Software Requirement Specification for Web Based Integrated Development Environment

DEVCLOUD Web Based Integrated Development Environment

TinTin

Alican Güçlükol

Anıl Paçacı

Meriç Taze

Serbay Arslanhan

Table of Contents

UPDATES

Section Name	Section Number	Status
Interface Requirements	2.2.5	changed
External Interface Requirements	3.1	changed
GitHub Synchronization Interface	3.1.1	changed
Command Line Interface Requirements	2.2.4	changed
Command Line Interface	3.2.4	changed

1. Introduction

This document is a Software Requirement Specification (SRS) for the DEVCLOUD Web Based IDE project. This is the initial draft for the SRS and it will be used for the extensions. This document is prepared by following IEEE conventions for software requirement specification.

Integrated Development Environment is an application which provides facilities to programmer for software development such as code completing and fixing, source code editing and management, automated testing, etc. Cloud computing, according to the Wikipedia's definition, is usage of computer resources (both hardware and software) which is served over the internet.

The purpose of this project is provide an easy-to-use web service providing many powerful feature of desktop IDE's by combining with the power of Cloud Computing for application developers.

1.1. Purpose

The aim of this document is to specify complete description of the Integrated Development Environment based on Cloud to be developed. It is basis for agreement between suppliers and customers about the product to be developed. Through this document, the workload needed for development, validation and verification will ease. To be specific, this document is going to describe functionality, external interfaces, performance, attributes and the design constraints of the system which is going to be developed. Therefore, intended reader groups for this software requirement specification are customers, suppliers and users.

1.2. Scope

This project is intended for making use of today's popular technology Cloud Computing for Integrated Development Environment. Currently, there are lots of IDE's, both open-source and commercial, in the market. Usually they provide lots of extensive features to developers to ease application developers life. However, there are two simple but substantial problems with today's IDE's. First is they require intensive CPU and memory usage which is not available all the time and since these applications are installed on specific system, it prevents portability.

By combining Cloud Computing technology, this project will remove the requirement for powerful systems and provide portability to developer.

1.3. Overview

We are going to focus on describing the system in terms of product perspective, product functions, user characteristics, assumptions and dependencies on the following section of this document. Next, we will address specific requirements of the system, which will enclose external interface requirements, requirements of the system, performance requirements, and other requirements.

2. Overall Description

This section gives background information about specific requirements of the web based integrated development environment service to be developed in brief. Although we will not describe every requirement in detail, this section will describe the factors that affect the final product.

2.1. Product Perspective

This software product is eventually intended for the software developers. Product will be deployed to web site and all users of the product will access by use of the website. Website will be main user interface where users can operate all the provided functionality. However, this web site will be only a part of a larger system. There will be cloud server where all the user data is kept and all the execution is done. Website will only be the interface for the user data and the execution of provided functionalities.

To use product, users are required to register through the web interface. Whenever a new user registered, all the required data will be created in the database and a predefined workspace will be assigned for the user. Later, user will be able to login and logout the system anytime he wants. Since every operation that user perform reflected to our database, user will find his workspace however he leaves last time.

From the user point of view, user will have to functionality to create and edit files in his own workspace. User will be able to run predefined programs on these files such as language specific compilers and debuggers (gdb, g++, javac, etc) and project management tools (mvn, svn, git, etc).

All of the files users created will be kept in cloud server and all the CPU requiring work such as compiling and running the programs will be executed on cloud server so that user will be able to access his own integrated development environment with his specific setting anywhere he wants.

2.2. Product Functions

This new product, web based integrated development environment, must have number of features which will allow users to use functionalities which have been explained above. Required functionalities of the product can be summarized in five categories; user management requirements, code editor requirements, debugger requirements, command line interface requirements and interface requirements. Overall description of the requirements can be found below;

2.2.1 User Management Requirements

This category of requirements is related to user authentication mechanism and workspace management of users. Each user will have credentials to connect their workspace on cloud and will be assigned to workspace. Users will perform all the functionality over this workspace using his credentials.

2.2.2 Code Editor Requirements

One of the most important functionality expected from an integrated development environment is a code editor which will ease the developer's life. Code editor will be the main interface that developers deal with. It supports variety of programming language with highlighting, syntax checking, auto-indentation and language specific auto-complete.

2.2.3 Debugger Requirements

Debugger is the main tool that developers can test and debug their target program. Debugger of the product should allow setting and displaying breakpoints on the code. It will also provide functionality of stopping/continuing of the execution of debugger. Finally, it will provide an expression interface where user can enter an expression and observe the value of expression at each step.

2.2.4 Terminal Requirements

As an important part of the software development process, an integrated development environment should provide a command line interface where user can work in old fashion and accomplish complicated tasks such as configuring git synchronization. Main component of CLI will be the terminal. Terminal will allow user to run UNIX command on his own workspace and also run predefined programs such as mvn, svn etc. Terminal will also provide auto-complete by list of available commands and browse in the command history.

2.2.5 Interface Requirements

This group of requirements is related to external interaction of the workspace with outer world. For user to interact with the workspace, product will provide both command line interface and graphical interface. Command line interface will be UNIX like and graphical interface will allow tabbed navigation of windows, hierarchical view of workspace etc.

Again as an external interface, product will support a synchronization interface for external-services. Through this interface, user will be able to synchronize his workspace with external services like GitHub and SVN.

2.3. User Characteristics

Users of this web based integrated development environment will mainly be software developers. Since it is reasonable to assume that an average developer has knowledge about functionalities and usage of IDE, we assume that our users will already be informed about basic functionality of the product. Also clear documentation and tutorials about the product feature will be provided.

2.4. Constraints

Developers of the product should be aware that main feature of the intended product is portability. So they should use common libraries and tools that can work with all the common internet browser application with no problem.

Developers should also be careful about the privacy of users. Since product will be cloud application, all user data will be kept on cloud server and necessary precautions should be taken to protect user data.

Since product will be cloud application and all user programs will be executed on cloud server, developers should limit the privileges of the users so that they cannot harm other users' data and system server.

3. Specific Requirements

With this section and later, we will describe the requirements of the software in detail. Basically, we will categorize requirements in 3 which are namely external interface requirements, functional requirements and non-functional requirements. Except non-functional requirements, requirements of the product will be detailed under this section with brief information and later sample input-output sequence and low of events will be given.

3.1. External Interface Requirements

In this sub section, we will describe the external interface requirements of the product in three categories which are CLI support, workspace explorer and code editor and communication interfaces. The purpose of this section is to identify and document this interfaces and interaction of the software with external entities in detail.

Workspace explorer and code editor interface will be the main graphical user interface where developers will interact with their workspace and its including. It will allow graphically create, edit and delete files, running and debugging their programs and other basic feature that and average integrated development environment provided.

With CLI support, we basically aim to provide a UNIX like command line interface which is usually preferred by software developers. With this CLI, users will be able to run UNIX commands on their workspaces under the privileges given to them.

Cloud IDE is going to provide a communication interface to external services. Through these interface users will be able to synchronize their workspaces with version control systems and online code repositories such GitHub, SVN and SourceForge.

