Inference Firewall System (IFS)

IEEE Standard 1016:2009

Software Design Descriptions

Supervisor:
Jason Jaskolka

Team Members:
Aleksandar Savic - 100999110
Hasan Issa - 100921446
Ryan Zheng - 100996797
Calvin Soong - 100999332
Tashfiq Akhand - 101004428

Table of Contents

1 Introduction

IFS is a middleware for web and database applications, available for any provided databases and their respective users that perform queries on those database. IFS is used to block possible inference attacks on the database through the use of preset administrator defined database policies. The IFS application interacts with a DBMS through the use of existing APIs in the backend and shall only be running if the database and web server are active.

The following sections of the document give a detailed design of the IFS system. It will identify and explain the overall architecture of the system, and provide the reasoning and justification of the design architectures chosen. This document will also identify the main subsystems/components of the system, and provide a brief description of each subsystem.

1.1 Scope

This software system to be produced is the Inference Firewall System (IFS). IFS aims to provide database users with more advanced security features to prevent any sensitive data from being exposed to unauthorized personnel or intruders, but also blocking authorized users from seeing information they are unauthorized to see. The proposed system is a web interface used to provide access control to the database at hand and manage the level of exposure to sensitive data depending on the set security clearance role of an authorized user. The system itself does not insert or modify any data in the database itself, it is merely an interface for users to query data, and provide inference free results. The main objective of this system is to prevent inference attacks on the targeted database through the use of preset database policies and stored query logs.

1.2 Definitions, Acronyms, and Abbreviations

IFS	Inference Firewall System, which is this application name.
API	“Application Programming Interface”. A set of functions and procedures allowing access to the features or data of an operating system, application, or other service.

2 Design Description

IFS will be implemented using the layer pattern, with an interface layer, logic layer, and a data layer. The interface layer includes a WebAPI class, and a SQLWrapperAPI class. The logic layer consists of a Coordinator Class, ResultController Class, PolicyController Class, LogController Class, ConfigController Class, and a DecisionEngine Class. The data layer consists of a Configuration file, and a Log file.

The interface layer is responsible for receiving the user input such as a queries from the search bar, administrator commands such add/modify users and add/modify policies using the WebAPI Class, and retrieving the results of queries from the database after the logic layer decides that the results are safe to be returned to the user using the SQLWrapperAPI Class.

The logic layer uses the DecisionEngine Class which is responsible for validating whether the queries follow the set database policies with the PolicyController Class, are legal based on the user role/security level using the ConfigController Class, and are safe to return based on the previous user queries stored in the log using the LogController Class.

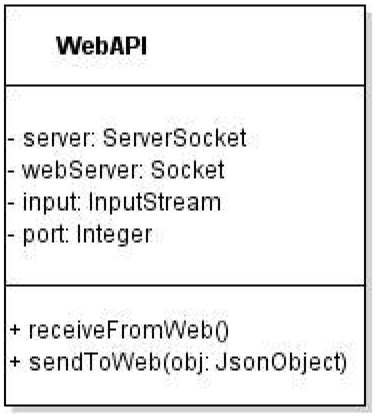
The data layer is responsible for storing the log files, configuration file, and policy rules, for access by the logic layer.

3 Design Viewpoints

3.1 Interface Layer

3.1.1 WebAPI Class

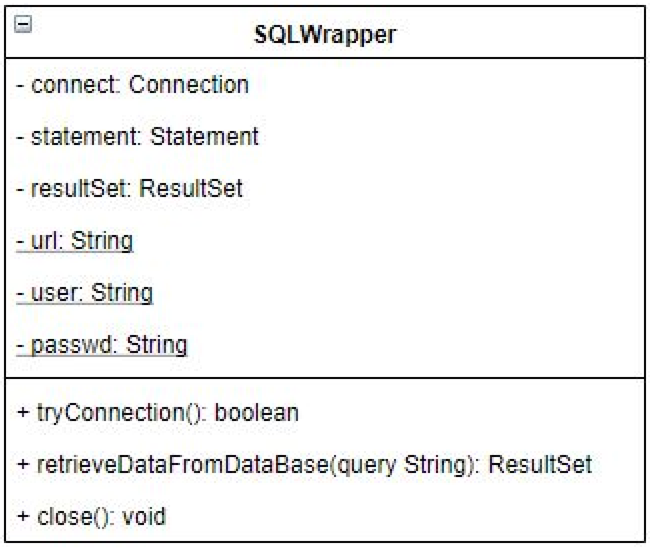
The WebAPI class provides an API to the web server to send and receive requests from the GUI.



Name	Definition
server	Java socket for receiving/sending
webServer	Socket of web server to connect
input	Input received from web server
port	The socket port used
receiveFromWeb()	Receives input from web server and forwards the object to the coordinator
sendToWeb()	Sends the results to the web server

3.1.2 SQLWrapperAPI Class

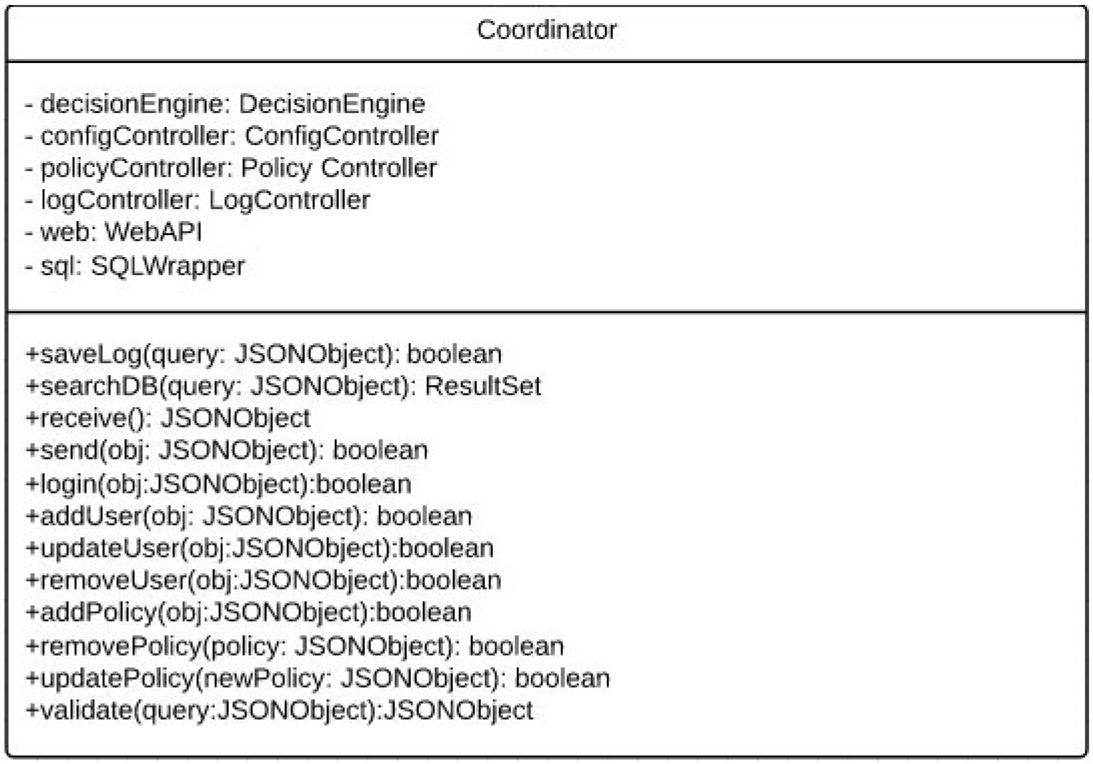
The SQLWrapperAPI class sends the user query to the database and returns the data retrieved.



