Software Design Description for Web Based Integrated Development Environment

DEVCLOUD Web Based Integrated Development Environment

TinTin

Alican Güçlükol
Anıl Paçacı
Meriç Taze
Serbay Arslanhan

UPDATES

Section Name	Section Number	Status
Module Decomposition	3.1	changed
ExternalInterface Module Description	3.1.1	changed
ExternalInterface Interface Description	5.1.1	changed
ExternalInterface Detailed Design	6.1.1	changed
UserManager Module Description	3.1.3	changed
UserManager Interface Description	5.1.3	changed
UserManager Detailed Design	6.1.3	changed
FileModule Module Description	3.1.5	changed
FileModule Interface Description	5.1.5	changed
FileModule Detailed Design	6.1.5	changed
CommandLineManager Module Description	3.1.9	changed
CommandLineManager Interface Description	5.1.9	changed
CommandLineManager Detailed Design	6.1.9	changed

1. INTRODUCTION

This document describes the design for the web based integrated development environment. To be able to modularize the software in proper way, first we will decompose project into modules which are different than the others in terms of structure and functionality. Purpose of this decomposition is as follows, when software decomposed into separate modules, than it is easier to implement these modules separately and test them. Also any possible change on one of the modules will have a minimal effect on the other modules. So decomposition is obviously a necessary and beneficiary part of software design. After giving decomposition of the system, we will describe the dependencies of among these separate modules. In most cases, there would be dependency between modules even if they have been separated in a best possible way. These dependencies will be in form of the relationship between different modules. After dependency section, interface description follows. These interface descriptions will be enough for a user to use that module. Hence, these 3 chapters will describe the modules without going into too much detail from different perspectives. Later, as the last part of these document, detailed description about design of modules will be given. In this section, each component and its internal details will be described in detail

1.1 Purpose

Main purpose of this software design description document is to encompass a design model with architectural, interface, component level and deployment representations. Design model will be contained in this software design description document, which later will be used as a medium for communicating software design information, assessed for quality, improved before code is generated. Many graphical representations such as class diagrams and verbal explanations were added to this document to achieve the goal of producing the web based integrated development environment in the context of design model.

1.2 Scope

In this document, we will specify architecture and detailed design for the web based integrated development environment. We will give identification, type, purpose, function, subordinates, dependencies, interface, resources, processing and data attributes of each

component. We will not, however, give a prototype, which can easily be constructed following the implementation details specified in detailed design part of this document.

2. REFERENCES

IEEE standard 10161998 recommended practice for software design descriptions.

Freeman, P. and A. I. Wasserman , Tutorial on Software Design Techniques. 4th Edition, IEEE Computer Society Press.

Gamma,E.,etal ., Design Patterns, AddisonWeslet, 1995.

UML: The Unified Modeling Language User Guide by G.Brooch, J. Rumbaugh and I.Jacobson, AddionWeslay Pub Co; ISBN:0201571684.

Roger S. Pressman , Software engineering : a practitioner's approach. 6th edition, McGrawHill international edition.

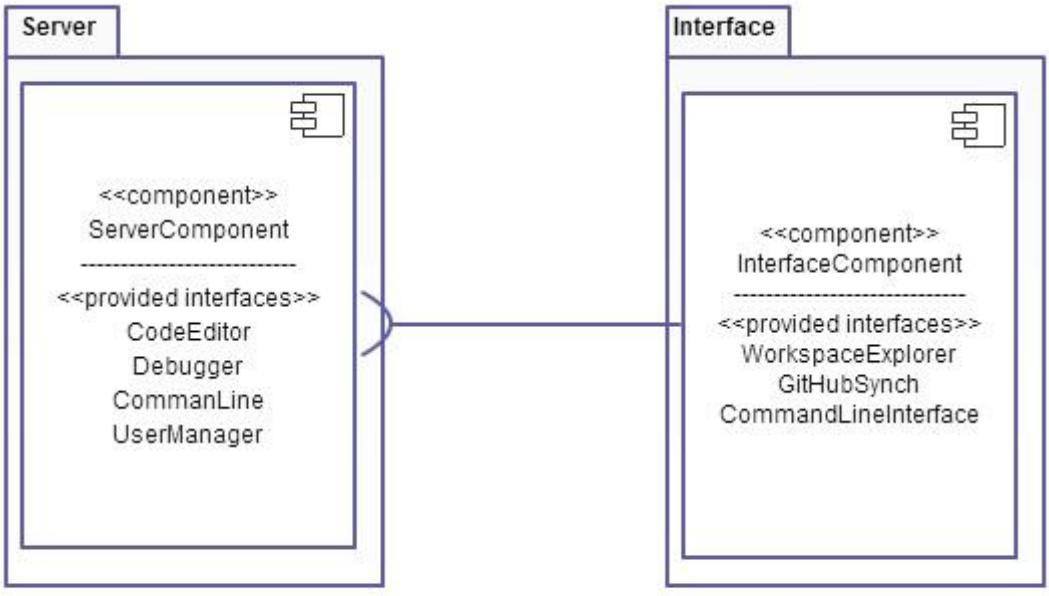
Doğru,A.,Component oriented software engineering , The Atlas Publishing,2006.

3. DECOMPOSITION DESCRIPTION

In this section of the document, we will describe the design entities that composes the whole web based integrated development environment. System is not a one big component which does all functionality itself; instead mainly system decomposes of two main components which are namely Server Component and Interface Component. These components also decomposed into smaller modules according to their structure and functionality.

As the name suggests, Server Component is the part responsible from holding all the user and system data and doing the business logic. There will be one huge server for the system which will hold all the data about the web based integrated development environment. Also, with the help of modules it included, server component will be responsible from doing all the business logic, like user management, compiling and running project etc.

