigned to reviews can approve changes | 3.2.4 – File Change System | User Client, Code Review System |  |  |  |  |  |
| 22 | Users assigned to reviews can reject changes | 3.2.4 – File Change System | User Client, Code Review System |  |  |  |  |  |
| 23 | Users assigned to a review can leave inquiries | 3.2.4 – File Change System | User Client, Code Review System |  |  |  |  |  |
| 24 | The app can handle a minimum of 10k users | 3.3 – Performance requirements | Server |  |  |  |  |  |
| 25 | Servers have enough storage space for user information and files | 3.3 – Performance requirements | Database |  |  |  |  |  |
| 26 | 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 |  |  |  |  |  |

### 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.