Chapter-1 Introduction to company

Ishwerdas is a small software company that believes in changing the way the world works. With it's humble origins in a small town named Samrala, this startup aims to change the system by providing excellent services and creating the the most intuitive, user friendly software on web.

Ishwerdas Softwares is set up to make a difference at 3 fronts. Firstly, Ishwerdas softwares is trying to bring a change in education system. Ishwerdas believes in the ideas of Sir Ken Robinson to inches Robinson and alike who believe that the education system has been going into wrong direction for almost a century now. Three inherent properties of humans a.k.a "Diversity-Each human is inherently different", "Curiosity-Each human (child) is born curious", "Creative-Each child is a born artist" are being attacked by education system under the names of "Compliance", "Conformity" and "Discipline". Ishwerdas software believes that teaching happens, when learning happens. Ishwerdas softwares thus believes that a student never fails to learn until teacher fails to teach. Ishwerdas softwares is trying to cure the education system of its disease by providing an alternative education system. First step being change in Industrial training. We have adapted ideas from various schools such as Shaolin education system, Ancient Indian Education System (Gurukuls), improving over them using latest technology and incorporating it in our daily teachings. Ishwerdas softwares will also soon be launching softwares and technology to enhance the basic pre-nursery education. We are having talks with various schools to bring a change in the way teaching happens.

Secondly, we see that many and most of Indian softwares suffer from bad taste. They have a bad user interface, a tasteless user experience and no one seems to care about those. So we,as a group of developers and designers, feel obligated to provide services in such a manner that each software is built at back-end in such a way that it loads fast, saves user's time and in front-end have an excellent and intuitive, emotion invoking interface. Our Industry needs to understand that users are human, and humans are driven by emotions (whether they admit or not). So emotionally intelligent interfaces can not only drive sales high but also provide user a relief and ease. No human deserves a crappy interface with 100 input fields and never ending dropdowns.

Last but not the least, Ishwerdas softwares is set out to be an example of organisation structure. Yes, we agree with benefits of hierarchy and vertical ladder of growth but our managers would rather like to challenge status quo and work with the likes of 37signals, valve and github to create an organisation with flat structure. Ishwerdas is set out to lead by example in this area by becoming one of the very few companies in India by adopting a completely flat organisation structure.

Ishwerdas softwares is yet small but ambitious, and is fearless in choosing the road less traveled by.

Chapter-2 Introduction to project

2.1 Overview

The world calls for efficient programmers who can write a seamless amount of clean and optimized code in the given time-frame. For writing an optimized and error free code, it is very necessary for a programmer to choose among the best code editors. We have quite a few impressive ones out there in the market for open-source peeps like atom, sublime, etc. All of them help in modifying and saving our content in a pretty simple yet efficient manner. Then comes the task of management. In the open source world, source code is meant to be a collaborative effort, where programmers improve upon the source code and share the changes within the community.

Github is one of the best version control repository and Internet hosting service. It offers all of the distributed version control and source code management functionalities such as wikis, task management, and bug tracking and feature requests for every project. Our modifications can be easily uploaded by committing or pushing the files to github. Anyone can review, inspect, modify and enhance your code. These two things are staple in almost every open-source programmer's life.

Edithub further simplifies this two-step process. This high-performance code editor for the web aims at making a programmer's life lot easier but combining these two tasks. It is a fascinating, vivacious developer-friendly editor. Inspired by the way you work, you can fetch, edit and push back your repository to GitHub.

It is easy to use for beginners and highly powerful for advanced users. It comes with syntax highlighting for many languages including PHP, JavaScript, HTML, and CSS. It comes with a very intuitive user interface that makes it super easy to browse files and work on projects. Instead of first cloning or downloading the files, updating the code for bug fix issues, etc. and then pushing it back to your github account, it does all this work in one-go. Once authenticated, you can upload files either from your local machine or via your GitHub account.

After uploading the files, you can open a specific project and view all the files simultaneously in different tabs. The beginner friendly interface has a black screen that supports distraction free editing mode. The turbolinks make the navigation quick, smooth and easy.

This feature-rich editor provides lightweight fuzzy-search library, highlight matching parenthesis and live syntax checker. It provides GitHub omni Authentication, save and push support. It has a drag and drop text using the mouse. It makes it really tough to write a messy code due to its auto-indentation feature. The line-wrapping and code-folding enhance the readability of the code. It enables you to see the preview update of your HTML documents live in a different browser window. This compact editor can handle large files upto the size of 4 million lines.

2.2 User-Requirement analysis

2.2.1 Software Requirement Specification

A software requirements specification (SRS) is a description of a software system to be developed. It lays out functional and nonfunctional requirements, and may include a set of use cases that describe user interactions that the software must provide.

Software requirements specification establishes the basis for an agreement between customers and contractors or suppliers (in market-driven projects, these roles may be played by the marketing and development divisions) on what the software product is to do as well as what it is not expected to do. Software requirements specification permits a rigorous assessment of requirements before design can begin and reduces later redesign. It should also provide a realistic basis for estimating product costs, risks, and schedules. Software requirements specification prevents software projects from failure since it captures complete description about how the system is expected to perform.

An SRS minimizes the time and effort required by developers to achieve desired goals and also minimizes the development cost. A good SRS defines how an application will interact with system hardware, other programs and human users in a wide variety of real-world situations. Parameters such as operating speed, response time, availability, portability, maintainability, footprint, security and speed of recovery from adverse events are evaluated.

2.2.2 SDLC Model

SDLC, Software Development Life Cycle is a process used by software industry to design, develop and test high quality softwares. The SDLC aims to produce a high quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.

A typical Software Development life cycle consists of the following stages:

Stage 1: Planning and Requirement Analysis

Stage 2: Defining Requirements

Stage 3: Designing the product architecture

Stage 4: Building or Developing the Product

Stage 5: Testing the Product

Stage 6: Deployment in the Market and Maintenance

2.2.3 SDLC Iterative Model

EditHub is built using SDLC iterative model. In Iterative model, iterative process starts with a simple implementation of a small set of the software requirements and iteratively enhances the evolving versions until the complete system is implemented and ready to be deployed.

An iterative lifecycle model does not attempt to start with a full specification of requirements. Instead, development begins by specifying and implementing just part of the software, which is then reviewed in order to identify further requirements. This process is then repeated, producing a new version of the software at the end of each iteration of the model.

Iterative Model Design

Iterative process starts with a simple implementation of a subset of the software requirements and iteratively enhances the evolving versions until the full system is implemented. At each iteration, design modifications are made and new functional capabilities are added. The basic idea behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental).

Following is the pictorial representation of Iterative and Incremental model:

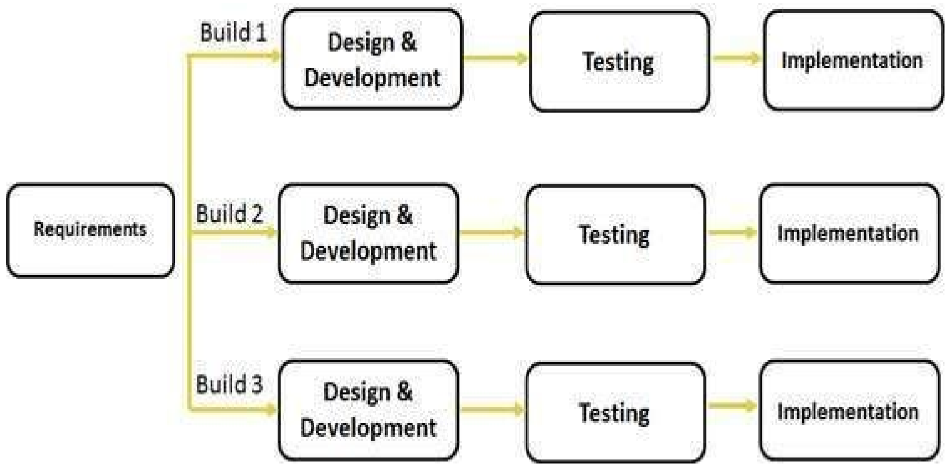


Figure 1: Iterative Model

In incremental model the whole requirement is divided into various builds. During each iteration, the development module goes through the requirements, design, implementation and testing phases. Each subsequent release of the module adds function to the previous release.

The key to successful use of an iterative software development lifecycle is rigorous validation of requirements, and verification & testing of each version of the software against those requirements within each cycle of the model. As the software evolves through successive cycles, tests have to be repeated and extended to verify each version of the software.

2.3 Feasibility Study

For an all new system, the requirement engineering process starts with Feasibility Study. The input to the feasibility study is an outline description of the system and how it will be used. A feasibility study is short, focused study, which aim to answer a number or questions:

- Does the system contribute to the overall objectives of the organization?
- Can the system be implemented using current technology and within cost and schedule constraints?
- Can the system be integrated with other systems that are already in place?

2.3.1 Introduction

A feasibility analysis involves a detailed assessment of the need, value and practicality of a proposed enterprise, such as systems development. The process of designing and implementing recordkeeping systems has sufficient accountability and resource implications for an organization. Feasibility analysis will help you make inform and transparent decision at crucial points during the developmental process to determine whether it is operationally, economically and technically realistic to produce with a particular course of action.

2.3.2 Cost Effectiveness

The cost encompasses both designing and installing the system. It includes user training, updating the physical facilities & documenting. System Performance criteria are evaluated against the cost of each system to determine which system is likely to be the most cost effective and also meets the performance requirements. Cost are most easily determine when the receipts if the systems are tangible and measurable. An additional factor to consider is the cost of the study design and requirements. Each candidate systems performance is evaluated against the system performance requirements set prior to the feasibility study. Whatever the Criteria, there has to be as close a match as practicable, although trade-off are often necessary to select the best system. In weight system performance each candidate system is weighted by it performance and cost data by applying a rating gure. Then candidate system with the highest total score is selected. The procedure for weighing candidate system is simple:

1. Assign a weighting factor to each evaluation criterion based on the criterias effect on the success of the system.
2. Assign a quantitative rating to each criterion qualitative rating.
3. Multiply the weight assigned to each category by the relative rating to determine the score.
4. Sum the score column for each candidate system.

2.4.3 Information sources

To find out the information about our project is called Information gathering. It is an art and science. The approach and manner in which the information is gathered required person with sensitivity, common science and knowledge of what and when to gather and what channels to use in secure the information. The methodology and tools for information gathering required training and the experience that the analyst is expected to have. This means that information gathering is neither easy nor routine. Much preparation, experience and training is required. The phases for information gathering while system analysis are:

1. Familiarity with the present system through the available information such as procedure manuals, documents and their flow, interviews with staff and on side observation.
2. Definition of the decision-making is associated with managing the system. Conducting interviews clarifies the decision point and how decisions are made in user area.
3. Once decision points are defined, a series of interviews conducted define the information requirements of user. The information is analyzed and document.

2.4.4 Time Feasibility

As you study above what is meant by feasibility study, the Time Feasibility is the sub point of it. It means the study of time period, which is required for the system. Using this time feasibility analyst will design a proper life cycle of the system. In this life cycle it is mentioned that at which time the system will be ready to use. This Time Feasibility will be at the initial stage of system. The CBA time period should match the system life cycle. The system life cycle ends when the system is terminated or is replaced by a system that has significant differences in processing, operational capabilities, resource requirement, or system outputs. Significant differences is a very subject term, and some organization may feel that a 10% change is significant, while others may be that the change must be over 30% to be significant.

2.4.5 Software & Hardware availability

When you design a system at that stage there are some requirements for the systems, for that you should study on your system. While studying you determine what are the requirements for your system. There are two types of requirements Software & Hardware requirements. For example in database project they require database software for storing the data and any big game will require some amount of RAM for graphics card is called Hardware requirements.

Steps in feasibility analysis:

1. **Prepare system flowcharts**: Information oriented charts and data flow diagrams prepared in the initial investigation are reviewed at this time. The charts bring up the importance of the

input, output and data flow among key points in the existing system. All other flowcharts needed for detailed evaluation are completed at this point.

2. **Enumerate potential candidate systems**: This step identifies the candidate system that is capable of producing the outputs included in the generalized flowcharts. This requires a transformation from logical to physical system models. Another aspect of this step is consideration of the hardware that can handle total system requirements.
3. **Describe and identify characteristics of candidate system**: From the candidate systems considered, the team begins a preliminary evaluation in an attempt to reduce them to a manageable number. The information along with additional data available through the vendor highlights the positive and negative features of the system. The constraints unique to each system are also specified.
4. **Determine and evaluate performance and cost-effectiveness of each candidate system**: Each candidate systems performance is evaluated against the system performance requirements set prior to the feasibility study. Whatever the Criteria, there has to be as close a match as practicable, although trade-off are often necessary to select the best system. The cost encompasses both designing and installing the system. It includes user training, updating the physical facilities & documenting. System Performance criteria are evaluated against the cost of each system to determine which system is likely to be the most cost effective and also meets the performance requirements. Cost are most easily determine when the receipts if the systems are tangible and measurable. An additional factor to consider is the cost of the study design and requirements.
5. **Weight system performance and cost data**: In this step each candidate system is weighted by its performance and cost data by applying a rating figure. Then the candidate system with the highest total score is selected. The procedure for weighing candidate system is simple:
    a. Assign a weighting factor to each evaluation criterion based on the criterias effect on the success of the system.
    b. Assign a quantitative rating to each criterion qualitative rating.
6. **Select the best candidate system**: The system with the highest total score is judged the best system.This assumes the weighing factors are fair and rating of each evaluating criteria is accurate. Most feasibility study select from more candidate systems. The criteria chosen and the constraints are also more complex. In any case, management cooperation and comments, however, are encouraged.

7. **Feasibility report**: The feasibility report is a formal document for management use, brief enough and sufficiently non-technical to be understandable, yet detailed enough to provide the basis for system design. There is no standard format for preparing feasibility reports. Analyst usually decides on a format that suits the particular user and the system.

2.4 Objectives of project

- **High Performance editor:** This lightweight editor launches quickly and gives rich editing features like automatic indentation, highlighting matching parentheses, line wrapping, drag and drop text using the mouse, code folding, capability to handle huge documents etc.
- **Smart Syntax Highlighting:** This feature displays text, especially source code, in different colors according to the category of terms for many languages like javascript, ruby on rails, HTML, CSS, etc.. It facilitates writing in a structured language such that both structures and syntax errors are visually distinct. It provides a strategy to improve the readability and context of the text; especially for code that spans several pages. The reader can easily ignore large sections of comments or code, depending on what they are looking for.
- **Save and push support:** It allows the user to save the changes made in the files in his/her local machine easily. It also gives you backup by pushing it to GitHub in a simple two-step process the instant you're done modifying your code. This gives you the modified file at both, GitHub and your local machine, making it hard to loose the data.
- **Fuzzy lightweight search:** It provides the technique of finding strings that match a pattern approximately (rather than exactly). The problem of approximate string matching is typically divided into two subproblems: finding approximate substring matches inside a given string and finding dictionary strings that match the pattern approximately.
- **User Authentication:** Allowing users to login with multiple authentication providers brings great benefits but also results in some annoying edge cases. For example, when they login with one provider, logout and then login with another. This is managing session data since a logged in user is simply a person who has some session data confirming that they have been logged in. The OmniAuth callback which a provider will redirect to upon authenticating a user it is created to be powerful, flexible, and do as little as possible. We used omniAuth to authenticate users via disparate systems.

Chapter-3 Product Design

3.1 Product Perspective

Our project's perspective is to make the life of an open-source-developer easy. Github is one of the best version control repository and Internet hosting service. It offers all of the distributed version control and source code management functionalities such as wikis, task management, and bug tracking and feature requests for every project. Our modifications can be easily uploaded by committing or pushing the files to github. Anyone can review, inspect, modify and enhance your code. However, cloning or downloading the files first to your local machine, then editing and then pushing it becomes a tedious three-step process. Thus comes the EditHub. This beginner friendly interface has a black screen that supports distraction free editing mode. The turbolinks make the navigation quick, smooth and easy. This feature-rich editor provides lightweight fuzzy-search library, highlight matching parenthesis and live syntax checker. It provides GitHub omni Authentication, save and push support. It has a drag and drop text using the mouse.