Now, detailed explanation of all the external interfaces can be found below;

3.1.1 External Services Synchronization Interface

Initially we are going to provide only GitHub synchronization since GitHub is today's one of the most popular online repository where developers can create their code base and share with other people by using the functionality of git source code management system.

Product is going to provide an interface to publish the content of the workspace to external systems. For the initial version of the product this interface works well with the GitHub web site. If user also has a GitHub account and wants to synchronize his repository with the GitHub, product should be able to provide this functionality.

To use this functionality, user should be able to login to system. Later, user should choose the GitHub synchronization button and enter the credentials of his GitHub account. Later workspace of the user will be synchronized to a GitHub workspace with same name, if it does not exist, new one will be created. Later any changes on the synchronized workspace will be reflected on the GitHub immediately.

When user login to system via web browser, if his workspace has already been synchronized with GitHub before, then system should reflect changes at GitHub workspace to user workspace, if there is any.

Since GitHub uses git and git is a transaction based source code management system, user should be able to return back to previous point in his git history.

During all this processes, user should be notified about stage of the process. And user should be given an alert or error message if anything goes wrong during the GitHub synchronization process. User also can choose to cancel a current synchronization with GitHub, in such case, no further automatic synchronization will be done.

3.1.2 Command Line Support

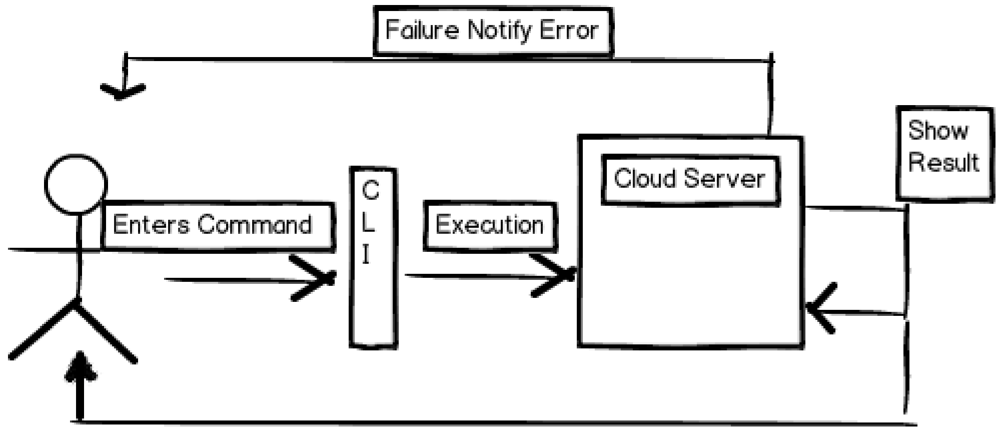
Except the graphical user interface, system will provide a command line support for the users. Although most of the functionality will be served on graphical user interface, some complex tasks may be accomplished through the command line as in the Linux OS distributions.

To use the command line interface, user first should login to the system with his own credentials. After login user will be redirected to the main page of the Cloud IDE. In this main page user can go to the below part of the page which will be a command line where user can execute UNIX commands on the current directory.

Other way to use command line will be the click on "Open Terminal" button on the main page. After clicking on this button, terminal window on a new tab will be opened and the current directory will be the root directory of the user workspace. User can run UNIX commands another programs in this terminal just as in the any Linux distribution. Only exceptions are going to be the workspace and user privilege restrictions which are to protect other user and system from any harm.

During any process, user should be able to receive feedback about operation. If the command or program runs successfully, then user should be able to see the effects either on the CLI or the in the Workspace explorer. If an error occurs because of the user limitations, user should be able to given a detailed notification about the cause of the error and solutions if possible.

Below diagram presents the functionality of the Command Line Interface;



3.1.3 Project/Workspace Explorer

Project/Workspace explorer is going to be the main user interface of the web based integrated development environment. User will be able to their file hierarchy and edit it.

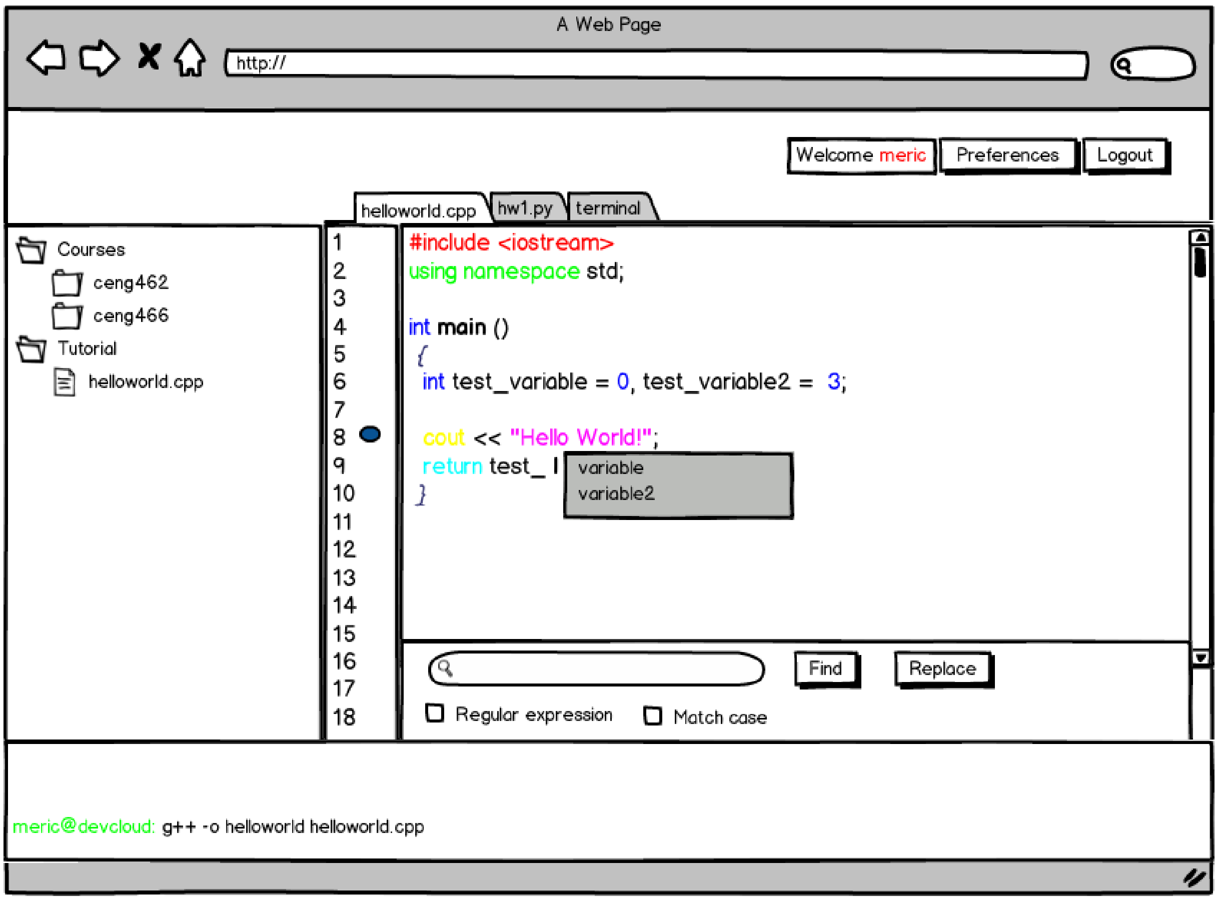
To be able to use this main graphical user interface, user should have login to the system. After login, user will be directed to a main page specialized for himself. In this main interface there will be a workspace explorer where users can see his file hierarchy and create and edit the files in the workspace.

There is also be a code editor in this main interface, so that users can edit their source codes by using the functionality provided by the code editor. Also they can use the command line interface to execute commands on the workspace which is also in the main interface of the system.

User should be able to specify preferences for the appearance of the main interface. After user specified a preference, user should be able to directed his own specialized main page whenever he/she logins to the system.

During all these operations, user should be notified about the status of the operation. If an operation operates successfully user should get a message or directly see the effect on workspace explorer or on the code editor. If an error occurs, user should get an error message indication the cause of error.

Below, you can find a mockup for the overall appearance of the external interfaces of the Cloud IDE;



```
#include <iostream>
using namespace std;

int main ()
 {
 int test_variable = 0, test_variable2 =  3;

 cout << "Hello World!";
 return test_ l
 }
```

```
meric@devcloud: g++ -o helloworld helloworld.cpp
```

3.2. System Features

In this subsection, we will examine the features of the system in detail by categorizing them according to their functionality. For each of the feature, we will give an introduction, purpose, diagram and a stimulus/response sequence. Introduction part will give basic background information about the feature. After that, we will show a diagram for the feature representing flow of events. Alternative flow of events will be given in stimulus/response subsection.

3.2.1. User Management

3.2.1.1 User Authentication

3.2.1.1.1 Background Information

Product will be used via a web browser. Each user will have his own workspace, and he must be logged in to the server to access his workspace. Hence, first-time users must be complete registration process. To register to the system, user must specify some information asked during registration. After validation, registration will completed, and user will be informed.

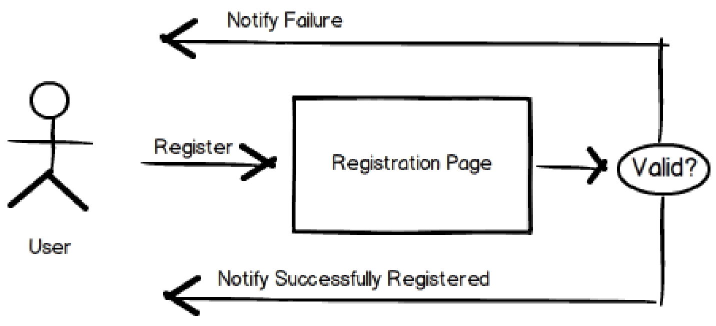
There will be a login page so that user can type into his login information, and login to the system. Login information will be username or e-mail address and password specified in registration process. Server let through the user if the given username and password are matched with the ones in database saved in registration. If specified information is not matched, an error dialog will be shown. Otherwise, user will be redirected to his personal workspace.

When user forgets his password, he can request new one from the system by specifying his username or e-mail.

3.2.1.1.2 Stimulus/Response Sequences

3.2.1.1.2.1 Register

Diagram



Description

Primary Actor	User
Goal in Context	Purpose of this feature is to register user to the system.
Trigger	User wants to register to the system

Normal Flow of Events

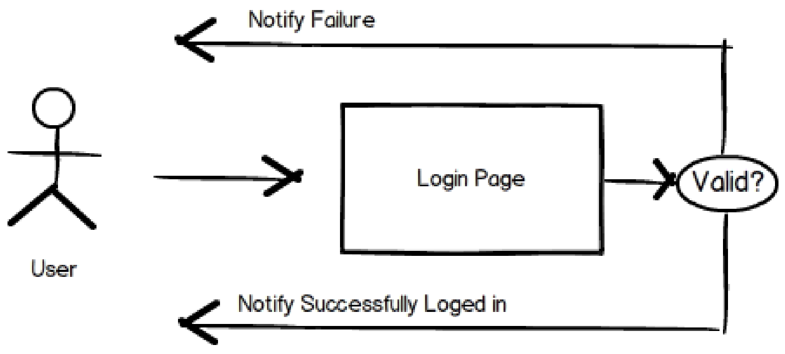
1. User opens the registration page
2. User specifies his information
3. System validate the specified information
4. User is registered to the system

Alternative Event Flow 1

4. User can not registered to the system due to inappropriate information

3.2.1.1.2.2 Login

Diagram



Description

Primary Actor	User
Goal in Context	Purpose of this feature is to login to the system with user credentials in order to use system.
Trigger	User wants to login to the system

Normal Flow of Events

1. User opens the login page
2. User tries to login to the system with his credentials
3. System validate the specified information
4. User is logged into the system

Alternative Event Flow 1

4. User cannot logged into the system due to incorrect credentials

3.2.1.2 Workspace Management / Ownership

3.2.1.2.1 Background Information

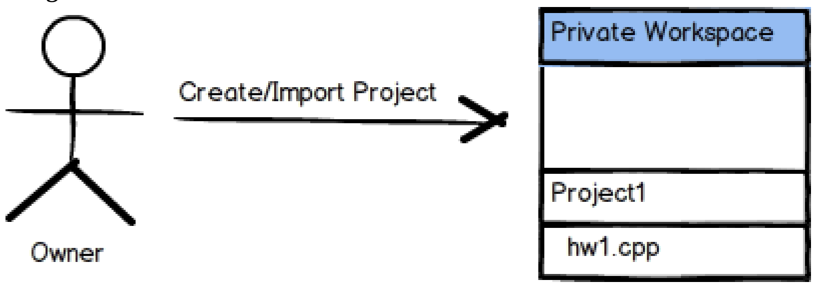
A workspace is a logical collection of projects. User can manage his projects within a workspace. User can create new projects, import existing projects, or create and delete files and folders in any of the Project at any time.

After registration of users, a private workspace will be available for him. Workspace will be accessed by workspace owner with full access. Owner can change visibility of his workspace. If workspace is set to be as public, any user can observe his projects.