Name	Definition
connect	The connection(session) with specified database
statement	The actual Object used for executing SQL statement and returning
resultSet	A table of data representing a database result set, which is usually generated by executing a statement that queries the database.
url	Address for MySQL database connection
user	Username for MySQL database connection
passwd	Password for MySQL database connection
tryConnection()	Try to make connection to SQL, return true if success
retrieveDataFromDatabase	Return the set of query data from the MySQL database
close()	Close all objects and sessions used for connection

3.2 Logic Layer

3.2.1 Coordinator Class


Coordinator

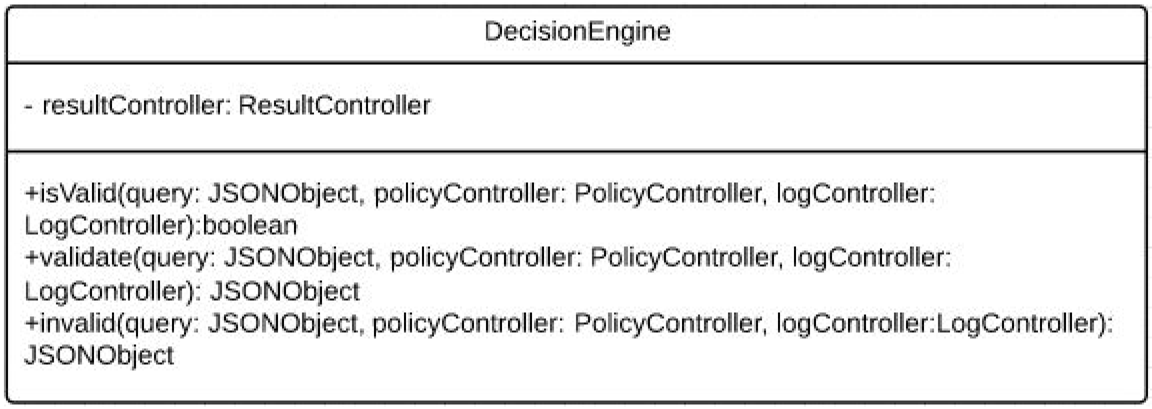
- decisionEngine: DecisionEngine
- configController: ConfigController
- policyController: Policy Controller
- logController: LogController
- web: WebAPI
- sql: SQLWrapper

```
+saveLog(query: JSONObject): boolean
+searchDB(query: JSONObject): ResultSet
+receive(): JSONObject
+send(obj: JSONObject): boolean
+login(obj:JSONObject):boolean
+addUser(obj: JSONObject): boolean
+updateUser(obj:JSONObject):boolean
+removeUser(obj:JSONObject):boolean
+addPolicy(obj:JSONObject):boolean
+removePolicy(policy: JSONObject): boolean
+updatePolicy(newPolicy: JSONObject): boolean
+validate(query:JSONObject):JSONObject
```

Name	Definition
decisionEngine	Controller that will validate whether a user's query breaks any security rules and assists in returning a valid result
configController	Controller to add/modify/remove user accounts
policyController	Controller to add/modify/remove policy rules as well as read the contents of the policy rules
logController	Controller to save and read user's logs
WebAPI	Interface to send and receive messages with web page

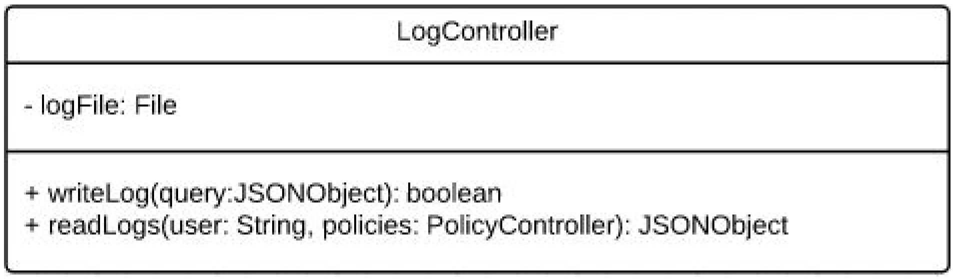
SQLWrapper	Wrapper used to search database
saveLog	Saves the query under a user's log with the help of the LogController, returns true if query was saved successfully
searchDB	Searches DB with given query and SQLWrapper, returns true if search was made successfully
receive	Receives JSONObject from WebAPI
send	Sends JSONObject through WebAPI
login	Verifies user's credentials using ConfigController and returns true if successful
addUser	Add's user to config file using ConfigController and returns true if successful
updateUser	Updates a user account's state using ConfigController and returns true if successful
removeUser	Removes a user account using ConfigController and returns true if successful
addPolicy	Adds a policy rules using PolicyController and returns true if successful
updatePolicy	Updates a policy rule using PolicyController and returns true if successful
removePolicy	Removes a policy rule using PolicyController and returns true if successful
validate	Validates a user's query using the Decision Engine and returns a valid result depending on the policy rules and user's logs given by the PolicyController and LogController

3.2.2 DecisionEngine Class


DecisionEngine
- resultController: ResultController
+isValid(query: JSONObject, policyController: PolicyController, logController: LogController):boolean
+validate(query: JSONObject, policyController: PolicyController, logController: LogController): JSONObject
+invalid(query: JSONObject, policyController: PolicyController, logController:LogController): JSONObject