There are many efficient editors out there on the web, but none of them have push support incorporated. And it comes extremely handy to all the open-source-developers for whom building software is as much about managing people as it is about code. We intended to combine these staple tasks so that one can simply open the file of the concerned project, modify it, get a live preview of it and push it directly.

3.2 Product Functions

EditHub is designed to fulfill the following functions:

- **Fuzzy lightweight search**: It provides the technique of finding strings that match a pattern approximately (rather than exactly). The problem of approximate string matching is typically divided into two subproblems: finding approximate substring matches inside a given string and finding dictionary strings that match the pattern approximately. For example, if you search for a 'phont' instead of 'font', it gives the results for 'font' only.
- **Syntax Highlighting**: This feature displays text, especially source code, in different colors according to the category of terms for many languages like javascript, ruby on rails, HTML, CSS, etc.. It facilitates writing in a structured language such that both structures and syntax errors are visually distinct. It provides a strategy to improve the readability and context of the text; especially for code that spans several pages. The reader can easily ignore large sections of comments or code, depending on what they are looking for.

- **Easy file upload and modification**: It allows the user to upload the files which he/she desires to modify in a simple manner from both, the local machine as well as GitHub account. The uploaded files can be viewed and changed as per the requirement. You can also get a live preview of your HTML files in a different browser window.
- **Save and push support**: It provides with the save option for the updated file in the local storage and direct push option to the GitHub. It stores the modified file at both, the local machine as well as GitHub account. GitHub shows all the commit histories with a brief about the changes made to each file. Thus, it makes it easier to track all the why's and what's of changes of the modified file.Automatic Indentation:
- **GitHub Omniauth**: It was created to be powerful, flexible, and do as little as possible. OmniAuth can authenticate users via disparate systems. It provides the user multiple ways to become authenticated.
- **Code folding**: It allows the user to selectively hide and display – "fold" – sections of a currently-edited file as a part of routine edit operations. This allows the user to manage large amounts of text while viewing only those subsections of the text that are specifically relevant at any given time. Identification of folds can be automatic, most often based on the syntax of the computer language being used. It becomes really useful to manage source code files and in data comparison to only view the changed text.

3.3 Constraints

A constraint is any restriction that defines a project's limitations; the scope, for example, is the limit of what the project is expected to accomplish. The three most significant project constraints -- schedule, cost and scope -- are sometimes known as the triple constraint or the project management triangle. A project's scope involves the specific goals, deliverables and tasks that define the boundaries of the project. The schedule (sometimes stated more broadly as time) specifies the timeline according to which those components will be delivered, including the final deadline for completion. Cost (sometimes stated more broadly as resources) involves the financial limitation of resources input to the project and also the overall limit for the total amount that can be spent.

Project constraints are also considered to be somewhat mutually exclusive. In the project management triangle, it is assumed that making a change to one constraint will affect one or both of the others. For example, increasing the scope of the project is likely to require more time and money.

GitHub has a vast community which creates well-maintained docs and make sure they receive the high level of care they deserve. And cloning or downloading those files to your machine first, then modifying, then push it back, takes a lot of storage and uploading time. For our team which wants to expand the open-source world, it had been an issue for a while now. It required a lot of brain-racking algorithms and logics since they were one of a kind. The main constraints faced during EditHub were largely because no such system exists yet. Existing systems give a glimpse of what-s, why-s and how-s of the project. It gives an insight into what has already been accomplished and where it is going wrong so that we can enhance it to reach a wider audience.

3.4 Data Flow Diagram/ Data Dictionary

3.4.1 Data Flow Diagram

A data-flow diagram (DFD) is a graphical representation of the flow of data. The purpose of DFD is to clarify system requirements and identify major transformations that will become programs in system design. So it is the starting point of the design phase that functionally decomposes the requirements specifications to the lowest level in detail. These diagrams help to understand the basic working of the system. It helps to make and recognize various parts and their inter relationships. It is a way of expressing system requirement in a graphical form; this leads to a modular design. It is also known as bubble chart. A DFD consists of series of bubbles joined by lines represent data transformations and the lines represent data flows in the system. The various DFD's are :

DFD level-0: DFD Level 0 is also called a Context Diagram. It is a basic overview of the whole system or process being analyzed or modeled. It is designed to be an at-a-glance view, showing the system as a single high-level process, with its relationship to external entities.

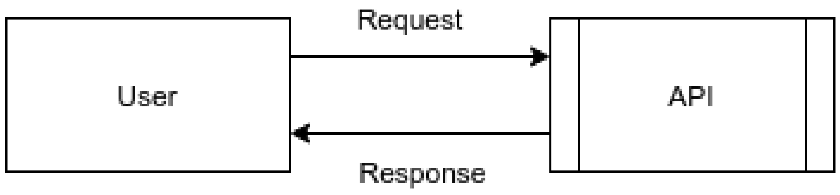


Figure 2 : DFD level-0

DFD level-1 : DFD level-1 provides a more detailed breakout of pieces of the Context Level Diagram. You will highlight the main functions carried out by the system, as you break down the high-level process of the Context Diagram into its subprocesses. The EditHub is broken into several sub modules:

- Upload via Github or your local machine
- Modify the file
- Commit changes
- Push to Github.

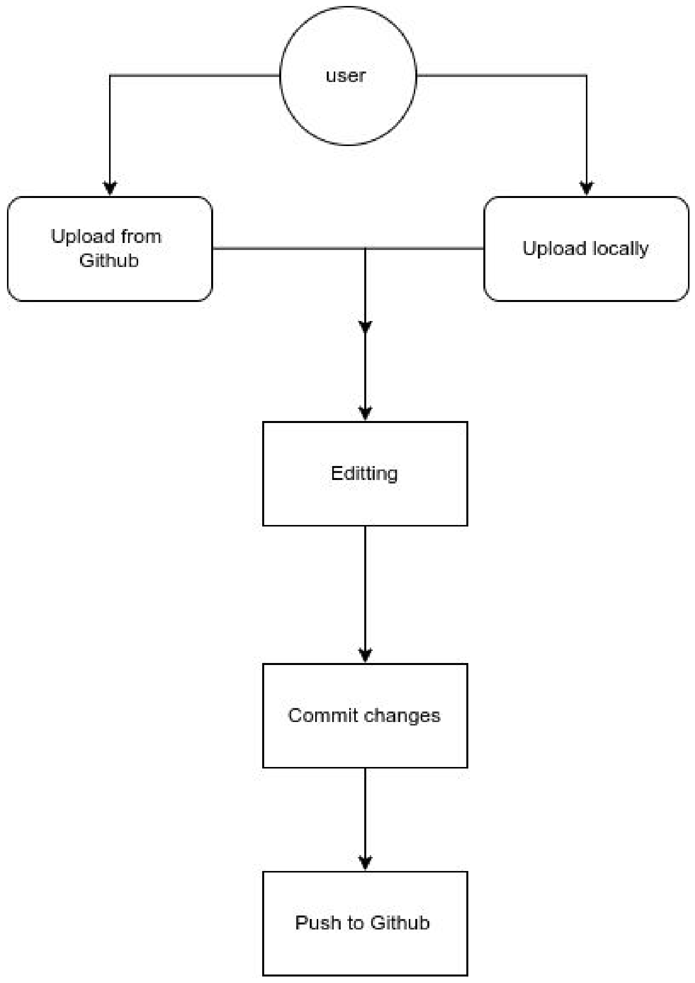


Figure 3 : DFD level-1

3.4.2 Data Dictionary

A data dictionary is a collection of descriptions of the data objects or items in a data model for finding what type of data objects to be used in system during development life cycle. A first step in analyzing a system of objects with which users interact is to identify each object and its relationship to other objects. This process is called data modeling and results in a picture of object relationships. After each data object or item is given a descriptive name, its relationship is described (or it becomes part of some structure that implicitly describes relationship), the type of data (such as text or image or binary value) is described, possible predefined values are listed, and a brief textual description is provided. This collection can be organized for reference into a book called a data dictionary.

Data dictionaries are important In order to manage the details in large-scale systems. Most systems are ongoing and dynamic and management of all the descriptive details is difficult, therefore an accurate and consistent recording technique is essential. It is necessary to communicate a common meaning for all of the elements in the system. Simply making sure that for all elements, the meaning will remain consistent. It is used to document features of the system. It also helps locate errors and omissions in the system.

Data Definition

Composite data items can be defined in terms of primitive data using the following data definition operators.

1. + : Denotes composition of two data items, e.g. a + b represents data a and b.
2. [,,] : Represent selection, i.e. any one of the data items listed inside the square bracker can occur.
3. ( ) : The contents inside the bracket represent optional data which may or may not appear.
4. { } : Represent iterative data definition e.g. name5 represent five name data.

3.4.3 Flow Chart

Flowcharts are maps or graphical representations of a process. Steps in a process are shown with symbolic shapes, and the flow of the process is indicated with arrows connecting the symbols. his diagrammatic representation illustrates a solution model to a given problem. Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields. Like other types of diagrams, they help visualize what is going on and thereby help understand a process, and perhaps also find flaws, bottlenecks, and other less-obvious features within it. This technique allows the author to locate the responsibility for performing an action or making a decision correctly, showing the responsibility of each organizational unit for different parts of a single process.

The flowchart for EditHub is depicted below:

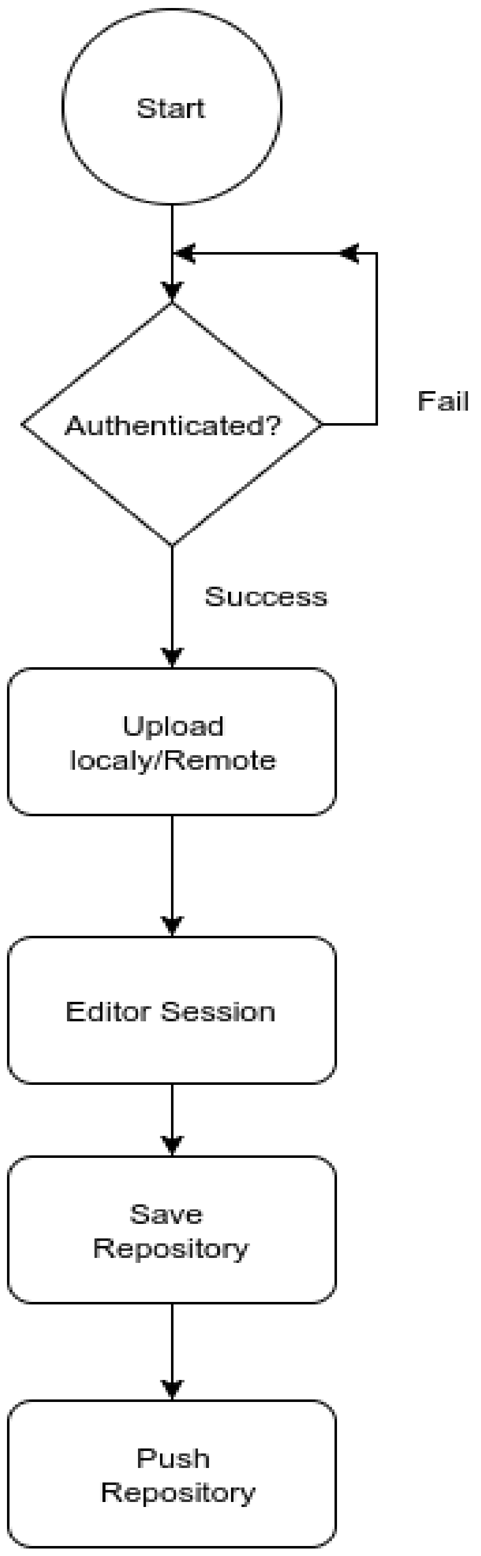


Figure 4: Flowchart for EditHub

3.4.4 Use Case model

The EditHub serves two main functionalities : Login and edit, and upload and push. The use case models for EditHub are depicted below:

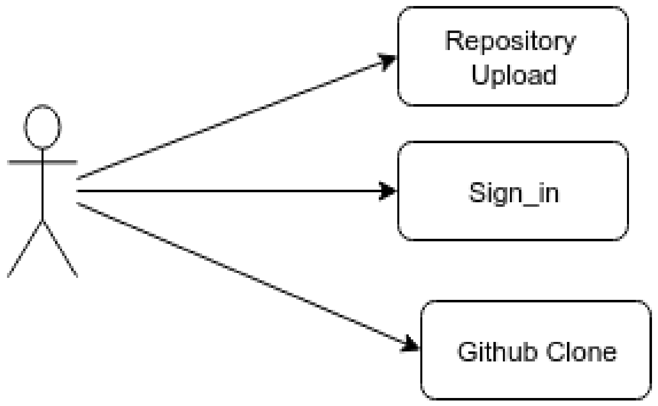


Figure 5: Use case model-1

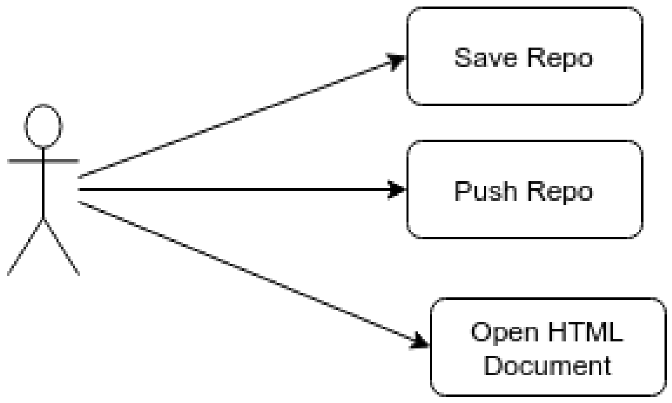
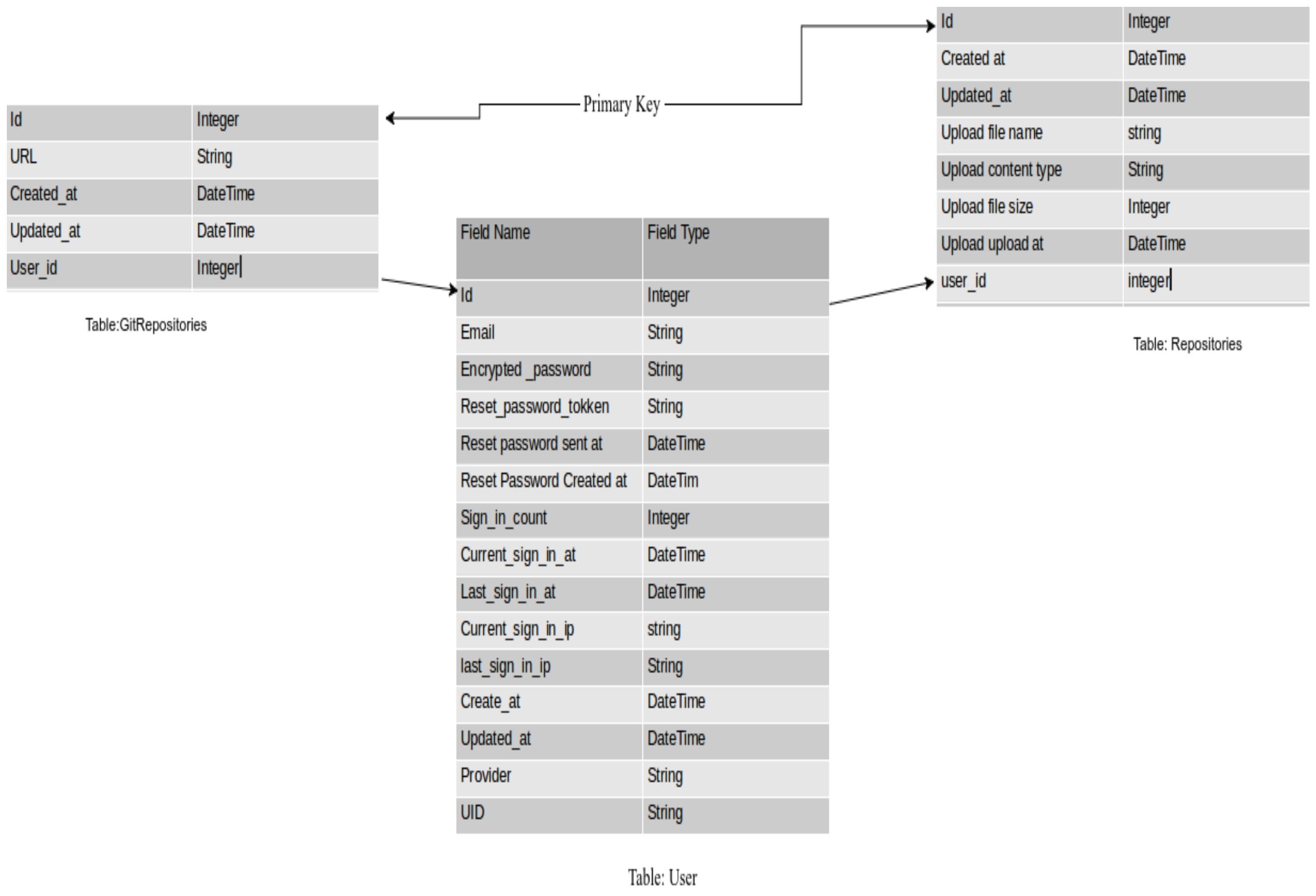


Figure 6: Use case model-2

3.5 Database Design

EditHub uses SQLite because it helps make good and beautiful products that are fast, reliable, and simple to use. In contrast to many other database management systems, SQLite is not a client–server database engine. Rather, it is an embedded SQL database engine. Unlike most other SQL databases, SQLite does not have a separate server process. SQLite reads and writes directly to ordinary disk files. A complete SQL database with multiple tables, indices, triggers, and views, is contained in a single disk file. The database file format is cross-platform - you can freely copy a database between 32-bit and 64-bit systems or between big-endian and little-endian architectures. These features make SQLite a popular choice as an Application File Format. Think of SQLite not as a replacement for Oracle but as a replacement for fopen().