3.2.1.2.2 Stimulus/Response Sequences

3.2.1.2.2.1 Create/Import Project

Diagram



Description

Primary Actor	User workspace owner
Goal in Context	Purpose of this feature is to create a new project or import existing one
Preconditions	User must be logged into the system
Trigger	User wants to create a new project, or import existing project

Normal Flow of Events

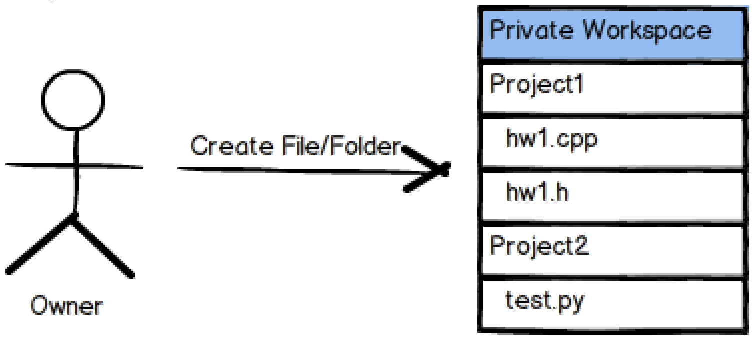
1. User logins in to the system
2. User opens his workspace
3. User creates a project
4. User creates a file into the new project

Alternative Event Flow 1

3. User imports an existing Project from his local file system

3.2.1.2.2.2 Create File/Folder

Diagram



Description

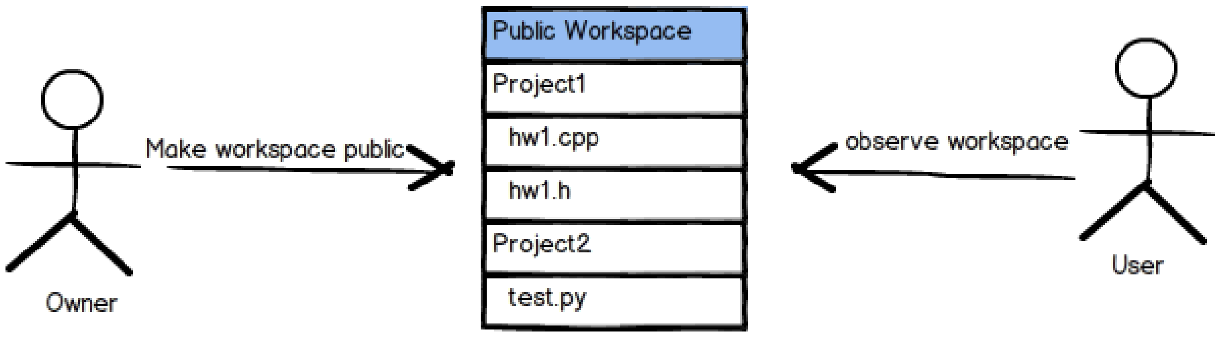
Primary Actor	User workspace owner
Goal in Context	Purpose of this feature is to create new files and folders into the projects
Preconditions	User must be logged into the system
Trigger	User wants to create a new file or folder to the workspace

Normal Flow of Events

1. User logins in to the system
2. User opens his workspace
3. User creates a folder
4. User opens newly created folder
5. User creates a file

3.2.1.2.2.3 Make Workspace Public

Diagram



Description

Primary Actor	Owner & User
Goal in Context	Purpose of this feature is to make workspace public so that any user can observe it.
Preconditions	User must be logged into the system, and be the owner of workspace
Trigger	User wants to make workspace public

Normal Flow of Events

1. Owner logins in to the system
2. Owner opens his workspace
3. Owner selects "make public" option
4. Users observe workspace

3.2.1.3 GitHub Synchronization

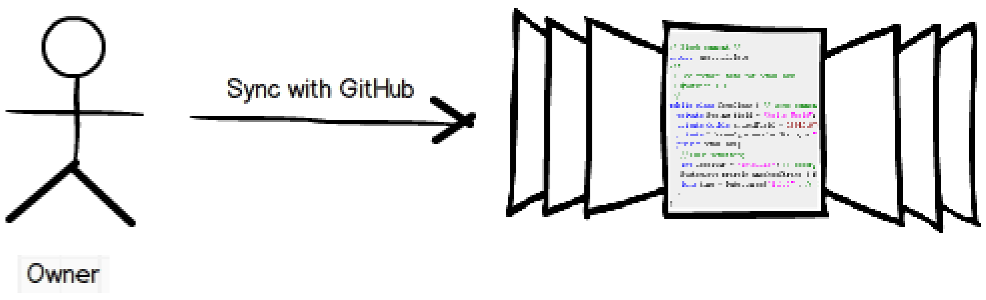
3.2.1.3.1 Background Information

Git is an efficient, distributed version control system ideal for the collaborative development of software. It makes enable to import and export projects, commit changes, switch branches and review version history. So that user can use these features, system will provide GitHub Integration. Thus, user will have an option to synchronize his workspaces with GitHub.

3.2.1.3.2 Stimulus/Response Sequences

3.2.1.3.2.1 Sync with GitHub

Diagram



Description

Primary Actor	User workspace owner
Goal in Context	Purpose of this feature is to synchronize workspace with GitHub
Preconditions	User must be logged into the system User must have a GitHub account
Trigger	User wants to synchronize workspace with GitHub

Normal Flow of Events

1. User logins in to the system
2. User opens his workspace
3. User selects "GitHub Sync" option
4. User types his GitHub credentials
5. User import his Project to the workspace
6. User makes some changes and commit it to master

Alternative Event Flow 1

6. User creates a branch
7. User make some changes and commit it to the newly created branch

Functional Requirements

REQ 1: The system shall provide a registration page

REQ 2: The system shall provide a login page

REQ 3: The system shall support creating and importing projects

REQ 4: The system shall support creation files or folders

REQ 5: The system shall support public workspace option

REQ 6: The system shall support GitHub synchronization

3.2.2. Code Editor

3.2.2.1 Background Information

This feature enables the user to edit his/her code with the help of several functionalities present in the code editor module of the IDE. While editing the source files, the user will be able to use the following functionalities of the code editor:

3.2.2.1.1 Syntax Highlighting:

The code editor will be able to highlight the code based on the programming language detected while user editing the source code. The user will be able to extend the list of the programming languages supported by the code editor.

3.2.2.1.2 Auto-indentation:

The source code will be indented automatically according to the characteristics of the programming language and widely used indentation style conventions.

3.2.2.1.3 Bracket/Brace matching:

The brackets and braces will be matched when the user clicks on them and the matching bracket or brace will be highlighted.

3.2.2.1.4 Auto completion:

The expressions will be auto completed while the user is editing the source file by using the dictionaries developed for the programming language and the user input.

3.2.2.1.5 Setting/Displaying breakpoints:

The user will be able to set a breakpoint by clicking the left side of the corresponding line. The breakpoints set before will be listed and shown in the left side of the corresponding line.

3.2.2.1.6 Find/Replace with regular expression support:

The user will be able to find and replace the desired parts of the source code matching with the regular expression.