Second main component of the system is the Interface Component as mentioned above. Interface component mainly will be responsible from interaction of user with the Server Component. Its main functionality is the serving the Web front end of the system. Other functionalities of these components will be providing the command line interface and GitHub synchronization interface. However, since these last two functionalities will be served over the Web Front end of the system, Web Front end is the main module of this Interface component. Component will be basically providing a web page will be the graphical user interface of the system. Also it will be the single entry point for all users. Before looking at the each module separately, let's have a look at the following Package Diagram;



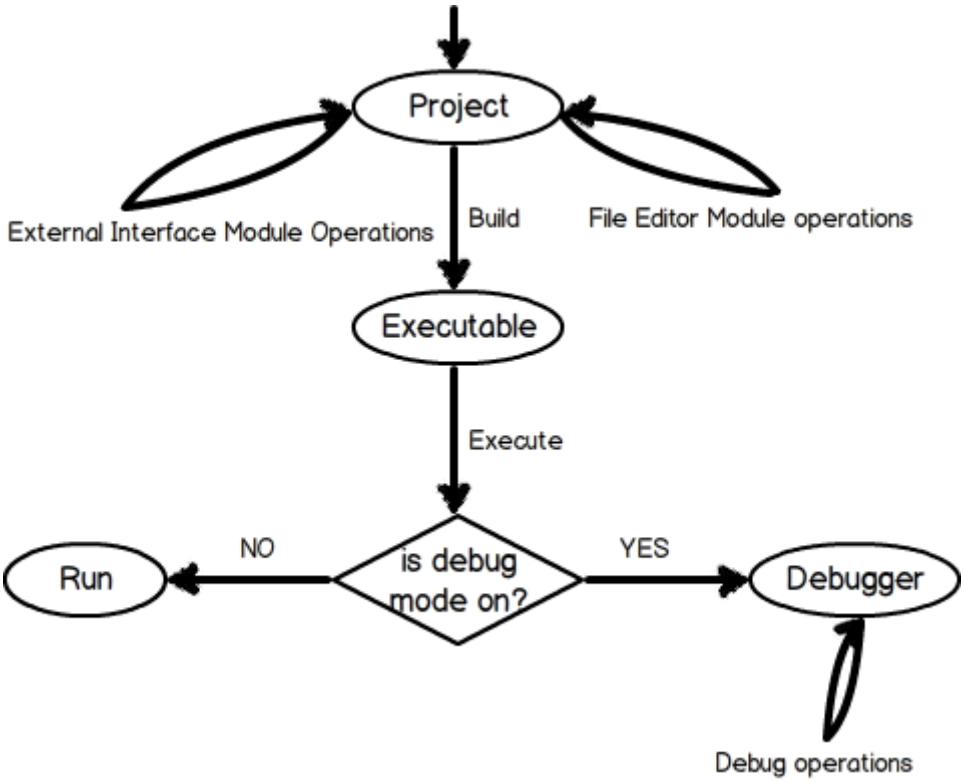
As we have stated before, you can see in above diagram that each main component is composed of other modules. Now we are going to describe each of these modules separately. For each, we will give identification, purpose, function and the subordinates design attributes.

3.1 Module Decomposition

In this section of the document, decomposition and the identification of components into modules can be found. Important point is that web based integrated development environment is not constructed of single module. System is composed of different modules, which are separated than each other by means of structure and functionality. Here, we will keep the design of modules simple so that it can be easily implemented and can serve full functionality of the module. For the each module that will be described name and identification will be given. Identification will not be same for the distinct modules. Later purpose of the module will be given. In the purpose, need for this module will be described. Then function of each module, what module does, will be described. Then subordinated of the modules will be given which are other modules that makes up the system. We will also present class diagram for the entities for the readers to have a better understanding.

Server Component of the web based integrated development environment will be the component where all business logic is done and all the information is kept. First of all, server component will act as a database for the web based integrated development environment system. All the user data such as projects and files of the user, preferences of user setting and all the system data such as the users itself will be kept on the server. Users are able to access only to their own data according to their user privileges. Server will also be responsible from the implementing all the features. Every operation such as compiling a project or running a task will be done on the server and only the effects of operations will be reflected to user. In this manner, Interface Component will only be a connection between the system itself and the user through the web page provided.

What interface component basically does can easily be understood by the preceding paragraphs. Main module of the Interface component will be the web front end which will be the graphical user interface of the system. Communication between server and the client will be done by using the http request, RESTful API. Other two modules of the Interface components will be provided over the web page, so they will actually be a part of web page interface according to their structure.



-State diagram shows available operations on a project-

3.1.1 ExternalInterface Module Description

Background:

External interface will be entry point of the web based integrated development environment for the users. Users will login, logout do the all operations through the external interface and operations will be done on the server.

External interface composes of three parts, namely web front end, GitHub synchronization and command line interface. These parts are not going to implement any business logic, they only will provide and API for users to perform their related operations on the Server component. Main component of the external interface will the web front end.

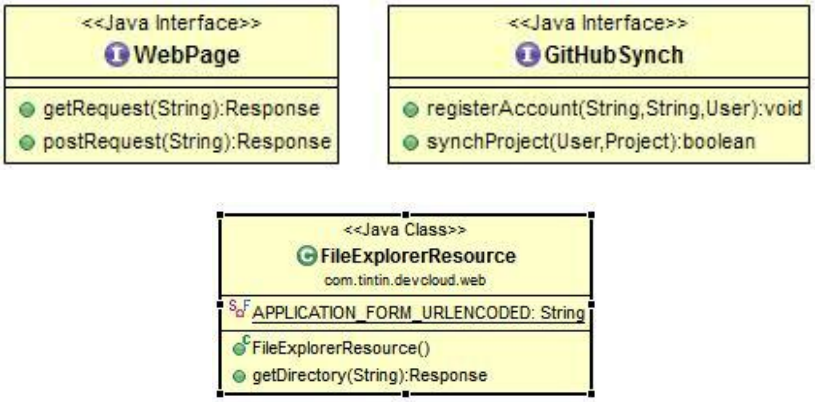
Web front will basically be a web page which will be the single entry point for the users. Users will login through the web page and create projects, edit projects, run and compile their source code and etc. Web page will not be a static web page, it will be a dynamic web page fully implemented in JavaScript. JavaScript application on the web page will perform operations on the server through the RESTful API.

One important functionality served by this module is directory listing. It is mainly intended to be used by the workspace explorer. It is a REST Service gets the name path or name of a directory and return its content in the form of HTML list.

Description:

Identification	ExternalInterface
Purpose	The purpose is the provide user interaction with Server Component
Function	This component provides an interface through a web page for user to perform operations on the server. It will only be a bridge between user and the server and is will only be responsible from performing operations on the server. It will not implement any business logic.