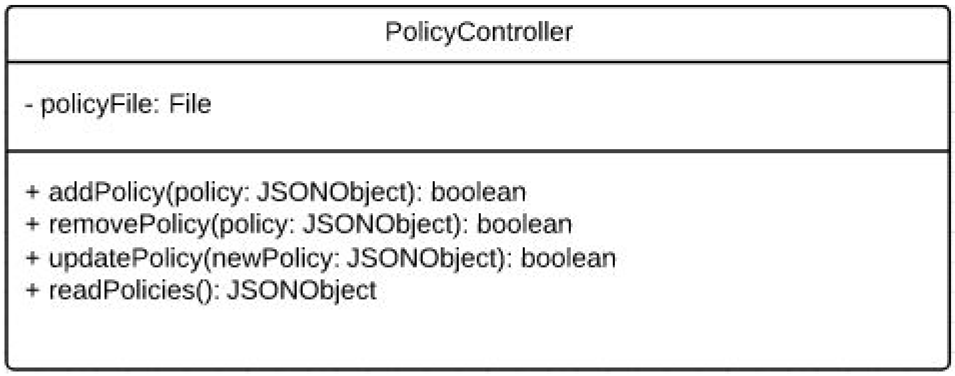
Name	**Definition**
resultController	Controller that will modify the result depending on the user's logs and policy rules
isValid	Validates whether the user's query log breaks the policy rules set by the system, returns true if valid and false otherwise
validate	Returns the original result of the user's query if query was found to be valid and returns a modified result if query was found to be invalid
invalid	Returns modified result of the user's query with the help of the resultController

3.2.3 LogController Class


LogController
- logFile: File
+ writeLog(query:JSONObject): boolean
+ readLogs(user: String, policies: PolicyController): JSONObject

Name	Definition
logFile	Log file containing all users' logs stored as JSON Objects
writeLog	Save a user's query in the log file
readLogs	Return a user's logs related to given policies and given username

3.2.4 PolicyController Class

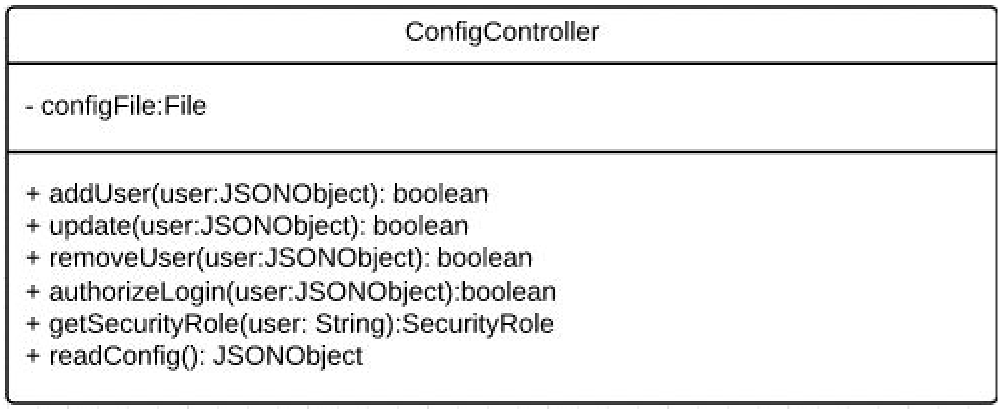

PolicyController

- policyFile: File

+ addPolicy(policy: JSONObject): boolean
+ removePolicy(policy: JSONObject): boolean
+ updatePolicy(newPolicy: JSONObject): boolean
+ readPolicies(): JSONObject

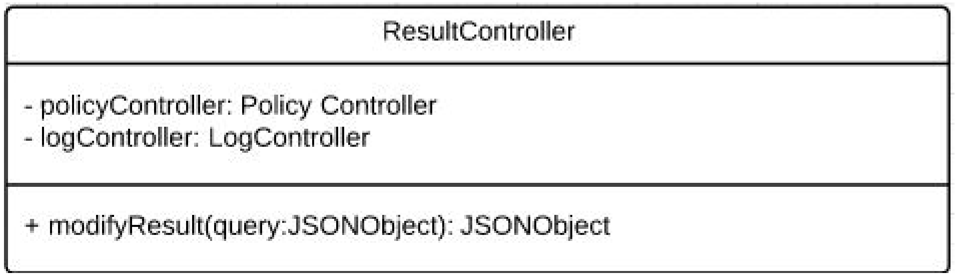
Name	Definition
policyFile	Policy file containing all policy rules stored as JSON Objects
addPolicy	Add policy rule to policy file, return true if the write is successful
removePolicy	Remove a policy rule from policy file, return true if removal was successful
updatePolicy	Update a policy rule from policy file, return true if the update was successful
readPolicies	Returns contents of Policy File as JSONObject

3.2.5 ConfigController Class


ConfigController
- configFile:File
+ addUser(user:JSONObject): boolean
+ update(user:JSONObject): boolean
+ removeUser(user:JSONObject): boolean
+ authorizeLogin(user:JSONObject):boolean
+ getSecurityRole(user: String):SecurityRole
+ readConfig(): JSONObject

Name	**Definition**
configFile	File where configuration data is stored as XML
addUser	Add user account to config file, return true if successful
update	Update a user account from config file, return true if the update was successful
removeUser	Remove the user account specified from config file, returns true if successful
authorizeLogin	Verifies whether user credentials passed as JSONObject are stored in configFile, returns true if successful
readConfig	Returns the contents of the config file as a JSON object
getSecurityRole	Returns a user’s security role given specified username

3.2.6 ResultController Class


ResultController
- policyController: Policy Controller
- logController: LogController
+ modifyResult(query:JSONObject): JSONObject

Name	**Definition**
policyController	Controller used to read the contents of the policy rules
logController	Controller used to read user's logs
modifyResult	Modifies the result of the user's query depending on the policy rule that is broken from the user's query log

3.3 Data Layer

3.3.1 Config file

The config file is an XML file that holds the settings from the Web UI. This includes the user accounts (username and password) and user roles.

3.3.2 Policy Rules

The policy file stores the administrator configured policy rules. The policy rules consists of the database column headers that when queried sequentially may lead to an inference attack. The policy file saves these rules as a text file and identifies the target column headers that the administrator is trying to protect using key words such as MODIFY.

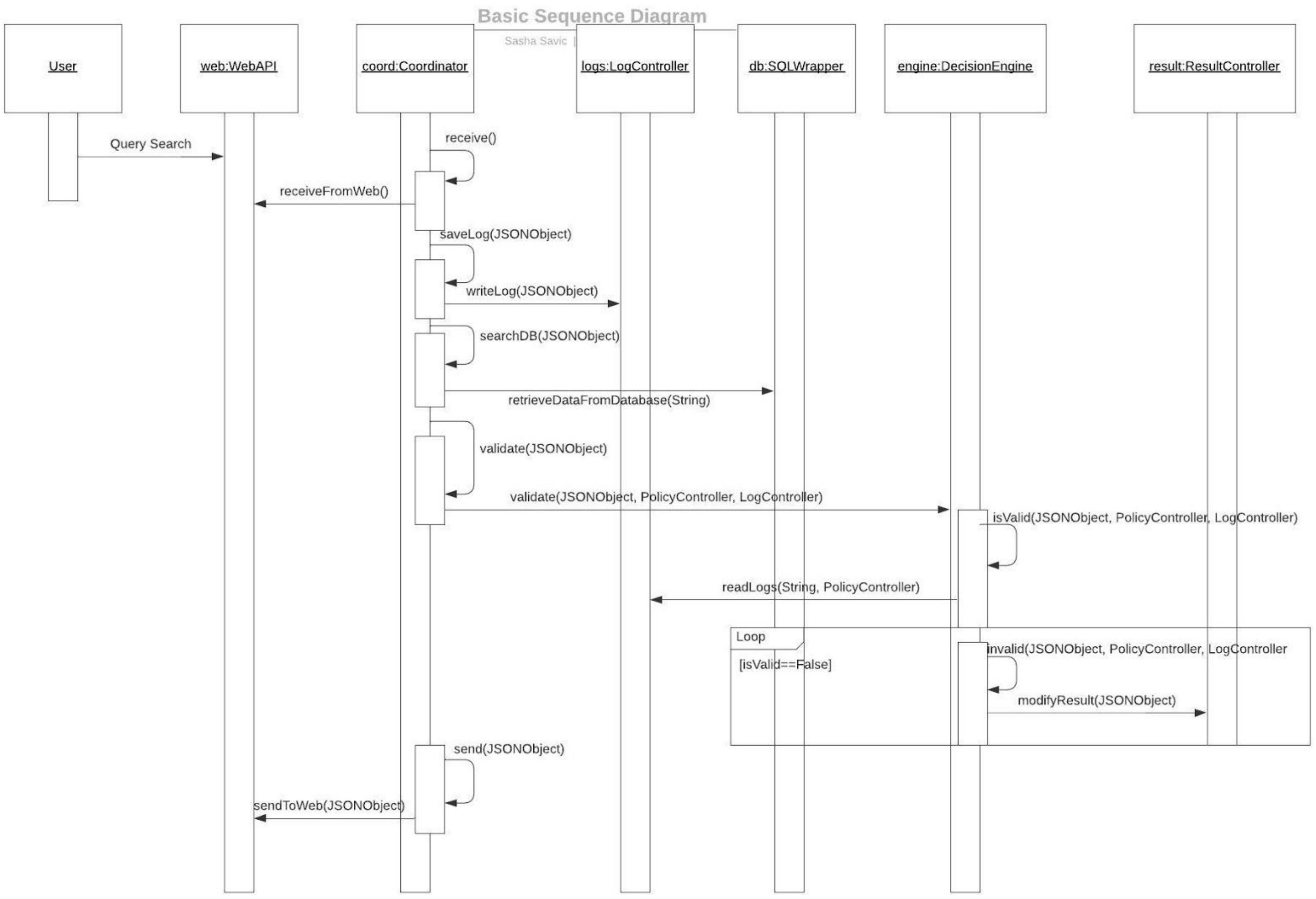
3.3.3 Logs

A log file will be created for each user account. The log file records the user search query history. The following information will be recorded:

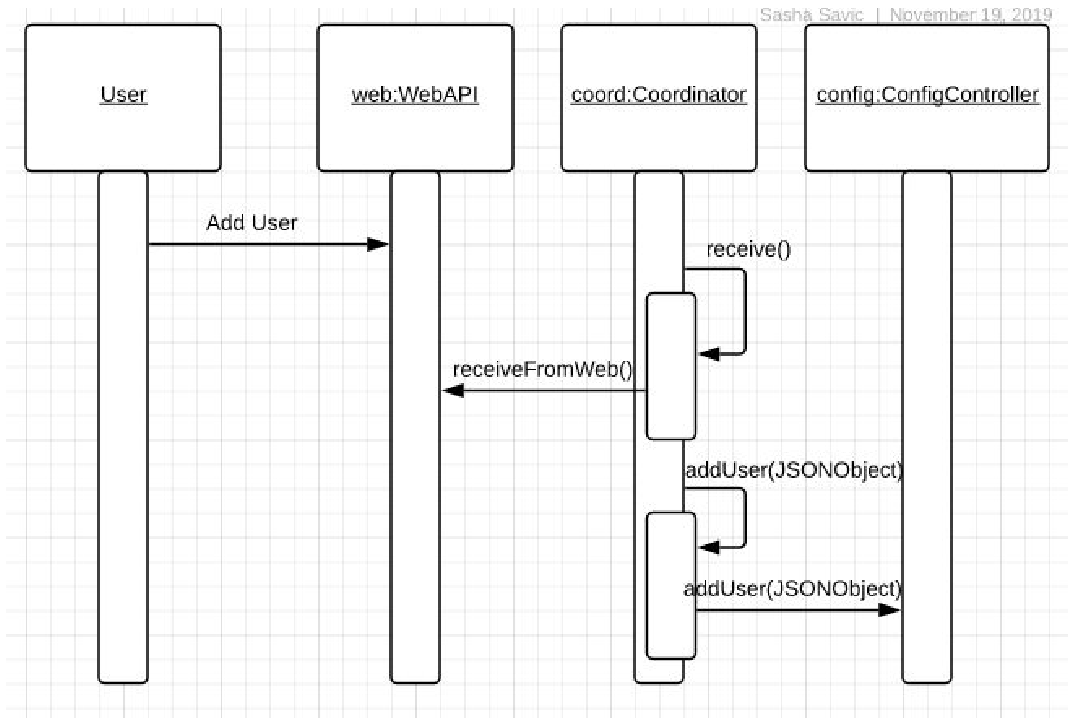
- User Account
- Timestamp
- Query Columns

