Matt Bichay
Vanessa Ulloa

Wormhole

Brief Overview:

Wormhole is a cloud storage solution for storing and managing files from one computer to another. The client is built to mimic a Dropbox-esk idea of being able to upload a file from one's computer and pass it over to the server. Then at a later date, the project gives the user the flexibility to remove the file from the server or download it to any machine.

Technology:

The Wormhole client is coded in Java using the Swing library for creating a user interactable interface. Under the hood, Wormhole uses a MySQL database to manage all of the user accounts and registration logic. It then will use HTTP Post and Get requests to interact with a Apache Tomcat7 Servlet for managing all of the file queries and backend logic for file storage and transmission. A key library used in this project is the Apache Servlet Upload Library. This library is a core dependency for decoding HTTP multipart requests when sending a file to the servlet.

Requirements:

ID	Date added	Priority	Description
REQ1	12/07/15	ESSENTIAL	Wormhole shall have a web based or local client for user interaction.
REQ2	12/07/15	HIGH	The client shall have a form of login and registration for new and existing users.
REQ3	12/07/15	ESSENTIAL	The user shall have the ability to upload and download files from a server.
REQ4	12/07/15	HIGH	The user shall be able to download files or upload files from multiple computers.
REQ5	12/07/15	MEDIUM	The files shall be encrypted prior to being uploaded.
REQ6	12/07/15	LOW	Wormhole shall allow the ability to share files amongst users.

Requirements fully realized:

All requirements have been met besides requirements 5 and 6. If the project lasted for more than 4 iterations, requirement 5 was in the design process and requirement 6 could be added with little hassle.

Use cases:

Use Case UC-#	Use Case #1 - Registering for Wormhole
Related Requirements	REQ1 / REQ2
Initiating Actor	Client/Customer/End-user
Actor's Goal	Register for a unique Wormhole ID and password.
Participating Actors	Working Tomcat servlet / backend.
Preconditions	Ability to launch our front-end GUI from the web or desktop (currently undecided).
Postconditions	Created a unique user ID and password for Wormhole.
Flow of events for main success scenario	1. The initiating actor selects the register button. 2. The initiating actor chooses a username and password. 3. The server registers the username and associates it with a specific password. 4. The user will then receive confirmation on a successful registration attempt.
Flow of events for extensions	1. User enters pre-existing username, the server will notify the user that the username has been taken and to try a new one. 2. The confirmation password doesn't match the initial specified password, the user will be prompted to to re-enter the password.

Use Case UC-#	Use Case #2 - Login to Wormhole
Related Requirements	REQ1 / REQ2
Initiating Actor	Client/Customer/End-user
Actor's Goal	Log into Wormhole to access/upload/download files
Participating Actors	Server/User Interface
Preconditions	User Registration has been completed