Diagram:



3.1.2 User Module Description

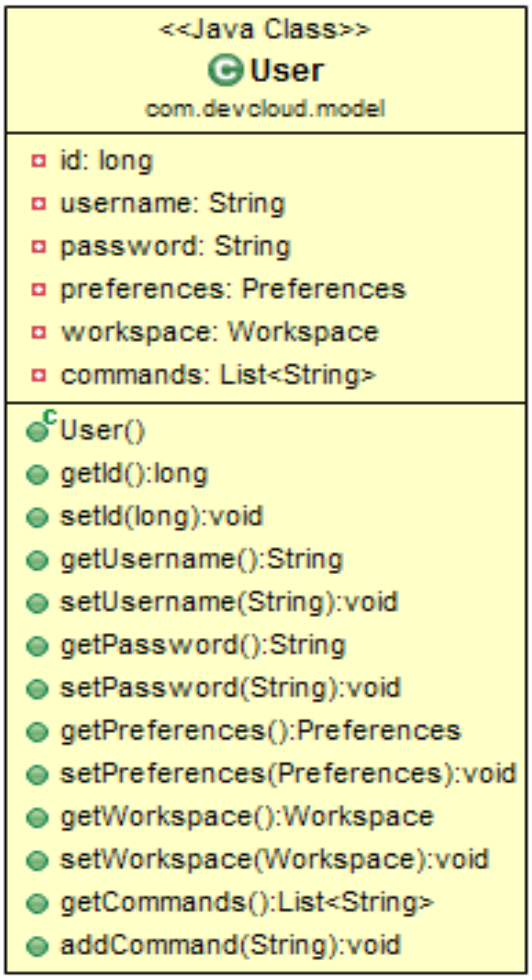
Background:

This module will provide functionality need by User. It contains basic user information, credentials, and basic functionalities to interact with the environment. Since user is the main actor of the system, all modules must be available to him. By using getter/setter of environment variables, he can interact with system at will.

Description:

Identification	User
Purpose	The aim of this module is to provide functionalities to the user if required.
Function	Both to set attributes of user in registration or update process and to get attributes of user in a request or preparing workspace to user with his preferences, getter and setter methods are defined for all user attributes.

Diagram:



3.1.3 User Manager Module Description

Background:

This module provides functionalities to the user to register, login to the system, and update his information.

This module also provides RESTful services for login, register and update operations.

Description:

Identification	User Manager
Purpose	The purpose of this module is to manage user authentication.
Function	In order to use system, a user must register to the system. After, he must login in to the system, and he could change his information if want. All these functionalities are provided by user manager module.

Diagram:

3.1.4 Execution Manager Module Description

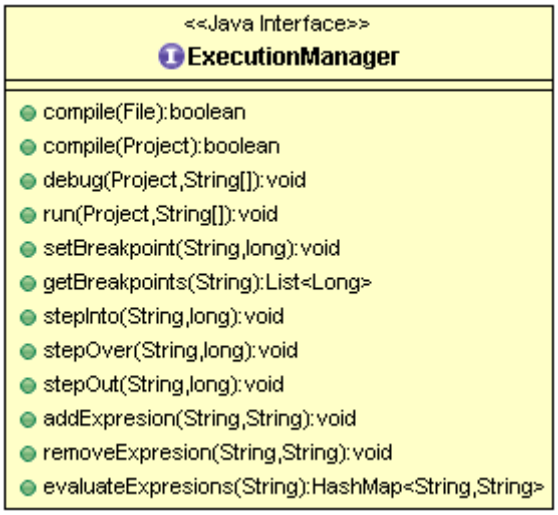
Background:

This module will provide all functionality to build, debug and run a project or a single file. This module will be available to all users. Users will be able to make all kind of execution processes using this module.

Description:

Identification	Execution Manager
Purpose	The aim of this module is managing all execution process by providing required functionality for building, debugging and running.
Function	This module will implement compile methods for both a file and a project. It will also implement debug, run, setBreakpoints, getBreakpoints, stepInto, stepOver, stepOut, addExpression, removeExpression and evaluateExpressions methods. Most of the functions' names are self-explanatory. Expression is anything to be evaluated in each step of debugging. stepInto, stepOver and StepOut methods will be used in debug mode. If the current statement is a function call, then stepInto steps into that function , otherwise it stops at the next statement. stepOver executes the whole function or script, and it stops at the next statement after the function call. stepOut steps out of the current function and up one level.

Diagram:



3.1.5 File Module Description

Background:

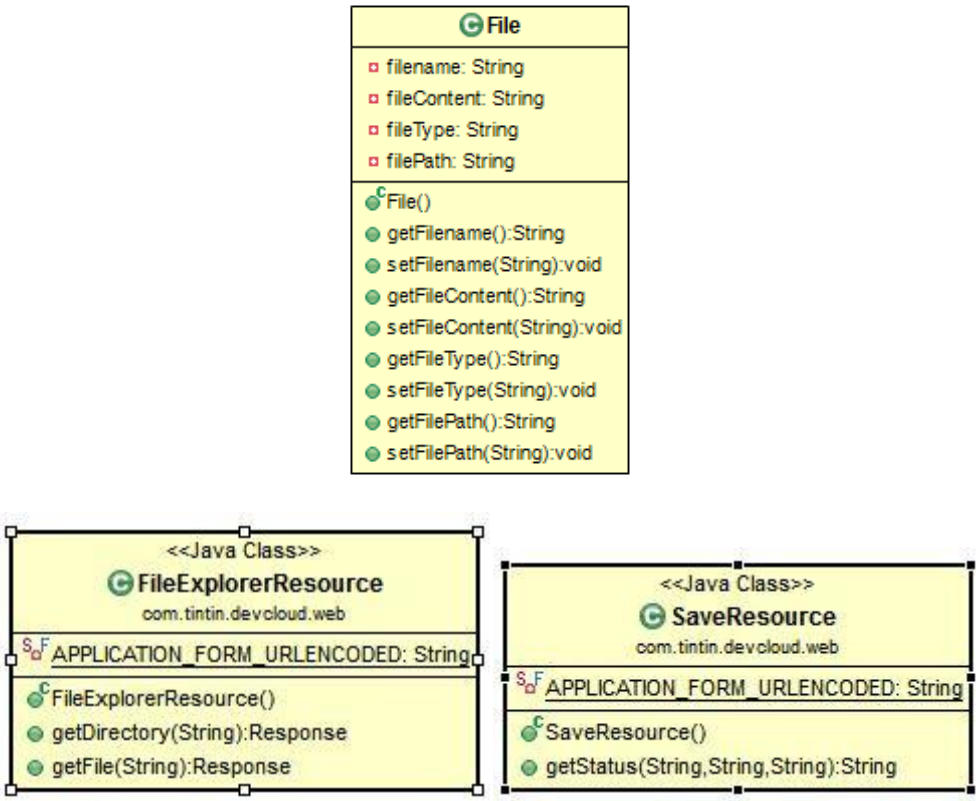
This module models the files edited and displayed in the editor. It consists of four attributes representing file name, file content, file type, and file path.