While basic queries are almost universal, there notable are nuances between MySQL, PostgreSQL, Microsoft SQL Server, Oracle Database, etc. But what's particularly notable about SQLite is that unlike all the others mentioned above, this database software doesn't come with a daemon that queries are passed through. This means that if multiple processes are using the database at once, they will be directly altering the data through the SQLite library and making the read / write data calls to the OS themselves. It also means that the locking mechanisms don't deal with contention very well. It provides with a very easy data retrieval which makes it worth it.



Id	Integer
URL	String
Created_at	DateTime
Updated_at	DateTime
User_id	Integer

Field Name	Field Type
Id	Integer
Email	String
Encrypted _password	String
Reset_password_tokken	String
Reset password sent at	DateTime
Reset Password Created at	DateTim
Sign_in_count	Integer
Current_sign_in_at	DateTime
Last_sign_in_at	DateTime
Current_sign_in_ip	string
last_sign_in_ip	String
Create_at	DateTime
Updated_at	DateTime
Provider	String
UID	String

Id	Integer
Created at	DateTime
Updated_at	DateTime
Upload file name	string
Upload content type	String
Upload file size	Integer
Upload upload at	DateTime
user_id	integer

Figure 7: Database design

3.6 Table Structure

Edithub uses three tables:

- Git Repos : Keeps track of the GitHub account URL of the user
- Repositories : Keeps track of the repositories uploaded by the user for modification
- Users : Keeps track of all the authenticated users

The table structure of each is depicted below:

Git Repos

Field name	**Field type**
Id	Integer

Url	String
Created_at	DateTime
Updated_at	DateTime
user_id	integer(Foreign Key)

Repositories

Field Name	**Field Type**
Id	Integer(Primary Key)
Created_at	DateTime
Updated_at	DateTime
Upload_file_name	String
Upload_content_type	String
Upload_file_size	Integer
Upload_updated_at	DateTime
User_id	Integer(Foreign Key)

Users

Field Name	**Field Type**
Id	Integer(Primary Key)
Email	String(Foreign Key)
Encrypted_password	String
Reset_password_tokken	String(Foreign Key)
Reset_password_sent_at	DateTime

Reset_password_created_at	DateTime
Sign_in_count	integer
Curent_sign_in_at	DateTime
Last_sign_in_at	DateTime
Current_sign_in_ip	String
last_sign_in_ip	String
Created_at	DateTime
Updated_at	DateTime
Provider	String
Uid	String

3.7 E-R Diagrams

Entity relationship model defines the conceptual view of database. It works around real world entity and association among them. At view level, ER model is considered well for designing databases. An entity-relationship diagram (ERD) is a graphical representation of an information system that shows the relationship between people, objects, concepts or events within that system. An ERD is a data modelling technique that can help define business process and can be use as the foundation for a relational database. While useful for organizing data that can be represented by a relational structure, an entity-relationship diagram cannot sufficiently represent semi-structured or unstructured data, and an ERD is unlikely to be helpful on its own in integrating data into a pre-existing information system.

Three main components of an ERD are the entities, which are objects or concepts that can have data stored about them, the relationship between those entities, and the cardinality which defines that relationship in terms of numbers.

1. **Entity Set:** An entity set is a collection of similar types of entities. Entity set may contain entities with attribute sharing similar values. For example, Students set may contain all the student of a school; likewise Teachers set may contain all the teachers of school from all

faculties. Entities sets need not to be disjoint. A weak entity is an entity that depends on another entity. Weak entity does not have a key attribute of their own. Double rectangle represents weak entity.

2. **Attributes:** Entities are represented by means of their properties, called attributes. All attributes have values. For example, a student entity may have name, class, age as attributes.
3. **Relationship:** Relationship are meaningful associations between or among entities. They are usually verbs, e.g. assign, associate, or track. A relationship provides useful information that could not be discerned with just the entity types.

E-R diagrams of EditHub are as follows-

- User has many repositories:

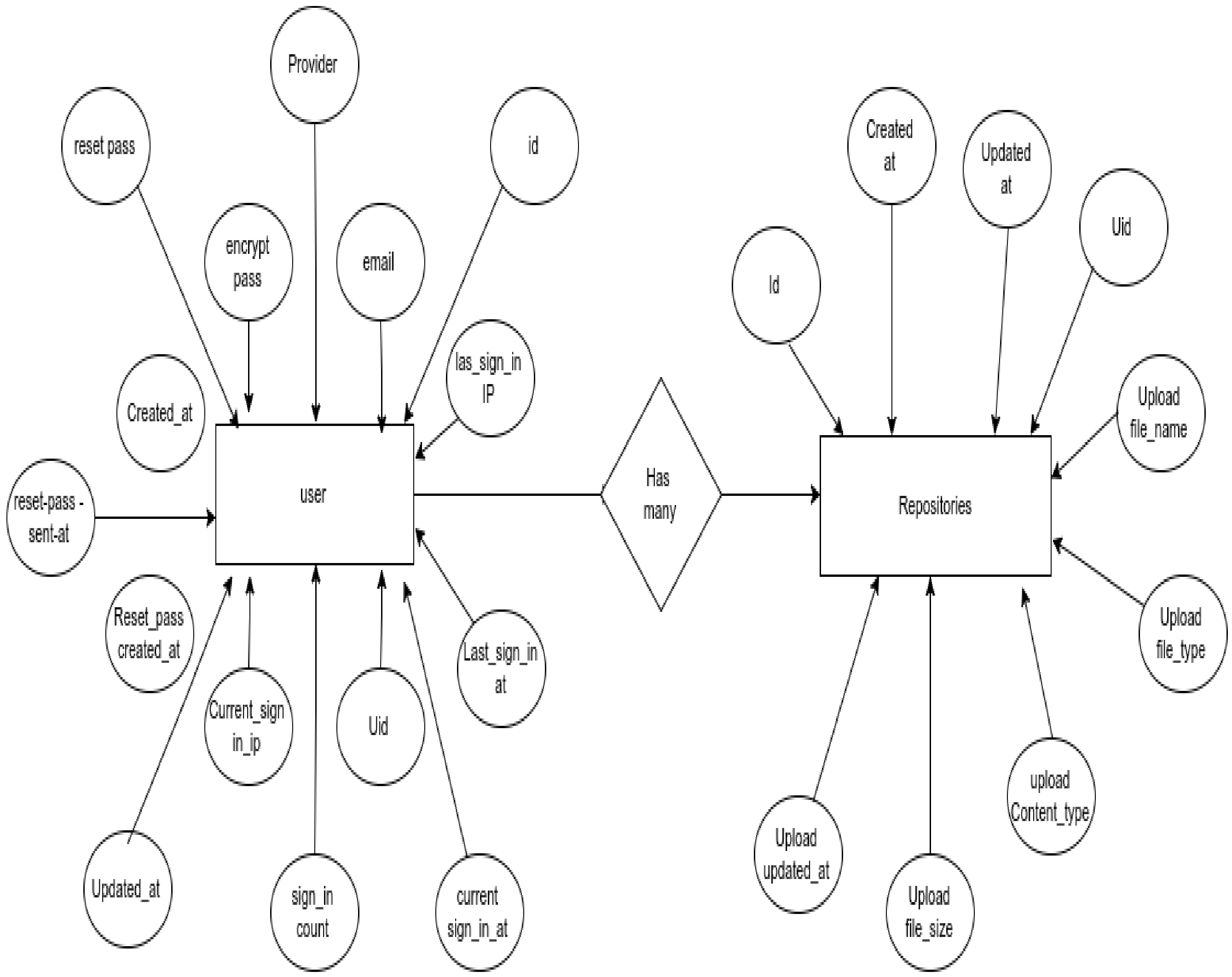


Figure 8: User to repositories E-R diagram

- User has many git repositories:

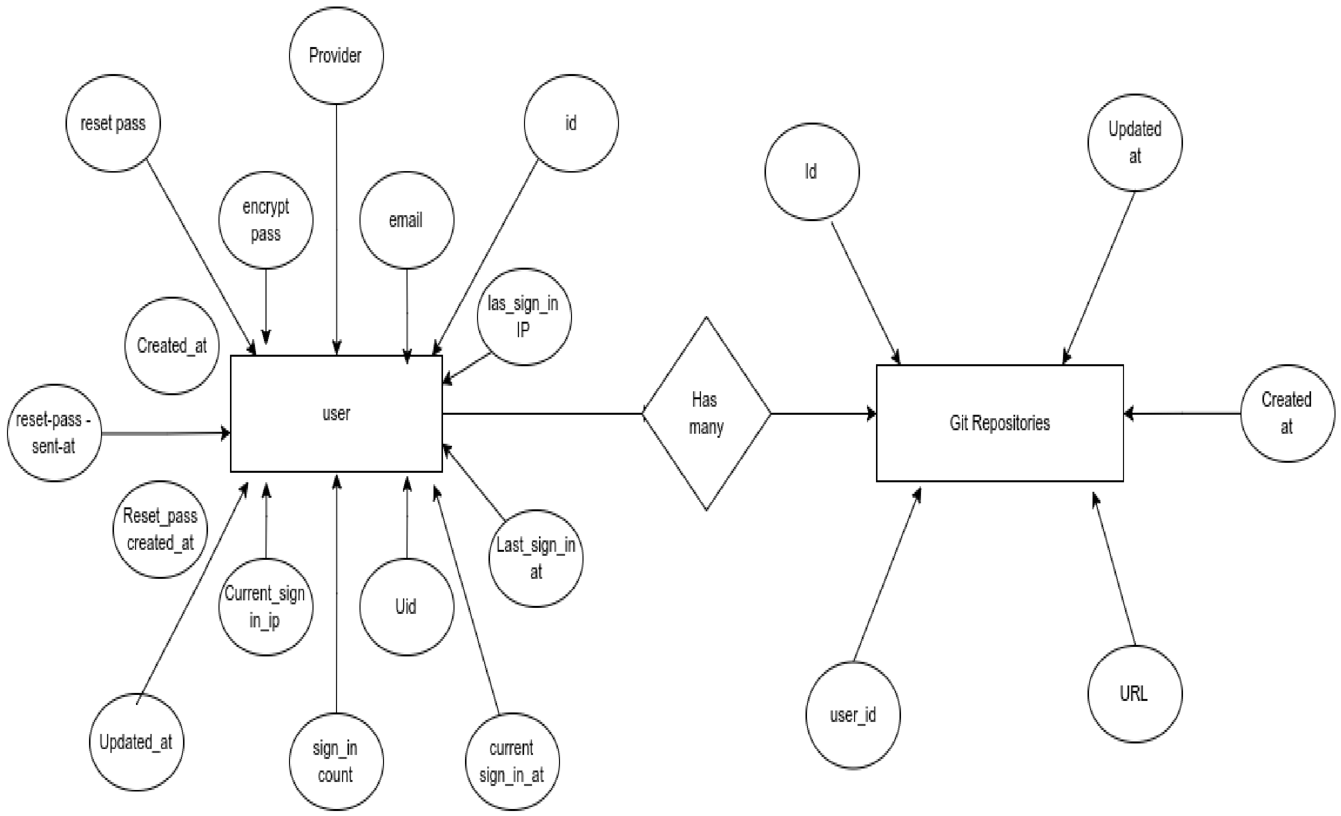


Figure 9: User to Git repositories E-R diagram

- Repositories belong to the user:

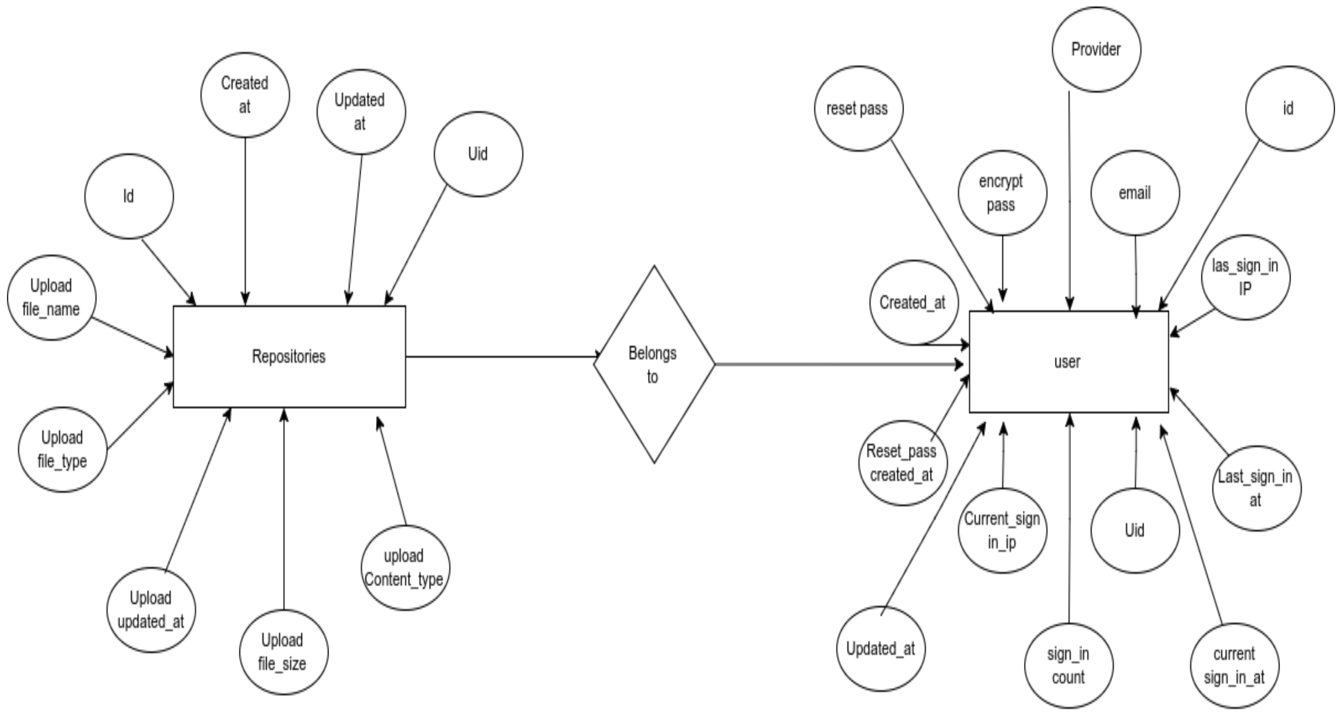


Figure 10: Repositories to user E-R diagram

- Git repositories belong to the user:

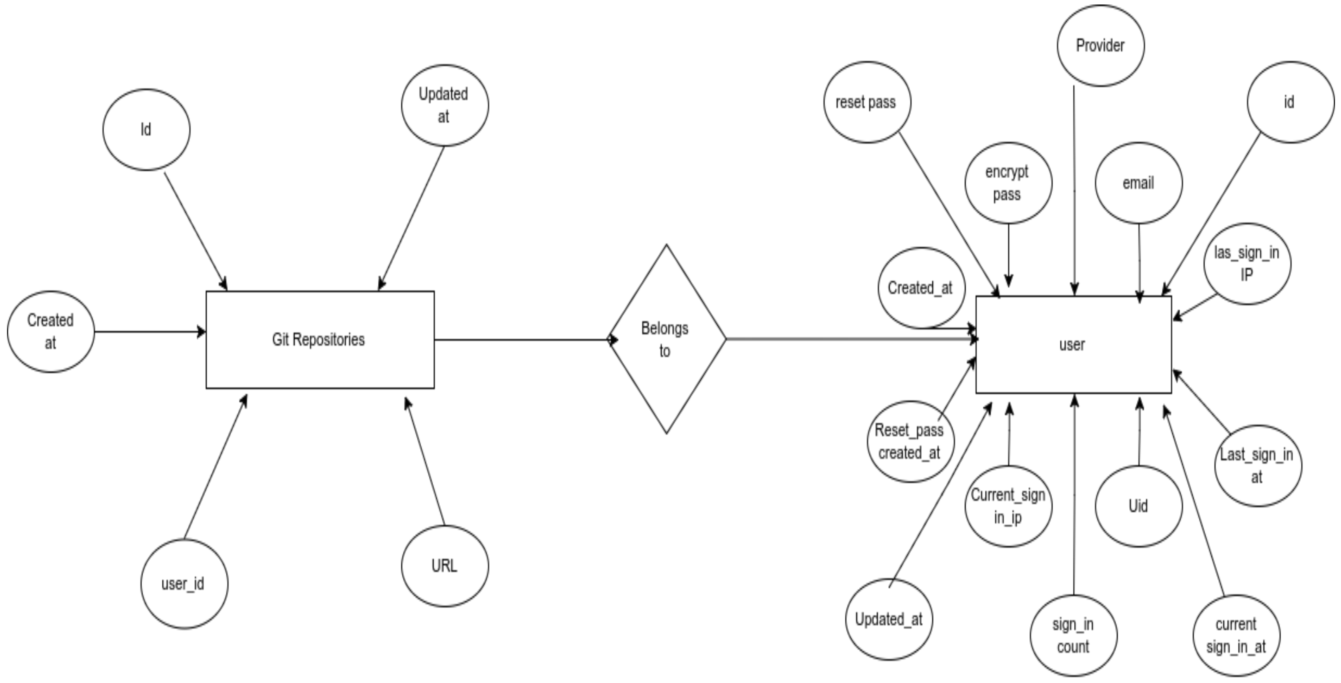


Figure 11: Git repositories to user E-R diagram

3.8 Specific Requirements

3.8.1 Software Requirements

1. **Operating System:** Windows, Linux OS
2. **Backend Software:** Puma Server
3. **Frontend Software:** Bundler package manager, JavaScript, Ruby on Rails, SQLite
4. **Browser:** Any modern web browser e.g. Google Chrome, Mozilla Firefox, Opera
5. **Text Editor:** Any text editor for editing html, css, javascript and ruby on rails files. Atom text editor is recommended.

3.8.2 Hardware Requirements

1. **Processor:** Intel i3 processor or above for local machine
2. **RAM:** 2 GB RAM or above for local machine
3. **HDD:** Minimum 5 GB of free Hard drive space
4. **Supported Architecture:**
    a. 32-bit(X86)
    b. 64-bit(X64)
5. **Another Devices**: Mouse, keyboard, modem (with internet connection)

3.8.3 Maintainability Requirement

Maintainability is the ease with which changes can be made to a website. These changes may be necessary for the correction of faults, adaptation of the system to a meet a new requirement, addition of new functionality, removal of existing functionality or corrected when errors or deficiencies occur and can be perfected, adapted or action taken to reduce further maintenance costs. It represents the cost required to maintain the website like yearly charges of web hosting and domain apart from this the updation charges of developer for new modifications in website. The typical objectives of a maintainability requirement are to:

1. Ensure that minor defects in an application or component are easy to correct.
2. Ensure that minor enhancements to an application or component are relatively easy to implement.
3. Minimize maintenance costs.
4. Minimize maintenance organization staffing needs.

3.8.6 Security Requirement

EditHub uses Devise for its security purposes. It is a flexible authentication solution for Rails based on Warden. It is Rack based and is a complete MVC solution based on Rails engines. It allows you to have multiple models signed in at the same time and is based on a modularity concept: use only what you really need. It's composed of the following modules:

- Database Authenticatable: hashes and stores a password in the database to validate the authenticity of a user while signing in.
- Omniauthable: adds OmniAuth (https://github.com/omniauth/omniauth) support.
- Recoverable: resets the user password and sends reset instructions.
- Registrable: handles signing up users through a registration process, also allowing them to edit and destroy their account.
- Rememberable: manages generating and clearing a token for remembering the user from a saved cookie.
- Trackable: tracks sign in count, timestamps and IP address.
- Timeoutable: expires sessions that have not been active in a specified period of time.
- Lockable: locks an account after a specified number of failed sign-in attempts. Can unlock via email or after a specified time period.
- Confirmable: sends emails with confirmation instructions and verifies whether an account is already confirmed during sign in.
- Validatable: provides validations of email and password. It's optional and can be customized.

Chapter 4 Development and Implementation

4.1 Introduction to languages

4.1.1 Ruby on rails

It is a web application development framework written in the Ruby language. It is designed to make programming web applications easier by making assumptions about what every developer needs to get started. It allows you to write less code while accomplishing more than many other languages and frameworks. Experienced Rails developers also report that it makes web application development more fun.It makes the assumption that there is a best way to do things, and it's designed to encourage that way - and in some cases to discourage alternatives.

It is an open source ruby framework for developing database-backed web applications, using the Model-View-Controller pattern.

In Ruby, everything is an object. Every bit of information and code can be given their own properties and actions. Object-oriented programming calls properties by the name instance variables and actions are known as methods. Ruby's pure object-oriented approach is most commonly demonstrated by a bit of code which applies an action to a number. For example:

```
5.times{ print “We *love* Ruby - it’s beautiful, artful, handy and practical!” }
```

In many languages, numbers and other primitive types are not objects. Ruby follows the influence of the Smalltalk language by giving methods and instance variables to all of its types. This eases one's use of Ruby, since rules applying to objects apply to all of Ruby.

This fast and portable language has a wealth of other features, among which are the following:

- **Flexibility**: Ruby is seen as a flexible language, since it allows its users to freely alter its parts. Essential parts of Ruby can be removed or redefined, at will. Existing parts can be added upon. Ruby tries not to restrict the coder. Ruby's block are also seen as a source of great flexibility. A programmer can attach a closure to any method, describing how that method should act. The closure is called a block and has become one of the most popular features for newcomers to Ruby from other imperative languages like PHP or Visual Basic.
- **Don't Repeat Yourself:** DRY is a principle of software development which states that "Every piece of knowledge must have a single, unambiguous, authoritative representation within a system." By not writing the same information over and over again, our code is more maintainable, more extensible, and less buggy.
- **Convention Over Configuration:** Rails has opinions about the best way to do many things in a

web application, and defaults to this set of conventions, rather than require that you specify minutiae through endless configuration files.

- **Built-in testing**: Rails gives you three default environments: development, testing, and production. Each behaves slightly differently, making your entire software development cycle easier. For example, Rails creates a fresh copy of the Test database for each test run. So, it creates simple automated tests you can then extend. It also provides supporting code called harnesses and fixtures that make test cases easier to write and run. Ruby can then execute all your automated tests with the rake utility.
- **Ruby and the Mixin**: Unlike many object-oriented languages, Ruby features single inheritance only, on purpose. But Ruby knows the concept of modules (called Categories in Objective-C). Modules are collections of methods. Classes can mixin a module and receive all its methods for free. For example, any class which implements the each method can mixin the Enumerable module, which adds a pile of methods that use each for looping. Example:

  ```
  class MyArray
  include Enumerable
  end
  ```
- **Easy visual appearance**: While Ruby often uses very limited punctuation and usually prefers English keywords, some punctuation is used to decorate Ruby. Ruby needs no variable declarations. It uses simple naming conventions to denote the scope of variables:
  - Var could be a local variable.
  - @var is an instance variable.
  - $var is a global variable.

These sigils enhance readability by allowing the programmer to easily identify the roles of each variable.

Installation:

Dependencies for ruby:

```
sudo apt-get update
sudo apt-get install git-core curl zlib1g-dev build-essential libssl-dev libreadline-dev libyaml-dev libsqlite3-dev sqlite3 libxml2-dev libxslt1-dev libcurl4-openssl-dev python-software-properties libffi-dev nodejs
```

Rvm:

```
sudo apt-get install libgdbm-dev libncurses5-dev automake libtool bison libffi-dev
gpg--keyserver hkp://keys.gnupg.net --recv-keys
409B6B1796C275462A1703113804BB82D39DC0E3
curl -sSL https://get.rvm.io | bash -s stable
source ~/.rvm/scripts/rvm
rvm install 2.4.0
rvm use 2.4.0 --default
ruby -v
```

Nodejs:

```
curl -sL https://deb.nodesource.com/setup_4.x | sudo -E bash -
sudo apt-get install -y nodejs
```

Rails:

```
gem install rails -v 5.0.1
```

Run to check you have everything installed :

```
rails -v
# Rails 5.0.1
```

4.1.2 Javascript

JavaScript is an easy-to-use object-based scripting language that can be embedded in the header of your web pages. This cross-platform and lightweight programming language can enhance the dynamics and interactive features of your page by allowing you to perform calculations, check forms, write interactive games, add special effects, customize graphics selections, create security passwords and more. It is not compiled but translated. The JavaScript Translator (embedded in browser) is responsible to translate the JavaScript code.

JavaScript itself is fairly compact yet very flexible. Developers have written a large variety of tools on top of the core JavaScript language, unlocking a vast amount of extra functionality with minimum effort. These include Browser Application Programming Interfaces (APIs) built into web browsers, providing functionality like dynamically creating HTML and setting CSS styles, collecting and manipulating a video stream from the user's webcam, or generating 3D graphics and audio samples. Third-party APIs to allow developers to incorporate functionality in their sites from other content providers. The merits of using JavaScript are:

- **Less server interaction**: You can validate user input before sending the page off to the server. It is executed on the client side (browser), this means that the JavaScript code is executed on the user’s computer processor/memory space instead of the web server. This saves bandwidth and load on the web server.

- **Interact with HTML forms**: Another important aspect of client-side JavaScript is its ability to interact with HTML forms. This capability is provided by the Form object and the form element objects it can contain: Button, Checkbox, Hidden, Password, Radio, Reset, Select, Submit, Text, and Textarea objects. These element objects allow you to read and write the values of the input elements in the forms in a document. For example, an online catalog might use an HTML form to allow the user to enter his order and could use JavaScript to read the input from that form in order to compute the cost of the order, the sales tax, and the shipping charge.
- **Browser support**: To use javascript, you don't have to use any plugin at all. This is because all browsers have accepted javascript as a scripting language for them and provides integrated support for it. All you need to do is to handle some of the tasks that are dependent on DOM (Document Object Model) of different browsers properly.
- **Richer interfaces and increased interactivity**: You can use JavaScript to include such items as drag-and-drop components, pop-up windows, alert boxes, sliders and client-side validation etc. The event handlers trigger functions based on user initiated events or on time. These event handlers can be attached to certain HTML elements or to the page itself. The most common event handlers are: onclick, onload, onmouseover, and onmouseout.
- **Interpreted Language**: It is an interpreted language, meaning that it can be used or executed with ease without pre-compilation. The newer version of JavaScript features built-in Generators and Iterators.
- **Easy-to-use**: The Javascript language is relatively easy to learn and comprises of syntax that is close to English. It uses the DOM model that provides plenty of prewritten functionality to the various objects on pages making it a breeze to develop a script to solve a custom purpose.
- **Faster**: As the code is executed on the user's computer, results and processing is completed almost instantly depending on the task (tasks in javascript on web pages are usually simple so as to prevent being a memory hog) as it does not need to be processed in the site's web server and sent back to the user consuming local as well as server bandwidth.

Syntax:

The <script> tag alerts the browser program to start interpreting all the text between these tags as a script. A simple syntax of your JavaScript will appear as follows.

```
<!DOCTYPE html>
  <head>
    <script>
   JavaScript code
    </script>
 </head>
  <body>
  </body>
</html>
```

4.1.3 SQLite

SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. SQLite is the most widely deployed SQL database engine in the world. The source code for SQLite is in the public domain. SQLite engine is not a standalone process like other databases, you can link it statically or dynamically as per your requirement with your application. The SQLite accesses its storage files directly.

- **SQLite is serverless**: SQLite doesn't require a different server process or system to operate. It comes with zero-configuration, which means no setup or administration needed. SQLite database is integrated with the application that accesses the database. The applications interact with the SQLite database read and write directly from the database files stored on disk.
- **Single-file cross-platform database**: A database in SQLite is a single disk file. Furthermore, the file format is cross-platform. A database that is created on one machine can be copied and used on a different machine with a different architecture. SQLite databases are portable across 32-bit and 64-bit machines and between big endian and little-endian architectures.
- **Fully ACID-compliant**: A transactional database is one in which all changes and queries appear to be Atomic, Consistent, Isolated, and Durable (ACID). SQLite implements serializable transactions that are atomic, consistent, isolated, and durable, even if the transaction is interrupted by a program crash, an operating system crash, or a power failure to the computer.
- **Variable length of columns**: The length of the columns is variable. It facilitates you to allocate only the space a field needs. For example, if you have a varchar(200) column, and you put a 10 characters' length value on it, then SQLite will allocate only 20 characters' space for that value not the whole 200 space.

- **Heavily commented source code**: The SQLite source code is over 35% comment. Not boiler-plate comments, but useful comments that explain the meaning of variables and objects and the intent of methods and procedures. The code is designed to be accessible to new programmers and maintainable over a span of decades.
- **Disaster planning**: Every byte of source-code history for SQLite is cryptographically protected and is automatically replicated to multiple geographically separated servers, in data centers owned by different companies. Thousands of additional clones exist on private servers around the world. The primary developers of SQLite live in different regions of the world. SQLite can survive a continental catastrophe.

Standard SQL commands are issued to operate in this database. Meta commands are issued to examine a database. Standard Commands can be classified into three groups:

- Data Definition Language: It provides the storage structure and methods to access data from the database system.
    - **CREATE** - Creates a new table in the database.
    - **ALTER** - Adds a new row into an existing table and renames the table.
    - **DROP** - Removes a table from the database.
- Data Manipulation Language: It enables users to manipulate data.
    - **INSERT** - Inserts rows into a table.
    - **UPDATE**- Updates data of the existing rows in a table.
    - **DELETE-** Removes existing rows from a table.
- Data Query Language: It enables users to retrieve required data from the database.
    - **SELECT** - Queries data from a single table.

4.1.4 Bundler package manager

Bundler provides a consistent environment for Ruby projects by tracking and installing the exact gems and versions that are needed. Bundler is an exit from dependency hell, and ensures that the gems you need are present in development, staging, and production. Starting work on a project is as simple as bundle install. It maintains a consistent environment for ruby applications. It tracks an application's code and the rubygems it needs to run, so that an application will always have the exact gems (and versions) that it needs to run.

It makes it easy to share your code across a number of development, staging and production machines. Of course, you know how to share your own application or gem: stick it on GitHub and clone it where you need it. Bundler makes it easy to make sure that your application has the dependencies it needs to start up and run without errors.

Installation:

To install bundler:

```
$ gem install bundler
```

Create a file named Gemfile in the root of your app specifying what gems are required to run it:

```
source "https://rubygems.org"
gem 'sinatra', '1.0'
```

This file should be added to the git repository since it is part of the app. You should also add the .bundle directory to your .gitignore file. Once you have added the Gemfile, it makes it easy for other developers to get their environment ready to run the app:

```
$ bundle install -j4
```

This ensures that all gems specified in Gemfile, together with their dependencies, are available for your application. Running bundle install also generates a Gemfile.lock file, which should be added to your git repository. Gemfile.lock ensures that your deployed versions of gems on Heroku match the version installed locally on your development machine. The flag -j4 will use 4 parallel jobs to install all of your dependencies.

Specify your dependencies in a Gemfile in your project's root:

```
source 'https://rubygems.org'
gem 'nokogiri'
gem 'rack', '~> 2.0.1'
gem 'rspec'
```

4.1.5 Sass (Syntactically Awesome StyleSheets)

Sass is an extension of CSS that adds power and elegance to the basic language. It is the most mature, stable, and powerful professional grade CSS extension language in the world. It allows you to use variables, nested rules, mixins, inline imports, and more, all with a fully CSS-compatible syntax. Sass helps keep large stylesheets well-organized, and get small stylesheets up and running quickly, particularly with the help of the Compass style library.

The rich features of SASS include:

- **CSS compatible**: Sass is completely compatible with all versions of CSS. So it can be used seamlessly with any available CSS libraries.
- **Import**: CSS has an import option that lets you split your CSS into smaller, more maintainable

portions. The only drawback is that each time you use @import in CSS it creates another HTTP request. Sass builds on top of the current CSS @import but instead of requiring an HTTP request, Sass will take the file that you want to import and combine it with the file you're importing into so you can serve a single CSS file to the web browser.

- **Mixins**: Some things in CSS are a bit tedious to write, especially with CSS3 and the many vendor prefixes that exist. A mixin lets you make groups of CSS declarations that you want to reuse throughout your site. You can even pass in values to make your mixin more flexible. A good use of a mixin is for vendor prefixes.
- **Inheritance**: This is one of the most useful features of Sass. Using @extend lets you share a set of CSS properties from one selector to another. It helps keep your Sass very DRY(Don't-repeat-yourself).
- **Partials**: You can create partial Sass files that contain little snippets of CSS that you can include in other Sass files. This is a great way to modularize your CSS and help keep things easier to maintain. A partial is simply a Sass file named with a leading underscore.
- **Variables**: Think of variables as a way to store information that you want to reuse throughout your stylesheet. You can store things like colors, font stacks, or any CSS value you think you'll want to reuse. Sass uses the $ symbol to make something a variable. For example:
  ```
  $font-stack:    Helvetica, sans-serif;
  $primary-color: #333;
  ```

Installation:

To install the sass gem, run the command:

```
$ gem install sass
```

To run Sass from the command line, just use:

```
sass input.scss output.css
```

You can also tell Sass to watch the file and update the CSS every time the Sass file changes:

```
sass --watch input.scss:output.css
```

If you have a directory with many Sass files, you can also tell Sass to watch the entire directory:

```
sass --watch app/sass:public/stylesheets
```

4.2 Supporting Languages

4.2.1 Fuse.js

It is a lightweight fuzzy-search, in JavaScript, with zero dependencies. It provides the technique of finding strings that match a pattern approximately (rather than exactly). The problem of approximate string matching is typically divided into two subproblems: finding approximate substring matches inside a given string and finding dictionary strings that match the pattern approximately.Fuse implements the CommonJS module system, which means that we can export variables to the outside world by attaching them to the module.exports object. Data-binding is then done using the curly brace syntax, Property="{variable}". The ways Fuse handle data binding, layout and animation are all tailored for the Fuse platform. Much like how any Node.js library that depends on modules like process and fs wouldn't be expected to work in a web page, Fuse's approach to JS is the same. Fuse is can be written in JavaScript or any language that compiles to JavaScript, for example using external transpilers like Babel or TypeScript.

It lets us choose which options to set. The following are supported:

- Case sensitivity: Indicates whether comparisons should be case sensitive.
- Sort: Whether to sort the result list, by score.
- Include matches: Whether the matches should be included in the result set. When true, each record in the result set will include the indices of the matched characters: indices: [start, end]. These can consequently be used for highlighting purposes.
- Find All Matches: When true, the matching function will continue to the end of a search pattern even if a perfect match has already been located in the string.
- Location: Determines approximately where in the text is the pattern expected to be found.
- Max pattern length: The maximum length of the pattern. The longer the pattern (i.e. the search query), the more intensive the search operation will be.
- ID: The name of the identifier property. If specified, the returned result will be a list of the items' identifiers, otherwise it will be a list of the items.
- Keys: List of properties that will be searched. This supports nested properties, weighted search, searching in arrays of strings and objects.
- Tokenize: When true, the algorithm will search individual words and the full string, computing the final score as a function of both. In this case, the threshold, distance, and location are inconsequential for individual tokens, and are thus ignored.
- Match all tokens: When true, the result set will only include records that match all tokens. Will only work if tokenize is also true.

4.2.2 Ace Editor API

Ace is a high performance code editor for the web. This embeddable code editor written in JavaScript. It matches the features and performance of native editors such as Sublime, Vim and TextMate. It can be easily embedded in any web page and JavaScript application. Ace is maintained as the primary editor for Cloud9 IDE and is the successor of the Mozilla Skywriter (Bespin) project.

It provides the following functionalities:

- Syntax highlighting for over 110 languages (TextMate/Sublime Text.tmlanguage files can be imported)
- Over 20 themes (TextMate/Sublime Text .tmtheme files can be imported)
- Automatic indent and outdent
- An optional command line
- Handles huge documents (four million lines seems to be the limit!)
- Fully customizable key bindings including vim and Emacs modes
- Search and replace with regular expressions
- Highlight matching parentheses
- Toggle between soft tabs and real tabs
- Displays hidden characters
- Drag and drop text using the mouse
- Line wrapping
- Code folding
- Multiple cursors and selections
- Live syntax checker (currently JavaScript/CoffeeScript/CSS/XQuery)
- Cut, copy, and paste functionality

Installation:

Installing ace editor is simple. Run the command :

```
git clone git://github.com/ajaxorg/ace.git
```

Ace can be easily embedded into any existing web page. You can either use one of pre-packaged versions of ace (just copy one of src* subdirectories somewhere into your project), or use requireJS to load contents of lib/ace as ace

The easiest version is simply:

```
 <div id="editor">some text</div>
    <script src="src/ace.js" type="text/javascript" charset="utf-8"></script>
     <script>
       var editor = ace.edit("editor");
     </script>
```

4.2.3 Puma server

Puma is a simple, fast, and highly concurrent HTTP 1.1 server for Ruby web applications. It can be used with any application that supports Rack, and is considered the replacement for Webrick and Mongrel. It was designed to be the go-to server for Rubinius, but also works well with JRuby and MRI. Puma is intended for use in both development and production environments.

Today, Puma runs on all Ruby implementations, but will always run best on any implementation that provides true parallelism. It looks to provide a simple and high performance request/response pipeline to Rack apps, allowing it to be used by nearly all ruby web applications.

Puma utilizes a dynamic thread pool which you can modify. You can set the minimum and maximum number of threads that are available in the pool with the -t (or --threads) flag. Puma will automatically scale the number of threads, from the minimum until it caps out at the maximum, based on how much traffic is present. The current default is 0:16.

Web applications that process concurrent requests make more efficient use of dyno resources than those that only process one request at a time. Puma is a web server that competes with Unicorn and allows you to process concurrent requests. It uses threads, in addition to worker processes, to make more use of available CPU. You can only utilize threads in Puma if your entire code-base is thread safe.

Puma includes the ability to restart itself allowing easy upgrades to new versions. When available (MRI, Rubinius, JRuby), Puma performs a "hot restart". This is the same functionality available in unicorn and nginx which keep the server sockets open between restarts. This makes sure that no pending requests are dropped while the restart is taking place.

To perform a restart, there are 2 built-in mechanisms:

- Send the puma process the SIGUSR2 signal
- Use the status server and issue /restart

No code is shared between the current and restarted process, so it should be safe to issue a restart any place where you would manually stop Puma and start it again. If the new process is unable to load, it will simply exit.

Installation:

The easiest way to get started with Puma is to install it via RubyGems. You can do this easily:

```
$ gem install puma
```

Now you should have the puma command available in your PATH, so just do the following in the root folder of your Rack application:

```
$ puma app.ru
```

4.2.4. Github API

GitHub is a Git repository web-based hosting service which offers all of the functionality of Git as well as adding many of its own features. Unlike Git which is strictly a command-line tool, Github provides a web-based graphical interface and desktop as well as mobile integration. It also provides access control and several collaboration features such as wikis, task management, and bug tracking and feature requests for every project.

GitHub offers both paid plans for private repto handle everything from small to very large projects with speed and efficiency. Repositories, and free accounts, which are usually used to host open source software projects. As of 2014, Github reports having over 3.4 million users, making it the largest code host in the world.

Version control systems keep these revisions straight, and store the modifications in a central repository. This allows developers to easily collaborate, as they can download a new version of the software, make changes, and upload the newest revision. Every developer can see these new changes, download them, and contribute.

GitHub has become such a staple amongst the open-source development community that many developers have begun considering it a replacement for a conventional resume and some employers require applications to provide a link to and have an active contributing GitHub account in order to qualify for a job.

GitHub supports the following formats and features:

- **Forking a repo**: Forking is when you create a new project based off of another project that already exists. This is an amazing feature that vastly encourages the further development of programs and other projects.
- **Pull requests with code review and comments**: Whenever you issue a pull request, GitHub provides a perfect medium for you and the project's maintainer to communicate. The authors of the original repository can see your work, and then choose whether or not to accept it into the official project.
- **Social networking:** The social networking aspect of GitHub is probably its most powerful feature, and is what allows projects to grow more than anything else. Each user on GitHub has their own profile, which can act like a resume of sorts, showing your past work and contributions to other projects via pull requests.

- **Changelogs**: When multiple people are collaborating on a project, it's really hard to keep track of who changed what, and to keep track of the revisions that took place. GitHub takes care of this problem by keeping track of all the changes that have been pushed to the repository.
- Documentation, including automatically-rendered README files in a variety of Markdown-like file formats
- Issue tracking (including feature requests) with labels, milestones, assignees and a search engine.
- **Wikis**: It is a website on which users collaboratively modify content and structure directly from the web browser. In a typical wiki, text is written using a simplified markup language and often edited with the help of a rich-text editor.
- **Graphs**: pulse, contributors, commits, code frequency, punch card, network, members.
- Updates about new work: Provides with an option to subscribe someone to notifications by @ mentioning them.
- Small Websites can be hosted from public repositories on GitHub. The URL format is http://username.github.io.
- Visualization of geospatial data
- 3D render files that can be previewed using a new integrated STL file viewer that displays the files on a "3D canvas". The viewer is powered by WebGL and Three.js.
- GitHub Pages: They are designed to host your personal, organization, or project pages directly from a GitHub repository.

4.3 Coding standards of Language used

The input to the Design Phase is the design document. During the coding phase, different modules identified in the design document are coded according to the module specifications. The Objectives of the coding phase is to transform the design of the system, as given by its module specifications, into a high level languages and code and then unit this code. Software developers adhere to some well-defined and standard style of coding called coding standards. The reasons for adhering to a standard coding style are the following.

- It gives a uniform appearance to the codes written by different engineers.
- It encourages good programming practices.
- It enhances the code readability and understanding.

Coding standards used:

- Don't use ; to separate statements and expressions. As a corollary—use one expression per line. For example:

```
# bad
puts 'foobar'; # superfluous semicolon
puts 'foo'; puts 'bar' # two expressions on the same line
# good
puts 'foobar'
puts 'foo'
puts 'bar'
puts 'foo', 'bar' # this applies to puts in particular.
```

- Don’t use ‘or’ and ‘and’, always use ‘&&’ and ‘||’.
- No space inside range literals. For example:

```
# bad
1 .. 3
'a' ... 'z'
# good
1..3
'a'...'z'
```

- Make the code a little more articulate.
- Prefer a single-line format for class definitions with no body.
- Avoid single-line methods. For example:

```
# bad
def too_much; something; something_else; end
# okish - notice that the first ; is required
def no_braces_method; body end
# good
def some_method
body
end
```

- Use spaces around operators, after commas, colons and semicolons. Whitespace might be (mostly) irrelevant to the Ruby interpreter, but its proper use is the key to writing easily readable code.

- No spaces after (, [or before],). Use spaces around { and before }. For example:

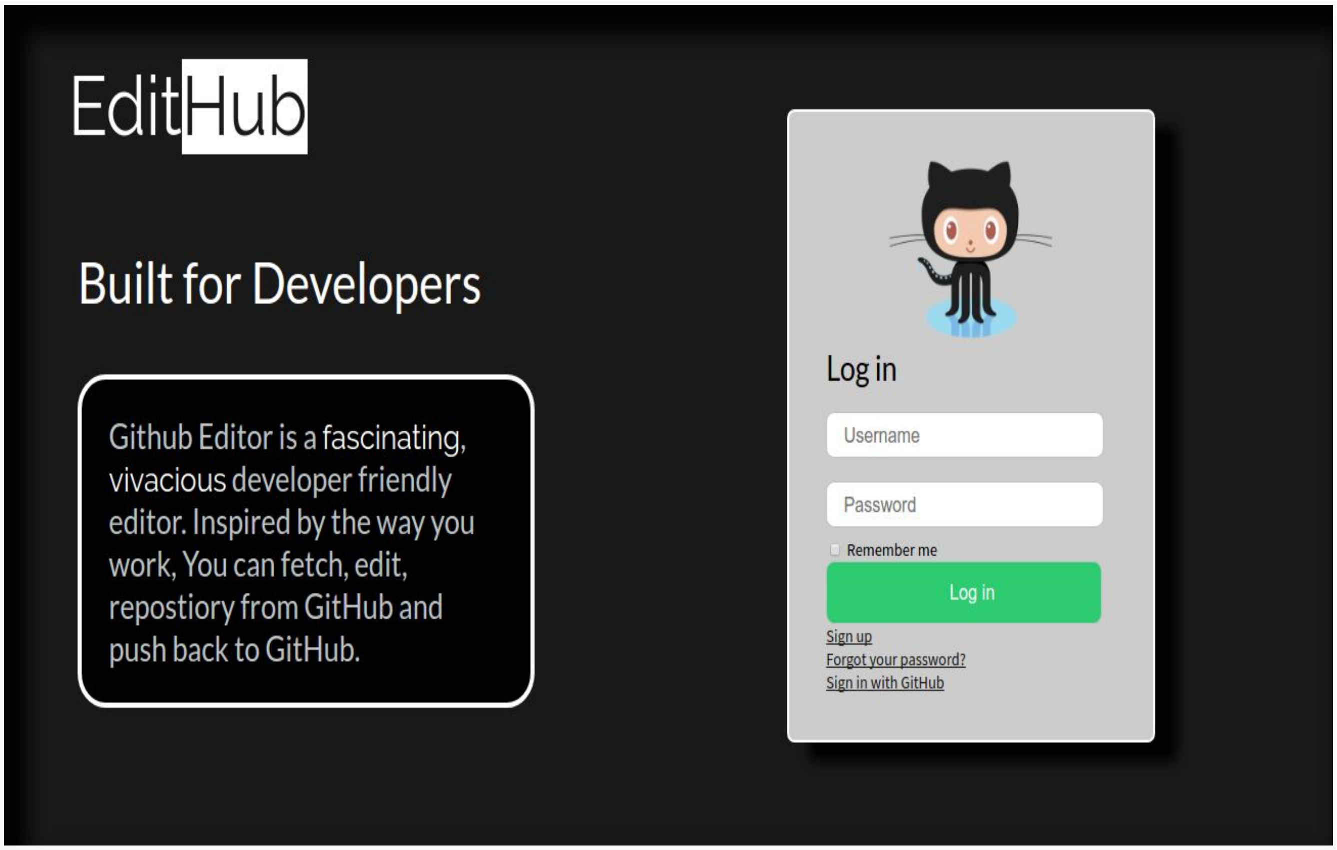
```
# bad
some( arg ).other
[ 1, 2, 3 ].each{|e| puts e}
# good
some(arg).other
[1, 2, 3].each { |e| puts e }
```

- When assigning the result of a conditional expression to a variable, preserve the usual alignment of its branches.
- Add underscores to large numeric literals to improve their readability. For example:

```
# bad - how many 0s are there?
num = 1000000
# good - much easier to parse for the human brain
num = 1_000_000
```

- Prefer small case letters for numeric literal prefixes.

4.4 Implementation with screenshots/ figures



Github Editor is a fascinating, vivacious developer friendly editor. Inspired by the way you work, You can fetch, edit, repostiory from GitHub and push back to GitHub.

Figure 12: Homepage

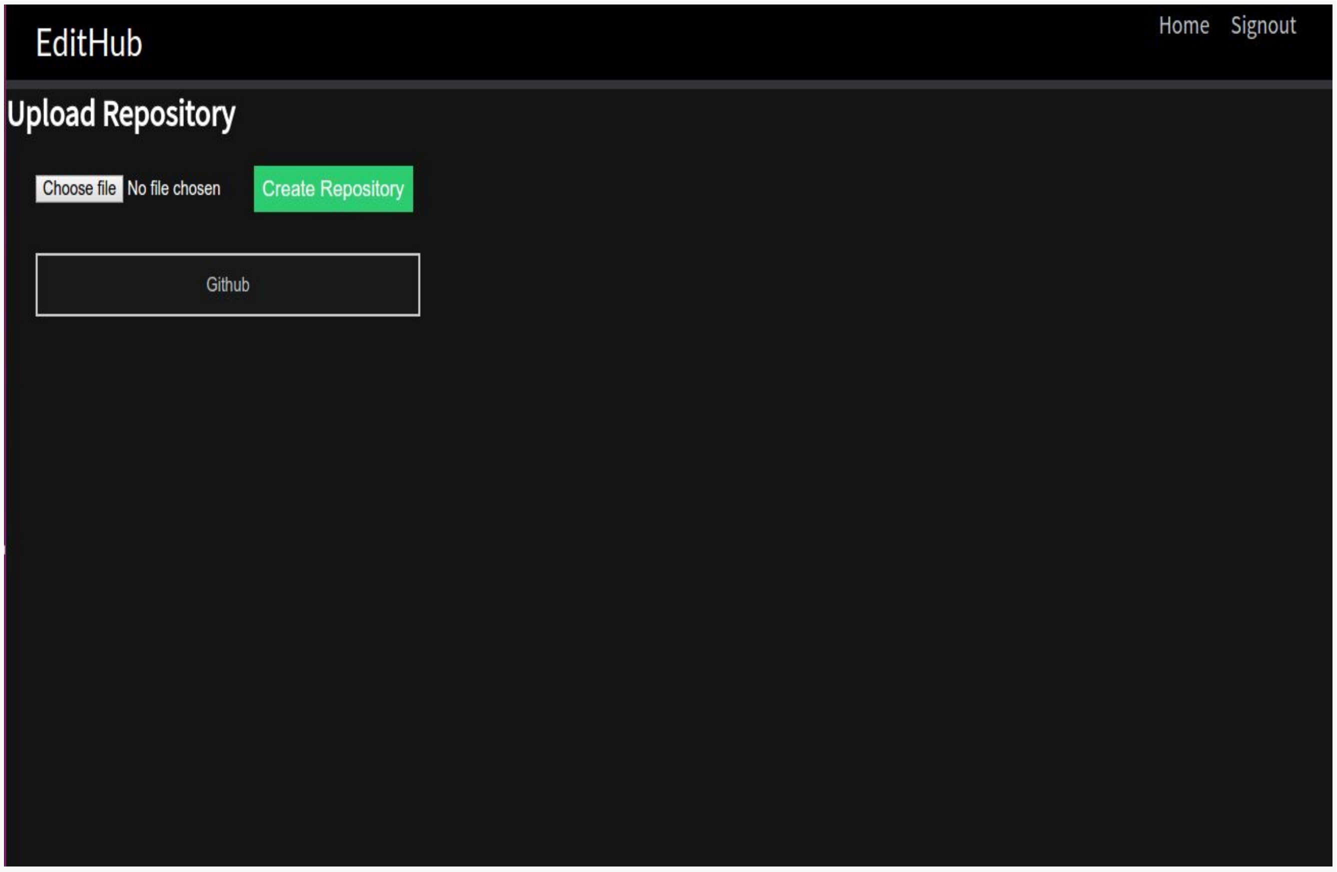


Figure 13: Uploading repository page

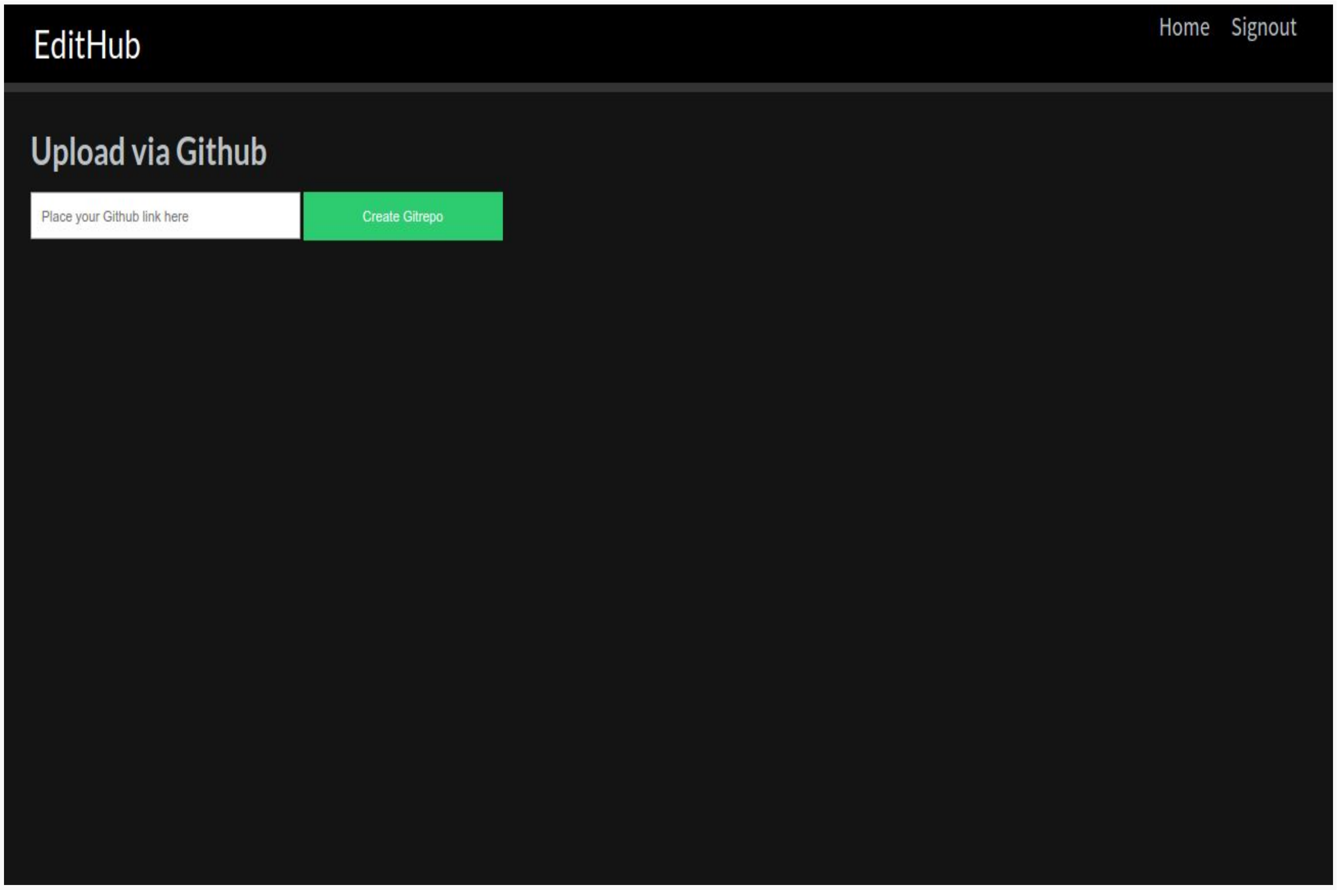


Figure 14: Upload via GitHub

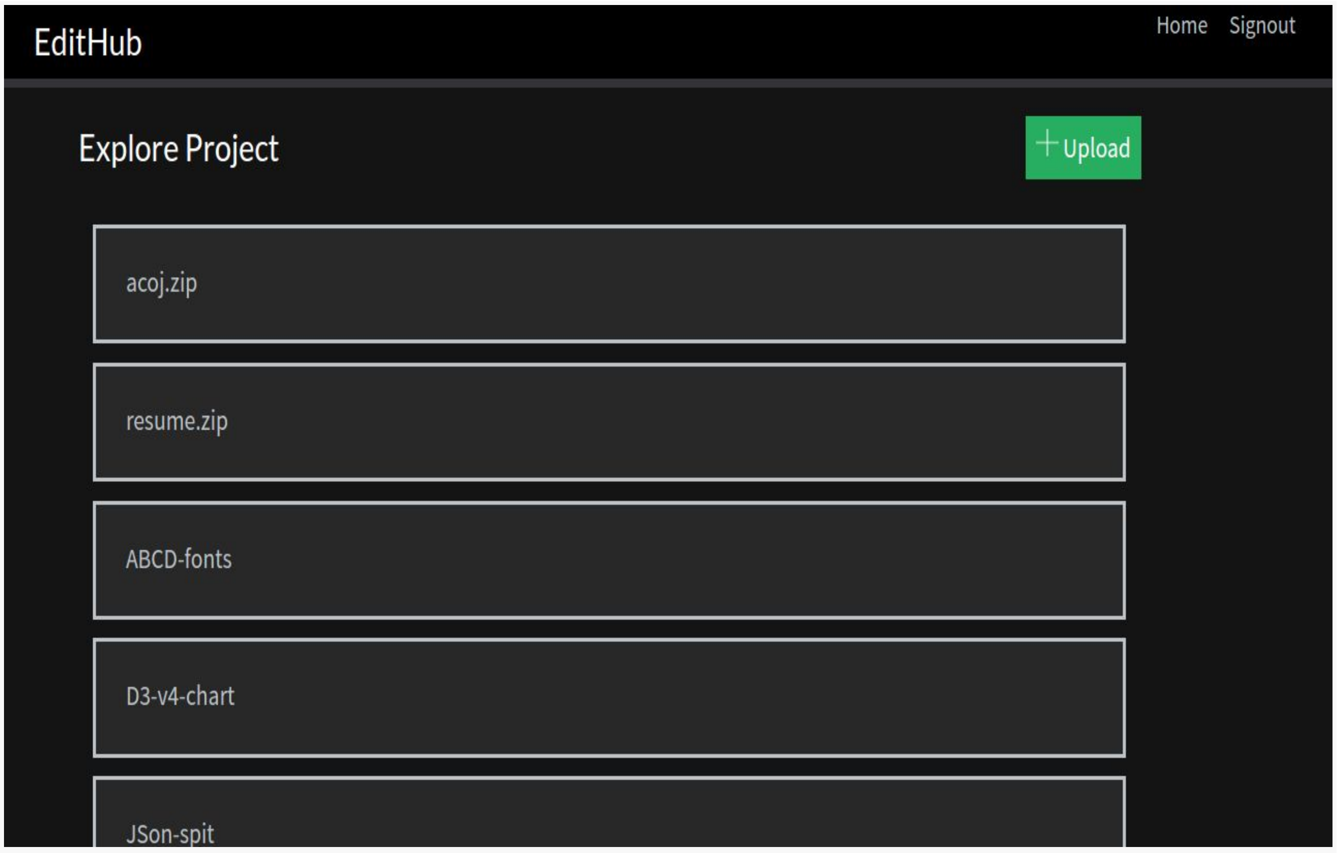
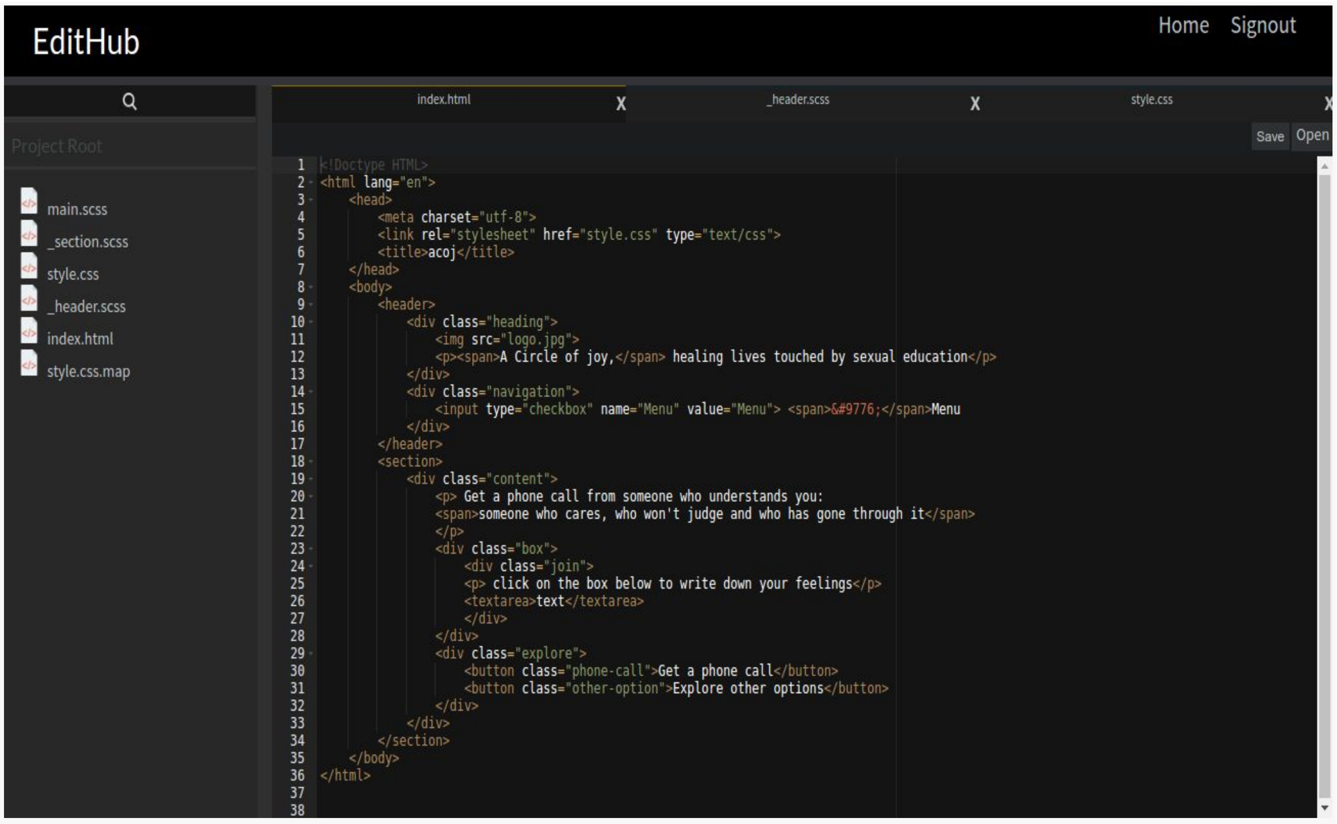


Figure 15: Index page



```html
<!Doctype HTML>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <link rel="stylesheet" href="style.css" type="text/css">
        <title>acoj</title>
    </head>
    <body>
        <header>
            <div class="heading">
                <img src="logo.jpg">
                <p><span>A Circle of joy,</span> healing lives touched by sexual education</p>
            </div>
            <div class="navigation">
                <input type="checkbox" name="Menu" value="Menu"> <span>&#9776;</span>Menu
            </div>
        </header>
        <section>
            <div class="content">
                <p> Get a phone call from someone who understands you:
                <span>someone who cares, who won't judge and who has gone through it</span>
                </p>
                <div class="box">
                    <div class="join">
                    <p> click on the box below to write down your feelings</p>
                    <textarea>text</textarea>
                    </div>
                </div>
                <div class="explore">
                    <button class="phone-call">Get a phone call</button>
                    <button class="other-option">Explore other options</button>
                </div>
            </div>
        </section>
    </body>
</html>
```

Figure 16: Edit mode

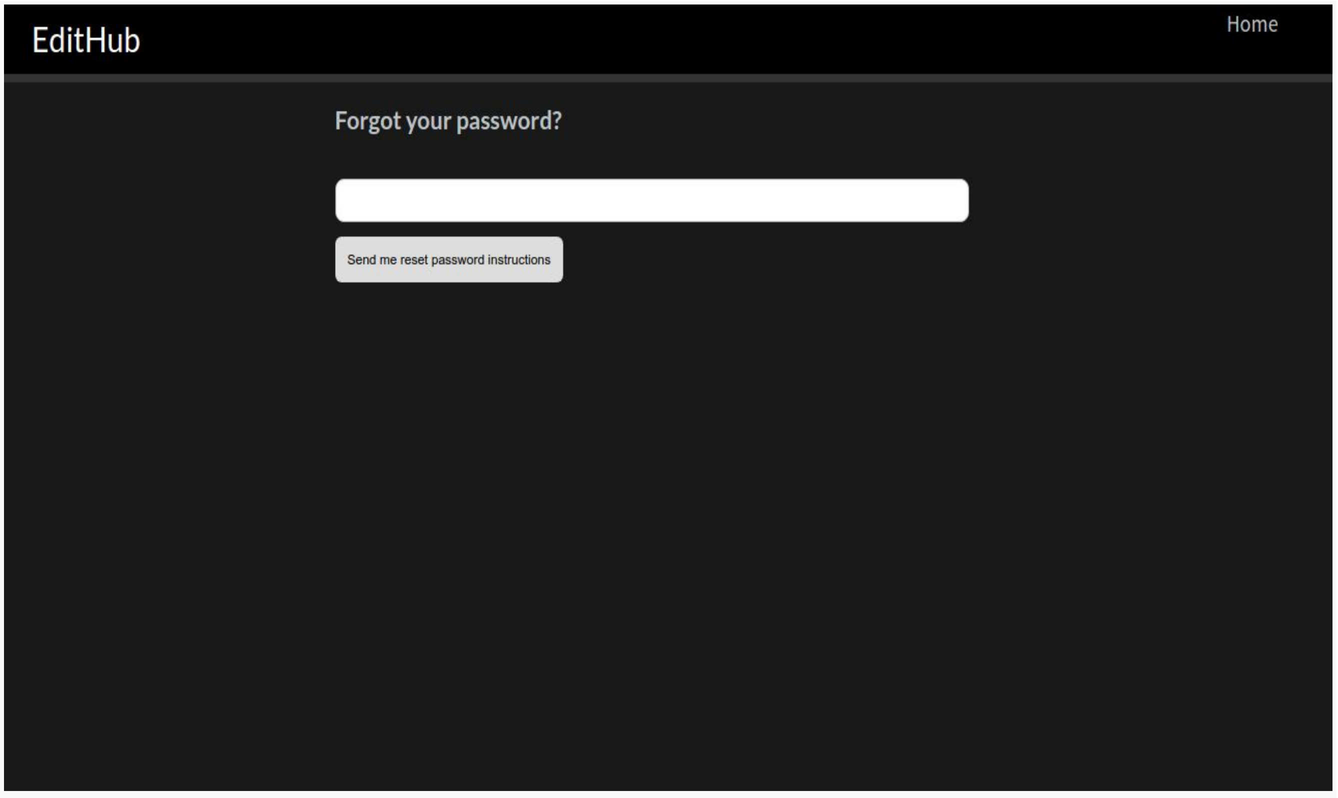


Figure 17: Password retrieval

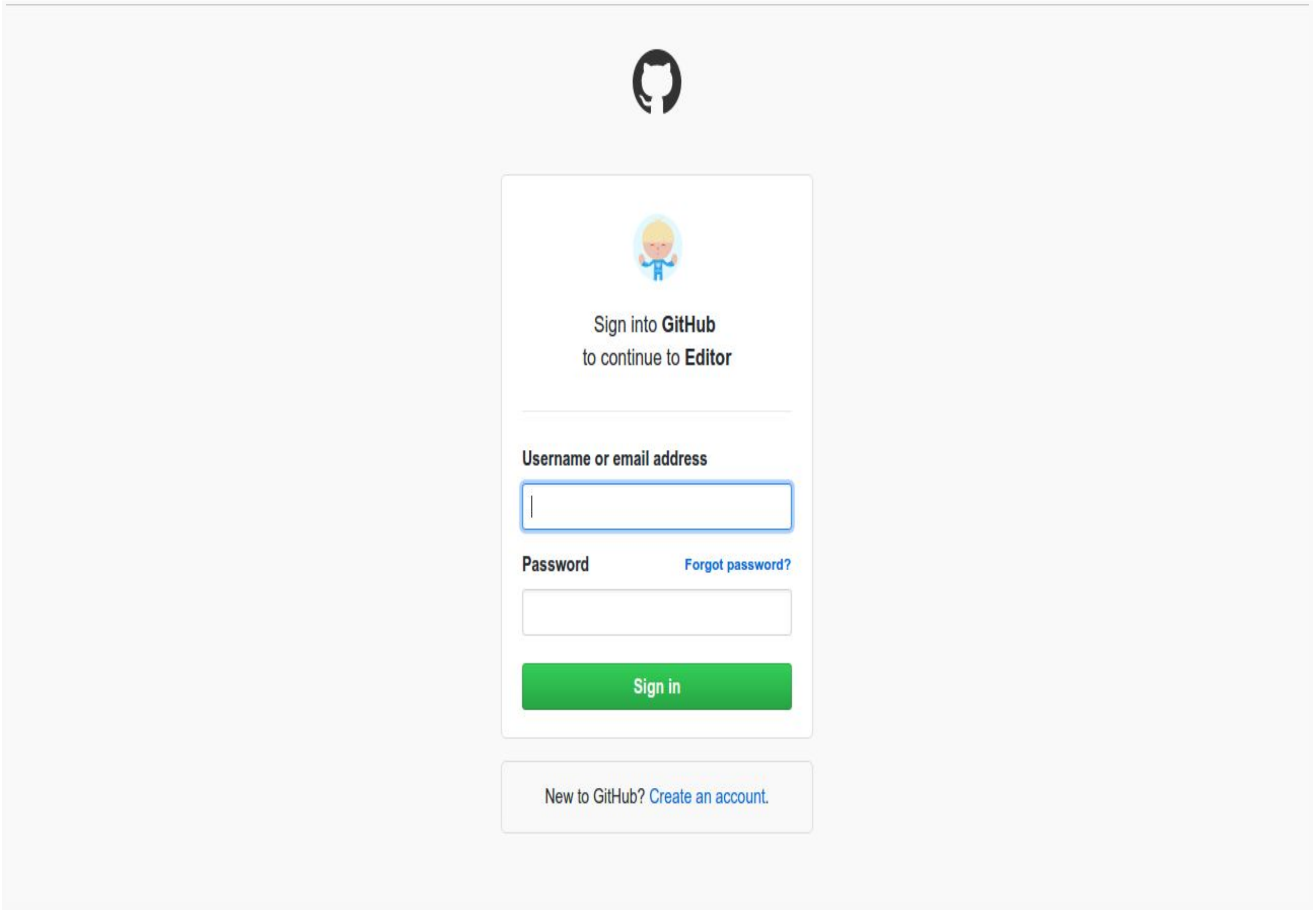


Figure 18: GitHub login

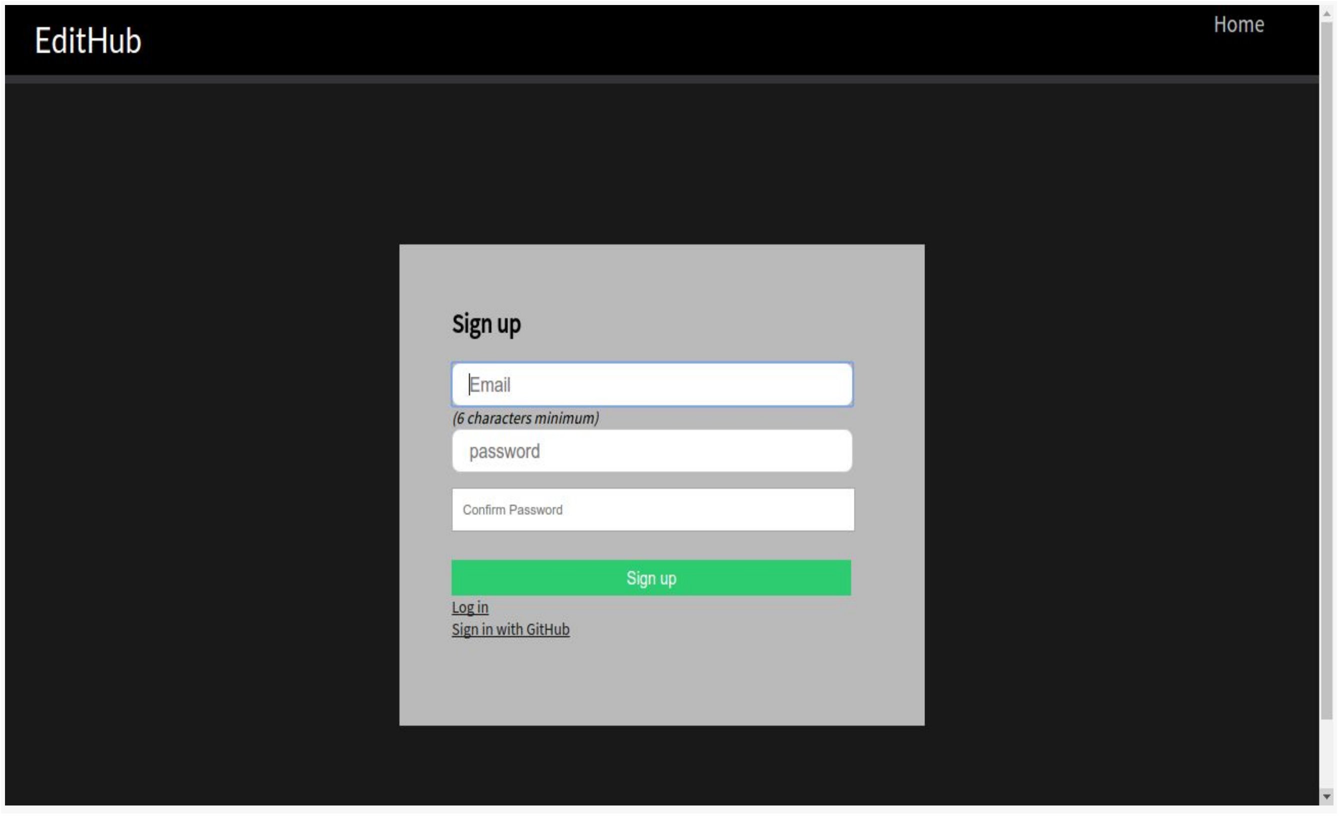
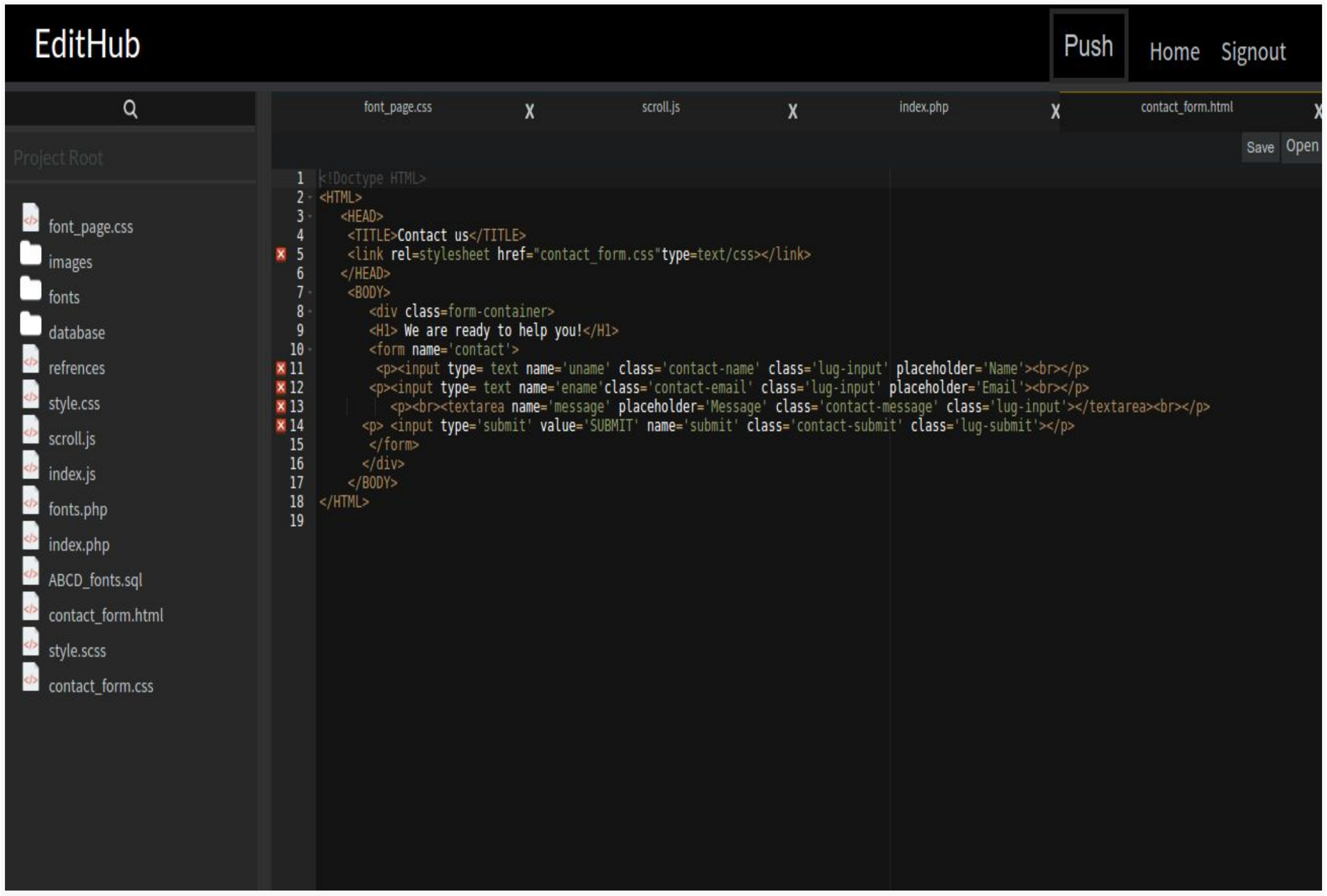
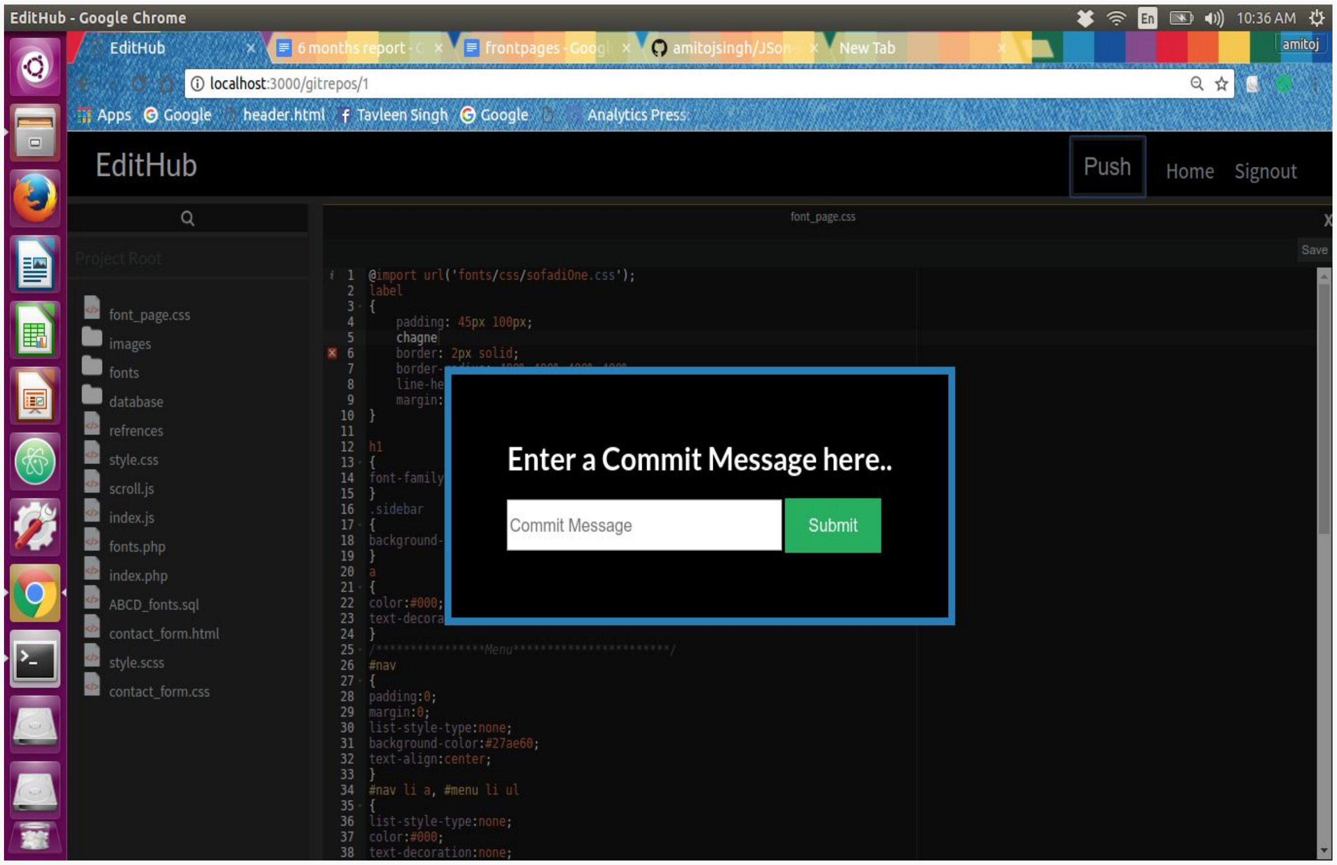