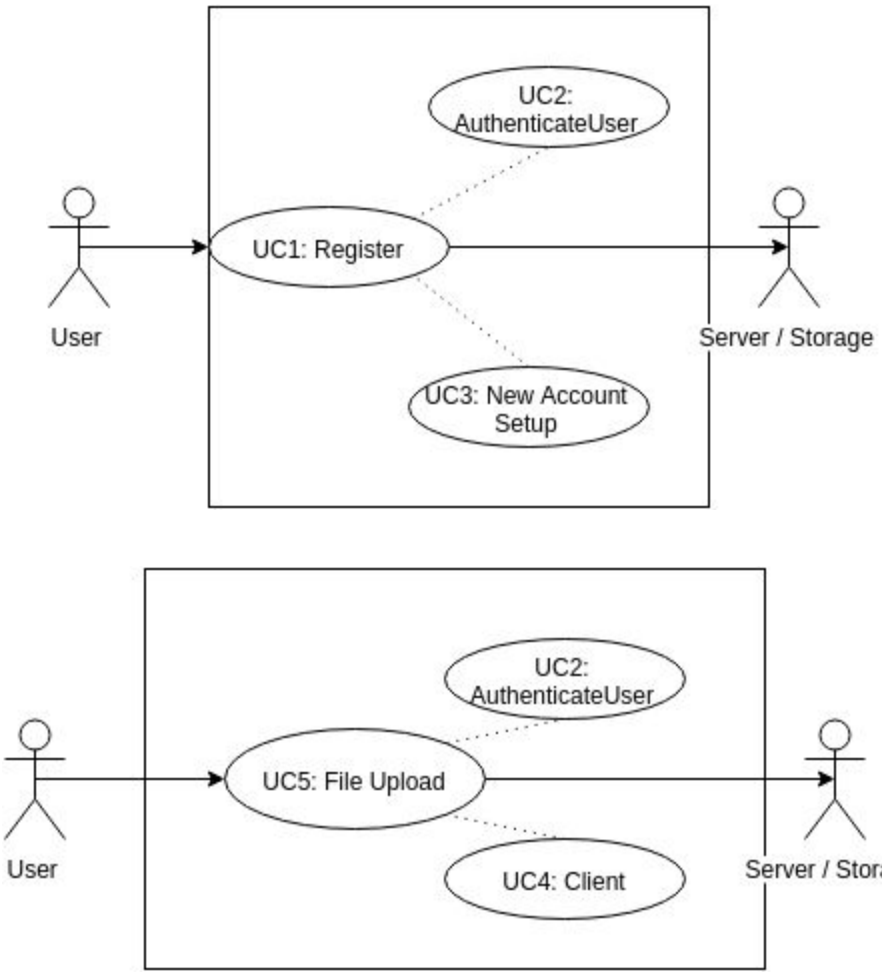
Postconditions	Receive confirmation of Successful Login
Flow of Events for Main Success Scenario	1. User enters username and password into Login screen. 2. Username and Password combination is verified against database information. 3. if Username and Password combination is correct then Login is successful.
Flow of Events for Extensions	1. User enters username and password into Login screen. 2. Username and Password combination is verified against database information. 3. If Username and Password combination is incorrect, then error message of incorrect combination is returned to the user. 4. The Username and Password can be entered again.

Use Case UC-#	Use Case #3 - File Upload to Wormhole
Related Requirements	REQ 2/3
Initiating Actor	Client/Customer/End-user
Actor's Goal	File to be stored on server
Participating Actors	Server/User Interface
Preconditions	User Login is successful
Postconditions	Confirmation of File Upload to server
Flow of Events for Main Success Scenario	1. User is successfully logged into Wormhole 2. User selects file for upload. 3. File is checked for appropriate extension, maximum file size, user has not met maximum uploads. 4. If conditions are met (no disqualifying factors) file is added to the server for that user. 5. Confirmation of successful file upload is displayed.
Flow of Events for Extensions	1. User selects file for upload. 2. File is over the maximum capacity file size or has reached their upload limit. 3. Client relays error message to prompt the user to compress or delete other files to comply with the maximum number of uploads.

	4. Upload process is terminated.

Use Case UC-#	Use Case #4 - Access wormhole from any computer.
Related Requirements	REQ 1/2/3
Initiating Actor	Client/Customer/End-user
Actor's Goal	Enact Use-Case 1,2, and 3 from a GUI client to a AWS server.
Participating Actors	Server/User Interface
Preconditions	User Login is successful
Postconditions	Confirmation of File Upload to server and download from a server
Flow of Events for Main Success Scenario	1. User is successfully logged into Wormhole 2. User selects file for upload. 3. File is checked for appropriate extension, maximum file size, user has not met maximum uploads. 4. If conditions are met (no disqualifying factors) file is added to the server for that user. 5. Confirmation of successful file upload is displayed. 6. Task is repeatable from different box.
Flow of Events for Extensions	1. User is successfully logged into the client. 2. User is not connected to the internet. 3. User cannot make connection to the SQL or Apache servlet through HTTP requests 4. Error message displayed.

Scenario Diagrams:



CRC Cards:

Scenario#1

Class Name:	UtilityRegistration
Responsibilities:	#1 Retrieve user input from the view's controller, (String username_; String password_). #2 Check for a valid password schema. #3 Submit account creation request to the servlet and fire notification to update the view