Also there will be Ajax RESTFul servies for clients to handle file operations.

Description:

Identification	File
Purpose	The purpose is to model the files open in the code editor.
Function	This model is used to identify the files in the code editor by using the private attributes in the model. It can represent the opened files by the user in the browser.

Diagram:



3.1.6 Theme Module Description

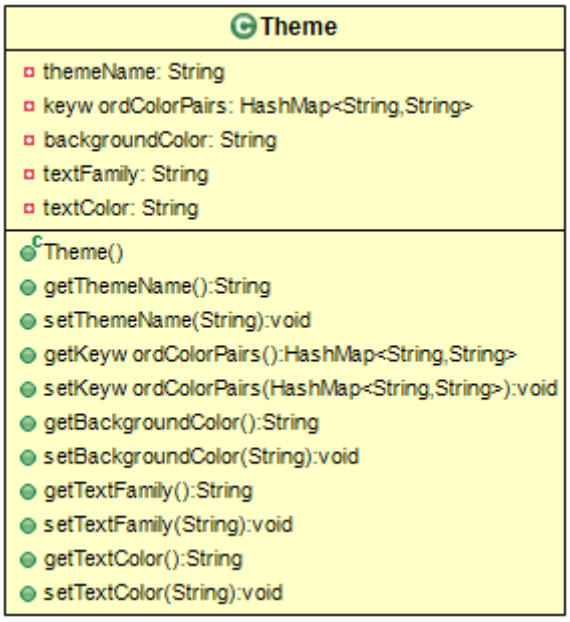
Background:

This module models the themes used in the code editor. It consists of five attributes representing theme name, keyword color pairs that maps keywords to the color codes they are going to be shown, background color, text font family, and text color.

Description:

Identification	Theme
Purpose	The purpose is to model the themes available for the code editor.
Function	This model is used to identify the themes available in the code editor by using the private attributes in the model.

Diagram:



3.1.7 Code Editor Module Description

Background:

Code editor module is the graphical user interface that covers the following product functions by using the interfaces of the other system modules and the methods in it as described in the system requirement specification:

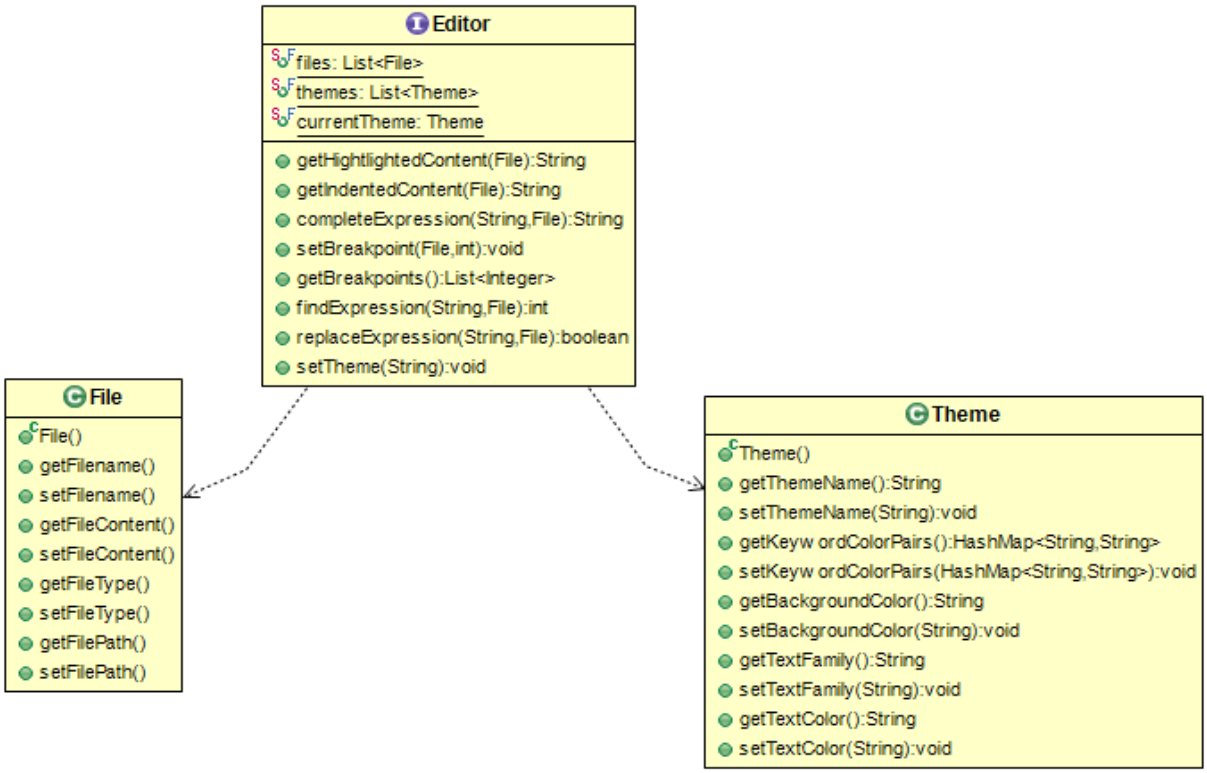
- Syntax highlighting
- Auto-indentation
- Bracket/Brace matching
- Auto-completion
- Setting/Displaying breakpoints
- Find/Replace with regular expression support
- Editor themes
- Displaying line numbers

It consists of three fields representing opened files, themes and current theme in the code editor.

Descripton:

Identification	Editor
Purpose	The purpose is to model the code editor itself.
Function	This model is used to identify the code editor. It uses File and Theme models to represent opened files, available themes, and the current theme for the code editor. It also consists of several public methods to achieve the requirements mentioned above.

Diagram:



3.1.8 Command Module Description

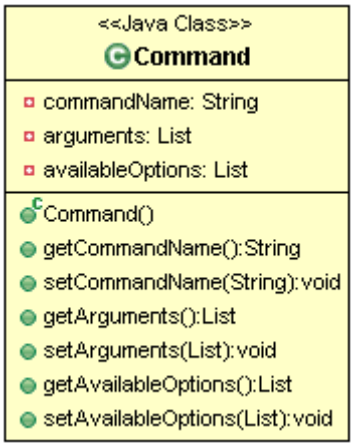
Background:

This module will be used to model a command as its name suggests. It contains name, arguments and options information for a valid command. This module will be used keep all information about a command in an organized way.

Description:

Identification	Command
Purpose	The aim of this module is to keep all information about a command in an organized way
Function	This module will implement only setter-getter methods for its attributes commandName, arguments and and availableOptions.

Diagram:



3.1.9 Command Line Manager Module Description

Background:

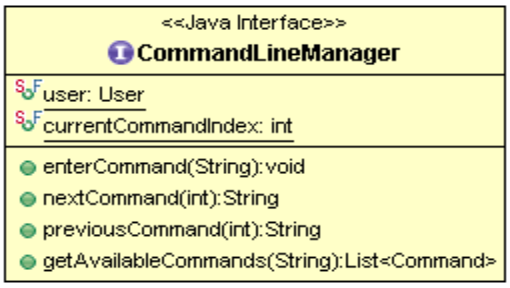
This module will provide users to required functionality to use command line interface. It contains basic functionalities which are provided by standard Unix-like command line. This module will be available to all users. Users will be able to made basic command line jobs using this module.

This module also provides WebSocket Interface to facilitate bidirectional data exchange between the web browser and the server.

Description:

Identification	Command Line
Purpose	The aim of this module is to provide required functionalities to use command line interface.
Function	This module will implement enterCommand, nextCommand, previousCommand and getAvailableCommands methods. enterCommand will get command entered by user and execute it on current workspace. previousCommand and nextCommand are return respectively previous and next commands of current selected command in command stack and getAvailableMethods gets string and returns commands contains this string.

Diagram:



3.2 TRACEABILITY MATRIX

In this section, cross reference of software requirements specification and software design description modules will be given. As it has been stated before, aim of to this design is to keep design as much as possible so that implementation, testing and maintaining would require minimal effort. Also system should provide user all the functionality. Purpose of the below table is to match requirements and modules so that one can trace which requirement is implemented by the modules in web based integrated development environment system.

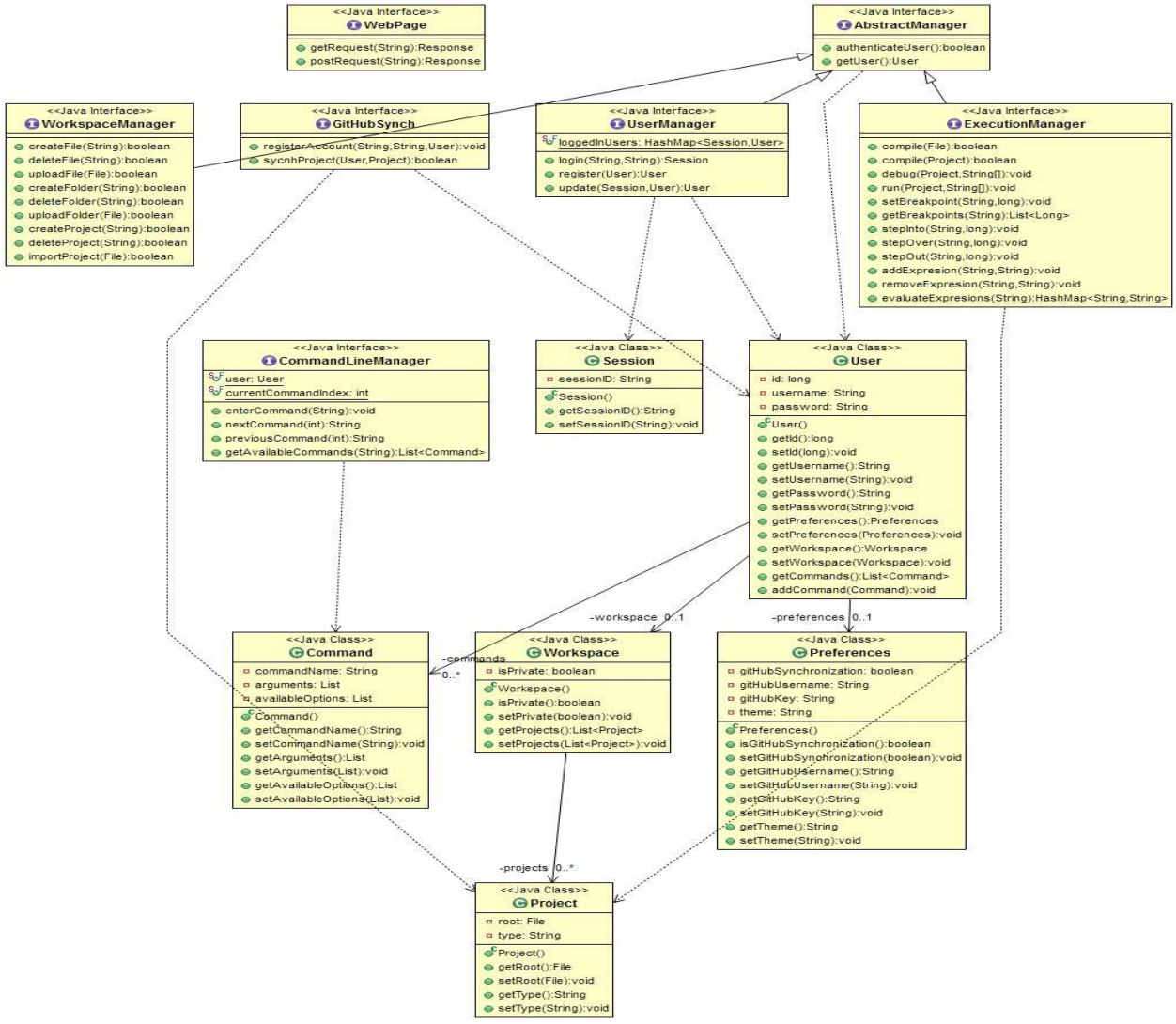
SRS REQUIREMENTS	SDD MODULES
3.1.1	3.1.1
3.1.2	3.1.1
3.1.3	3.1.1
3.2.1	3.1.2, 3.1.3
3.2.2	3.1.5, 3.1.6, 3.1.7
3.2.3	3.1.4
3.2.4	3.1.8 3.1.9

4. DEPENDENCY DESCRIPTION

In this chapter, dependencies and relationships among the module described above will be given

4.1 Inter-module Dependencies

In the previous chapter, design entities which decompose the web based integrated development environment was given. Now we will start to put those modules into together to make the system. Below class diagram represents the relationships and dependencies among the modules of our Server component. Since interface component is simply the bridge between the user and system, in this chapter we will focus on the modules of Server component. Below mentioned class diagram and rest of the chapter will basically be detailing the diagram.