3.4 Interaction Viewpoints

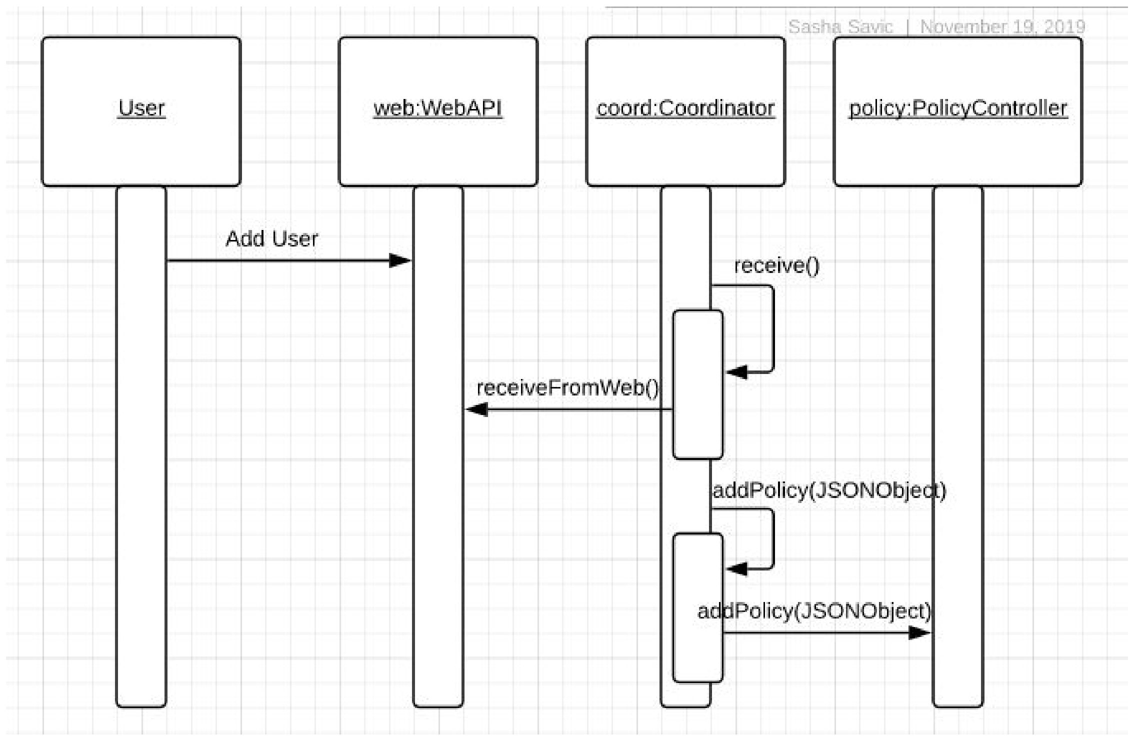
3.4.1 Query Search



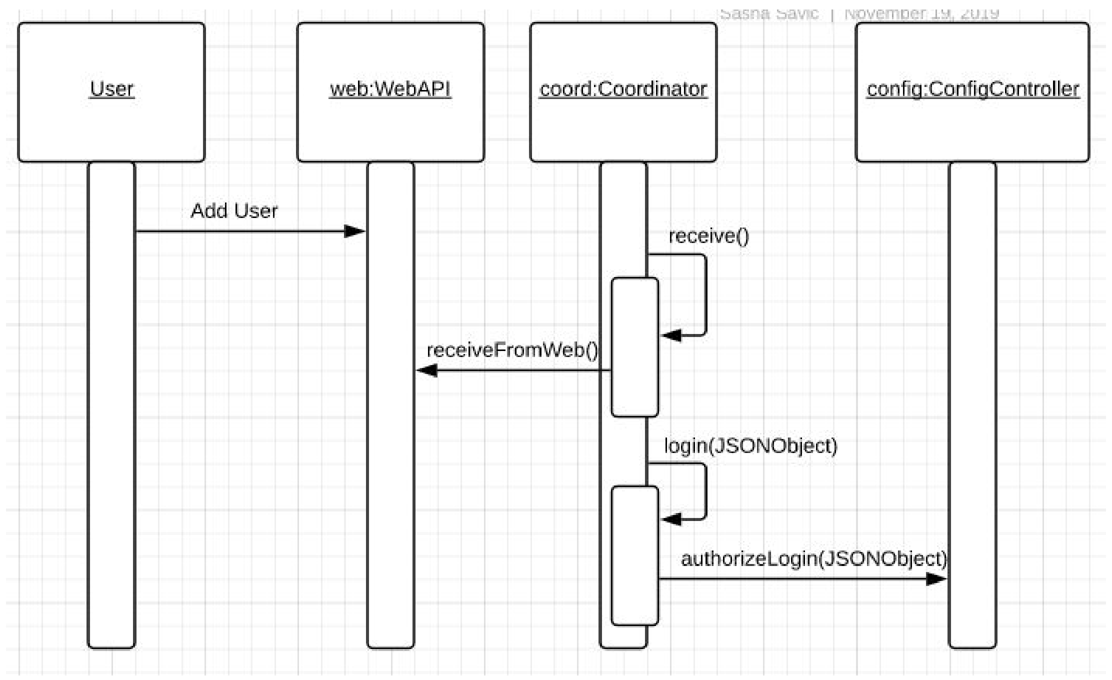
3.4.2 Add/Modify User Account



3.4.3 Add/Modify Policy



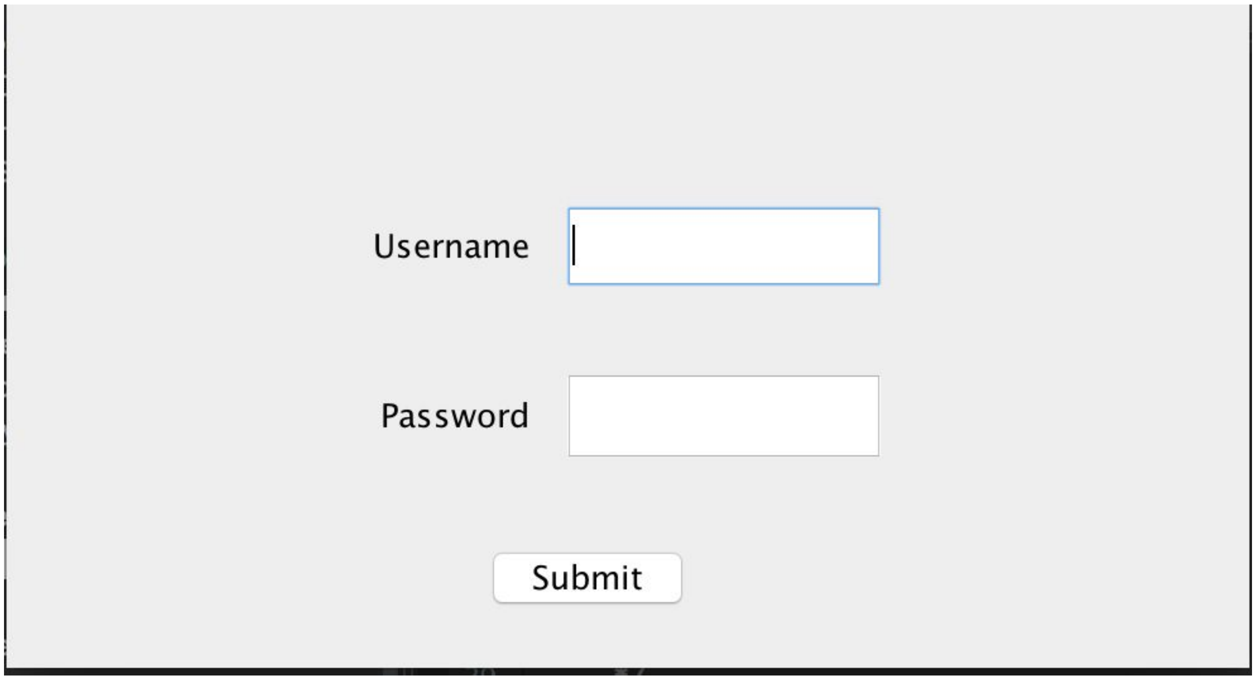
3.4.4 User Login



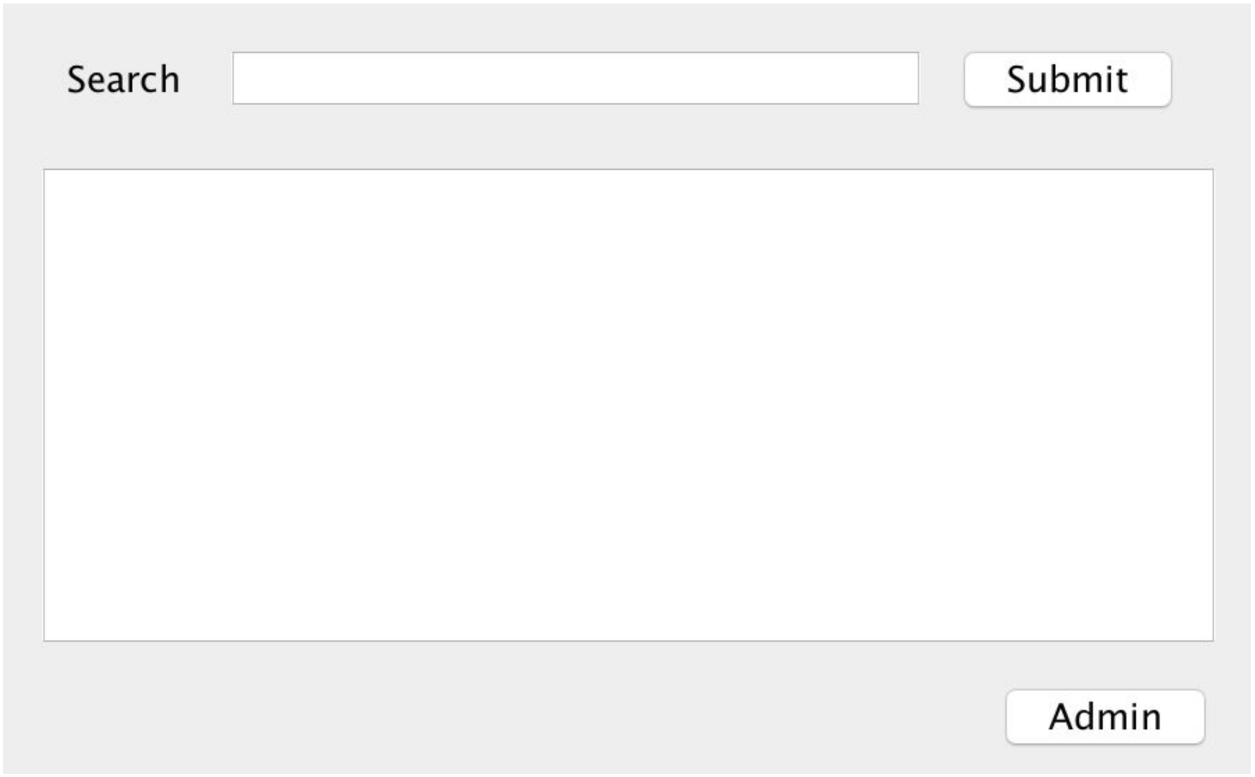
3.5 Demo User Interface

3.5.1 Graphical User Interface Mockup

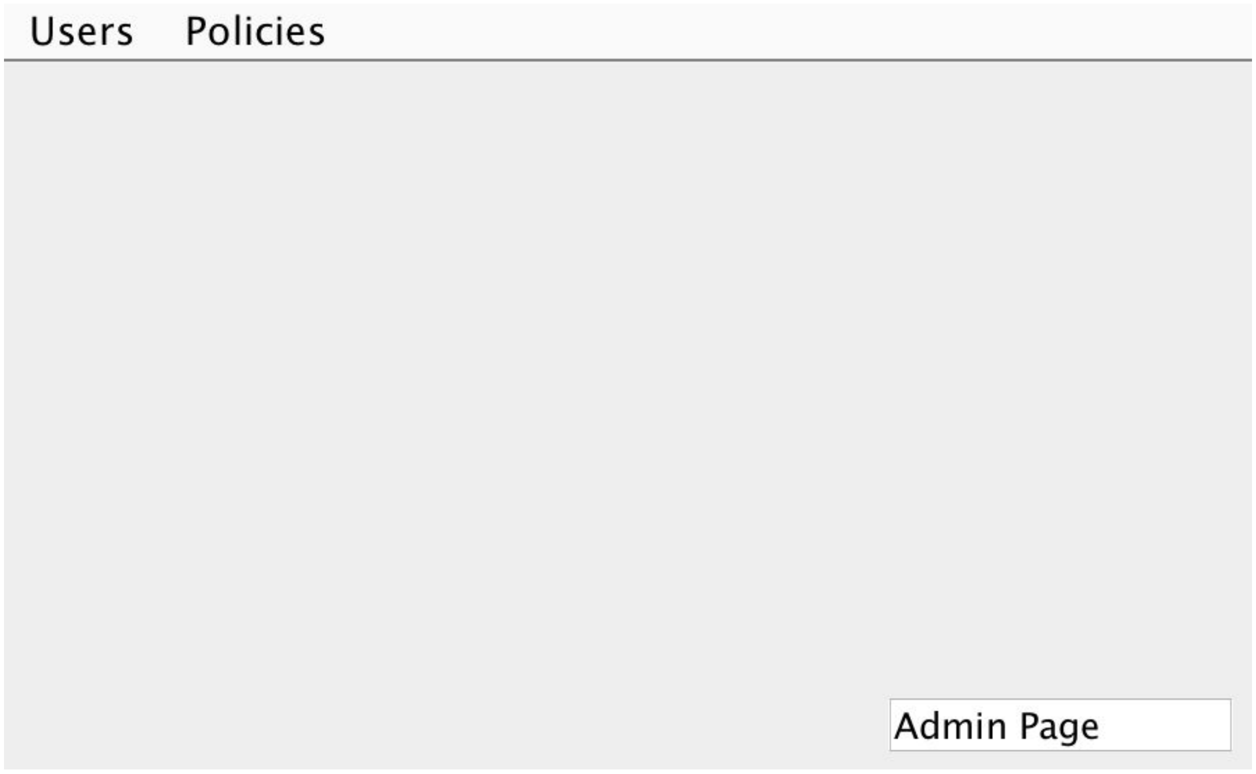
All Mockup Demos were created in a Java GUI application for anticipated future use.



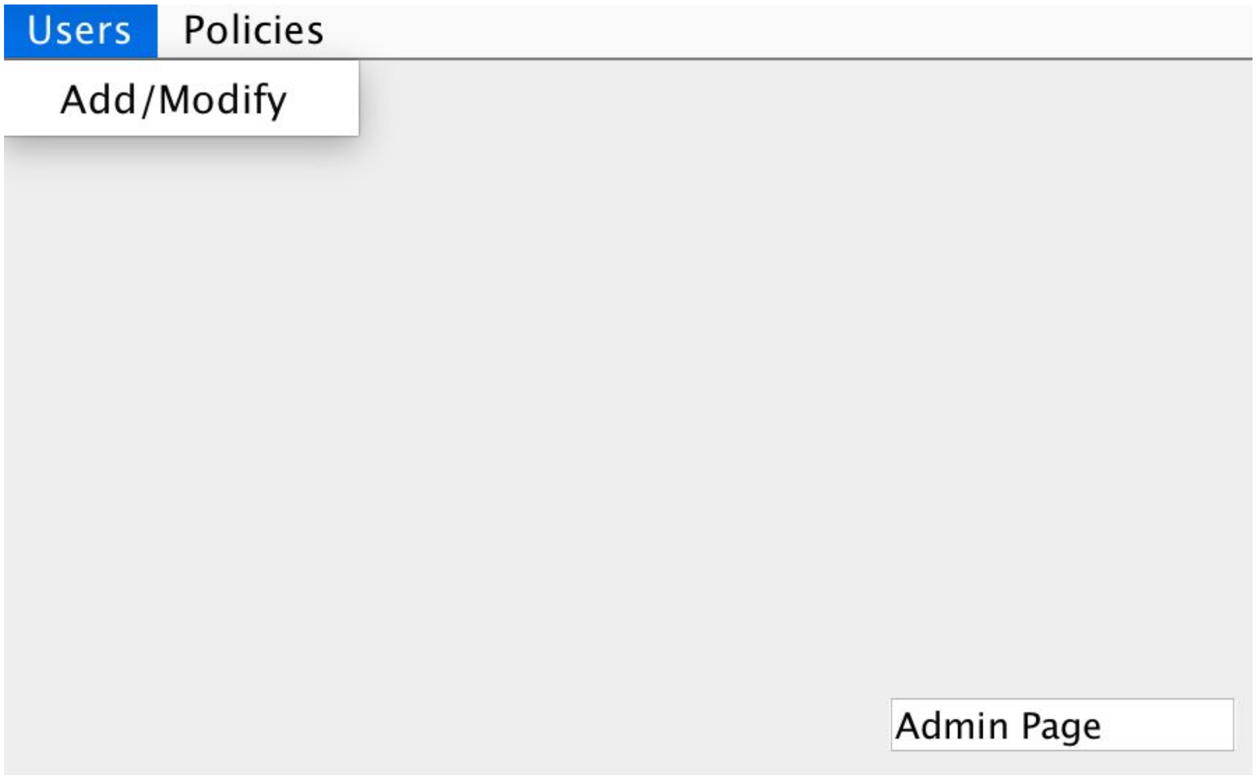
All users will arrive at the login page when first connecting to the system. The Login Page requires a username and password to get into the system.



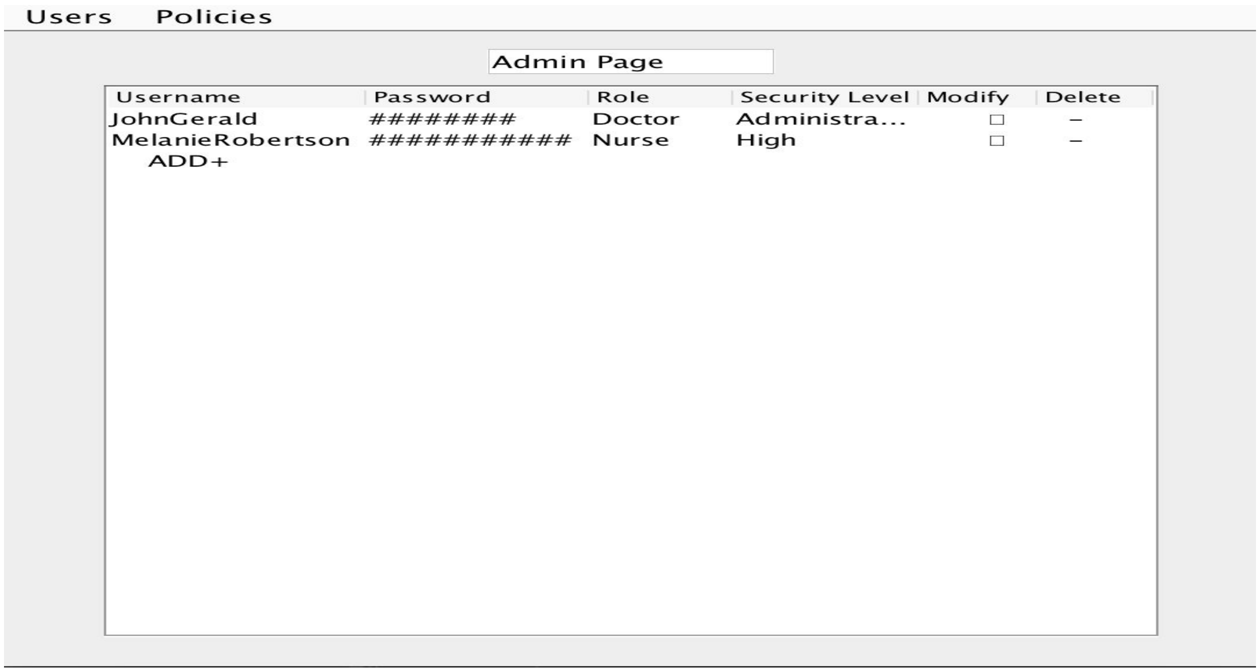
The next page is for the user to search the information that the user needs to acquire, with an admin button for the administrators to use.



This is the Admin page.

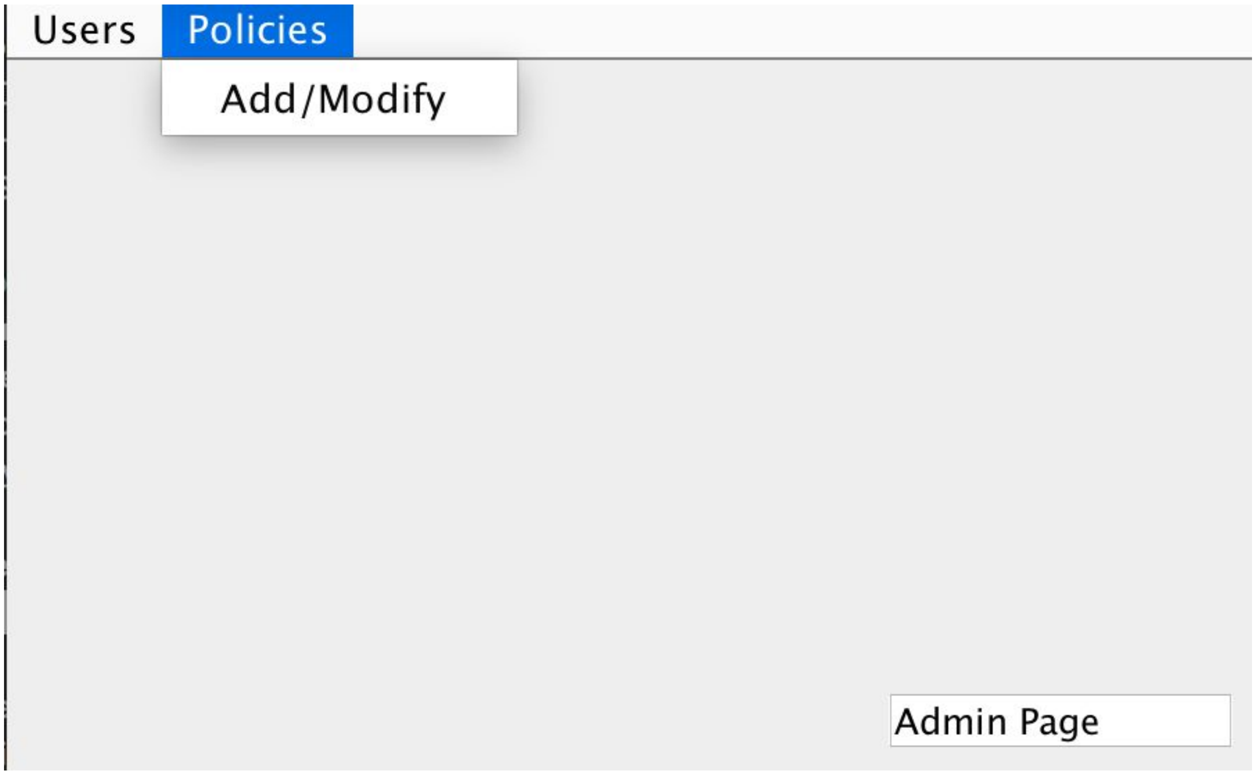


The Administrator can add or modify user accounts.

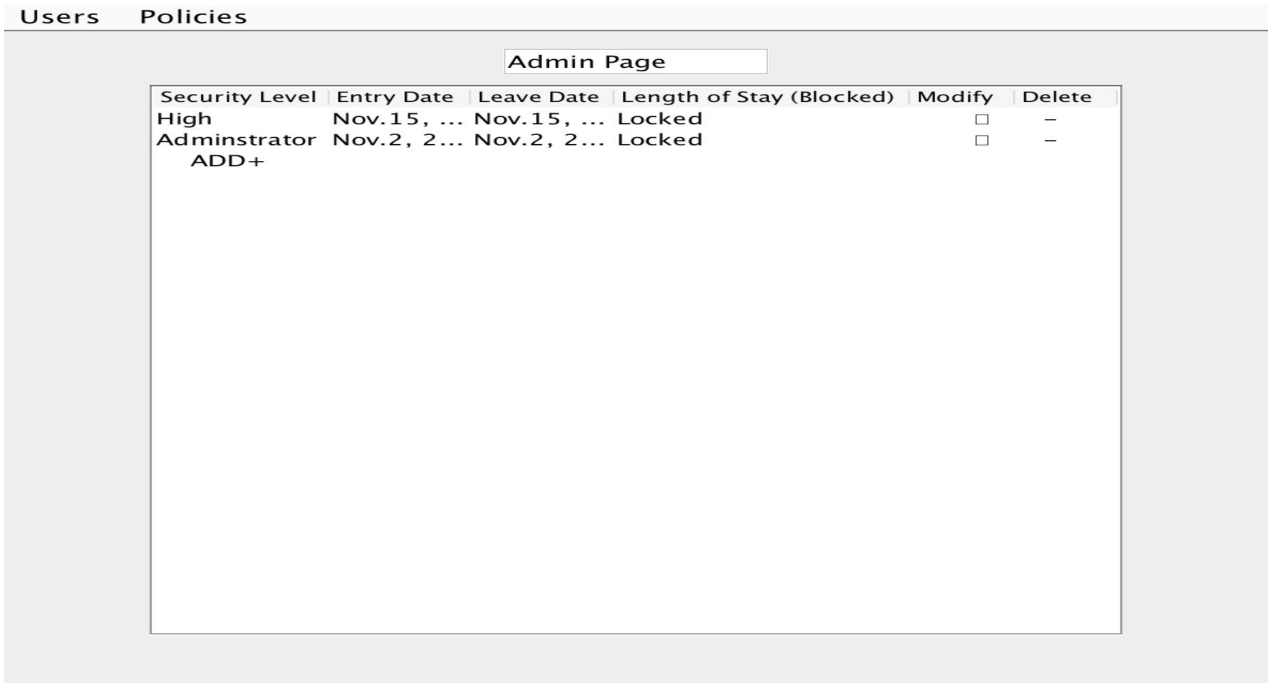


Username	Password	Role	Security Level	Modify	Delete
JohnGerald	########	Doctor	Administra...	□	–
MelanieRobertson	###########	Nurse	High	□	–
ADD+					

This is a sample example of information that the user account has and the administrator can either add, modify, or delete the user information that is already in the database.



The administrator can add/modify policies.



Security Level	Entry Date	Leave Date	Length of Stay (Blocked)	Modify	Delete
High	Nov.15, ...	Nov.15, ...	Locked	□	–
Adminstrator	Nov.2, 2...	Nov.2, 2...	Locked	□	–
ADD+					

This is a sample example of information that the Policies has and the administrator can either add, modify, or delete the Policy information that the system already has. You can extend the date columns to see the whole dates with the month, day, year, hours, minutes, and seconds stated.

Contributions

Calvin

- 3.1.1, 3.3.1- 3.3.3

Ryan

- 3.1.2
- SQLWrapper code implementation

Aleksandar

- 3.2.1-3.2.6, 3.4.1-3.4.4

Hasan

- 1.1 - 1.2
- 2
- 3.2.2

Tashfiq

- 3.5

Calvin Soong Calvin Soong

Ryan Zheng RyanZheng

Aleksandar Savic A.Savic

Hasan Issa [illegible]

Tashfiq Akhand [illegible]