3.2.2.1.7 Editor themes:

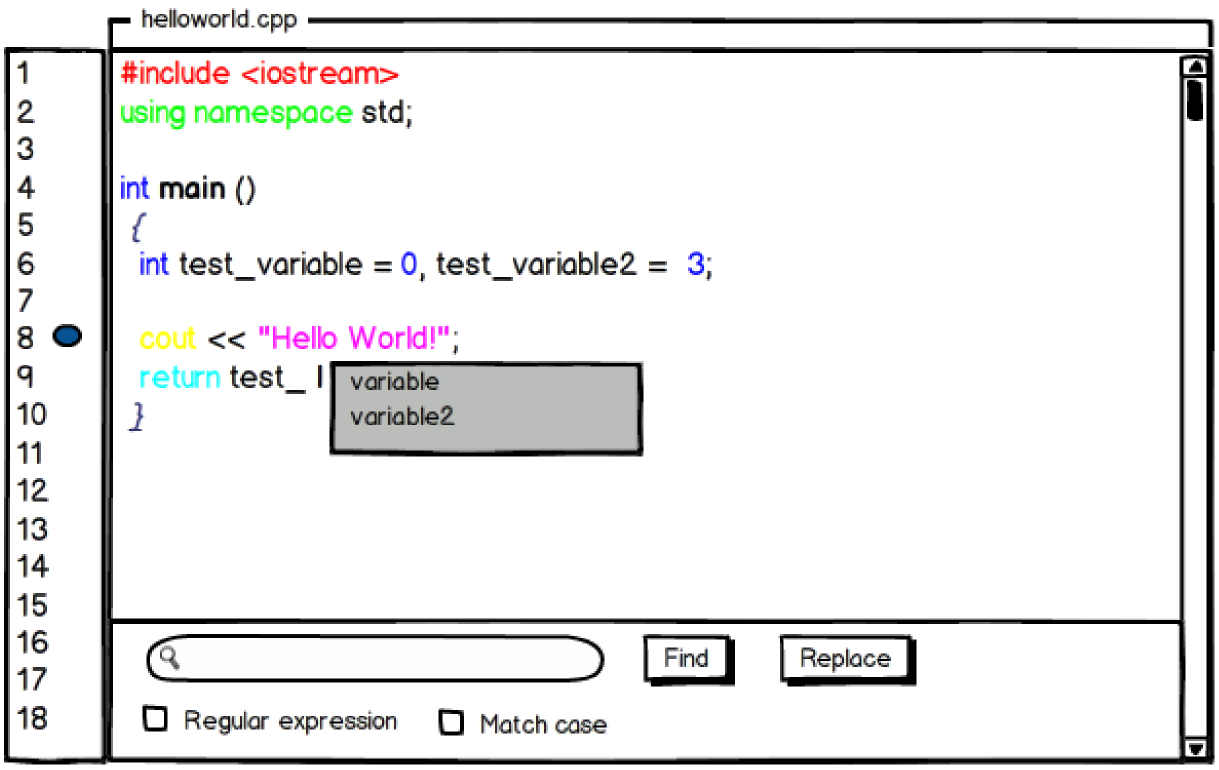
Several themes for the editor will be provided and the user will be able to choose the editor theme among the predefined themes.

3.2.2.1.8 Displaying line numbers:

The corresponding line numbers will be displayed on the left side of the source code.

3.2.2.2 *Stimulus/Response sequences*

3.2.2.2.1 Diagram



```
#include <iostream>
using namespace std;

int main ()
 {
  int test_variable = 0, test_variable2 =  3;

  cout << "Hello World!";
  return test_ I
 }
```

Figure 1A presentation of Functionalities of the Code Editor

3.2.2.2.2 Description

Primary Actor	User
Goal in Context	The purpose of this feature is to provide fundamental requirements for the code editor module of the IDE.
Preconditions	User has logged in to the system.
Trigger	User edits the source file.

3.2.2.2.3 Normal flow of events

i. User logs in to the system.
ii. User begins editing a source file.
iii. A programming language specific keyword is typed by the user.

iv. The keyword is detected by the code editor.
v. The keyword is highlighted.

3.2.2.2.4 Alternative Event Flows

Alternative Event Flow I

iii. User types an expression that needs to be indented.
iv. Code editor detects the need for the indentation.
v. The line is indented automatically.

Alternative Event Flow II

iii. User moves the cursor near a bracket or a brace.
iv. Code editor finds the matching bracket or brace and makes its appearance noticable by the user.

Alternative Event Flow III

iii. User presses CTRL+Space keys to make an auto completion request to the code editor while typing an expression.
iv. Code editor processes the auto completion request and shows the user possible matches if available.

Alternative Event Flow IV

iii. User clicks on the left side of a line to set a breakpoint at that line.
iv. A breakpoint is set at the desired line and a mark is shown on the left side of the the line.

Alternative Event Flow V

iii. User clicks corresponding button to display the breakpoints set before.
iv. A list of the breakpoints set before is shown to the user.

Alternative Event Flow VI

iii. User presses CTRL+F keys to search an expression in the file.
iv. A search bar appears on the bottom of the editor.
v. User types the expression to be searched.
vi. The matching expressions are marked if there is any. Otherwise a warning message showing that there are no matches appears on the search bar.

Alternative Event Flow VII

iii. User chooses the editor theme among the provided themes.
iv. The editor theme is changed according to the user selection.

Alternative Event Flow VIII

iii. Corresponding line numbers are displayed on the left side of the lines.

Functional Requirements

<u>REQ 7:</u> The system shall provide a code editor that supports programming language syntax coloring.

<u>REQ 8:</u> The system shall provide a code editor that is capable of auto indentation.

REQ 9: The system shall provide a code editor that matches brackets and braces when user clicks on one of them.

REQ 10: The system shall provide a code editor that is capable of completing the expressions typed by the user if available.

REQ 11: The system shall provide a code editor that can set a breakpoint at the desired line.

REQ 12: The system shall provide a code editor that can display the breakpoints set until that time.

REQ 13: The system shall provide a code editor that can search and replace the regular expressions provided by the user.

REQ 14: The system shall provide a code editor, appearance of which can be selected by the user.

REQ 15: The system shall provide a code editor that displays the corresponding line numbers on the left side of the editor view.

3.2.3. Debugger

3.2.3.1 Execution Control

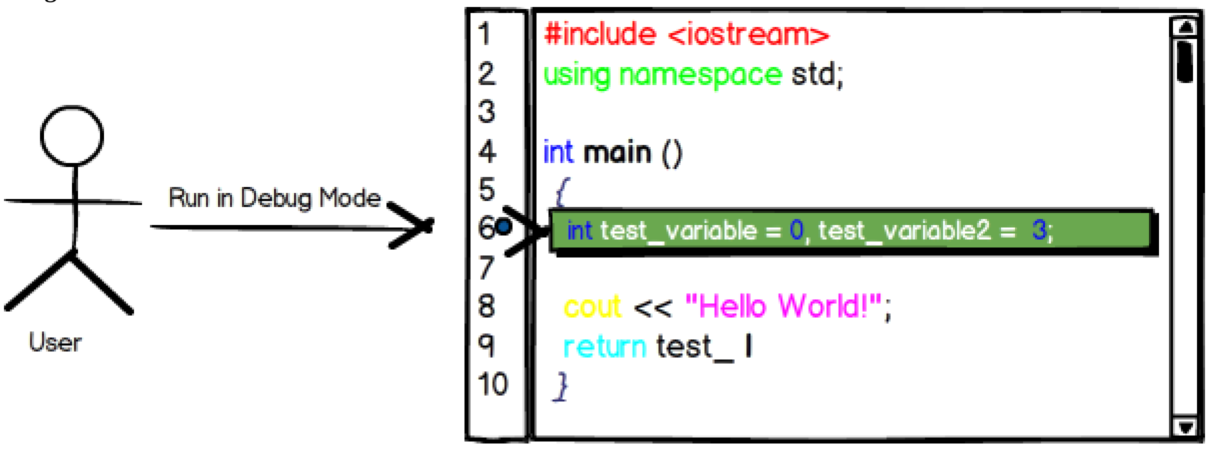
3.2.3.1.1 Background Information

Debugging is a process of finding bugs, defects, and errors by proceeding in program execution step by step. Since debugging is mission critical application in programming, system will provide a debug mode to the user. Different from normal execution, in debug mode user must be able to control to execution of the program. Therefore, system makes available to some functionality such as start, stop, step into, over, and out of the user code.

3.2.3.1.2 Stimulus/Response Sequences

3.2.3.1.2.1 Run in Debug Mode

Diagram



```
#include <iostream>
using namespace std;

int main ()
 {
  int test_variable = 0, test_variable2 =  3;

  cout << "Hello World!";
  return test_ l
 }
```

Description

Primary Actor	User
Goal in Context	Purpose of this feature is to run in debug mode to the selected project
Precondition	User has logged in to the system. There must be at least one executable project
Trigger	User wants to debug his project

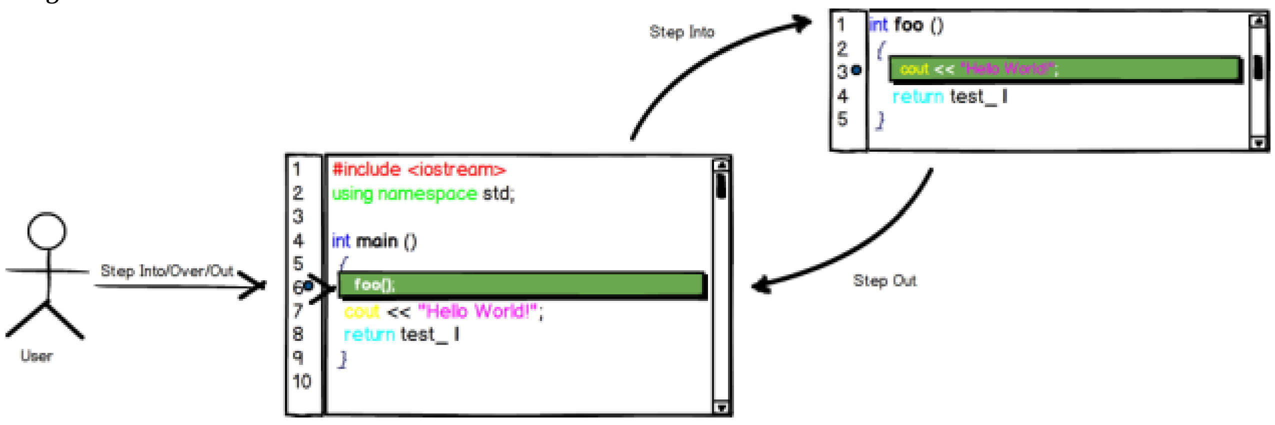
Normal Flow of Events

1. User logins in to the system
2. User selects a project from his workspace
3. User opens a file
4. User adds a breakpoint to the some rows
5. User run project in debug mode

6. Program pauses its execution in a breakpoint
7. Program waits an action from user

3.2.3.1.2.2 Step Into/Over/Out of the Code

Diagram



```
int foo ()
{
  cout << "Hello World!";
  return test_ I
}
```

```
#include <iostream>
using namespace std;

int main ()
{
  foo();
  cout << "Hello World!";
  return test_ I
}
```

Description

Primary Actor	User
Goal in Context	Purpose of this feature is to step into, over, or out from a code in debug mode
Precondition	User has logged in to the system. There must be at least one executable project Project must be run in Debug Mode
Trigger	User wants to step into, over, or out from a code

Normal Flow of Events

1. User logins in to the system
2. User selects a project from his workspace
3. User opens a file
4. User adds a breakpoint to the some rows
5. User run project in debug mode
6. Program pauses its execution in a breakpoint
7. Program waits an action from user
8. User step into a function
9. User step out from the function
10. User step over of current line

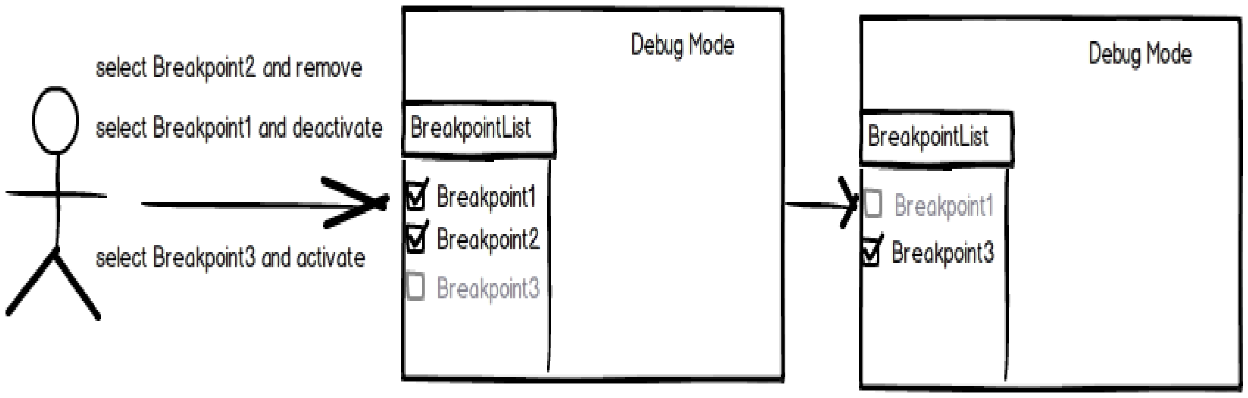
3.2.3.1 Breakpoint List

3.2.3.1.1 Background Information

In debug mode users shall be able to control execution of the program. For this purpose users will use breakpoints which are intentional stopping or pausing places in a program. All breakpoints decided by users will be listed in breakpoints view. When user adds a breakpoint it will be automatically shown in this view. In this view user will be able to activate and deactivate the breakpoints. Also he will be able to remove the breakpoint completely.

3.2.3.1.2 Stimulus/Response Sequences

3.2.3.1.2.1 Diagram



3.2.3.1.2.2 Description

Primary Actor	User
Goal in Context	Purpose of this feature is make easier manage of breakpoints by showing all of them in a list.
Trigger	User wants to see all break points together

3.2.3.1.2.3 Normal Flow of Events

i. User is in the debug mode
ii. Removes a breakpoint from list

3.2.3.1.2.4 Alternative Flow of Events
Alternative Event Flow 1
ii. Deactivate an active breakpoint

Alternative Event Flow 2
iii. Activate an inactive breakpoint

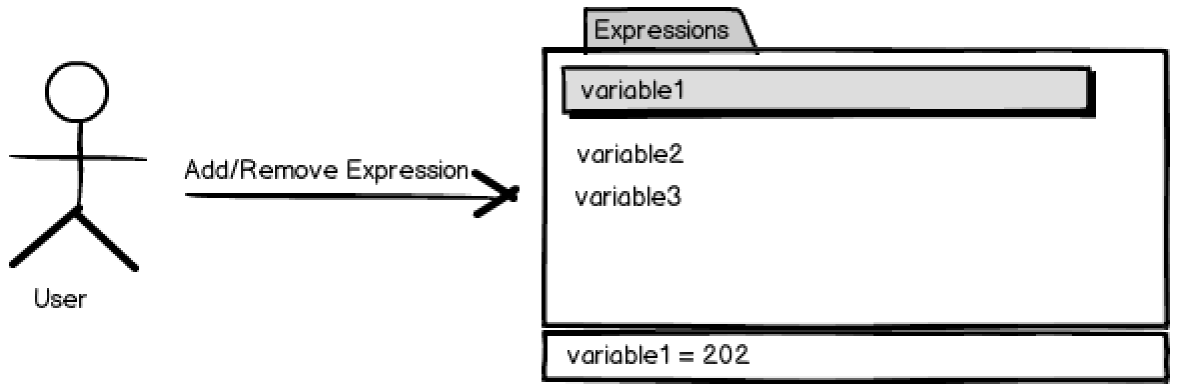
3.2.3.2 Expression Evaluation

3.2.3.2.1 Background Information

In debug mode following the value of certain expressions could be crucial for the user and he wants to follow the values of these expressions in every step of debugging. To make this processes easier values of specified expression automatically will be evaluated at each breakpoint and shown to user. Expression specifications will be handled in expression tab which will be a part of debug mode interface. User will be able to add and remove expressions thorough this tab. If user defines an invalid expression will not be added to expression list and user will be notified.

3.2.3.2.2 Stimulus/Response Sequences

3.2.3.2.2.1 Diagram



3.2.3.2.2.2 Description

Primary Actor	User
Goal in Context	Purpose of this feature is show values of wanted expression in every breakpoint
Trigger	User wants to follow the changes in values of some certain expressions

3.2.3.2.2.3 Normal Flow of Events

i. User is in the debug mode

ii. User opens the expression evaluation tab

iii. Defines an expression

iv. Expression added to evaluated expressions list

3.2.3.2.2.4 Alternative Flow of Events

Alternative Event Flow 1

iv. If the expression is not valid it is not added to list

v. User is notified about invalid expression

3.2.3.3 Functional Requirements

REQ 16: The system shall provide a debug mode that support breakpoints list demonstration

REQ 17: The system shall provide a debug mode that support expression evaluation

REQ 18: The system shall allow user to control the execution of the running process.

3.2.4. Terminal

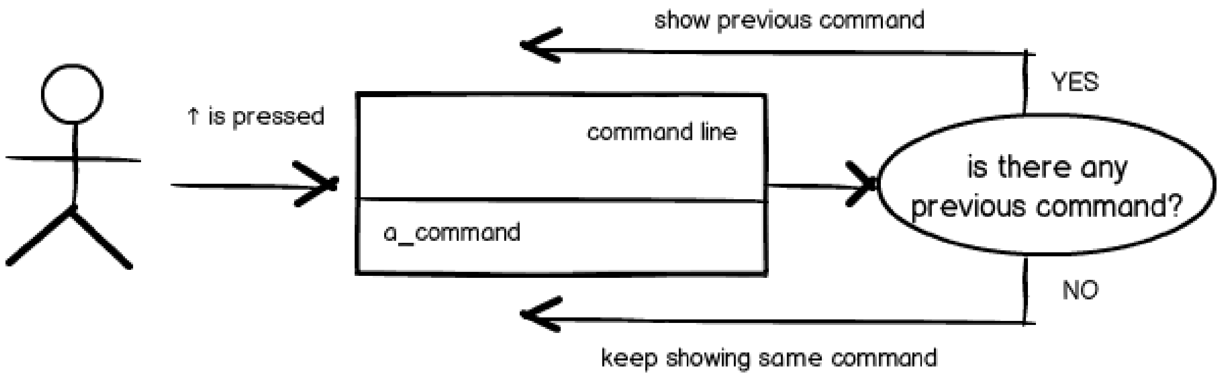
3.2.4.1 Viewing Previous Commands

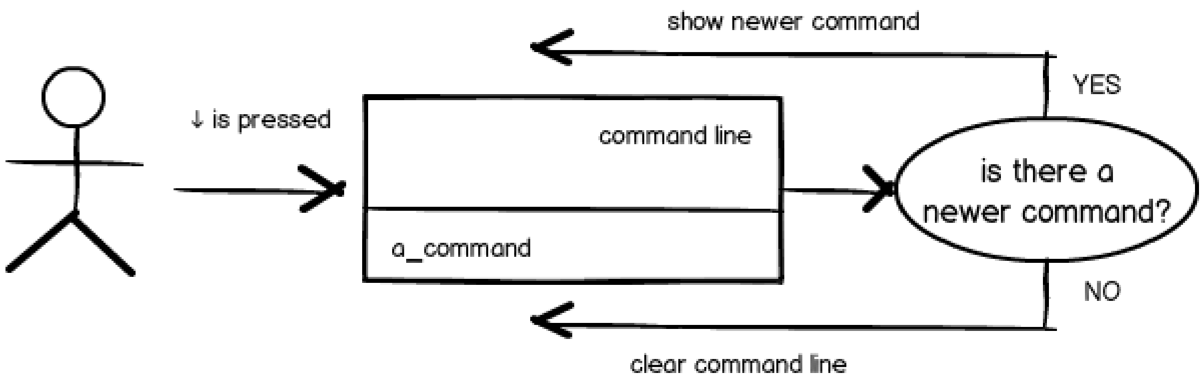
3.2.4.1.1 Background Information

Users will be able to view the previous commands they entered to terminal since the beginning of their sessions. In order to provide this feature, commands will be saved in a command stack. In user interface travelling through commands will be handled by keyboard shortcuts. By using up arrow key users will be able to view older commands while they will be able to view newer commands by using down arrow key. When the oldest command is reached up arrow key will not respond anymore and when the current command is the newest one and down arrow key is pressed command line will be cleared.

3.2.4.1.2 Stimulus/Response Sequences

3.2.4.1.2.1 Diagram





3.2.4.1.2.2 Description

Primary Actor	User
Goal in Context	Purpose of this feature is make user’s job easier by giving him availability to access his previous command easily.

Trigger	User does not want to type same commands again and again

3.2.3.1.2.3 Normal Flow of Events

i. User is on the command line

ii. Presses up arrow key

iii. Views his last command

iv. Presses up arrow key again

v. Views the previous command of the command currently viewing

vi. Keeps viewing the previous commands while presses the up arrow key until the oldest command is reached

vii. Keeps viewing the newer commands while presses the down arrow key until the newest command is reached

3.2.3.1.2.4 Alternative Flow of Events

Alternative Event Flow 1

ii. Presses down arrow key

iii. Nothing happens

Alternative Event Flow 2

iii. Nothing happens if there is no previous command

Alternative Event Flow 3

iv. Presses down arrow key

v. Command line is cleared

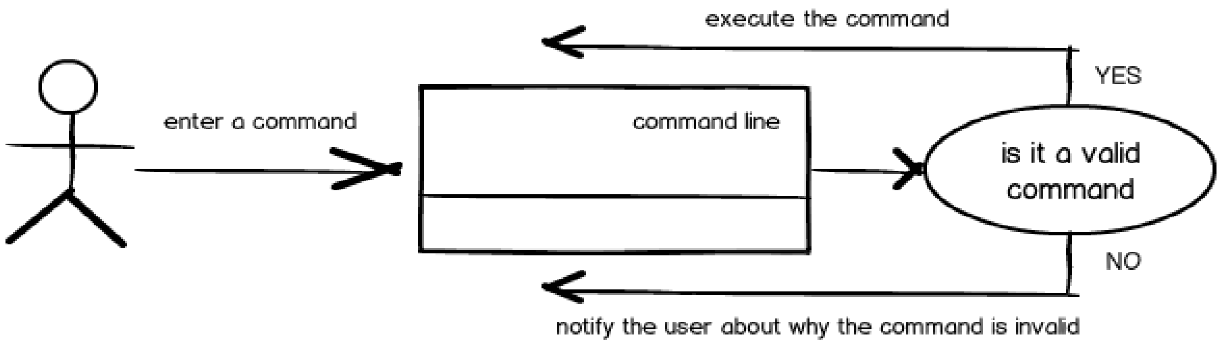
3.2.4.2 Support for Basic UNIX Terminal Commands

3.2.4.2.1 Background Information

Terminal will give opportunity to users to use basic UNIX-like terminal commands. With basic we mean the commands which do not require sudo privileges. By using these commands, in their workspaces user will be able to perform all actions they can perform in a UNIX machine.

3.2.4.2.2 Stimulus/Response Sequences

3.2.4.2.2.1 Diagram



3.2.4.2.2.2 Description

Primary Actor	User
Goal in Context	Purpose of this feature is giving users to chance to organize their workspaces with command line interface.
Trigger	User wants to use UNIX-like commands

3.2.4.2.2.3 Normal Flow of Events

i. User is on the command line

ii. Enters a command

iii. Command is executed

iv. Result of the command is shown in command line or workspace

3.2.4.2.2.4 Alternative Flow of Events

Alternative Event Flow 1

iii. Command is not executed if not all necessary parameters are specified
iv. An error message is shown in command line interface which indicates the required parameters for that command

Alternative Event Flow 2

iii. Command is not executed if there is no such defined command
iv. An error message is shown in command line interface which indicates this situation

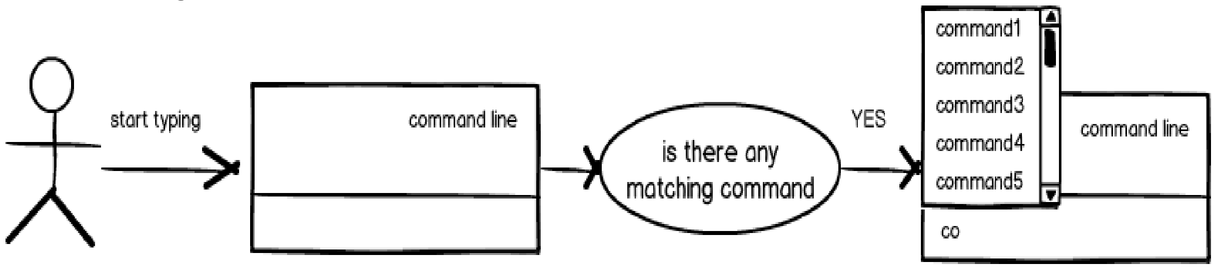
3.2.4.3 Listing Available Commands

3.2.4.2.1 Background Information

When users start typing in terminal, commands include the phrase typed so far in their names will be listed in a scrolled pop-up lightbox. Box will be open only if there is any command to be listed. Otherwise box will be closed. In this box commands will be ordered according to their relevancy to typed phrase and there will be short descriptions besides the commands. Also by typing "help" in the command line users will able to see the list of all commands. By this features users will be able to find the commands they need easily even they have not used these commands before.

3.2.4.2.2 Stimulus/Response Sequences

3.2.4.2.2.1 Diagram



3.2.4.2.2.2 Description

Primary Actor	User
Goal in Context	Purpose of this feature is to decrease the invalid command entrance by showing available commands
Trigger	User wants to know available commands and wants help to remember commands

3.2.4.2.2.3 Normal Flow of Events

i. User is on the command line
ii. Starts typing a command
iii. Pop-up lightbox which shows the relevant commands is opened
iv. User chooses a command from this box
v. Presses enter to execute command

3.2.4.2.2.4 Alternative Flow of Events
Alternative Event Flow 1
iii. Pop-up lightbox is not open if there is no matching command

Alternative Event Flow 2
iv. User does not choose any command from the list and keep typing
v. Presses enter when he wants to execute the command

3.2.4.4 Functional Requirements

REQ 19: The system shall provide a terminal that supports previous command viewing

REQ 20: The system shall provide a terminal that supports basic UNIX commands execution.

REQ 21: The system shall provide a terminal that supports available commands list demonstration.

3.3. Non-functional Requirements

In this section, last group of the requirements which is non-functional requirements will be explained in detail. Non-functional requirements include performance requirements, security requirements and portability requirements.

3.3.1 Performance Requirements

Since this software is going to web – based, it does require a powerful server machine with high band internet access.

Server machine should have a powerful CPU and high speed internet access so that it can handle multiple users at the same time. Another performance requirement is the storage space. Higher storage space means more user and bigger workspace per user so higher the storage, better the performance.

Performance requirement by the user side is, web application should be developed as a lightweight web app so that it can work on almost any platform even with slower internet connections.

Expected number of simultaneous user should be at least 100. System should be able to deal with 100 users at the same time. Also database of the system should handle at least a thousand of users at any periods.

3.3.2 Security Requirements

Since this software will be hosted on cloud server, all the user data will be kept on the cloud server. Product should be able to protect privacy of user data. Workspace of the user should only be accessed through user own credentials and any other user should not be able to access to the user private data.

Since execution will also be done in the machine in the cloud, user should be restricted in terms of user rights. User should only access to his own workspace and should not access to any other workspace with the programs they run on the cloud. Also rights of the user should be restricted so that user can not harm to system by the programs they run or by the commands they run on terminal.

Since all the data will be transferred on the web, system should also use an encryption and decryption mechanism only intended user can decode the data and work on the data.

3.3.3 Portability Requirements

Main purpose of developing web-based IDE is to improve the portability of software development process. To improve portability, software should run on variety of platforms and variety of connection speeds.

As explained in the performance requirements section, software should be lightweight so that it can run on a machine with slow internet connection. To make the web application lightweight, simple libraries and tools should be used at developing phase. Such as using JavaScript and HTML5 instead of Apache Flex.

Portability also means running on most number of different platform without an additional effort. To achieve this, web application should be developed by using the common technologies and tools which are provided by all common web browsers and operating system such as HTML5, JQuery etc.