4.1.1 ExternalInterface Module Dependency

Dependencies	Type	Description
None	-	-

4.1.2 User Module Dependency

Dependencies	Type	Description
Workspace	aggregation	User objects have a Workspace object.
Preferences	aggregation	User objects have a Preference object.
UserOperationException	association	User objects can throw UserOperationException if needed.

4.1.3 User Manager Dependency

Dependencies	Type	Description
User	association	User Manager objects have an access to User objects.
Session	association	User Manager objects have an access to Session objects.
UserOperationException	association	User Manager objects can throw UserOperationException if needed.

4.1.4 Execution Manager Dependency

Dependencies	Type	Description
None	---	---

4.1.5 File Module Dependency

Dependencies	Type	Description
None	-	-

4.1.6 Theme Module Dependency

Dependencies	Type	Description
None	-	-

4.1.7 Code Editor Module Dependency

Dependencies	Type	Description
File	association	Editor objects have an access to File objects.
Theme	association	Editor objects have an access to Theme objects.

4.1.8 Command Module Dependency

Dependencies	Type	Description
CommandLineManager	association	Each CommandLineManager could return list of Command objects when needed.

4.1.9 Command Line Manager Module Dependency

Dependencies	Type	Description
User	aggregation	CommandLine objects have a User object.
Command	associacition	Each CommandLineManager could return list of Command objects when needed.

5. INTERFACE DESCRIPTION

In this chapter, interfaces of the web based integrated development environment will be given. This chapter will provide everything that users, implementers, testers need to know to use the system

5.1 Module Interfaces

In this section, we will provide an interface table for each module, which contains name, type, accessibility and type signature of methods and attributes. Then we will describe methods and attributes in detail.

5.1.1 ExternalInterface Module Interface Description

Name	Type	Accessibility	Signature
getRequest	method	public	Response(string)
postRequest	method	Public	Response(string)

Name	Type	Accessibility	Signature
registerAccount	method	public	void(String, User, User)
synchProject	method	public	boolean(User, Project)

Name	Type	Accessibility	Signature
getDirectory	method(POST)	public	Response((FormParam)String)

getRequest:

Performs an http get request on the Server.

postRequest:

Performs a post request on the Server.

registerAccount:

Registers a GitHub account for further synchronizations.

synchProject:

Synchronizes the project with GitHub repository which has been matched before.

getDirectory:

Returns the content of the directory in form of HTML list to be used by Workspace explorer.

5.1.2 User Module Interface Description

Name	Type	Accessibility	Type signature
id	attribute	private	long
username	attribute	private	String
password	attribute	private	String
preferences	attribute	private	Preferences
workspace	attribute	private	Workspace
commands	attribute	private	List<Commands>
getId	method	public	long (void)
setId	method	public	void (long)
getUsername	method	public	String (void)
setUsername	method	public	void (String)
getPassword	method	public	String (void)
setPassword	method	public	void (String)
getPreferences	method	public	Preferences (void)
setPreferences	method	public	void (Preferences)
getWorkspace	method	public	Workspace (void)
setWorkspace	method	public	void (Workspace)
getCommands	method	public	List<String> (void)
addCommand	method	public	void (String)

id:

Stores id of User.

username:

Stores username of User

password:

Stores password of User.

preferences:

Stores preferences of User.

workspace:

Stores workspace of User.

commands:

Stores previously entered commands of User.

getId:

Returns id attribute of the user from User object.

setId:

Sets id attribute of the user in registration process. If given id is already exists, it throws an UserOperationException with an error message.

getUsername:

Returns username attribute of the user from User object.

setUsername:

Sets username attribute of the user in registration process, and user can update it later on.

getPassword:

Returns password attribute of the user from User object.

setPassword:

Sets password attribute of the user in registration or update process.

getPreferences:

Returns Preferences object of the user from User object.

setPreferences:

Sets the user preferences

getWorkspace:

Returns Workspace object of the user from User object.

setWorkspace:

Sets the user workspace

getCommands:

Returns commands attribute of the user from User object.

addCommand:

Adds new command to the command attribute.

5.1.3 User Manager Module Interface Description

Name	Type	Accessibility	Type signature
loggedInUsers	attribute	public	HashMap<Session, User>
login	method(POST)	public	Response (String, String)
register	method(PUT)	public	Response(User)
update	method(POST)	public	Response (Session, User)
getUser	method(GET)	public	Response(Session)

loggedInUsers:

Stores Session and User objects of currently logged in users in a HashMap

login:

POST method that returns Session in Response object of the logged in user if given credentials are matched with database, otherwise gives an error message.

register:

PUT method that returns success in Response if specified information is in correct format, otherwise gives an error message.

update:

POST method that returns updated User object in Response if updated information are in correct format, otherwise throw an User Operation Exception.

getUser:

GET method that returns Session in Response after login process

5.1.4 Execution Manager Module Interface Description

Name	Type	Accessibility	Type signature
compile	method	public	Boolean (File)
compile	method	public	Boolean (Project)
debug	method	public	void (Project, String)
run	method	public	void (project, String)
setBreakpoint	method	public	void (String, long)
getBreakpoints	method	public	List<Long> (String)
stepInto	method	public	void (String, long)
stepOver	method	public	void (String, long)
stepOut	method	public	void (String, long)
addExpresion	method	public	void (String, String)
removeExpresion	method	public	void (String, String)
evaluateExpresions	method	public	HashMap<String, String> (String)

compile:

Returns true if the given file or project successfully compiled, if not, it throws Execution Operation Exception.

debug:

Runs the executable compiled from project in debug mode.

run:

Runs the executable compiled from project in normal mode.

setBreakpoint:

Sets breakpoint to specified line in the given executable.

getBreakpoints:

Returns list of breakpoint lines from the given executable.

stepInto :

Execution steps into the function at the current line while running in debug mode.

stepOver:

Execution steps over the current line while running in debug mode.

stepOut:

Execution steps out from the current function while running in debug mode.

addExpresion:

Adds expression to watch its value when it is available.

removeExpresion:

Removes expression from watched list.

evaluateExpresions:

Evaluates value of expressions in each step.

5.1.5 File Module Interface Description

Name	Type	Accessibility	Type signature
filename	attribute	private	String
fileContent	attribute	private	String
fileType	attribute	private	String
filePath	attribute	private	String
setFilename	method	public	void (String)
getFilename	method	public	String (void)
setFileContent	method	public	void (String)
getFileContent	method	public	String (void)
setFileType	method	public	void (String)
getFileType	method	public	String (void)
setFilePath	method	public	void (String)
getFilePath	method	public	String (void)

Name	Type	Accessibility	Type signature
getFile	Method(GET)	Public	Response()
saveFile	Method(POST)	public	String

filename:

Stores the name of the file.

fileContent:

Stores the content of the file.

fileType:

Stores the type of the file.

filePath:

Stores the path of the file in the server.

setFilename:

Setter method for the filename attribute of the File object.

getFilename:

Getter method for the filename attribute of the File object.

setFileContent:

Setter method for the fileContent attribute of the File object.

getFileContent:

Getter method for the fileContent attribute of the File object.

setFileType:

Setter method for the fileType attribute of the File object.

getFileType:

Getter method for the fileType attribute of the File object.

setFilePath:

Setter method for the filePath attribute of the File object.

getFilePath:

Getter method for the filePath attribute of the File object.

saveFile:

POST method to save given content under the given file

getFile:

GET method to obtain content of a file

5.1.6 Theme Module Interface Description

Name	Type	Accessibility	Type signature
themeName	attribute	private	String
keywordColorPairs	attribute	private	HashMap<String, String>
backgroundColor	attribute	private	String
textFamily	attribute	private	String
textColor	attribute	private	String
setThemeName	method	public	void (String)
getThemeName	method	public	String (void)
setKeywordColorPairs	method	public	void (HashMap<String, String>)
getKeywordColorPairs	method	public	HashMap<String, String> (void)
setBackgroundColor	method	public	void (String)
getBackgroundColor	method	public	String (void)
setTextFamily	method	public	void (String)
getTextFamily	method	public	String (void)
setTextColor	method	public	void (String)
getTextColor	method	public	String (void)

themeName:

Stores the name of the theme.

keywordColorPairs:

Stores the keyword-color code pairs for the theme.

backgroundColor:

Stores the background color of the theme.

textFamily:

Stores the text font family of the theme.

textColor:

Stores the text color of the theme.

setThemeName:

Setter method for the themeName attribute of the Theme object.

getThemeName:

Getter method for the themeName attribute of the Theme object.

setKeywordColorPairs:

Setter method for the keywordColorPairsattribute of the Theme object.

getKeywordColorPairs:

Getter method for the keywordColorPairsattribute attribute of the Theme object.

setBackgroundColor:

Setter method for the backgroundColor attribute of the Theme object.

getBackgroundColor:

Getter method for the backgroundColor attribute of the Theme object.

setTextFamily:

Setter method for the textFamily attribute of the Theme object.

getTextFamily:

Getter method for the textFamily attribute of the Theme object.

setTextColor:

Setter method for the textColor attribute of the Theme object.

getTextColor:

Getter method for the textColor attribute of the Theme object.

5.1.7 Code Editor Module Interface Description

Name	Type	Accessibility	Type signature
files	attribute	public	List<File>
themes	attribute	public	List<Theme>
currentTheme	attribute	public	Theme
getHighlightedContent	method	public	String (File)
getIndentedContent	method	public	String (File)
completeExpression	method	public	String(String, File)
setBreakpoint	method	public	void (File, int)
getBreakpoints	method	public	List<Integer> (void)
findExpression	method	public	Int (String, File)
replaceExpression	method	public	Boolean (String, File)
setTheme	method	public	void (String)

files:

Stores the opened files in the code editor.

themes:

Stores the themes available for the code editor.

currentTheme:

Stores the current theme for the code editor.

getHighlightedContent:

This method returns the highlighted version of the file content according to the file type and theme by using color codes.

getIndentedContent:

This method returns the indented version of the file content according to the file type.

completeExpression:

This method returns the possible completion of the given expression in a file.

setBreakpoint:

This method sets a breakpoint at a line by using the execution manager interface.

getBreakpoints:

This method gets a list of breakpoints by using the execution manager interface.

findExpression:

This method returns the first occurrence of the given expression in a file.

replaceExpression:

This method returns true if the replacement of the first occurrence of the given expression in a file. Otherwise it returns false.

setTheme:

This method sets the currentTheme to the theme with the corresponding name.

5.1.8 Command Module Interface Description

Name	Type	Accessibility	Type signature
commandName	attribute	Private	String
arguments	attribute	Private	String
availableOptions	attribute	Private	String
setCommandName	method	Public	void (String)
setArguments	method	Public	void (List<String>)
setAvailableOptions	method	Public	void (List<String>)
getCommandName	method	Public	String (void)
getArguments	method	Public	List<String> (void)
getAvailableOptions	method	Public	List<String> (void)

commandName:

Stores name of the command.

arguments:

Stores list of the arguments command is supposed to take.

setCommandName:

Sets the commandName.

setArguments:

Sets the arguments.

setAvailableOptions:

Sets the availableOptions.

getCommandName:

Returns the name of the command.

getArguments:

Returns the list of arguments.

getAvailableOptions:

Return the list of available options.

5.1.9 Command Line Manager Module Interface Description

Name	Type	Accessibility	Type signature
User	attribute	private	User
enterCommand	method	public	void (String)
nextCommand	method	public	String (int)
previousCommand	method	public	String (int)
getAvailableCommands	method	public	Command (String)

Name	Type	Accessibility	Type signature
sendRequest	method	public	void(String)
getResponse	method	public	String()

user:

Stores User who is using the command line.

enterCommand:

Gets the command entered by user and executes on the current workspace.

nextCommand:

Gets current command's index and return the String value at user's command list's (index+1) position if there is an available command

previousCommand:

Gets current command's index and return the String value at user's command list's (index-1) position if there is an available command

getAvailableCommands:

Gets a string and among available commands return commands which contains this string

6. DETAILED DESIGN

In this chapter, we will provide internal details of the modules. We will explain identification, processing and data attributes of each module. Since interfaces and dependencies of modules have been specified previously, we will not dwell much on these subjects. We will, however, describe sequencing of events, actual process conditions and paths.

6.1 Module Detailed Design

In this section, we will basically provide internal details of each module. To achieve this, we will specify processing and data attributes of each module.

6.1.1 ExternalInterface Module Detailed Design

Processing:

As it has been specified in previous chapters, external interface will provide user interaction with the Server component. Users will be required to login web page to use the web based integrated development environment. Later any action user performed on the Web page will reflected to the server.

Web front end of the web based integrated development environment will be a Backbone.Marionette Application. Marionette is a JavaScript framework which eases the development of dynamic web pages.

In the Server component RESTful services will be implemented. Through these services web page will interact with the Server. Since Marionette provides huge flexibility, every operation is not required to handle by the Server. However, when data changes on the web page or user wants to perform an operation over the Server component, Marionette application will make get and post requests to RESTful API and perform the necessary changes on the web GUI according to the response sent back by the Server.

Also RESTful services for the workspace explorer will be served by this module. POST method will be used to content of a directory. Since jqueryFileTree is used as front end architecture of the workspace explorer and it requires directory content in form of HTML list, this method is required to take a path of a directory as parameter and return content of specified directory in form of HTML list.

Data:

There will be any data attribute specifically kept by the external interface module. All the incoming data will be served to appropriate module. For example when user gave the GitHub repository credentials, these data will be saved on the User module. Since web front end will be dynamic JavaScript application, all the data will be sent to server through the RESTful API.

6.1.2 User Module Detailed Design

Processing:

When user login to the system, his id and password are needed to identify him, and create his session object. Therefore, we need getters and setters for id, username, and password attributes. In addition, after login, system must prepare workspace according to the predefined Preferences and Workspace object by user. System must be able to get, and user must be able to change these objects. Hence, this module also has getters and setters for Preferences and Workspace objects. Lastly, in order to keep user's pre-entered commands to provide quick access later on, the module has getter and adder for commands.

Data:

This module has six attributes namely Id that specifies id of the user as a long, Username that specifies username of the user, Password that specifies password of the user, Preferences that specifies preferences of the user such as theme, GitHub sync, Workspace that specifies workspace and its properties of the user, Commands that specifies commands entered previously by the user.

6.1.3 User Manager Module Detailed Design

Processing:

This module is responsible for user authentication. When user wants to use system, he must be registered to the system if it is not previously, right after login to the system with his credentials. Thus, User Manager module provides register and login RESTful services. In addition, in case user wants to change his credentials, the module provides update method for user. This module also provides getUser method to return User object.

Data:

This module has only one attribute namely LoggedInUsers that specifies online users. In other words, when a user logged in to the system, his Session and User objects will be stored in this attribute.

6.1.4 Execution Manager Module Detailed Design

Processing:

This module manages all execution process. For execution, system must create executable. For this purpose, since it must be able to compile a File or Project object, this module provides compile function with File or Project arguments.

After compilation, system provides run in normal mode or debug mode with created executable. Hence, there are run and debug functions in this module.

To control execution of debug mode, system provides setter and getter for breakpoints, and stepInto, stepOver, stepOut functionalities. During any step of the execution, so that user is able to watch values of attributes, system also provides add remove and evaluate expression functions.

Data:

This module must have a list of long numbers to keep breakpoint lines in increasing order. In addition, in debug mode to store watched expressions and their values, it uses a HashMap.

6.1.5 File Module Detailed Design

Processing:

This module is responsible for file related operations in the code editor. This module provides a level of abstraction above the communication between the server and website on opened files in the editor. It is used by the Editor interface in operations like syntax highlighting, indentation, expression completion, find and replace, etc. Since the attributes of this module are private fields, it also consists of getter and setter methods for these attributes.

This module also provides two RESTFul services to handle request from clients. These services are responsible of file operations. Any client with appropriate permission can get or save a file to server by calling these AJAX services.

Data:

This module has four attributes that represents file name, file content, file type, and the file path on the server.

6.1.6 Theme Module Detailed Design

Processing:

This module is responsible for keeping the themes available for the code editor. It is used by the Editor interface for keeping available themes and current active theme. It is also used in a method that sets the current editor theme by using the theme name.

Data:

This module has five attributes representing the theme name, keyword-color code pairs in a hash map that is used for coloring the specific keywords, background color, text font family, and text color for the editor.

6.1.7 Code Editor Module Detailed Design

Processing:

This module is responsible for keeping information about opened files, available themes and current active theme in the code editor by using File and Theme modules. It also handles the highlighting, indentation, expression completion, find and replace operations, breakpoint setting and displaying by using the execution manager interface.

Data:

This module consists of three public attributes representing list of opened files that is using the File module, list of available themes and a current theme that using the Theme module.

6.1.8 Command Module Detailed Design

Processing:

This module will be used as a model for commands. It has no methods except setter-getter methods for command name, arguments and available options.

Data:

The module will keep name of the command, arguments command need to take and available options which can e used with the command.

6.1.9 Command Line Manager Module Detailed Design

Processing:

Command line manager module provides basic functionality that a standard Unix-like command line have. For this purpose it has four methods which are enterCommand, nextCommand, previousCommand and getAvailableComands.

It will take the command entered by user using enterCommand method and execute it in Unix based server's terminal. If the command is invalid, the situation will be handled by exception class.

previousCommand and nextCommand methods will provide opportunity to travel through commands entered before. These methods will use User's command list to get command history.

When user start typing a command, every time a new character typed getAvaialableCommands method will be used and it will return available commands.

This module also provides two WebSocket methods to exchange data between client and server. Client sends the commands entered to the server while the server sends results when they are generated.

Data:

The module will keep a User object since command history is special for every user.