Figure 19: EditHub Sign up



```html
<!Doctype HTML>
<HTML>
   <HEAD>
    <TITLE>Contact us</TITLE>
    <link rel=stylesheet href="contact_form.css"type=text/css></link>
   </HEAD>
    <BODY>
       <div class=form-container>
       <H1> We are ready to help you!</H1>
       <form name='contact'>
        <p><input type= text name='uname' class='contact-name' class='lug-input' placeholder='Name'><br></p>
       <p><input type= text name='ename'class='contact-email' class='lug-input' placeholder='Email'><br></p>
          <p><br><textarea name='message' placeholder='Message' class='contact-message' class='lug-input'></textarea><br></p>
      <p> <input type='submit' value='SUBMIT' name='submit' class='contact-submit' class='lug-submit'></p>
       </form>
      </div>
    </BODY>
</HTML>
```

Figure 20: Edit mode 2



font_page.css
images
fonts
database
refrences
style.css
scroll.js
index.js
fonts.php
index.php
ABCD_fonts.sql
contact_form.html
style.scss
contact_form.css

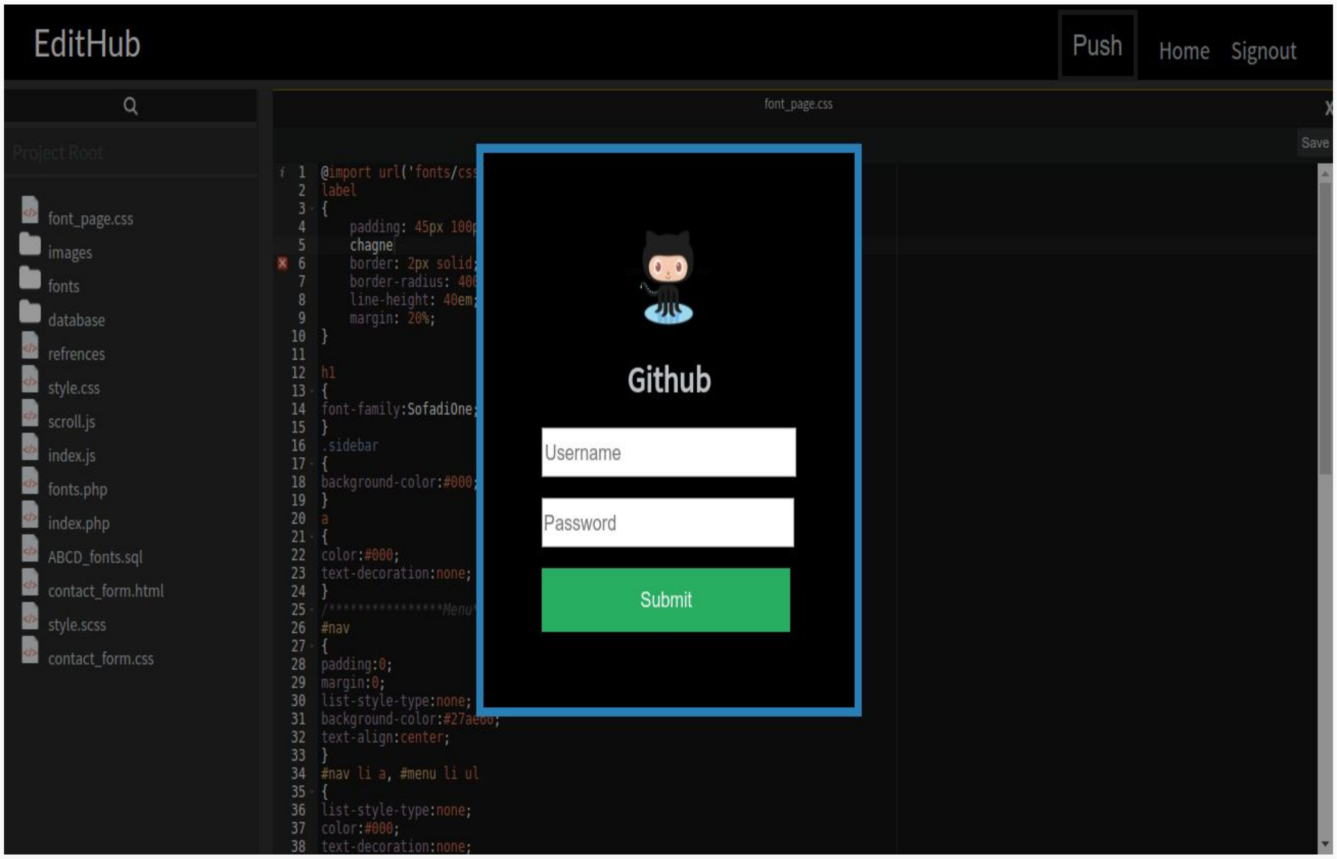
```
@import url('fonts/css/sofadiOne.css');
label
{
    padding: 45px 100px;
    chagne
    border: 2px solid;
    border-
    line-he
    margin:
}

h1
{
font-family
}
.sidebar
{
background-
}
a
{
color:#000;
text-decora
}
/***************Menu**********************/
#nav
{
padding:0;
margin:0;
list-style-type:none;
background-color:#27ae60;
text-align:center;
}
#nav li a, #menu li ul
{
list-style-type:none;
color:#000;
text-decoration:none;
```

Enter a Commit Message here..

Commit Message

Submit

Figure 21: Commit file



font_page.css
images
fonts
database
refrences
style.css
scroll.js
index.js
fonts.php
index.php
ABCD_fonts.sql
contact_form.html
style.scss
contact_form.css

```
@import url('fonts/css
label
{
    padding: 45px 100p
    chagne
    border: 2px solid;
    border-radius: 40
    line-height: 40em;
    margin: 20%;
}

h1
{
font-family:SofadiOne;
}
.sidebar
{
background-color:#000;
}
a
{
color:#000;
text-decoration:none;
}
/***************Menu
#nav
{
padding:0;
margin:0;
list-style-type:none;
background-color:#27ae60;
text-align:center;
}
#nav li a, #menu li ul
{
list-style-type:none;
color:#000;
text-decoration:none;
```

Github

Username

Password

Submit

Figure 22: GitHub push

4.5 Testing

There are different methods that can be used for software testing.

4.5.1 Test-driven development

Test-driven development (TDD) is a software development process that relies on the repetition of a very short development cycle: requirements are turned into very specific test cases, then the software is improved to pass the new tests, only. This is opposed to software development that allows software to be added that is not proven to meet requirements.

Kent Beck, who is credited with having developed or 'rediscovered' the technique, stated that TDD encourages simple designs and inspires confidence. Programmers also apply the concept to improving and debugging legacy code developed with older techniques.

BDD includes the practice of writing tests first, but focuses on tests which describe behavior, rather than tests which test a unit of implementation. The idea behind behaviour-driven development BDD (child of TDD) was that instead of always writing tests for some code we already have, we work in a red-green loop:

- Write the smallest possible test case that matches what we need to program.
- Run the test and watch it fail. This gets you into thinking how to write only the code that makes it pass.
- Write some code with the goal of making the test pass.
- Run your test suite. Repeat steps 3 and 4 until all tests pass.
- Go back and refactor your new code, making it as simple and clear as possible while keeping the test suite green.

Test structure: Effective layout of a test case ensures all required actions are completed, improves the readability of the test case, and smooths the flow of execution. Consistent structure helps in building a self-documenting test case. A commonly applied structure for test cases has setup, execution, validation, and cleanup.

- Setup: Put the Unit Under Test (UUT) or the overall test system in the state needed to run the test.
- Execution: Trigger/drive the UUT to perform the target behavior and capture all output, such as return values and output parameters. This step is usually very simple.
- Validation: Ensure the results of the test are correct. These results may include explicit outputs captured during execution or state changes in the UUT & UAT.

- Cleanup: Restore the UUT or the overall test system to the pre-test state. This restoration permits another test to execute immediately after this one.

RSpec is a testing tool created for behavior-driven development (BDD). It is the most frequently used testing library for Ruby in production applications. Even though it has a very rich and powerful DSL (domain-specific language), at its core it is a simple tool which you can start using rather quickly.

This workflow implies a "step zero": taking time to think carefully about what exactly it is we need to build, and how. When we always start with the implementation, it is easy to lose focus, write unnecessary code, and get stuck.Behavior-driven development is a concept built on top of TDD. The idea is to write tests as specifications of system behavior. It is about a different way of approaching the same challenge, which leads us to think more clearly and write tests that are easier to understand and maintain. This in turn helps us write better implementation code. A common problem newcomers face when starting with testing is falling into the trap of writing tests which do too much, test too little and require deep concentration in order to understand what is going on.

```
def test_making_order
book = Book.new(:title => "RSpec Intro", :price => 20)
customer = Customer.new
order = Order.new(customer, book)
order.submit
assert(customer.orders.last == order)
assert(customer.ordered_books.last == book)
assert(order.complete?)
assert(!order.shipped?)
end
```

The example above is written with test/unit, a unit testing framework that's part of Ruby's standard library.

With RSpec, we can get a little more verbose, describing behavior for the sake of clarity:

```
describe Order do
describe "#submit" do
before do
    @book = Book.new(:title => "RSpec Intro", :price => 20)
    @customer = Customer.new
    @order = Order.new(@customer, @book)
    @order.submit
end
describe "customer" do
it "puts the ordered book in customer's order history" do
expect(@customer.orders).to include(@order)
expect(@customer.ordered_books).to include(@book)
end
end
describe "order" do
it "is marked as complete" do
expect(@order).to be_complete
end
it "is not yet shipped" do
expect(@order).not_to be_shipped
end
end
end
end
```

It is worth noting that for a full BDD cycle we would need a tool such as Cucumber to write an outside-in scenario in human language. This also acts as a very high level integration test, making sure the application works as expected from the user's perspective. Now that we've covered the idea behind BDD, it's time to continue our quest to learn the basics of RSpec.

Installation: Create a new directory and put the following code in your Gemfile:

```
# Gemfile
source "https://rubygems.org"
gem "rspec"
```

Open your project's directory in your terminal, and type bundle install --path .bundle to install the latest version of RSpec and all related dependencies. You'll see output similar to the one below:

```
$ bundle install --path .bundle
Fetching gem metadata from https://rubygems.org/....
Resolving dependencies...
```

```
Installing diff-lcs 1.2.5
Installing rspec-support 3.1.2
Installing rspec-core 3.1.7
Installing rspec-expectations 3.1.2
Installing rspec-mocks 3.1.3
Installing rspec 3.1.0
Using bundler 1.6.0
It was installed into ./.bundle
```

Basics: RSpec gives you a way to encapsulate what you're testing via the describe block, and it's friend context. In a general unit testing sense, we use describe to describe the behavior of a class:

```
describe Hash do
end
```

Tests are written using the it block. Here's an example of how you might write a spec for the Hash class:

```
describe Hash do
it "should return a blank instance" do
Hash.new.should == {}
end
end
```

We usually use the describe keyword to describe methods. Using a "." will signify that you're testing a class method, and using "#" will signify that it's an instance method. Here's how it might look for a made up class:

```
describe MyClass do
describe ".class_method_1" do
end
describe "#instance_method_1" do
end
end
```

The context method does the same thing by letting you contextualize a block of your tests. This is extremely powerful for test states when you add more complicated setup and teardown code to really get in to your objects. One neat thing about RSpec is that the built in matchers will let you declaratively specify methods in your tests if they conform to a certain naming convention.

Chapter-5 Conclusion and future scope

5.1 Conclusion

This high-performance code editor for the web aims at making a programmer's life lot easier but combining the two tasks of editing the files and commiting to GitHub. It is a fascinating, vivacious developer-friendly editor. Inspired by the way you work, you can fetch, edit and push back your repository to GitHub.

It is easy to use for beginners and highly powerful for advanced users. It comes with syntax highlighting for many languages including PHP, JavaScript, HTML, and CSS. It comes with a very intuitive user interface that makes it super easy to browse files and work on projects. Instead of first cloning or downloading the files, updating the code for bug fix issues, etc. and then pushing it back to your github account, it does all this work in one-go. Once authenticated, you can upload files either from your local machine or via your GitHub account.

After uploading the files, you can open a specific project and view all the files simultaneously in different tabs. The beginner friendly interface has a black screen that supports distraction free editing mode. The turbolinks make the navigation quick, smooth and easy.

This feature-rich editor provides lightweight fuzzy-search library, highlight matching parenthesis and live syntax checker. It provides GitHub omni Authentication, save and push support. It has a drag and drop text using the mouse. It makes it really tough to write a messy code due to its auto-indentation feature. The line-wrapping and code-folding enhance the readability of the code. It enables you to see the preview update of your HTML documents live in a different browser window. This compact editor can handle large files upto the size of 4 million lines.

This powerful editor would be really handy for the GitHub users which is such a large community that people use their work as a resume and hunt for jobs within! Though we can easily clone or download the file, modify our content and commit it again, it is a three-step process.

It would simplify this task and reduce the tiresome authentication processes too via GitHub omniAuth. It also gives you a backup at your local machine. The database stores all the git repos and the commit history, which, if someone wants to publish it later, can use it for that cause. It really has a chance to create a difference in the open-source world, especially the vast GitHub community which lives to browse interesting repositories and solve all types of interesting problems. Because the world calls for efficient programmers who can write a seamless amount of clean and optimized code in the given time-frame. For writing an optimized and error free code, everyone needs an editor.

5.2 Future Scope

- Provide full functionality of GitHub branches
- Sharing the code on other platforms as well
- Real time collaboration
- Integration issues
- Fetch repositories from Gitlab, Bitbucket, etc
- Export the modified files as .zip and other formats.

References

1. http://rubyonrails.org/
2. https://www.w3schools.com/js/
3. http://bundler.io/
4. https://www.sqlite.org/
5. http://sass-lang.com/guide
6. https://ace.c9.io/#nav=about
7. http://puma.io/
8. https://devcenter.heroku.com/articles/deploying-rails-applications-with-the-puma-web-server
9. https://github.com/
10. https://try.github.io/levels/1/challenges/1