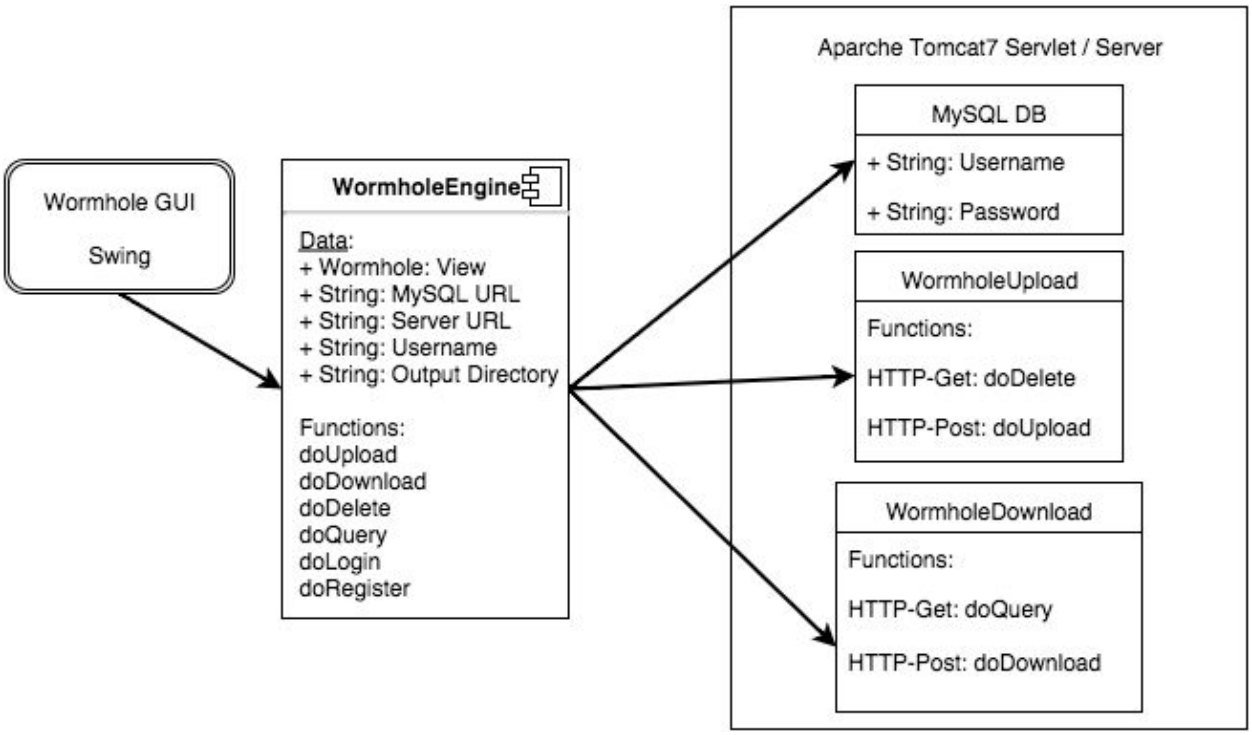
	on creation results.
Collaborations:	WormholeController - GUI Controller for all I/O WormholeRegistration -Servlet class for checking valid registration information and initiating the process for account creation.

Scenario#2

Class Name:	UtilityUpload
Responsibilities:	#1 Send post requests to the Tomcat Servlet for uploading the file. #2 Report upload completion status as well as progress reports.
Collaborations:	WormholeUploadFileController - Class for retrieving filepath from user input. WormholeUploadFileTask - Class giving command to UtilityUpload to make HTTP post requests.

(Note. Some of these CRC cards are out of date in terms of lexicon, classes have since been renamed, please refer to UML/Architecture diagram.)

Back-end UML diagram:



Description:
The GUI is abstracted out of this diagram, it is only used for gathering input from the user and taking command via buttons, which will be forwarded to the Wormhole Engine class. This engine class is full of onClick listeners to listen for requests and inputs from the user to perform 6 basic functionalities:

doRegister: This will ask the MySQL DB if this is a valid registration name, IE: is the account name taken and is the password a good password, if so, it will input the name into the database and log the user in.

doLogin: Similar to doRegister, it will take the input from the user and validate that against the database login and password and decide if the user is a valid user.

doQuery: This one doubles as a server-end registration form, it will create the user a place on the disk if there isn't already one, and it will retrieve all the files stored in the user's account to display them for downloading and deletion purposes.

doUpload: This will take a file from the user's computer, chop it up into 4096 byte chunks and pass it to the server in miniature HTTP multipart requests. The server will then use the Apache servlet upload library to take these incoming requests and assemble the file and output it into the user's directory.

doDownload: This is the exact reverse of the doUpload process and all the user specifies is the name of the file they would like to download.

doDelete: This will pass a filename to the servlet, and the servlet will pass a response back explaining if the deletion was a success or failure.

We do some more backend authentication not displayed on this diagram for authenticating the URL and user inputs.

Large picture architectural diagram: