

**GO IN ACTION II (RUN BATCH # 4)**

**ASSIGNMENT SUBMISSION**

**DUE DATE: 9th MAY 2021 (SUNDAY) – 2359HRS**

|  |  |  |
| --- | --- | --- |
| **SUBMITTED BY** | **:** | **AMANDA SOH CHIEW PHENG** |
| **TRAINER** | **:** | **MS LEE CHING YUN** |

**Content Page**

[INTRODUCTION 4](#_Toc71492258)

[GO SECURITY 4](#_Toc71492259)

[INPUT VALIDATION 4](#_Toc71492260)

[FILE MANIPULATION 14](#_Toc71492261)

[POST-VALIDATION ACTIONS 15](#_Toc71492262)

[SANITIZATION 15](#_Toc71492263)

[CONTEXT AWARENESS 15](#_Toc71492264)

[CROSS SITE SCRIPTING 16](#_Toc71492265)

[HTTP/TLS 18](#_Toc71492266)

[ERROR HANDLING & LOGGING 21](#_Toc71492267)

[LOGGING 23](#_Toc71492268)

[SESSION MANAGEMENT 24](#_Toc71492269)

[AUTHENTICATION & PASSWORD MANAGEMENT 26](#_Toc71492270)

[COMMUNICATING AUTHENTICATION DATA 26](#_Toc71492271)

[VALIDATION AND STORING AUTHENTICATION DATA 28](#_Toc71492272)

[STORING PASSWORD SECURELY 28](#_Toc71492273)

[GO DOCUMENTATION 29](#_Toc71492274)

[IDIOMATIC GO 30](#_Toc71492275)

[FORMATTING 30](#_Toc71492276)

[COMMENTARY 30](#_Toc71492277)

[NAME 32](#_Toc71492278)

[SEMICOLONS 32](#_Toc71492279)

[CONTROL STRUCTURES 33](#_Toc71492280)

[FUNCTIONS 33](#_Toc71492281)

[DEFER 33](#_Toc71492282)

[BLANK IDENTIFIER 34](#_Toc71492283)

[UNUSED PACKAGE IMPORT 34](#_Toc71492284)

[REFERENCES 35](#_Toc71492285)

[APPENDIX 36](#_Toc71492286)

[INSTRUCTIONS ON HOW TO RUN THE APPLICATION 36](#_Toc71492287)

[SAMPLE WALK-THROUGH ON THE APPLICATION 38](#_Toc71492288)

[NON-ADMIN USER 38](#_Toc71492289)

[ADMIN USER 43](#_Toc71492290)

# INTRODUCTION

For Go In Action I assignment, I have converted the assignment for Go Advanced from a cmd prompt based application into a client-server based web application. For this assignment, we will apply the key concepts learnt in Go In Action II, namely Go Software Security, Go Documentation, Idiomatic Go and enhanced on the assignment submitted for Go In Action I.

The application I have developed is a pizza ordering system. The application is created on the idea of a first-in-first-out (FIFO) concept whereby an order is processed on a first-come-first-serve queue basis i.e. the first order that comes into the system must be processed before the next one can be processed. Thus the code will make use of the Queue data structures to manage orders. To manage the pizzas, I have selected to use a pointer-based LinkedList data structure.

*See APPENDIX for a SAMPLE WALK-THROUGH OF THE APPLICATION.*

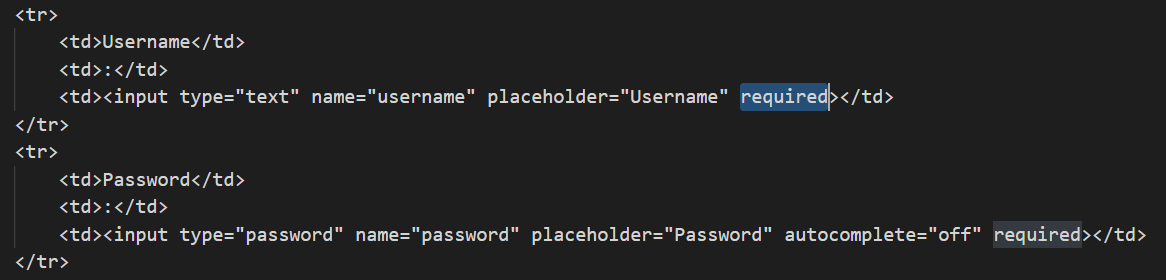
# **GO SECURITY**

The following sections highlight the Go security features that I have implemented in my application.

## **INPUT VALIDATION**

For assignment submission in Go In Action I, I have made use of the html input attribute “required” (as illustrated below) in the input fields such as textbox and password to ensure that user keys in the fields that are required for form submission. We will start with the login page.

Login (Also the index/home page)



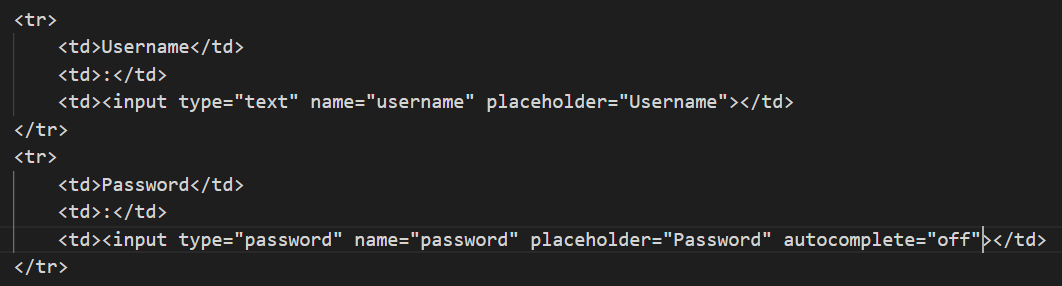
The output of using the “required” attribute is as such.



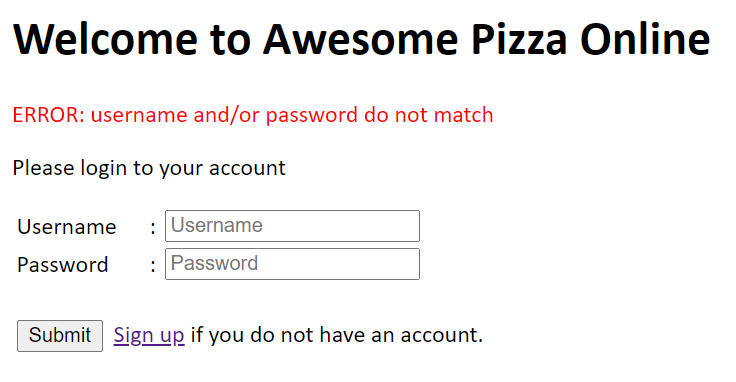
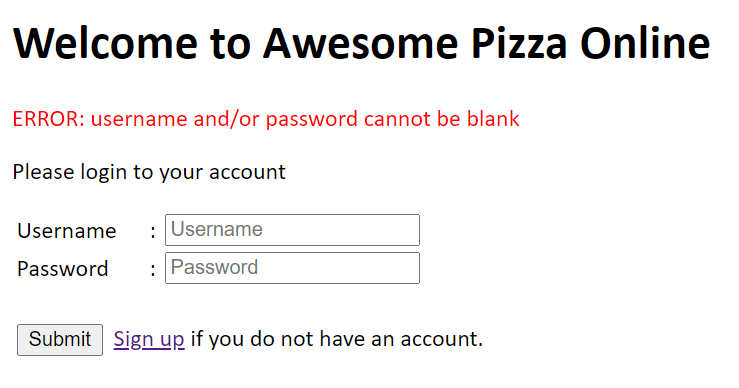
User is prompted to key in the “required” field

However, for this submission, in addition to the “required” attribute, I have also enhanced the validations into the codes. This is because we have to assume that all user inputs are unsafe and in case this feature is possibly broken, we will not be providing an opportunity for hacker to attack the application.

The “required” parameter is removed for testing on both username and password

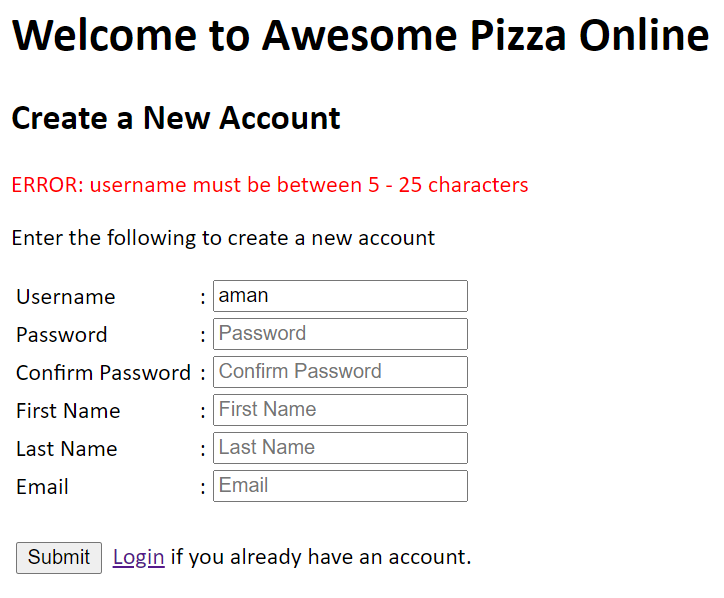
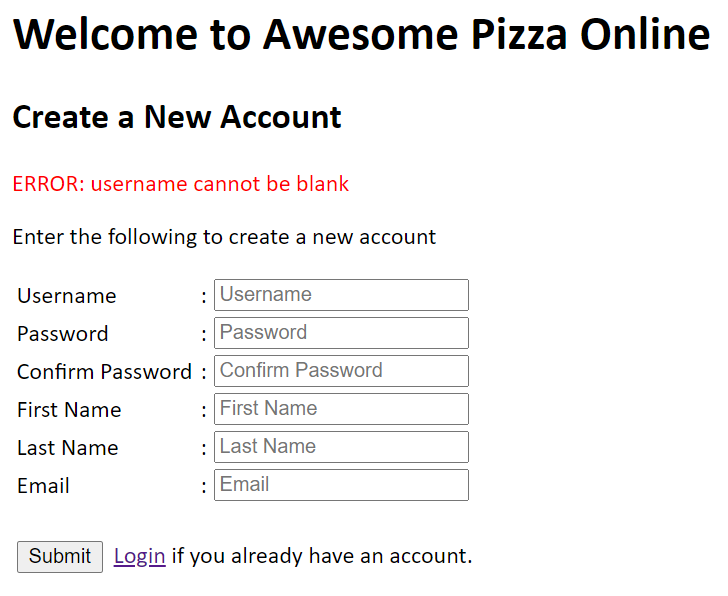


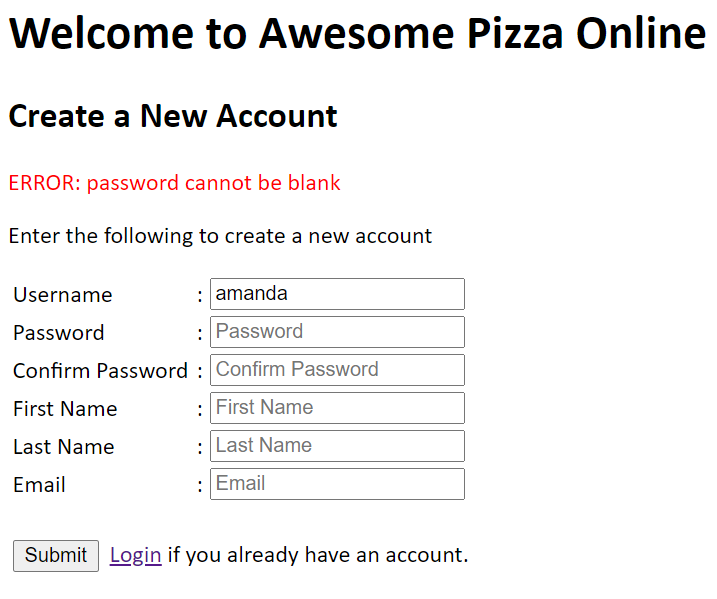
The following screenshots shown are the outputs when testing the various scenarios for invalid username/password in the login page.

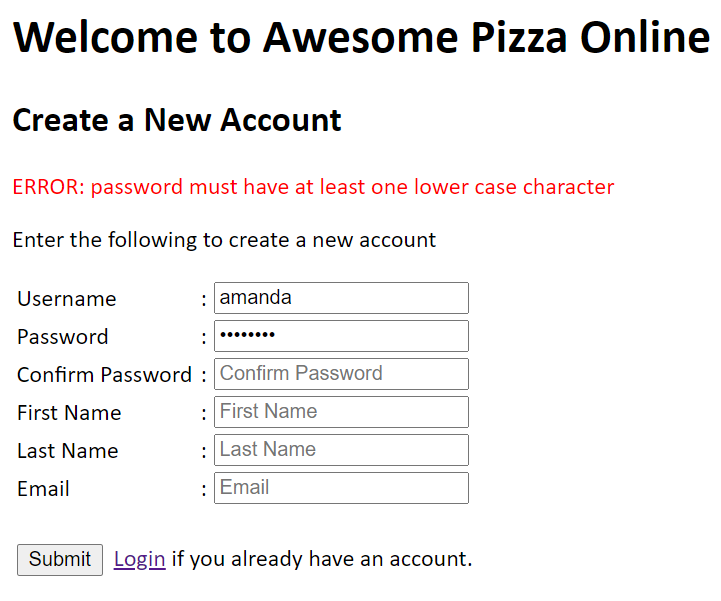
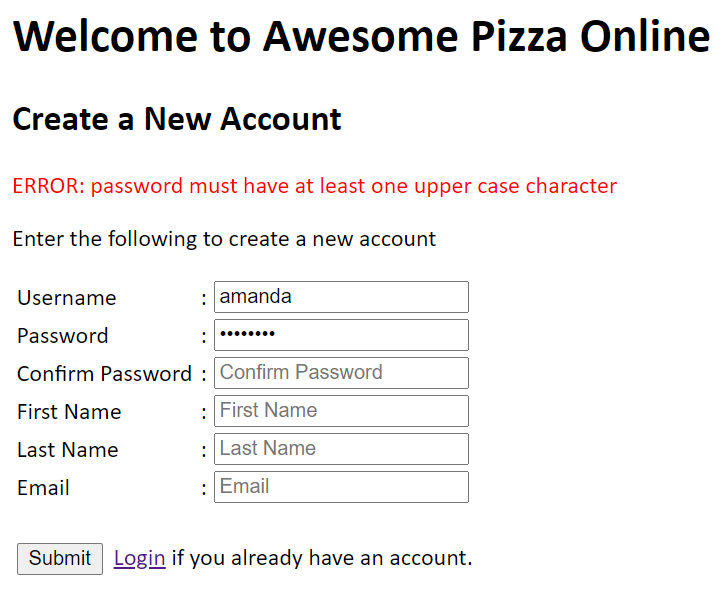


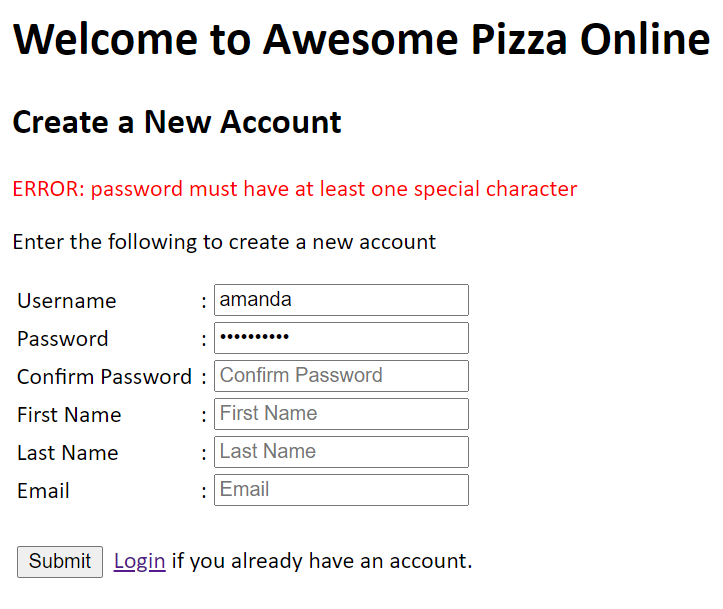
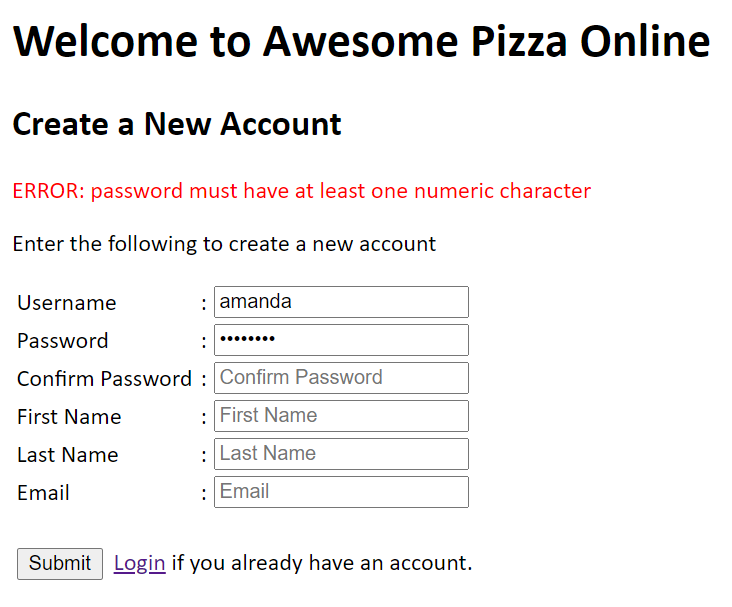
Signup

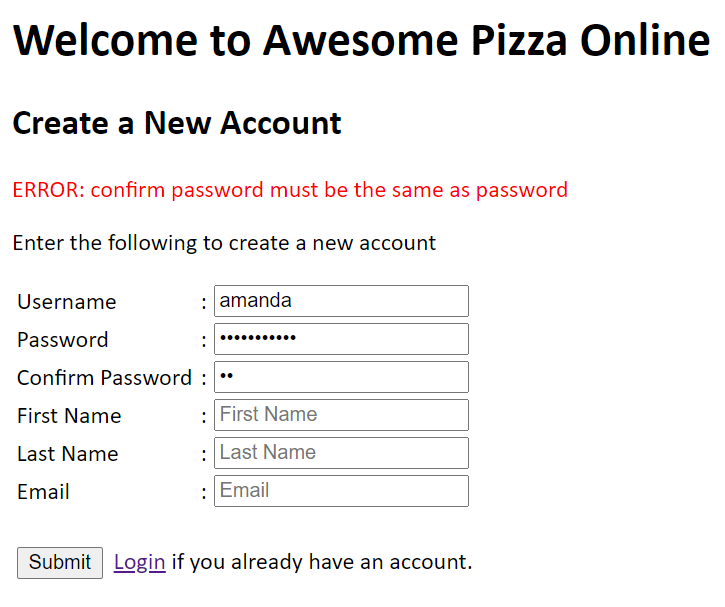
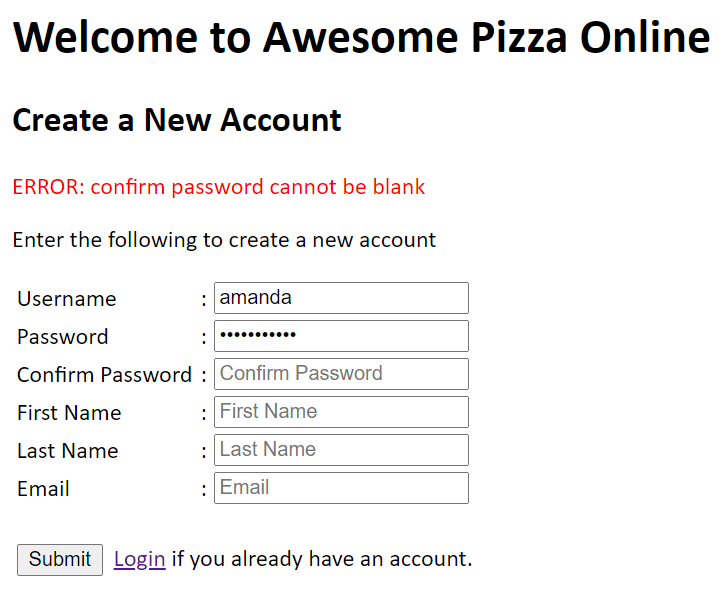
The following screenshots shown are the outputs when testing the various scenarios for invalid username information in the signup page. I will not be using the “required” attribute in the signup page. Note that I have set a min and max no. of characters for username (min 5, max 25) and password (min 8, max 25). This is to prevent bottlenecks when usernames and passwords get too long. Also it is hard for a user to remember an overly long username/password. To ensure that user can remember the password he/she keys in, the application will confirm the password again and both passwords should match for creation of user account to be successful. All other fields (i.e. First Name and Last Name) are also required. Email is not mandatory but when user input something, *regexp* is used to validate the email.

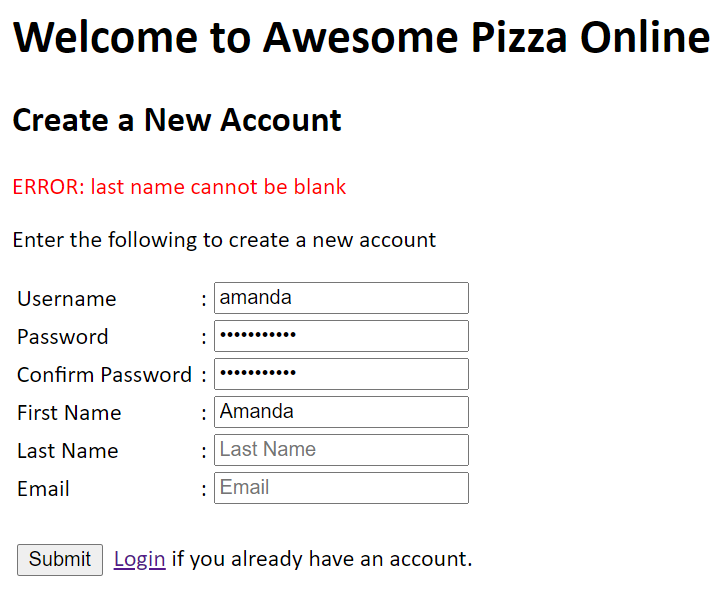
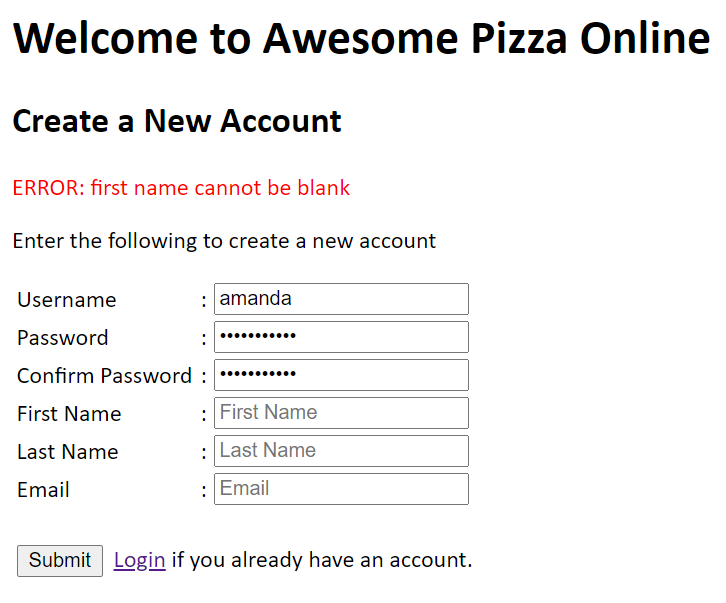




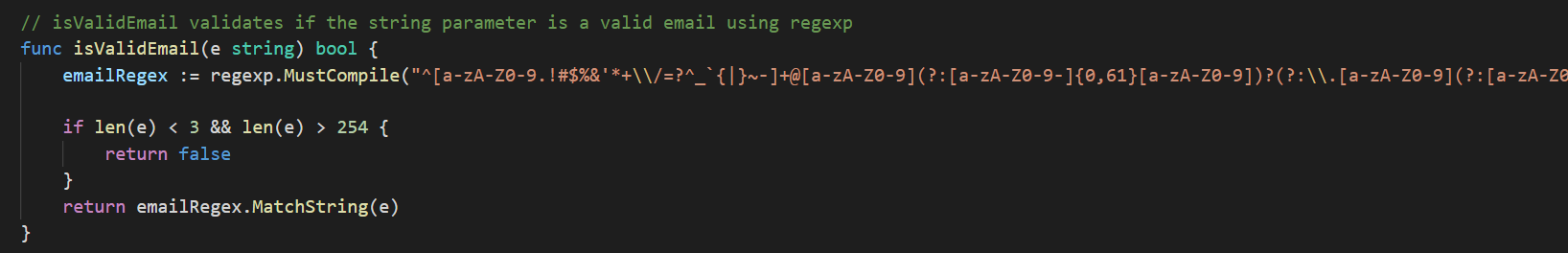


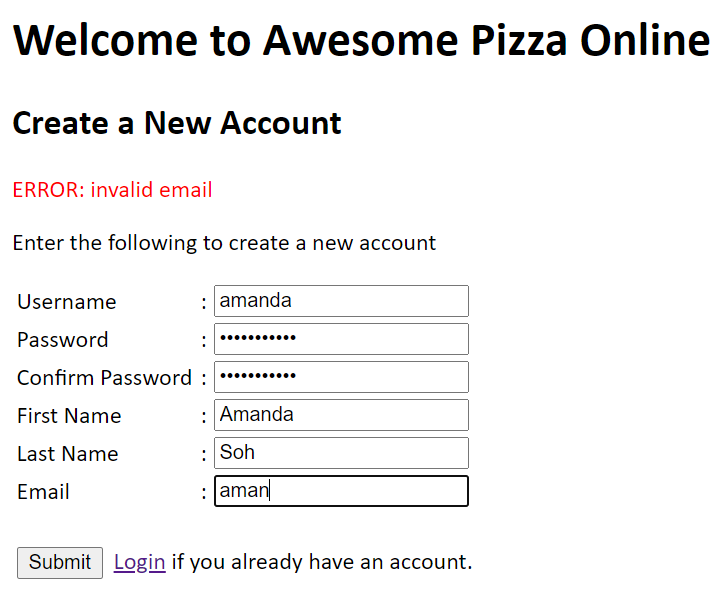




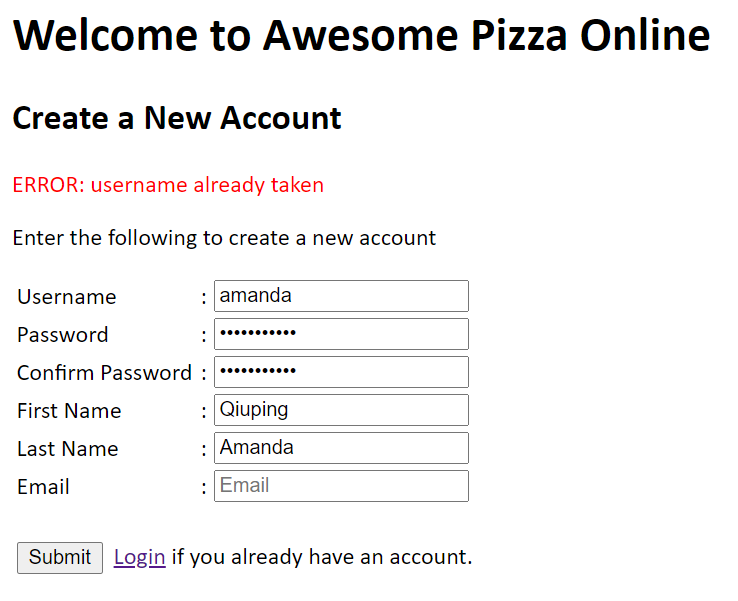


This is the function to validate email using regexp.



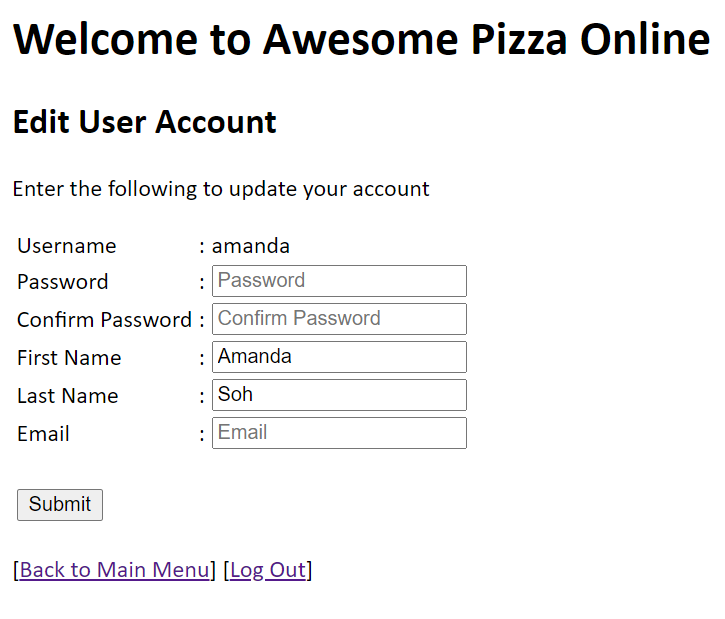


When a username has already been added before, user is not allowed to create a duplicate username.



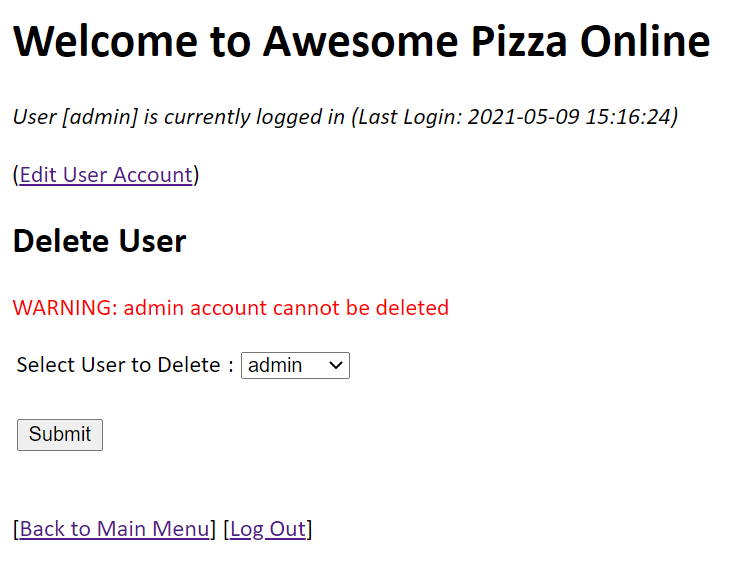
Edit User Account

Edit User Account validates the same information as Signup, except that username cannot be amended.



Delete User

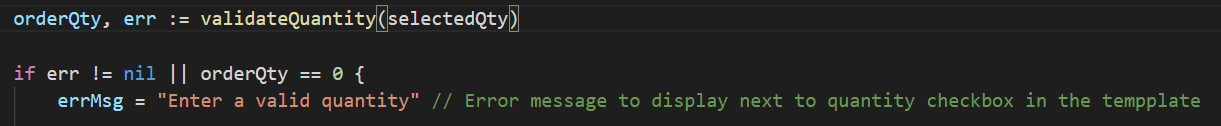
Only admin user can delete users but admin is not allowed to delete admin users.

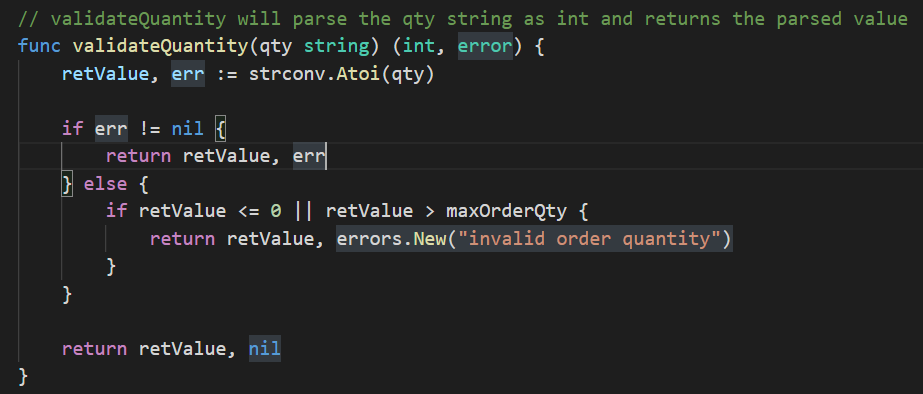


Add Order

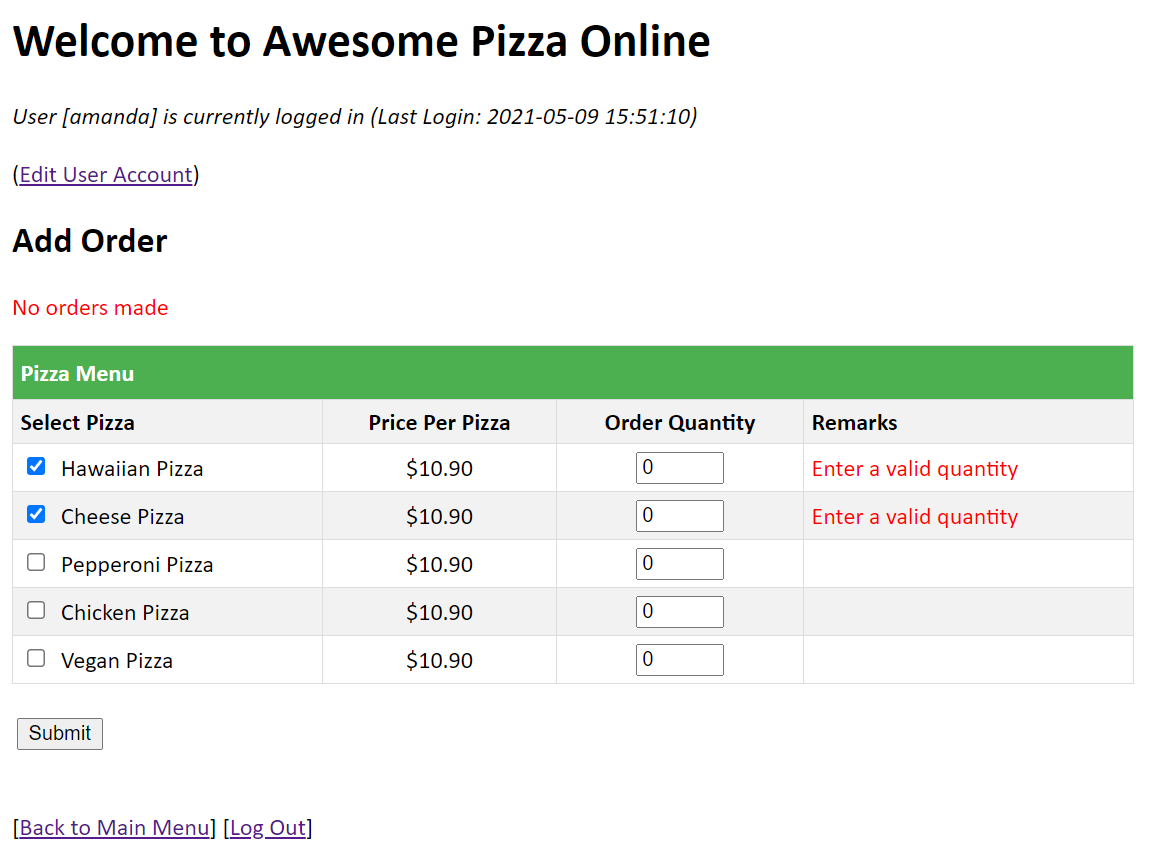
In the Add Order page, user selects the pizza to order by checking on the checkboxes next to the pizza names. Using checkboxes eliminates the need to validate the pizzas. In the same row of the selected pizza, user just has to key in a valid numeric quantity. Other validations incorporated for quantity includes that it is not empty and it has to be within 1 – 5. This is the limit that is set on the no. of pizzas a user can order for a selected pizza as it may be illogical for a user to order a large no. of pizzas. Error messages are displayed under the Remarks column.

The validations for quantity uses strconv.Atoi() to convert the quantity in string that is retrieved from the html input to an int with the following code. Validations is performed in validateQuantity() func and also checks that quantity is within the min and max allowed.



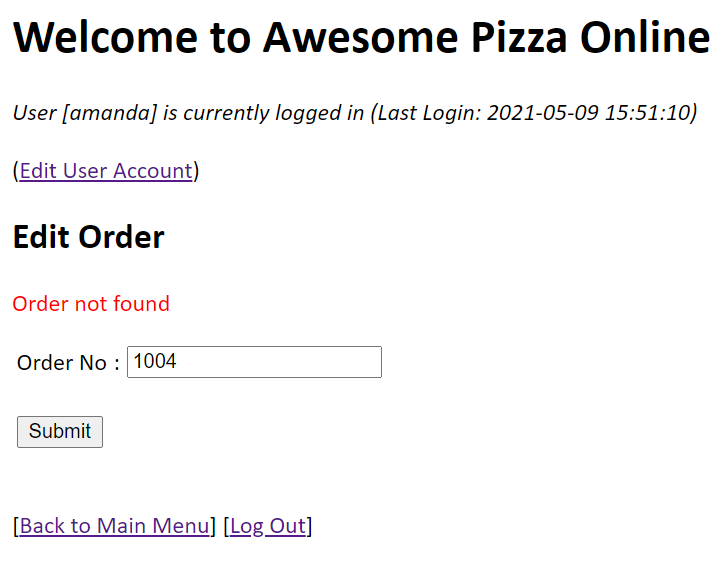
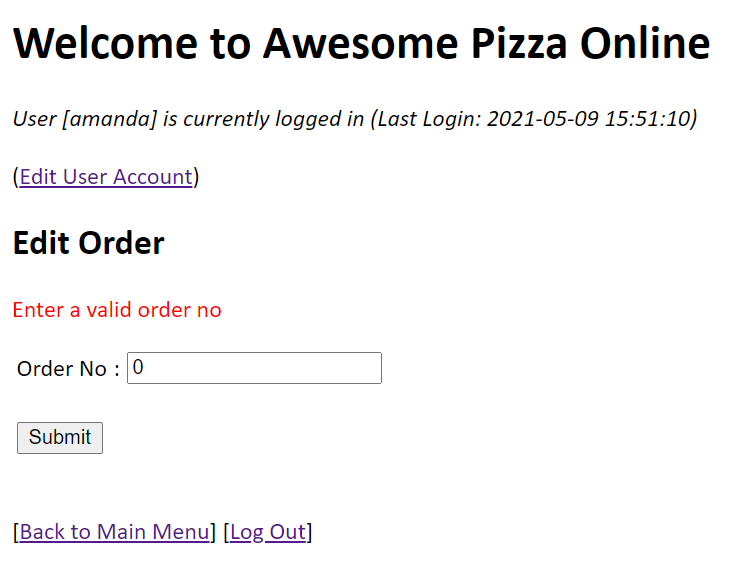


The output for an invalid quantity is as follows.

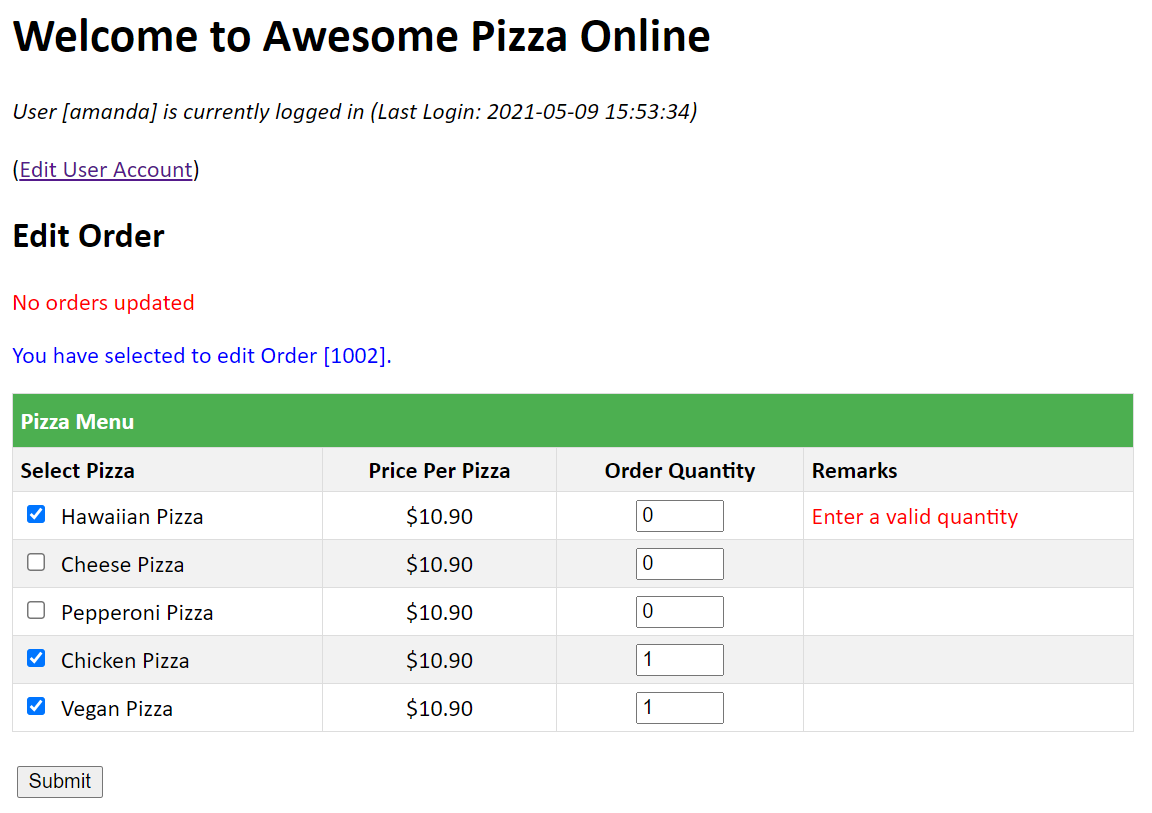


Edit Order

Edit Order requires user to enter a valid Order No. Otherwise it will prompt an error message.

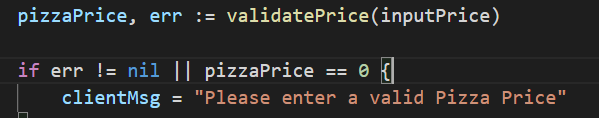


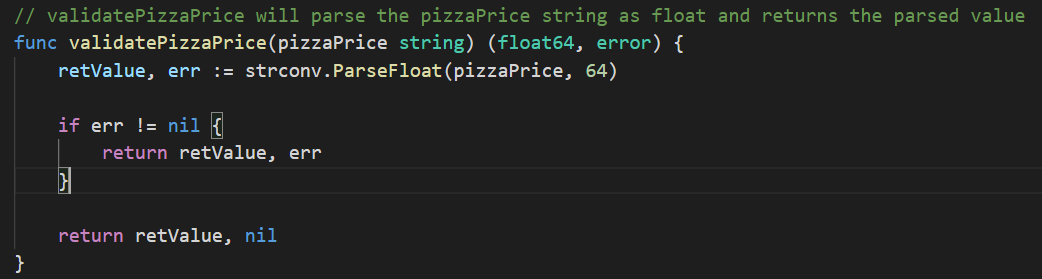
Once a valid Order No is keyed in, the page shown will be similar to Add Order and the pizzas made for the particular order will be checked and quantities displayed in the textbox. Validation for Order Quantity applies the same as Add Order.



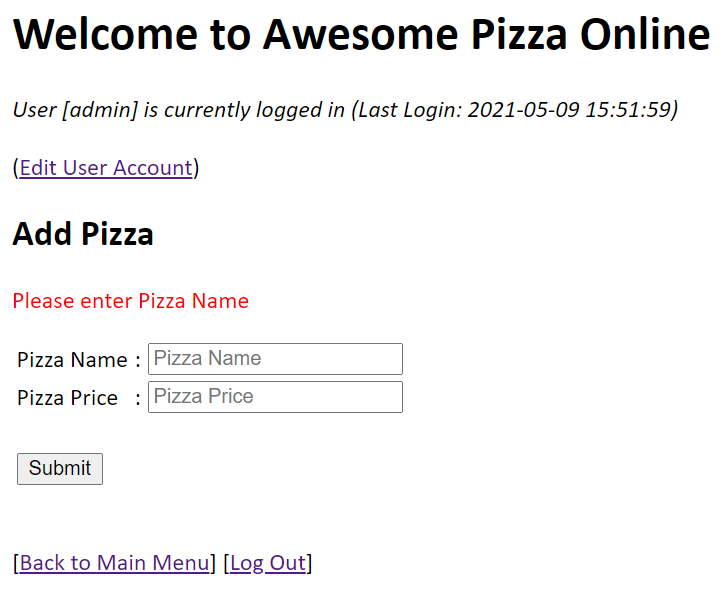
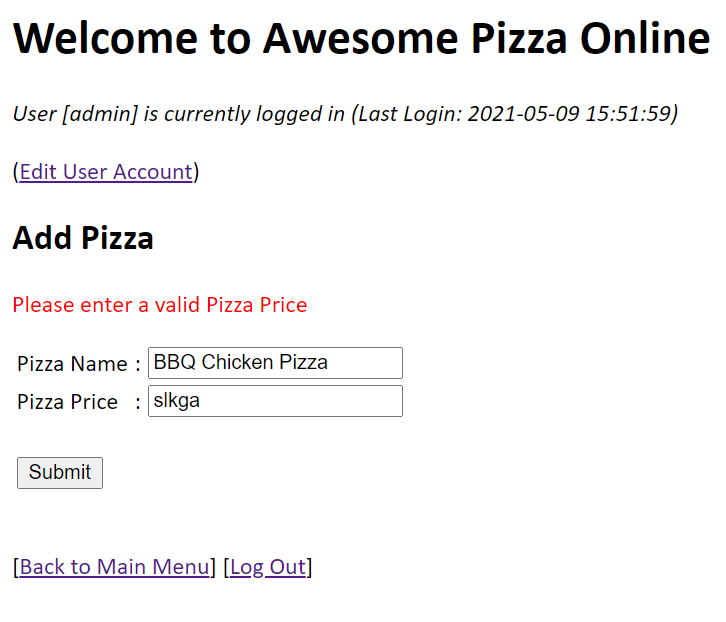
Add Pizza

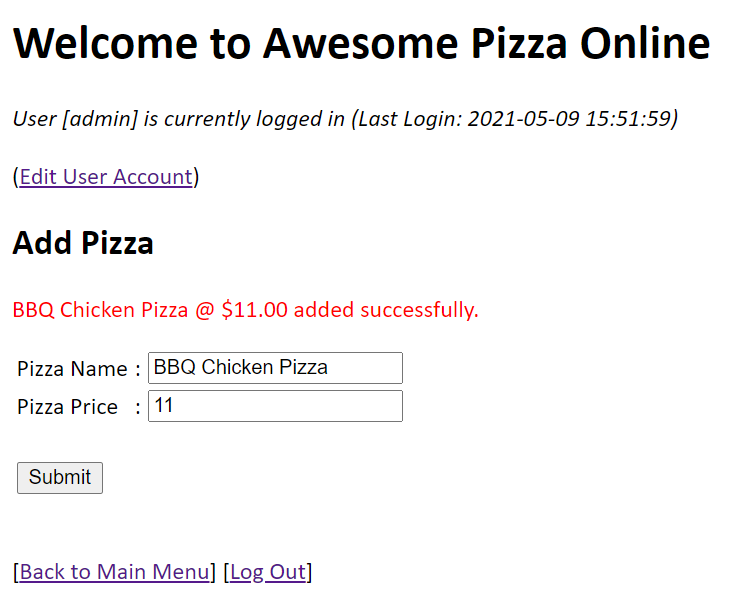
Add Pizza is only available to the admin user. It allows the admin to add a new pizza and its price. The validation code for price is performed in validatePrice() func and uses the strconv.ParseFloat to convert the price from a string to a float. Using func allows the validations to be expanded such as a providing a min and max price allowed.





The following outputs display the validations in place.

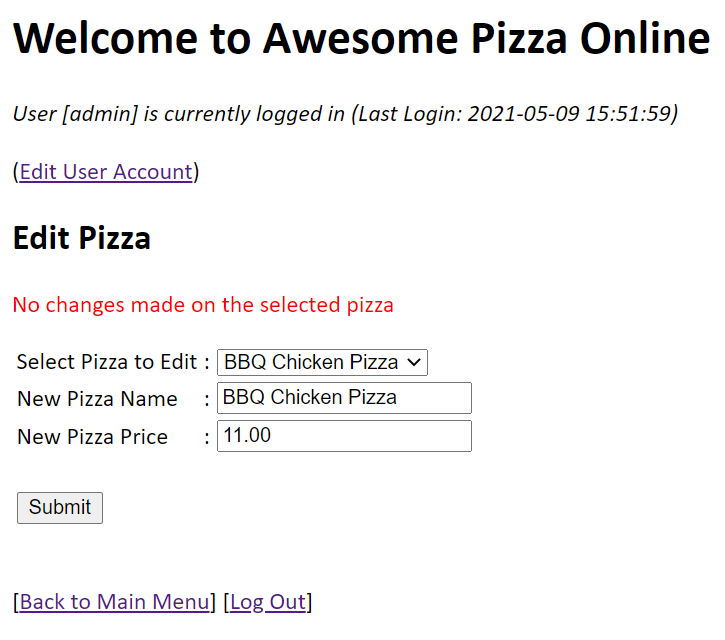
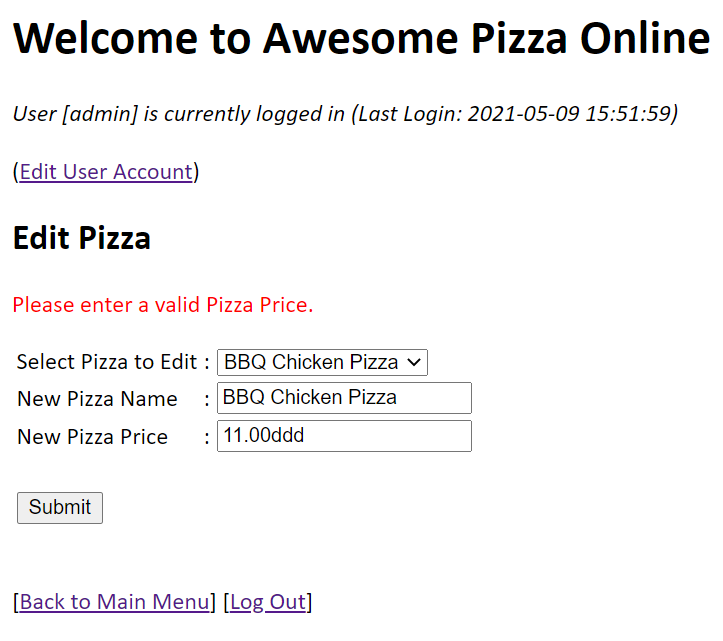
 



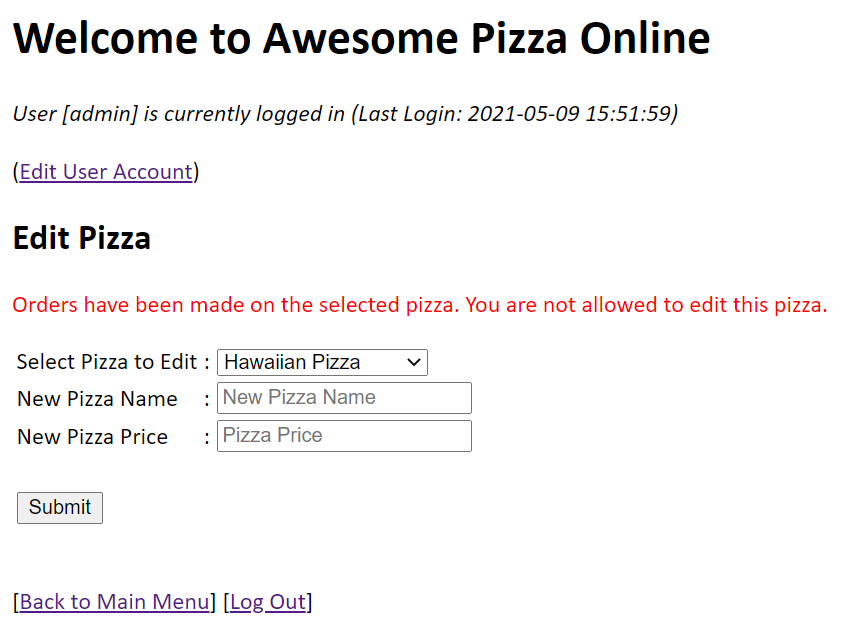
Edit Pizza

Edit Pizza is only available to the admin user. It allows the admin to edit a selected pizza and its price. The pizza to edit can be selected in a dropdown list box. User can enter a new pizza name and price. If the new pizza name or price is left blank, no changes will be made on the either one of these fields.

The following outputs display the validations in place.

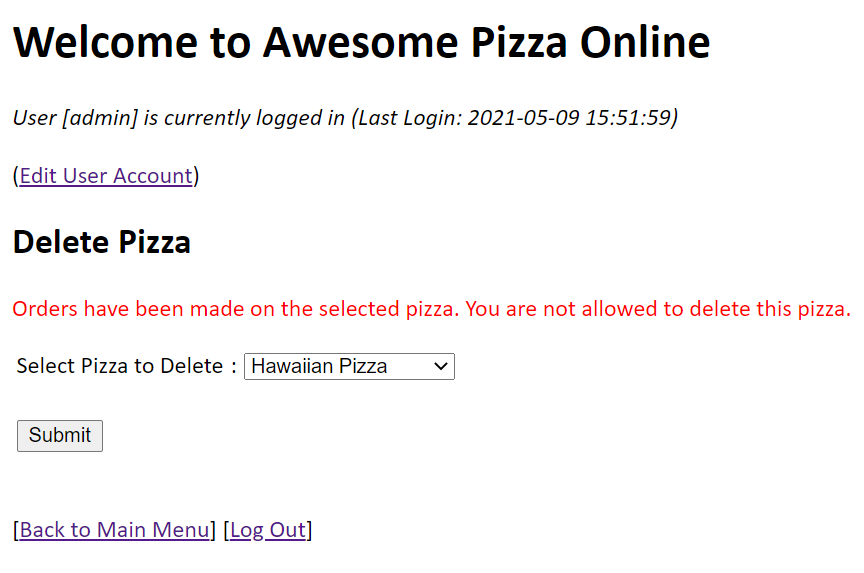
 

If an order has been made on the selected pizza, edit is not allowed.



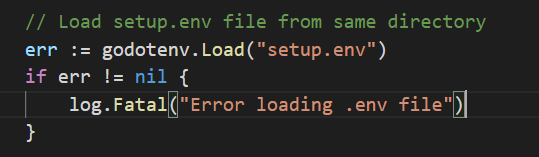
Delete Pizza

Delete Pizza is only available to the admin user. It allows the admin to delete a selected. If an order has been made on the selected pizza, delete is not allowed.

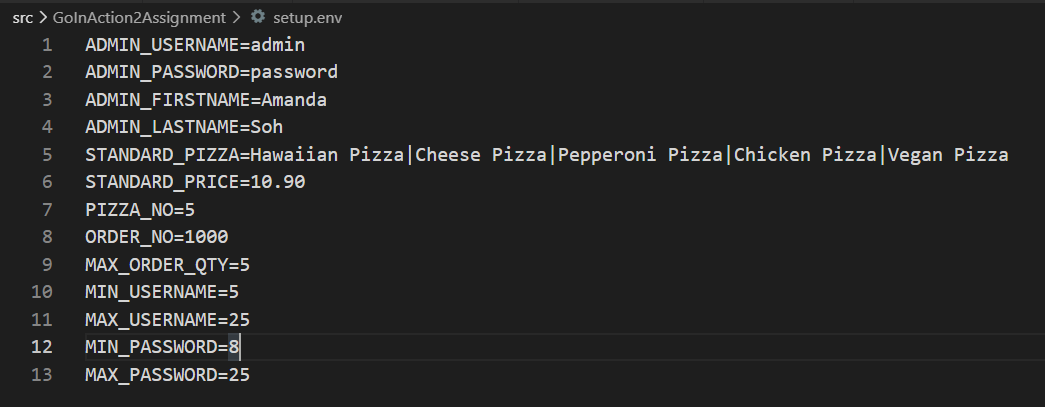


## **FILE MANIPULATION**

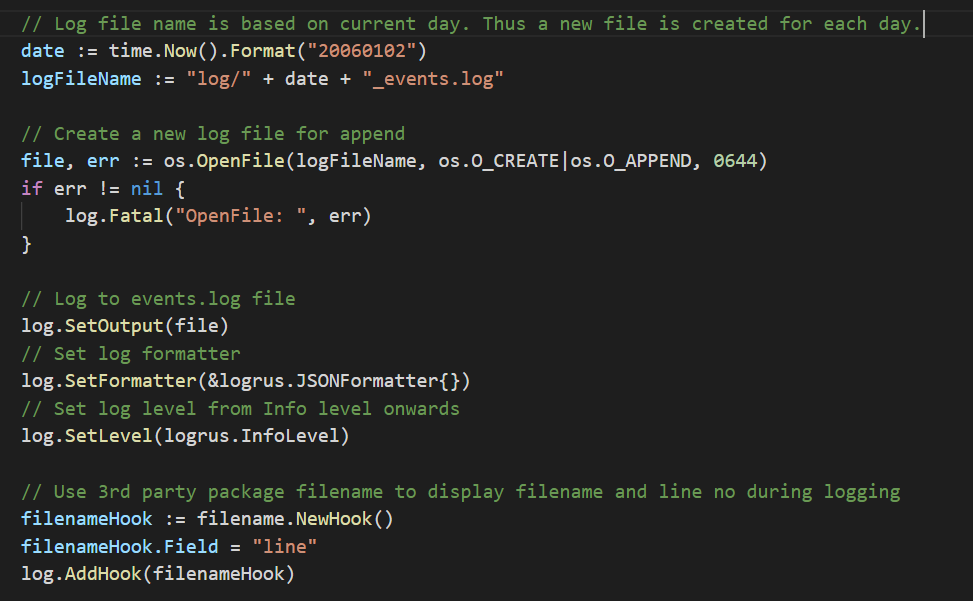
The application uses a .env file to extract environment variables for initialisation. File check procedures are in place and a log.Fatal is thrown when file error occurs.



The content in the .env file is as follows:



The application also creates a log file for logging the application events. Log file is created based on the current day. File check procedures are in place whereby a new file is created if not exist. Thus there will be no conflicts. The attempt to create/open the file will throw a log.Fatal if error occurs.



## **POST-VALIDATION ACTIONS**

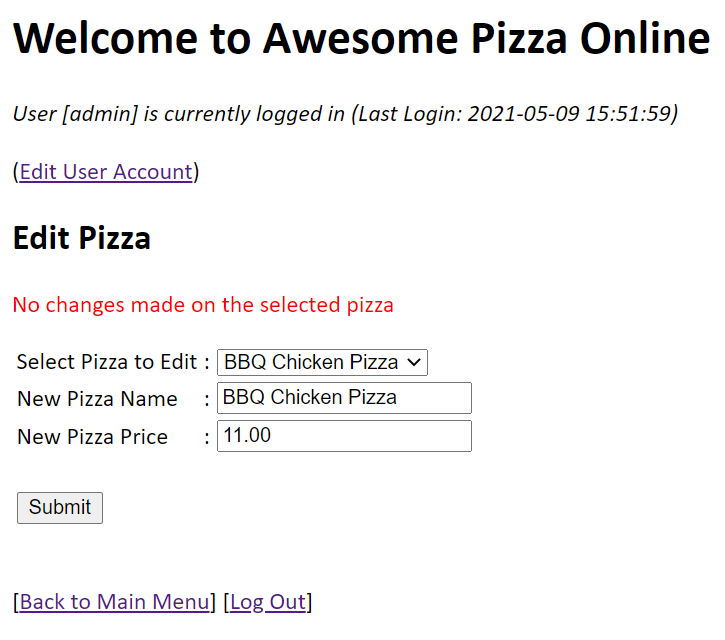
The following post validation actions have been performed:

Enforcement:

* User has been informed that submitted data failed to comply with requirements, and the data should be modified. Examples can be seen as previously described under Input Validation section.

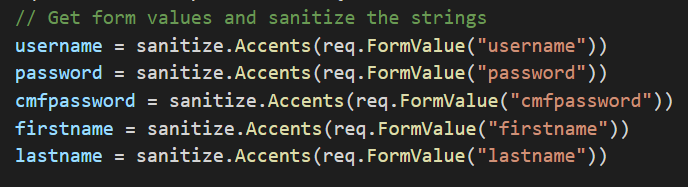
Advisory:

* Allow for unchanged data to be entered, but source actor informed user that data has issues. An example is in Edit Pizza.



## **SANITIZATION**

To perform sanitisation, I have imported third party package from “github.com/kennygrant/sanitize”.



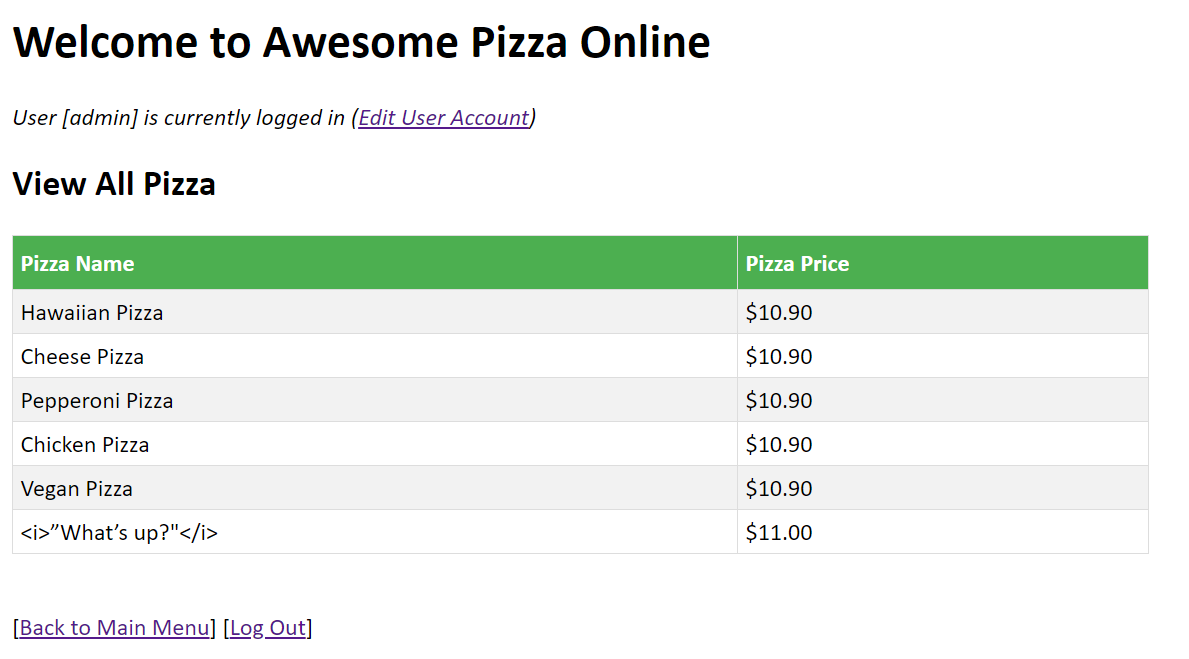
## **CONTEXT AWARENESS**

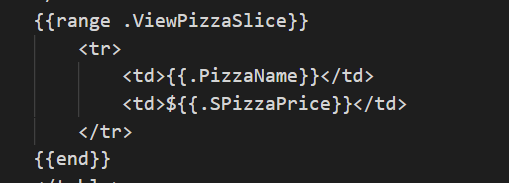
* Testing on context awareness:

Type ***“<i>”What’s up?"</i>***” into Pizza Name field in Add Pizza and submit.

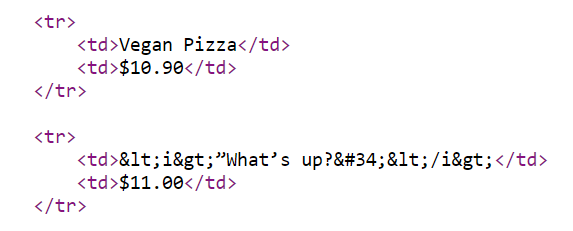


Go to View All Pizza menu option. The Go template engine is able to change the display of content according to context, i.e. where the content is placed in the document.





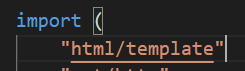
Following is the content shown in “View page source”.



In this example, it is shown that the context is able to recognize that the content ***"<i>”What’s up?"</i>***" is not a CSS style.

## **CROSS SITE SCRIPTING**

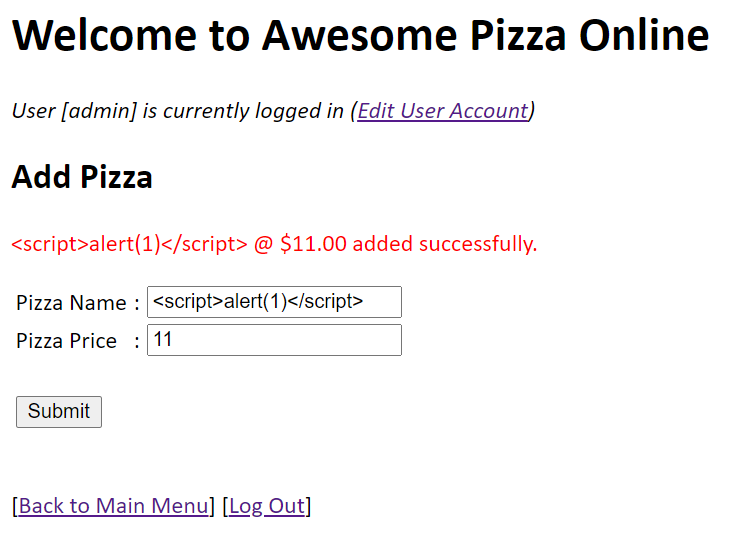
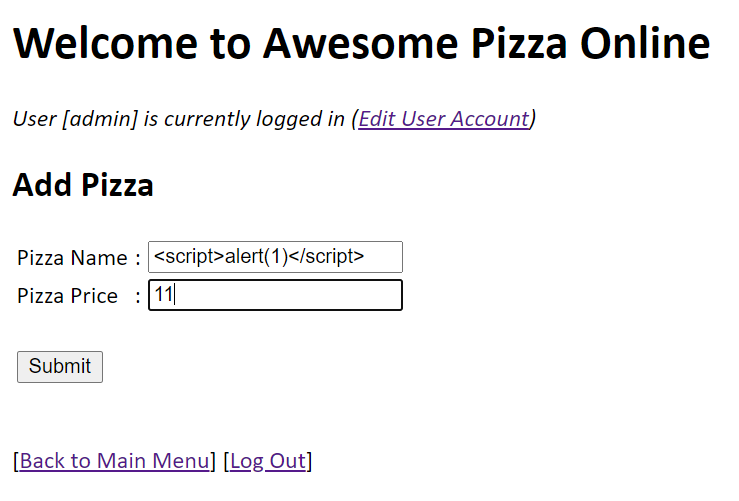
To prevent cross site scripting, it is important NOT to allow content type to be set automatically and to NOT use "text/plain". In the code for the application, “html/template” is used for proper escaping.



Within the html <head> tag, the <meta> data set for content is set with attribute content=”text/html” as shown.



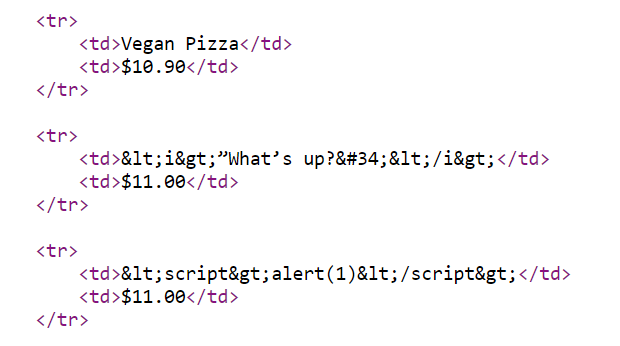
With the above, a test is done to check if cross site scripting will occurs by inputting “<script>alert(1)</script>” into Pizza Name field in Add Pizza and clicking submit.



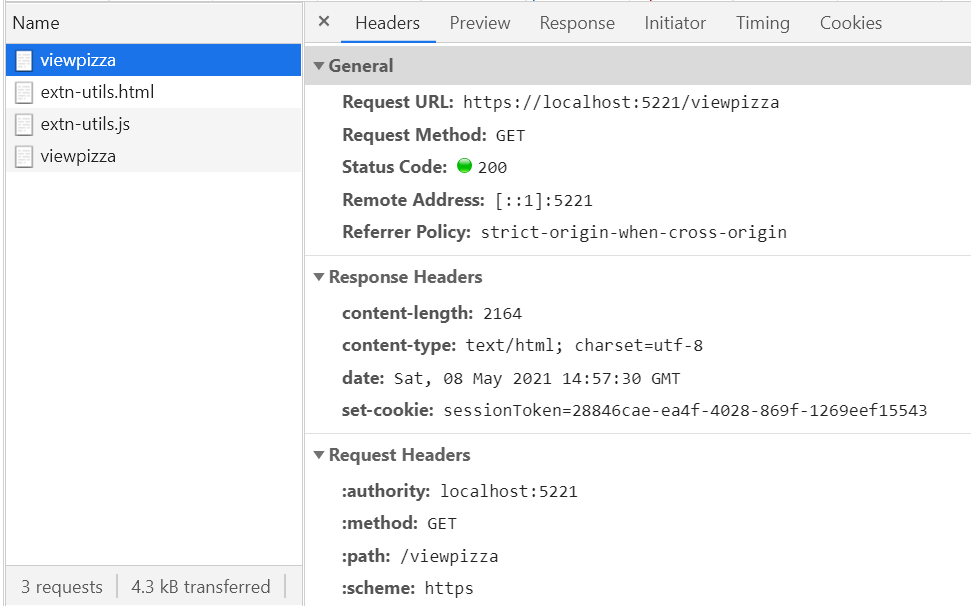
The following displayed shows the script is accepted as a line of text and was not executed with a pop-up alert.



The following is captured in “View page source”.



To check what is the Content Type, right-click on the browser and select “Inspect”. Select the “Network” tab then fire the request again. The following is observed. Content-type shown is “text/html” as intended.



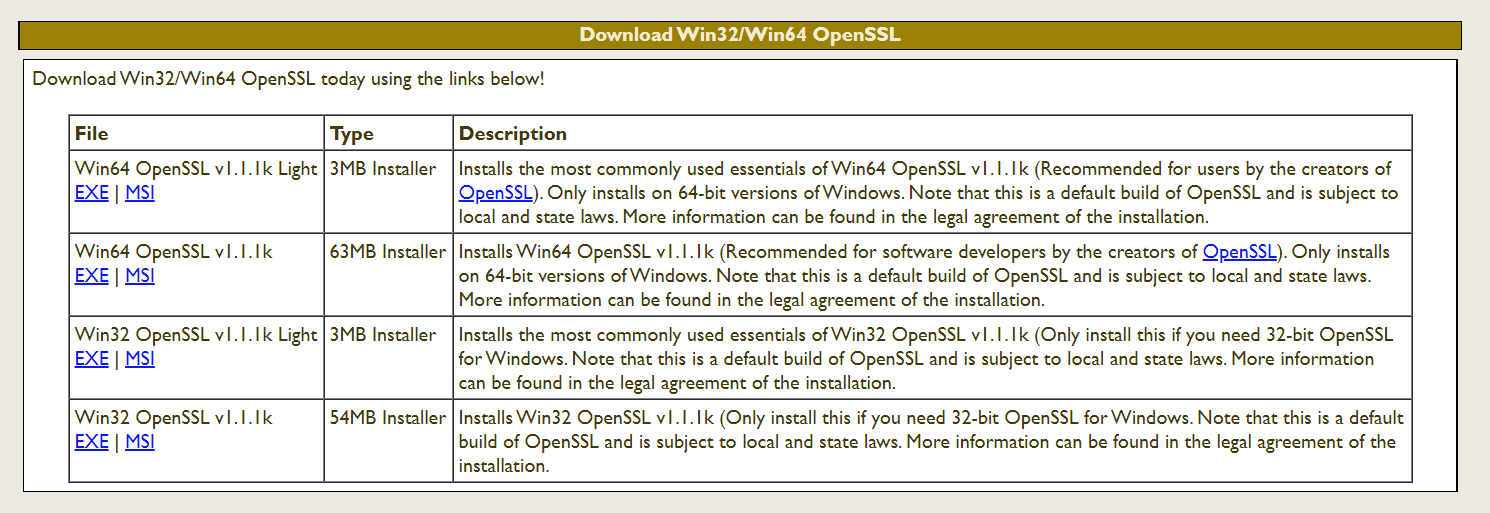
With context awareness, defensive programming is automated and aids to eliminate programmer mistakes thus defending against cross site scripting (XSS) attacks.

## **HTTP/TLS**

As developers, we need to be certain that channels for communication between server-client, server-database and all backend communications are secure. These communications must be encrypted to ensure data integrity, and protect against common attacks related to communication security. As with most major websites, this application uses ***HTTPS*** to encrypt and protect the communications between client and server. For testing purposes, OpenSSL is used to generate a certificate and key for establishing a handshake between the server and client to agree on a single session key to encrypt their packets.

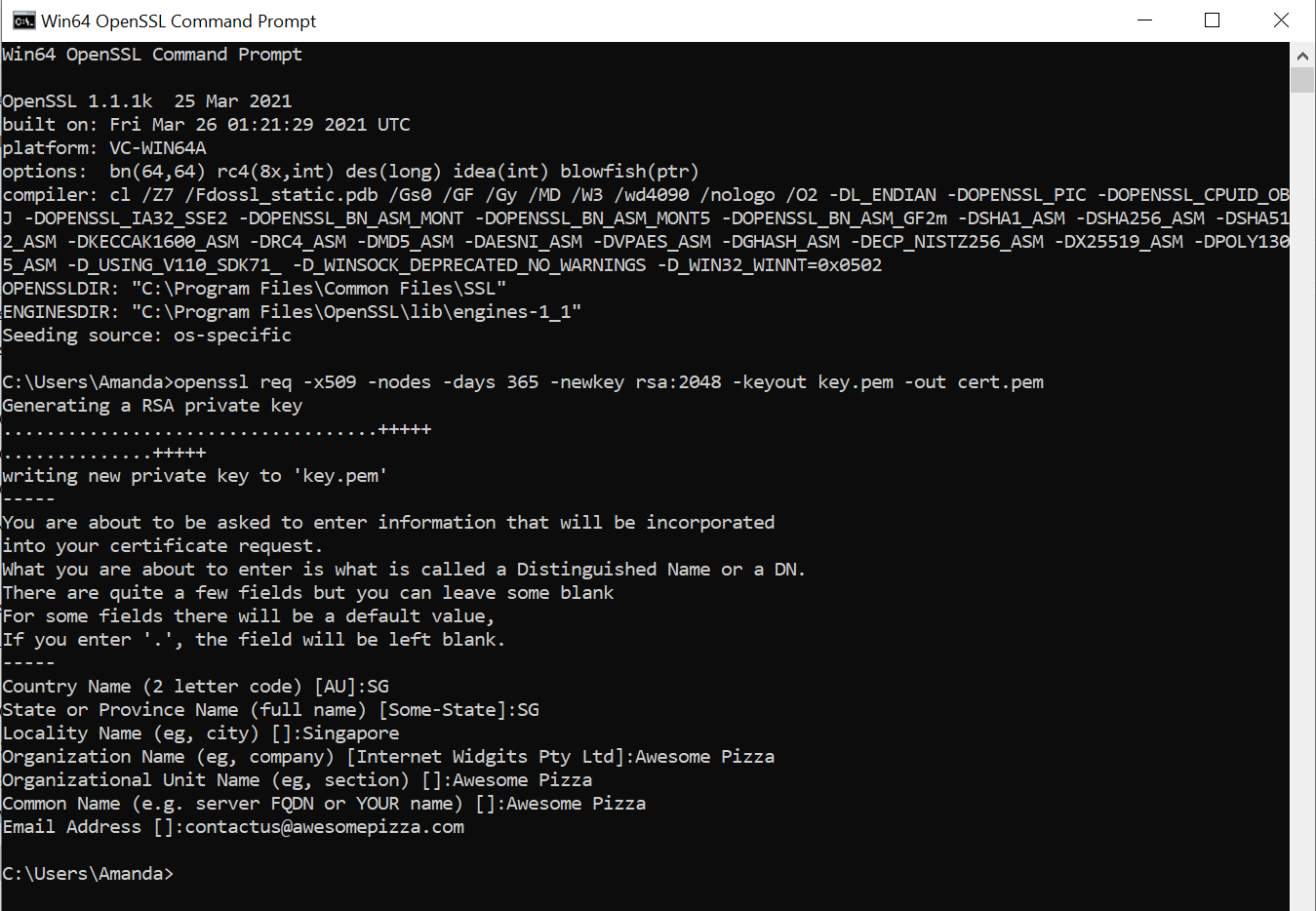
Note: The OpenSSL certificates is only used for development and testing. A CA certified SSL certificate is needed for production.

To use OpenSSL, we have to download and install it from <https://slproweb.com/products/Win32OpenSSL.html>

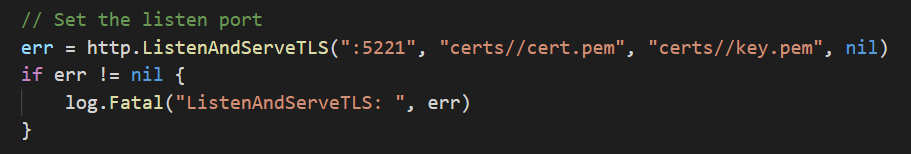


Once install, run the OpenSSL Command Prompt. Type the following to generate the server certificate and private key with OpenSSL:

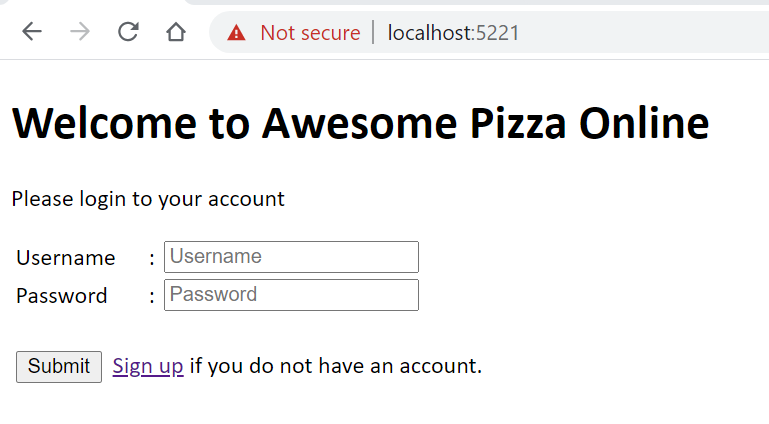
**> openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout key.pem -out cert.pem**



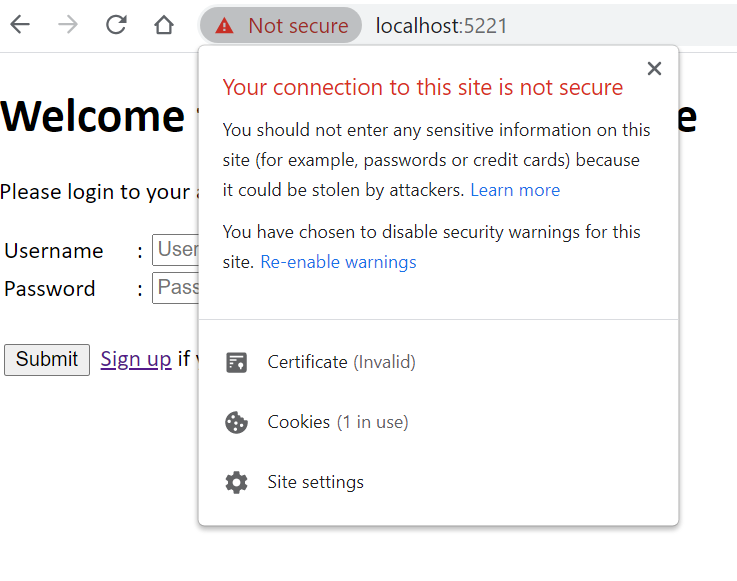
The code to use the certificate/key and start the server is a follows.

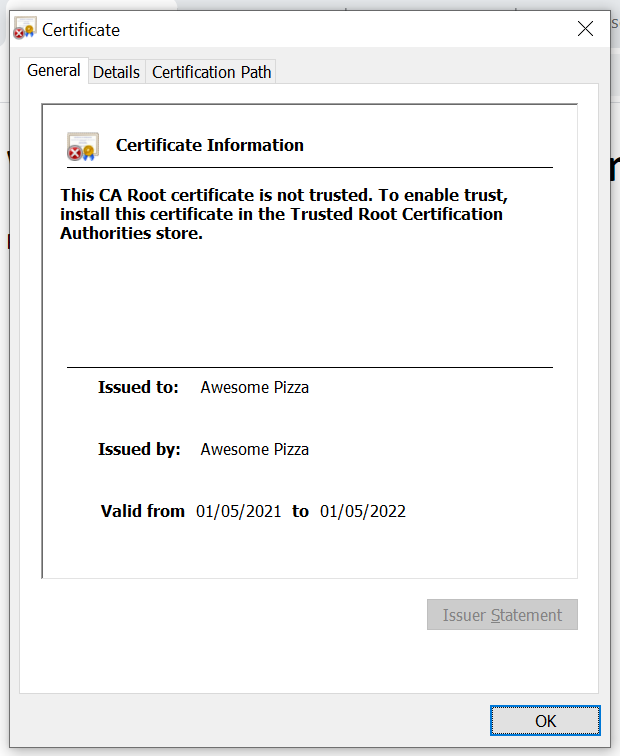


When the server is run, launch the client at <https://localhost:5221/>



As this is a self-signed certificate, the browser will not trust the connection. But the fact this appears, that means handshaking between server and client already done and thus the SSL/TLS is working.

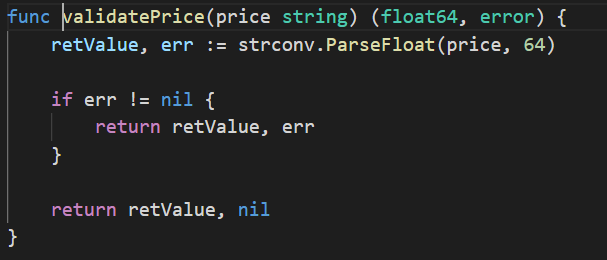




## **ERROR HANDLING & LOGGING**

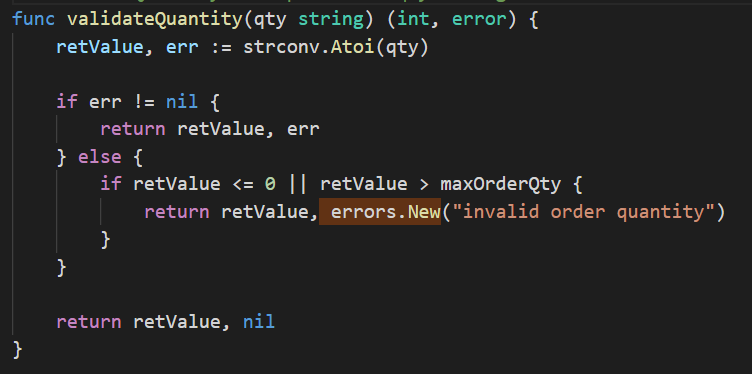
The following error handling ways are implemented in the application.

Using the error type in Go - Example



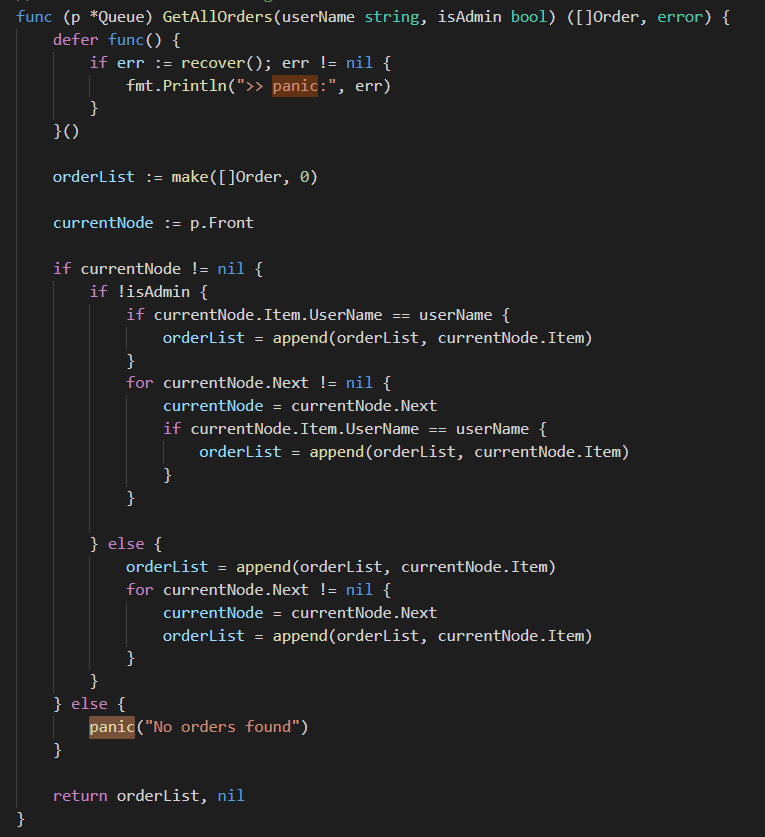
Package scope error variables using errors.New function – Example

errors.New creates a new error with its dedicated message.



Panic – Example

When panic occurs due to currentNode is nil, the defer func for recover() ensures that the application can continue to run.

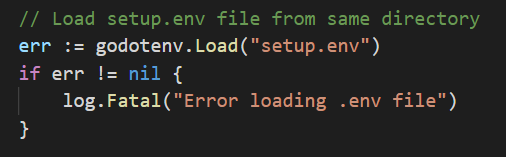




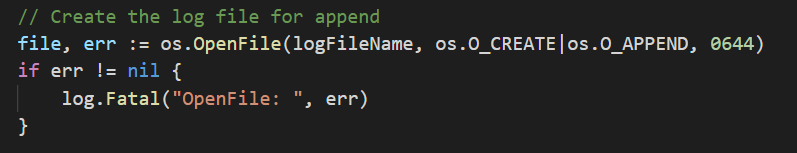
Log.Fatal – Example

Three occurrences of log.Fatal are used in the application with the understanding that Fatal effectively logged a message, then call os.Exit (1). It is used knowing that the error is essentially unrecoverable and the application will not be able to run properly. These three occurrences are used during the init func and right at the beginning of the main func.

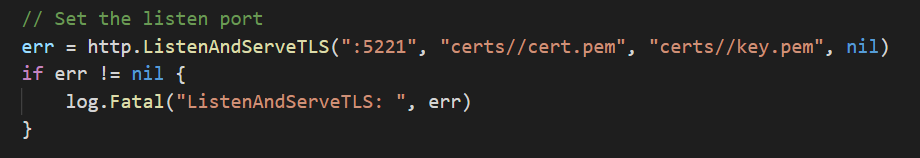
* In init() to load the .env file.



* In init() to opening/creating the log file.

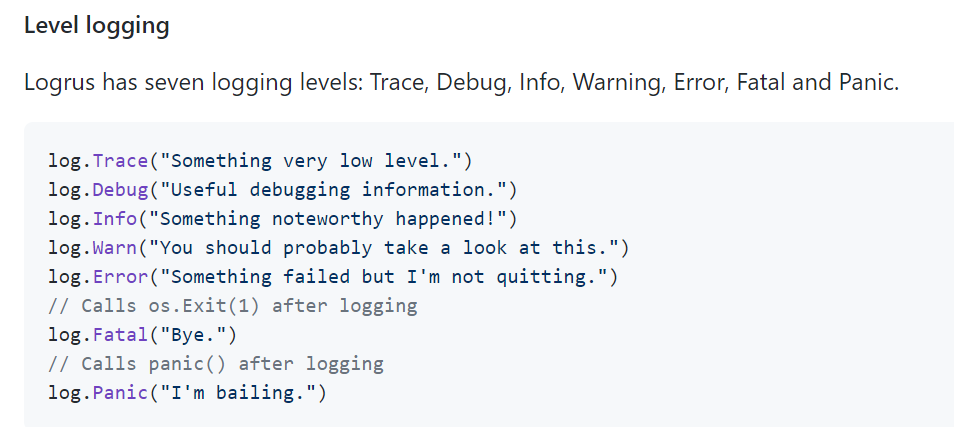


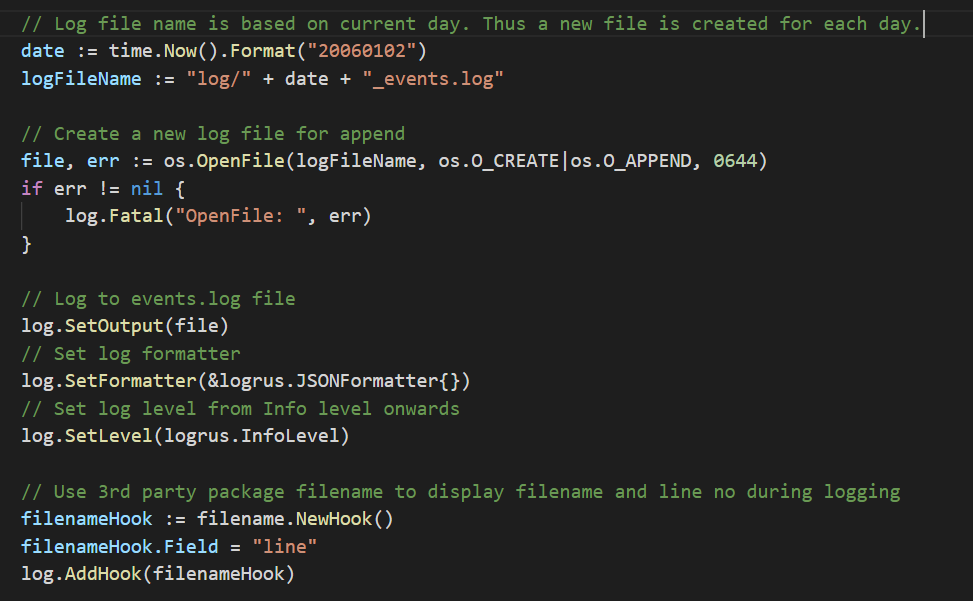
* In main() to listen to the server port.

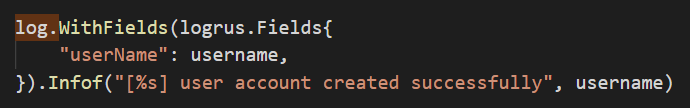


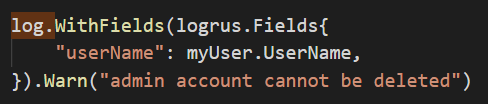
## **LOGGING**

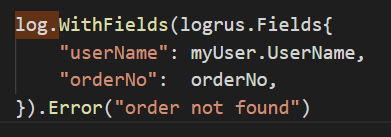
Logging file for the application is created at init () function and is executed before main() as part of program initialization. This is so that log package will be used immediately when program starts. I am using 3rd party library Logrus from <https://github.com/Sirupsen/logrus> to perform logging. Import "github.com/sirupsen/logrus". Logrus is able to use a JSONFormatter and is able to set the log level. In this application, we will log from Info level onwards. To be able to display filename and line in the logs, another 3rd party library is used – import filename "github.com/keepeye/logrus-filename"

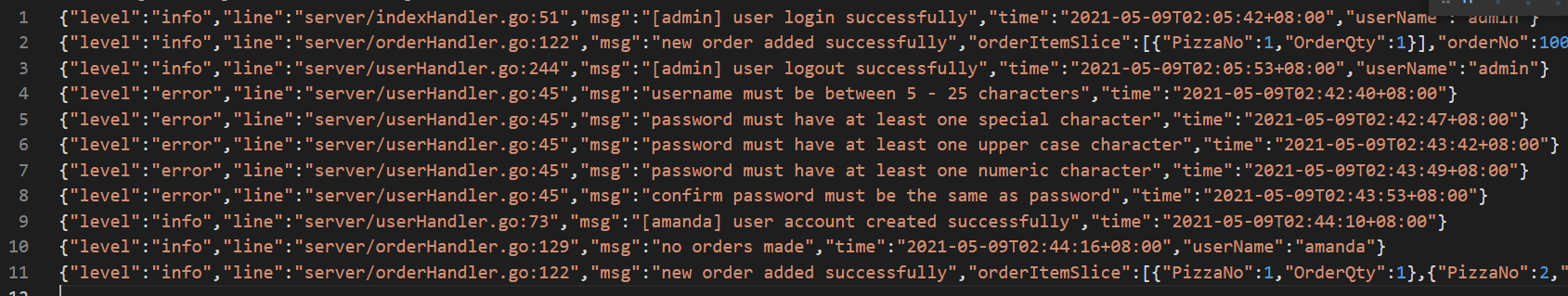






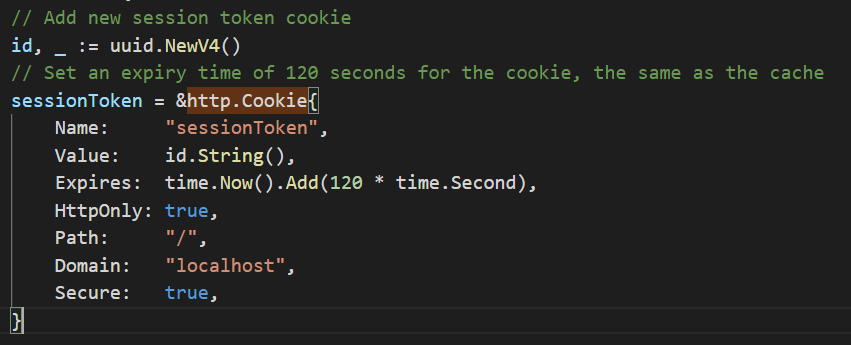


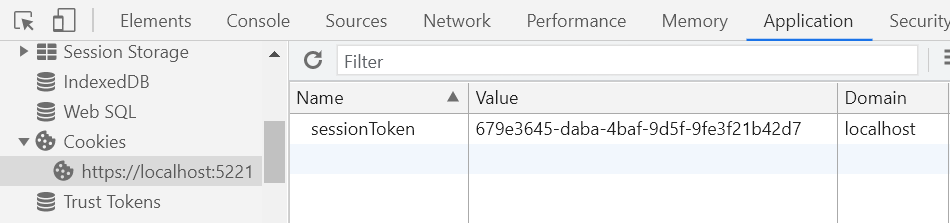




## **SESSION MANAGEMENT**

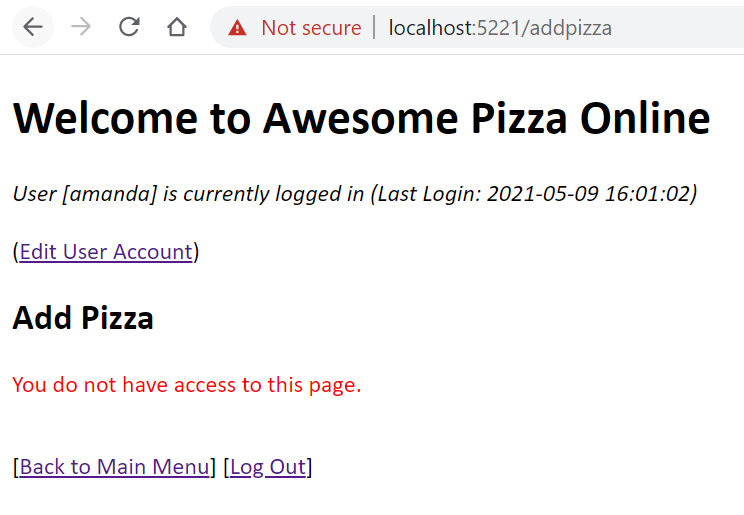
The application uses session token in the form of cookies to maintain the session state of users’ login to the system. It is encrypted during transport from server to client. The session token cookie name is “sessionToken” for this application. The cookie is created with the use of a unique ID generated using the Universally Unique Identifier (UUID) through the package: import uuid "github.com/satori/go.uuid". It is a 128-bit number and it is set upon user first access to the index page. The session token cookie is deleted when user logout and a new session token cookie is created to be set for the next login. The session token cookie is also set to expire in 120 seconds.



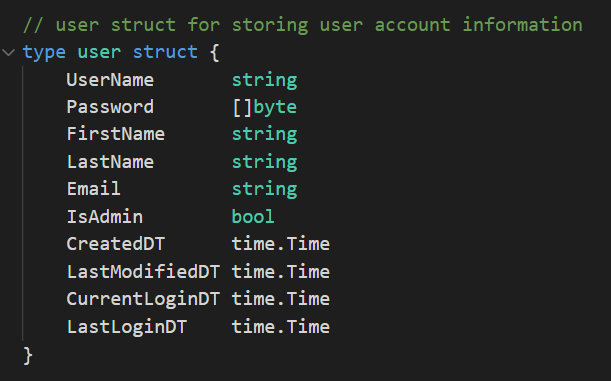


For session management, it is noted on the following:

* Upon sign in, a new session is always generated.
* Old session is never reused even if not expired.
* The Expire parameter enforces periodic termination of session to prevent session hijacking.
* Concurrent login for the same username is not allowed by maintaining a list of logged in users, and comparing the new login username against the list. The list is usually kept in database or other forms of persistent storage but for the current application, I am storing it in a map.
* Session identifiers are not exposed in URL’s but are located in the HTTP cookie header.
* Only POST and HTTPS is used in this application. No sensitive information is passed as HTTP GET parameters.
* Session data is protected from unauthorized access by other users of the server. Even if other user know the URL link of the webpages in the application, they will be denied access if they are not authorised for the page.



* Admin user is determined by the IsAdmin field setup in the user struct.

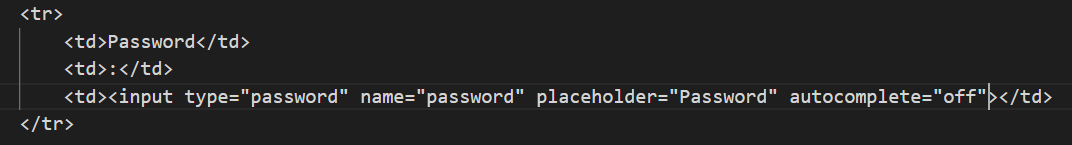


* Application uses HTTPS in all requests.

## **AUTHENTICATION & PASSWORD MANAGEMENT**

### **COMMUNICATING AUTHENTICATION DATA**

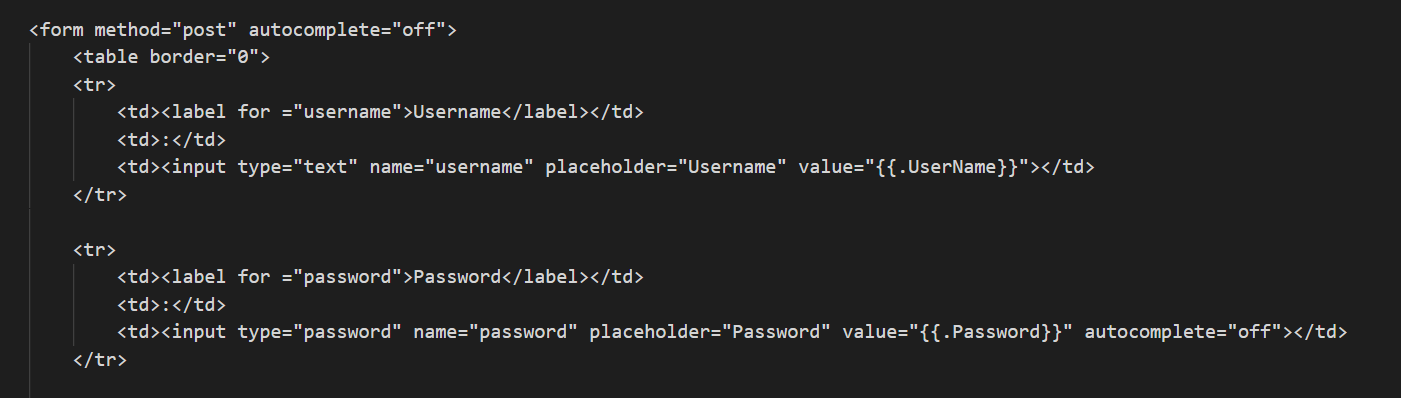
To ensure that password entry is obscured, and the “remember me” functionality is disabled, all input fields for “Password” have been set as follows.



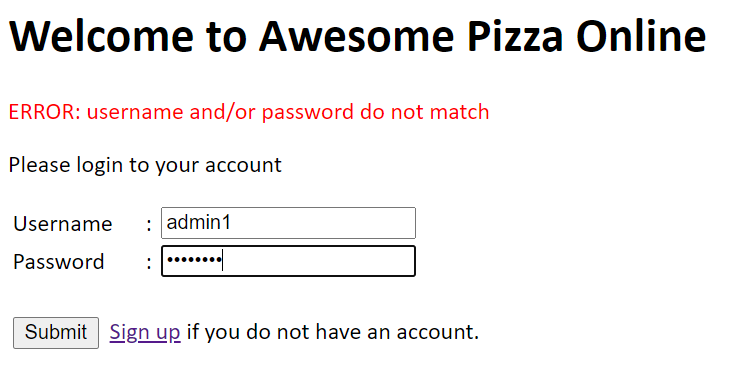
To prevent authentication credentials leakage to HTTP server logs, data is sent to the server using the HTTP POST method as follows.

Notes on POST:

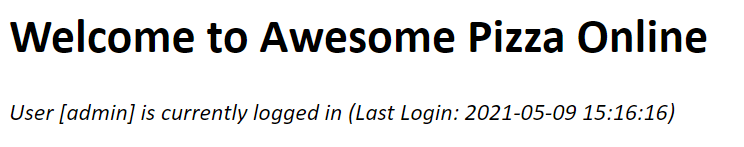
* Appends the form data inside the body of the HTTP request (the submitted form data is not shown in the URL).
* POST has no size limitations, and can be used to send large amounts of data.
* Form submissions with POST cannot be bookmarked.



When handling authentication errors, my application does not disclose which part of the authentication data was incorrect. "Invalid username and/or password" is used interchangeably.

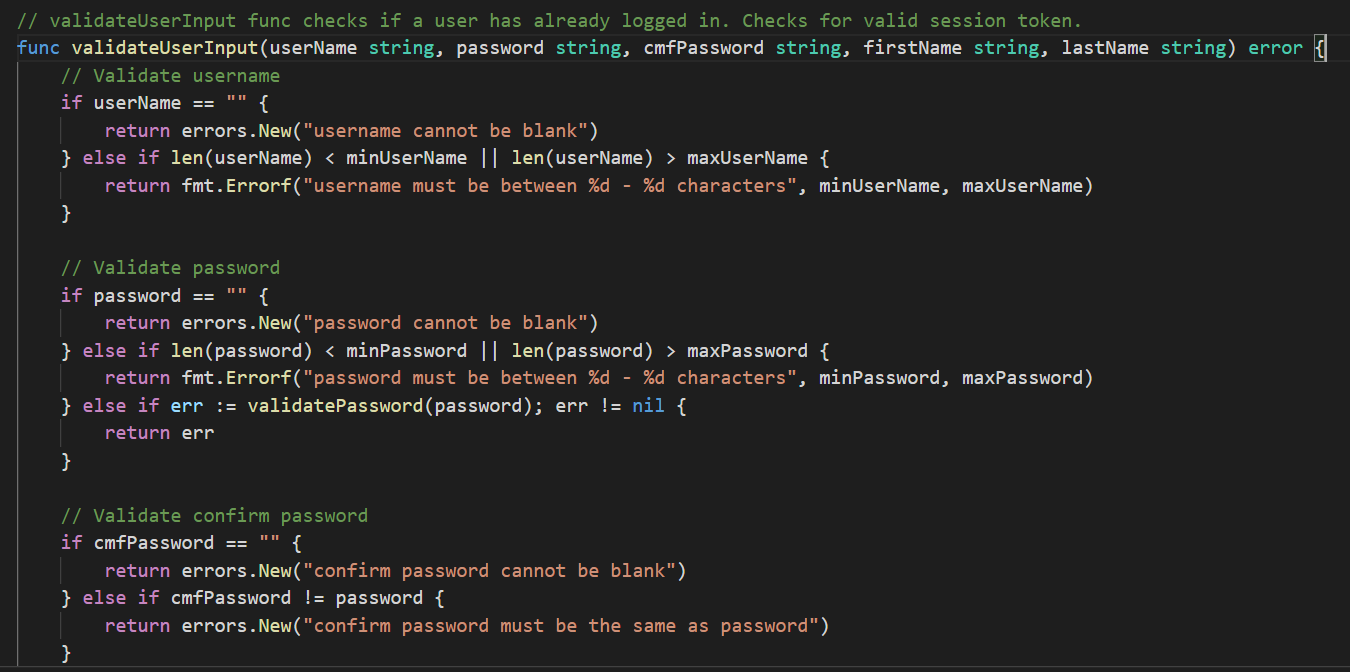


After a successful login, the user is informed about the last login date/time so that he/she can detect and report any suspicious activity.



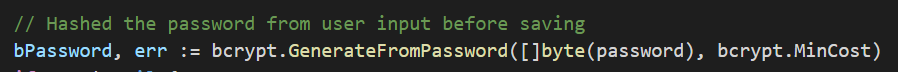
### **VALIDATION AND STORING AUTHENTICATION DATA**

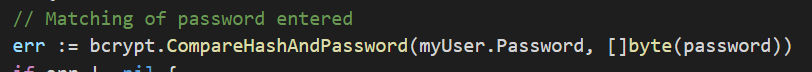
On validation should happen only on the completion of all data input, on a trusted system (e.g. the server), I have done this for the signup and edit user page.



### **STORING PASSWORD SECURELY**

For password storage, the hashing algorithm used in the application is *brcypt* which will take care of the hashing and salting of passwords in a robust way. The package to get is “golang.org/x/crypto/bcrypt"





# **GO DOCUMENTATION**

* Click on the following link to view the full documentation in pdf format.

[***GoInAction2Assignment - Go Documentation.pdf***](GoInAction2Assignment%20-%20Go%20Documentation.pdf)

* Alternatively, you can install the godoc library.

go get -u golang.org/x/tools/godoc

* Run the following command in the VSC terminal or cmd prompt. Ensure that you have godoc installed.



* Open up the browser, enter the URL:

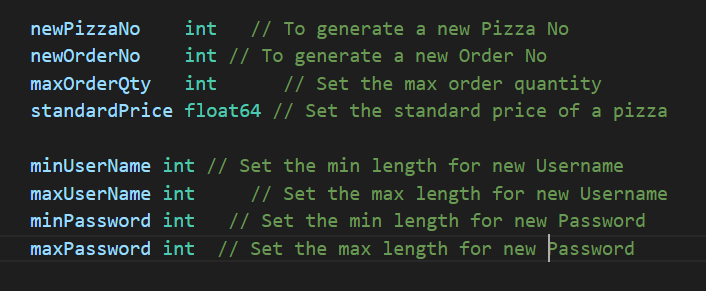
[***http://localhost:7001/pkg/GoInAction2Assignment/?m=all***](http://localhost:7001/pkg/GoInAction2Assignment/?m=all)

* Note: For illustration purposes to display the comments of unexported functions in godoc, I am appending ***“/?m=all***” in the URL link. Otherwise, the documentation will only shows the comments of exported functions (functions which are capitalized).

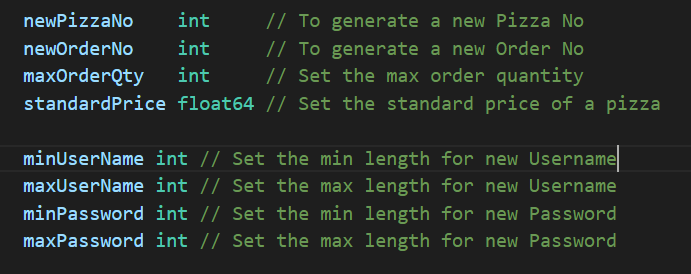
# **IDIOMATIC GO**

## **FORMATTING**

When using Visual Studio Code (VSC), the IDE will auto format the indentation for the whole codes including comments. Thus I will not be using Gofmt tool (which is a tool that automatically formats Go code) to line comments on the fields of a structure. The following screenshots show the before and after of VSC formatting.

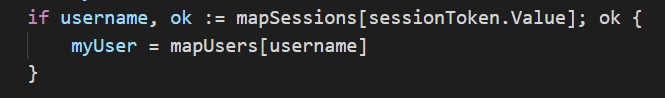


Before formatting



After formatting (when file is saved)

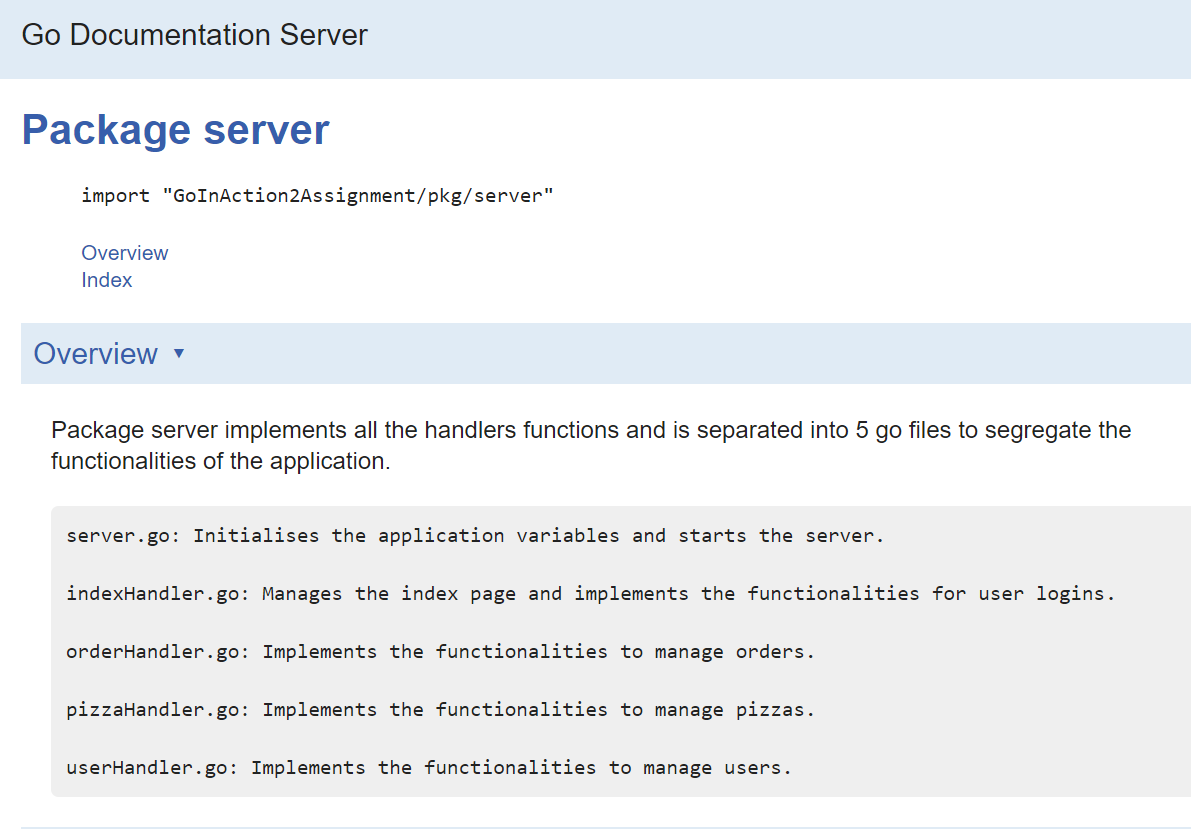
In addition because Go needs fewer parentheses than C and Java: the control structures (if, for, switch) do not have parentheses in their syntax. Example:

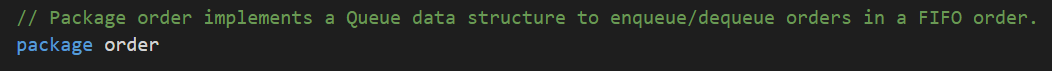


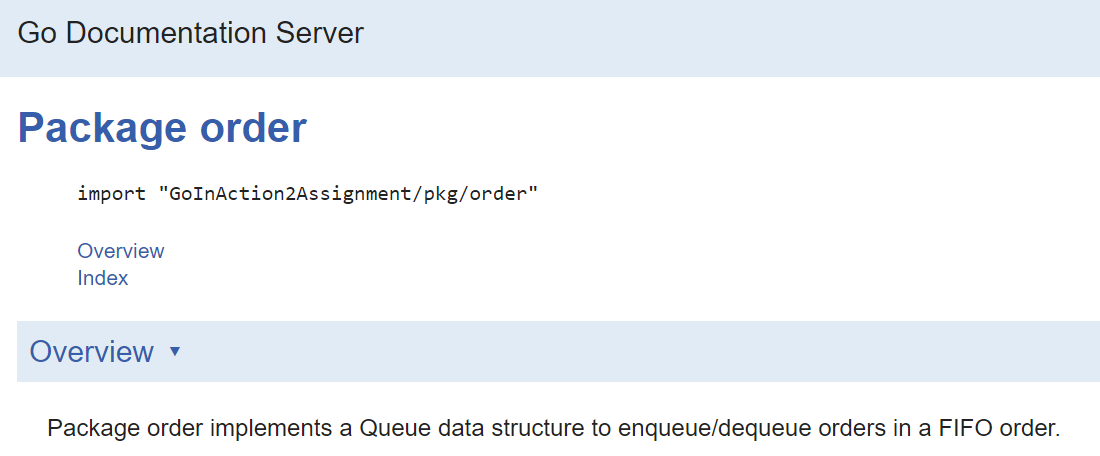
## **COMMENTARY**

Go provides C style /\* \*/ block comments and C++ style // line comments. In my code // style line comments are mostly used. Comments are put before top level declarations, with no intervening newlines, so that they can be extracted by godoc along with the declaration to serve as explanatory text for the item. Package comments that are in large blocks will be put with /\* \*/ block comments (examples as shown). And comments that are brief will be using the // style line comments.





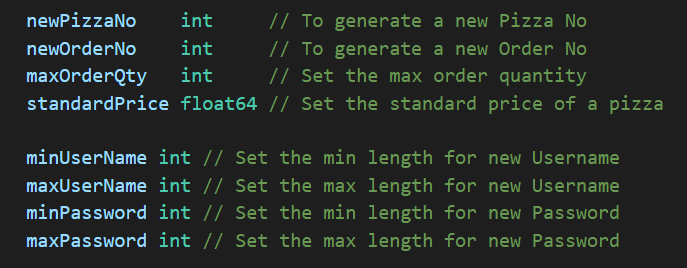




## **NAME**

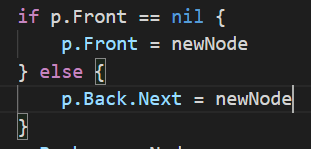
In my previous assignment, I have created two packages with the names adminOrder and adminPizza. However, after the lessons in Go In Action II, I am now aware that by Go convention, packages are given lower case, single word names; Convention is not to use underscores or mixedCaps. Thus for this assignment, I have renamed my packages to just order and pizza. I have also added a server package.

Also identifiers in my code use mostly MixedCaps or mixedCaps rather than underscores for multiword names. Example:

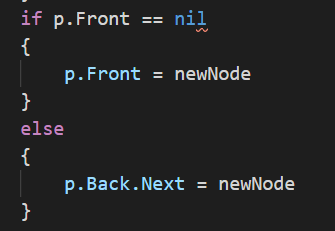


## **SEMICOLONS**

Semicolon insertion rules dissert that we cannot put the opening brace of a control structure (if, for, switch, or select) on the next line. If we do, a semicolon will be inserted before the brace, which could cause unwanted effects. Examples:



Correct way of using braces in Go



Incorrect way of using braces in Go

## **CONTROL STRUCTURES**

In the coding, it is noted that there is no ***do*** or ***while*** loop in Go, only a slightly generalized ***for***which is mostly used for the codes. Examples in the different ways a ***for*** is used.

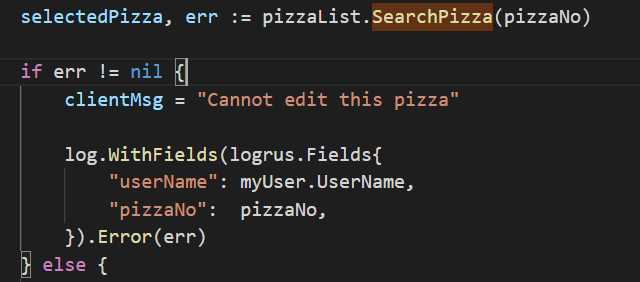






## **FUNCTIONS**

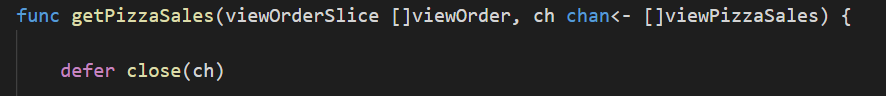
In Go, one unusual features is that functions and methods can return multiple values. Thus, a write can return a count and an error. Go programmers are encouraged to check for the error and handle it before writing any code for the case if no error occurs. This is implemented in most parts of the codes. Examples: the SearchPizza function returns a Pizza struct as well as the error. The error is handled by the calling function and handled with appropriate actions.





## **DEFER**

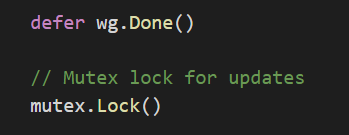
Go's defer statement schedules a function call (the deferred function) to be run immediately before the function executing the defer returns. Examples where defer are used in the codes are unlocking a mutex, closing a file or channel as shown in the following examples.



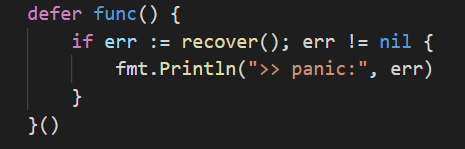
Defer a channel closure



Defer a file closure



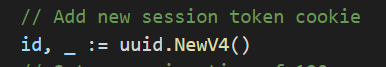
Defer WaitGroup.Done

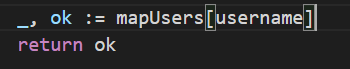


Defer func() for panic recovery

## **BLANK IDENTIFIER**

The blank identifier is assigned or declared with any value of any type, with the value discarded harmlessly in some parts of the codes. As described, it represents a write only value to be used as a place holder where a variable is needed but the actual value is irrelevant. It is used mostly for multiple assignments whereby an assignment requires multiple values on the left side, but one of the values is not used by the program. Using a blank identifier on the left hand side of the assignment avoids the need to create a dummy variable, and makes it clear that the value is to be discarded. Examples of its uses are as follows:







## **UNUSED PACKAGE IMPORT**

Another use of the blank identifier is to provide a workaround for unused imports and variables to prevent an error occurrence for not using it. This can allow the compilation to proceed and not have the unused imports and variables to be deleted, only to realise later that we need them again.

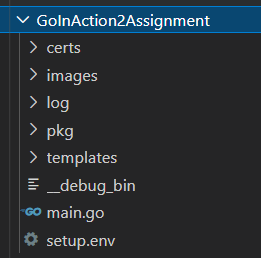
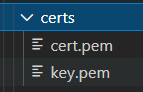
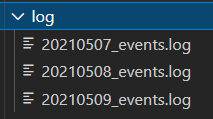
# REFERENCES

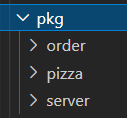
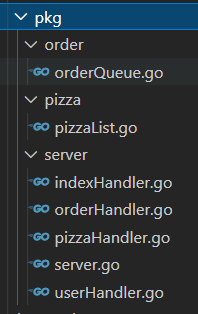
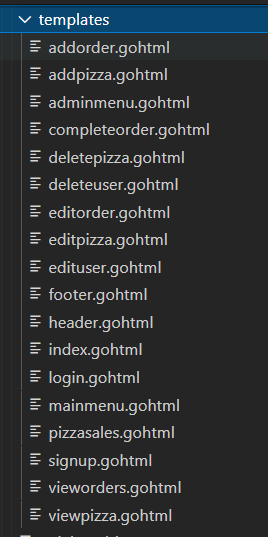
* <https://slproweb.com/products/Win32OpenSSL.html>
* <https://curtisvermeeren.github.io/2017/09/14/Golang-Templates-Cheatsheet#creating>
* <https://gist.github.com/kylelemons/2711541>
* <https://github.com/keepeye/logrus-filename>
* <https://www.scalyr.com/blog/getting-started-quickly-with-go-logging>
* <https://www.w3schools.com/html/default.asp>
* <https://golangcode.com/validate-an-email-address/>

# APPENDIX

## INSTRUCTIONS ON HOW TO RUN THE APPLICATION

* For information, my program is currently running Go version - Go1.16.3
* My current project files resides in my local **C:\Users\Amanda\go\src\GoInAction2Assignment\** with the following directories/files structures.

* Install created packages for ***order***, ***pizza*** and ***server***
  + Run in cmd prompt or VSC terminal: ***go install*** in the respective folders of the 3 packages that I have created.
  + Install ***pizza*** package in ***GoInAction2Assignment/pkg/pizza/***



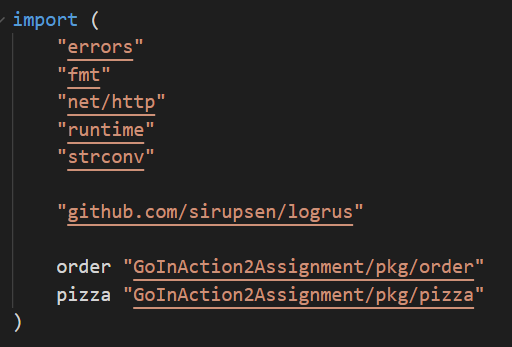
* + Install ***order*** package in ***GoInAction2Assignment/pkg/order/***



* + Install ***server*** package in ***GoInAction2Assignment/pkg/server/***



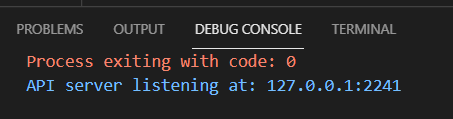
* It is important to use the same folder name structure because the import of the self-created packages in takes the following format.



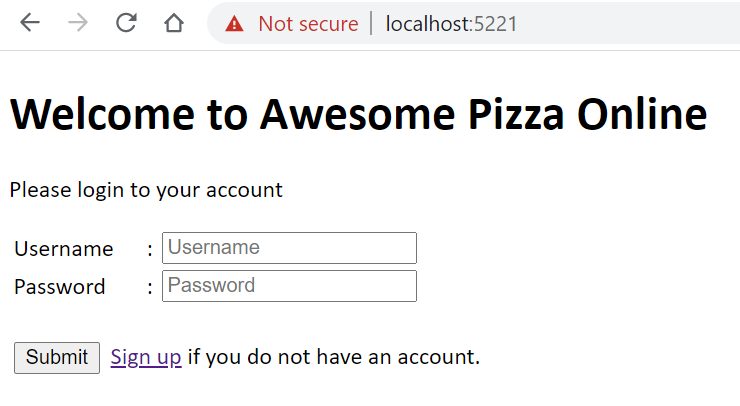
* Get 3rd party packages used in the application. Run in cmd prompt or VSC terminal: ***go get*** the following packages:
  + "github.com/joho/godotenv"
  + "golang.org/x/crypto/bcrypt"
  + "github.com/satori/go.uuid"
  + "github.com/sirupsen/logrus"
  + "github.com/keepeye/logrus-filename"
  + "github.com/kennygrant/sanitize"
* Once the packages are installed successfully, run F5 in Visual Studio Code (VSC) on ***main.go*** file to start the application. Alternatively, you can also type the following in cmd prompt or VSC terminal.



* The server will start running.



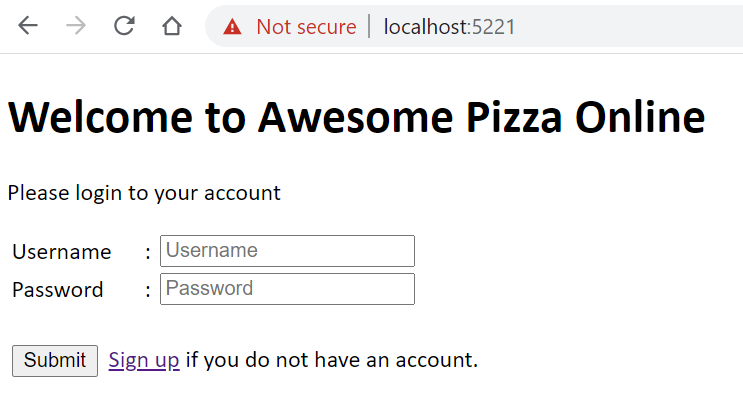
* Open the browser in Chrome and type in [***https://localhost:5221/***](https://localhost:5221/) to start the client. The index cum login page is displayed.



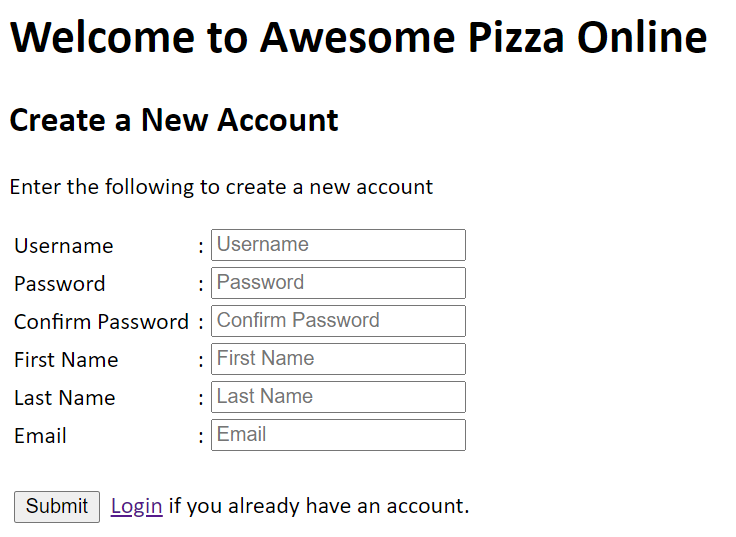
For testing purpose only, the default user is an admin and the login for username is “**admin**”, password is “**password**” (This is setup in the .env file)

## SAMPLE WALK-THROUGH ON THE APPLICATION

User starts the client by keying in the URL: [***https://localhost:5221/***](https://localhost:5221/)***.*** By default the index page which is also the login page will be displayed.

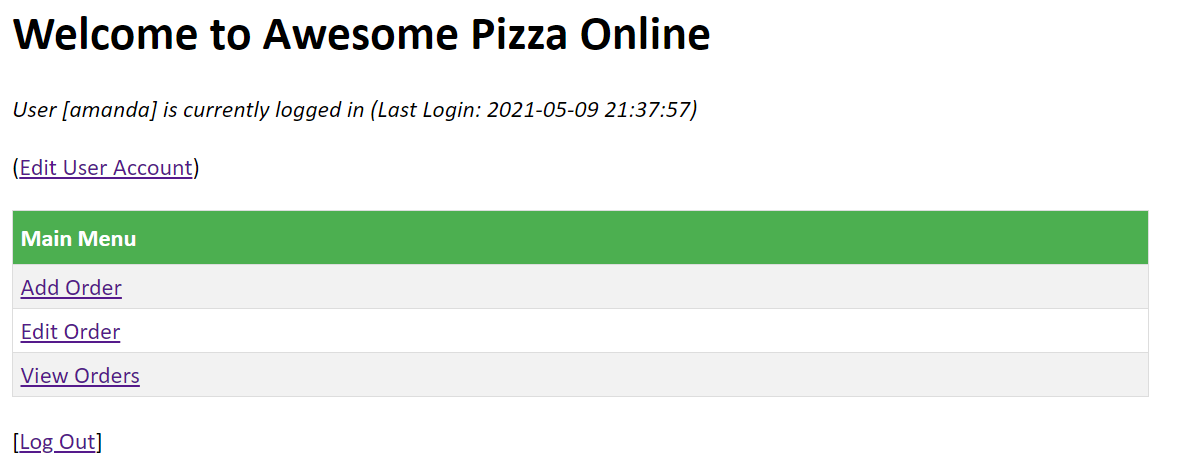


A new user can create an account by clicking on the [***Sign up***] hyperlink which will show the page below.

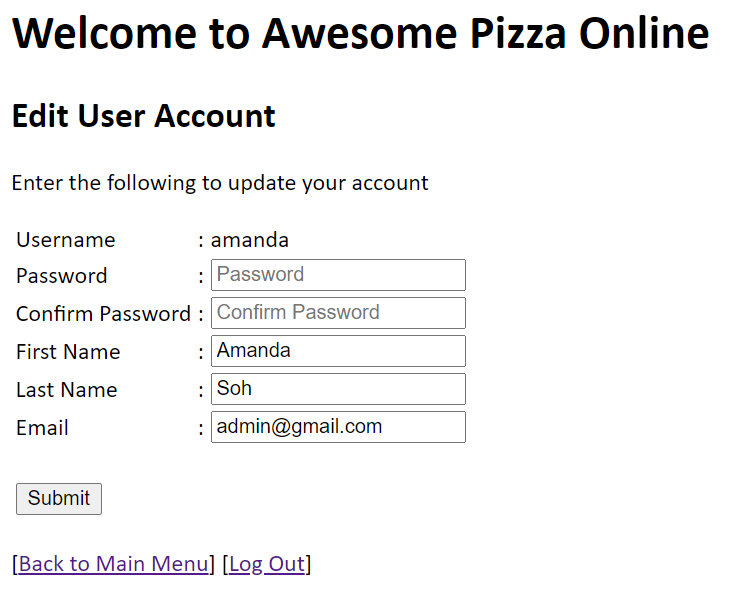


### NON-ADMIN USER

Upon valid submission of the account information, user will be directed back to the default index page now showing the user menu that is accessible by the user. Currently a user can add an order, edit an order and view orders that he/she has added.



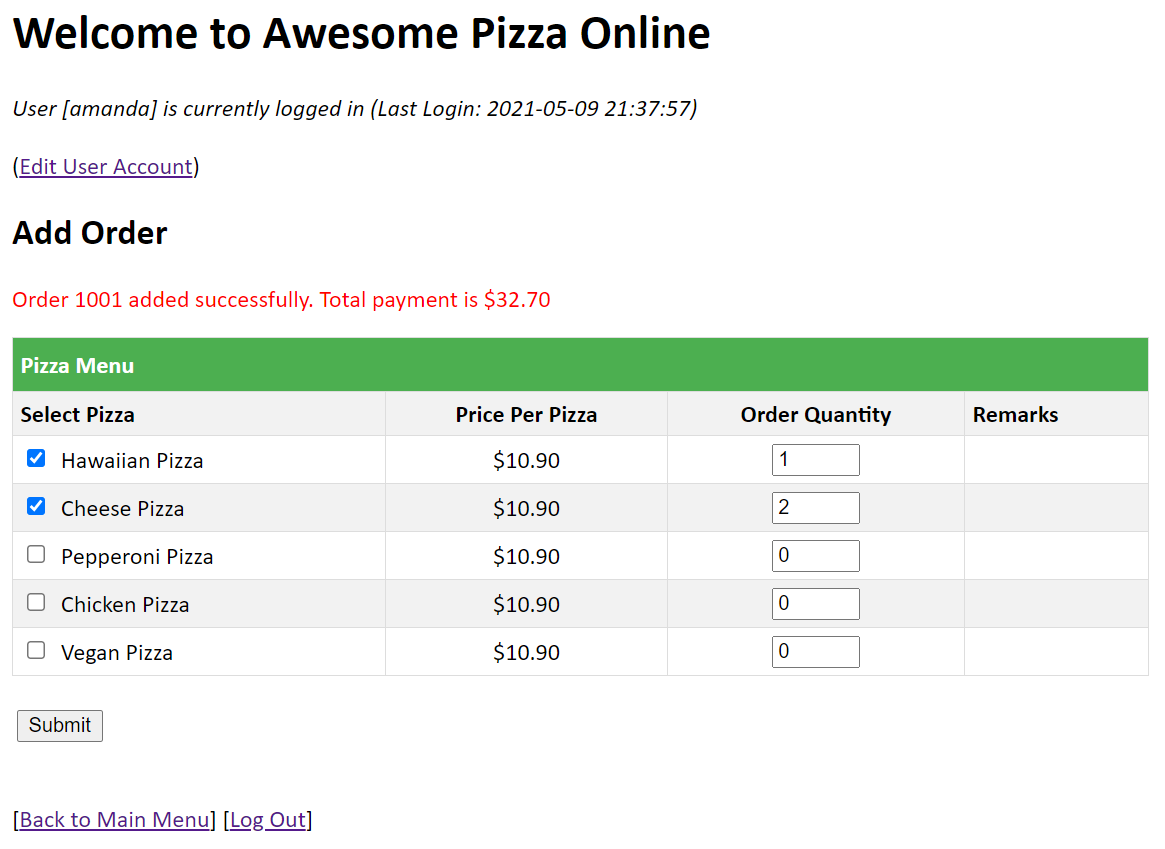
The user is also able to edit his/her account by clicking on the [***Edit User Account***] hyperlink. User is not allowed to change his/her user name but he/she can update their names, email and change their password. Due to time constraints, the user profile is limited to this information, however for future upgrades, information like addresses and phone numbers can be added to beef up the application.



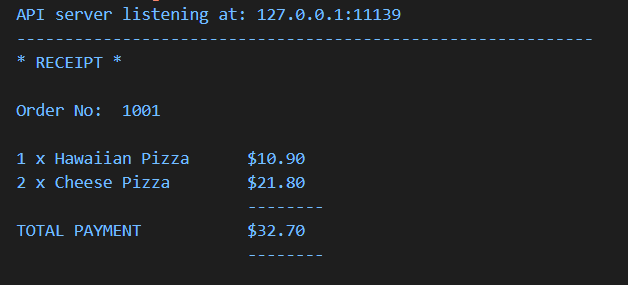
From the Main Menu, the [***Add Order***] hyperlink will bring user to the Add Order page where user can select the pizza he/she wants to order by checking on the checkbox next to the pizza name and entering a valid quantity. There is currently a limit on the quantity of 5 for each pizza selected. This is set in the .env file and can be changed easily.



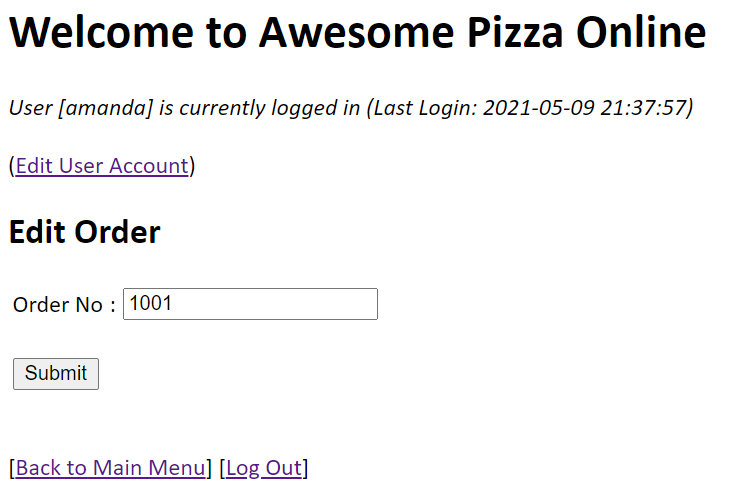
When a valid order is made, user will be prompted with a message on the same page. The refreshed page will also check on the pizza selected and its quantities ordered.



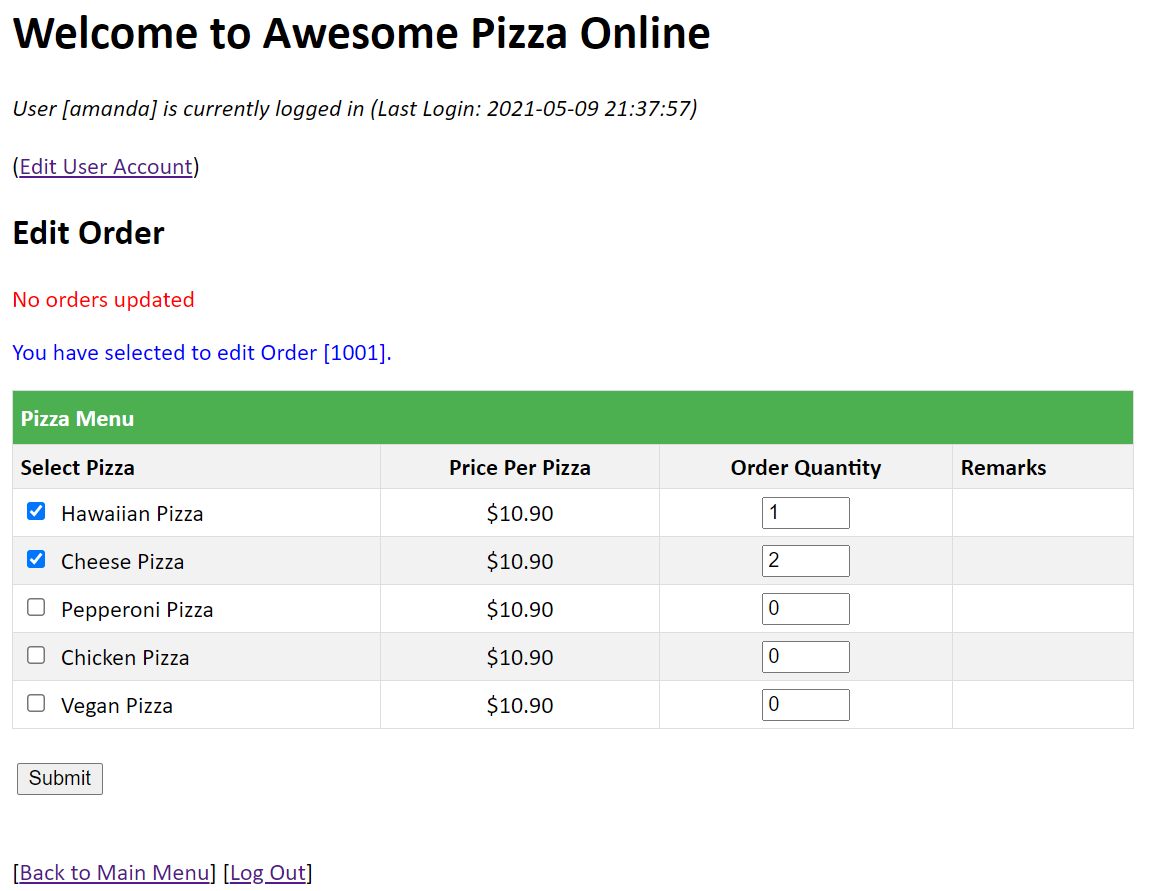
At the same time, a receipt will be printed at the server end to notify of the orders made.



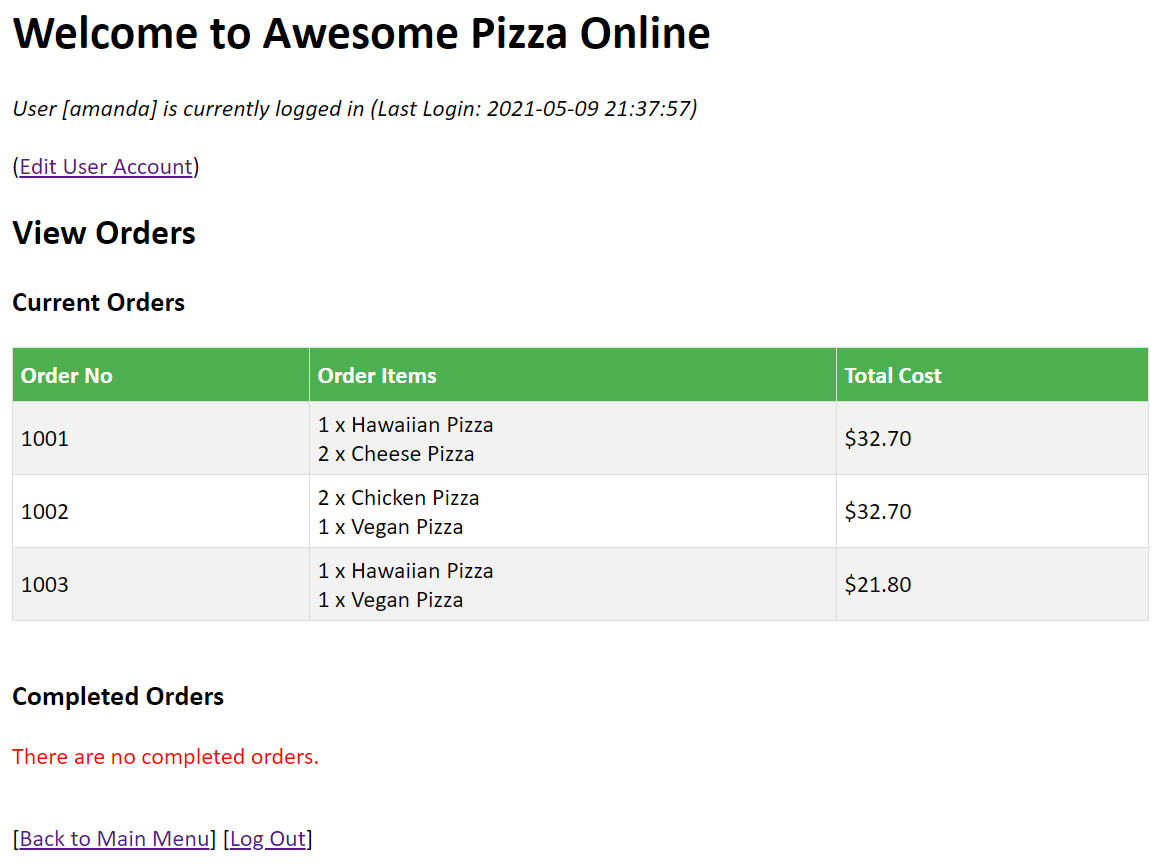
The [***Edit Order***] hyperlink in the main menu brings user to the edit order page where user has to enter the order number that he/she wants to edit. User is only allowed to edit orders that he/she had made.



When a valid order is found, the same page will be refreshed showing a similar page as [***Add Order***] for user to amend the order information. A new receipt will also be printed at the server.

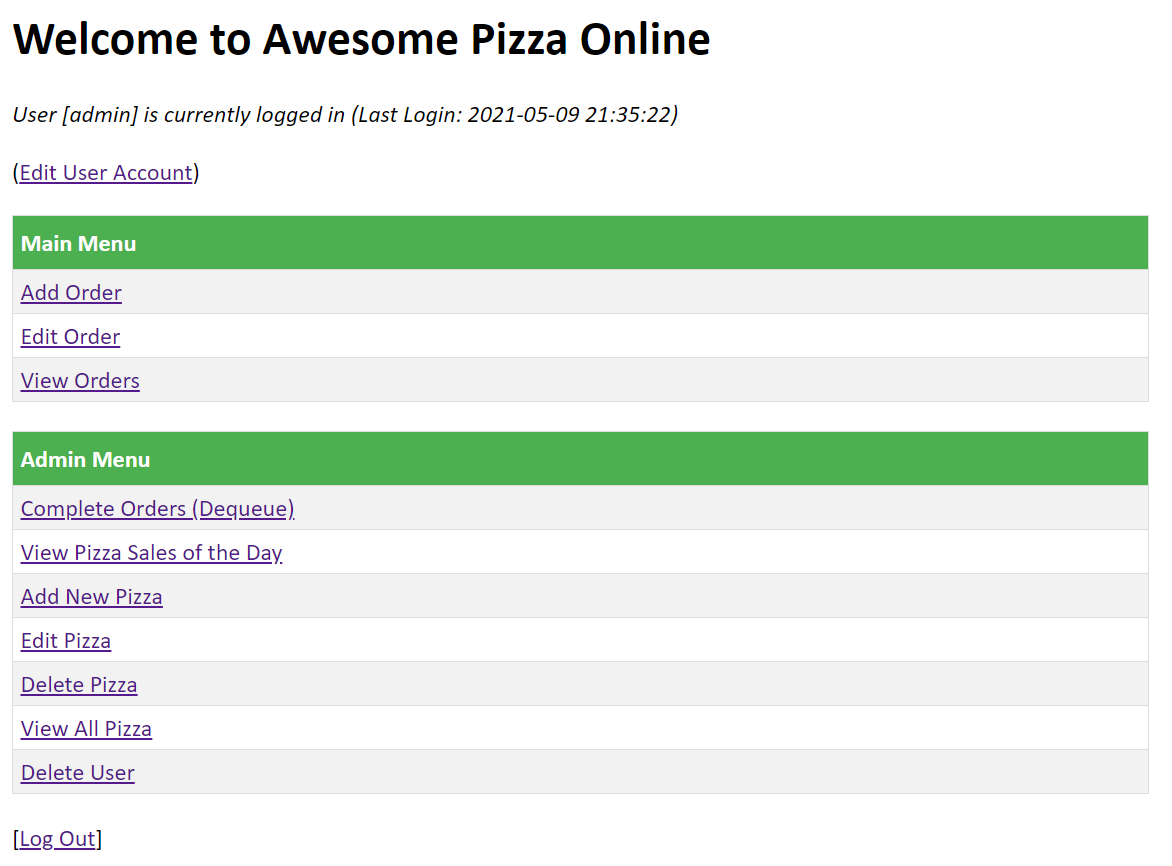


The last menu option for user is the [***View Orders***] hyperlink. This page will show the current orders as well as completed orders for the user (completed orders will be explained in admin user menu).

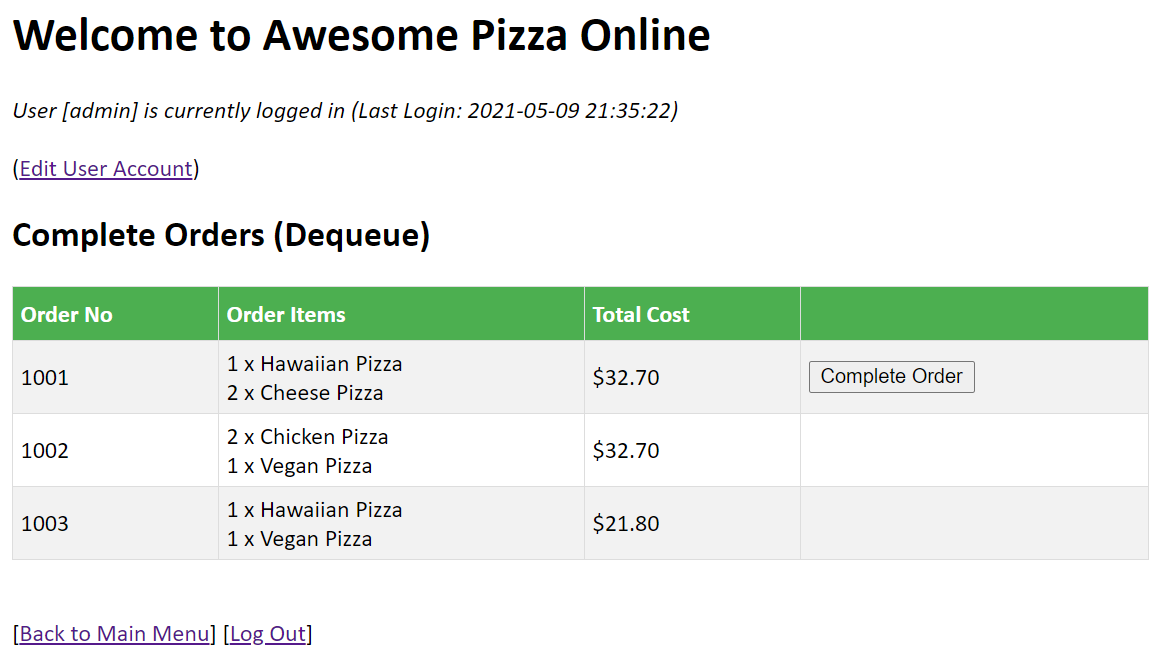


### ADMIN USER

The following shows menu options when the admin is logged in. Admin is able to edit and view orders of all users. Besides the Main Menu which is the same as what a typical user sees, the admin user will be able to see the Admin Menu as well.



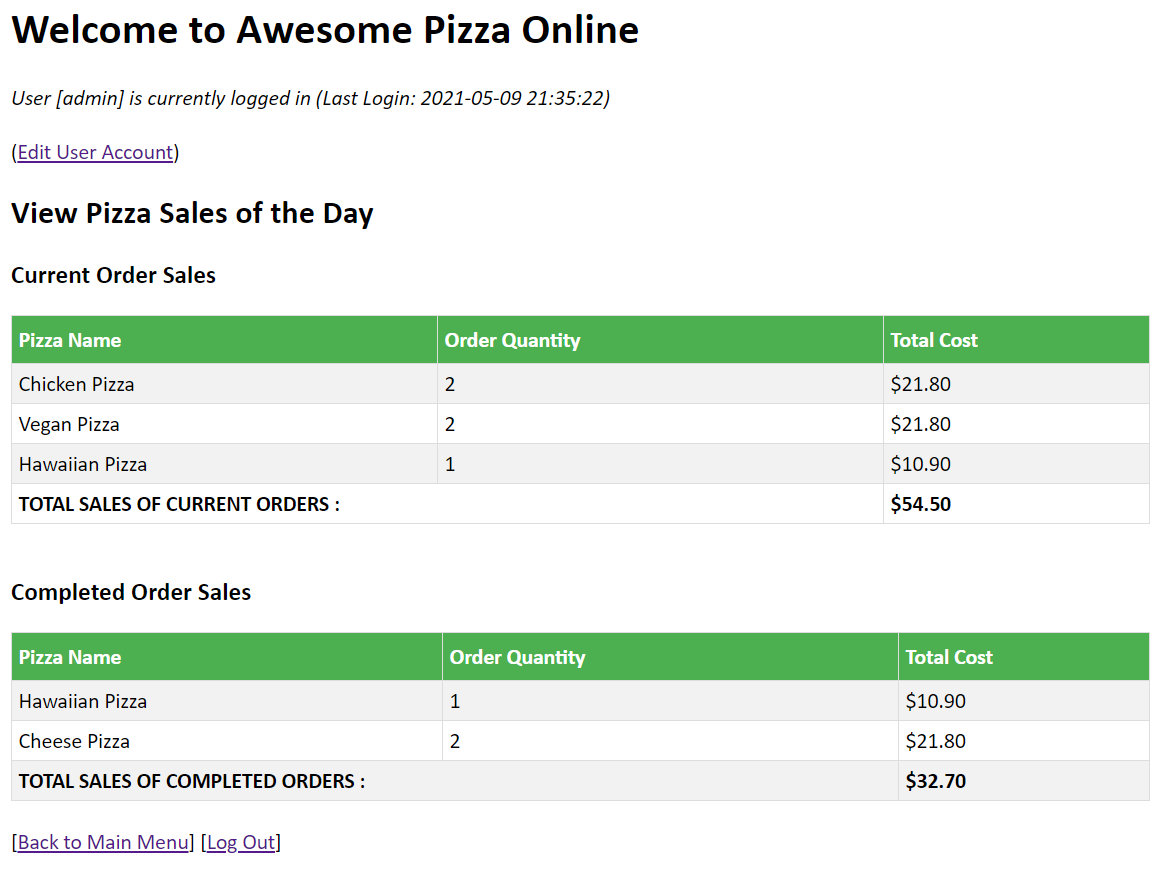
The [***Complete Orders (Dequeue)***] hyperlink allows the admin to remove an order from the queue using the Queue data structure. As mentioned in the introduction, a queue having a first-in-first-out concept will have the first order being dequeued one-by-one thus there is no need to enter the order no.



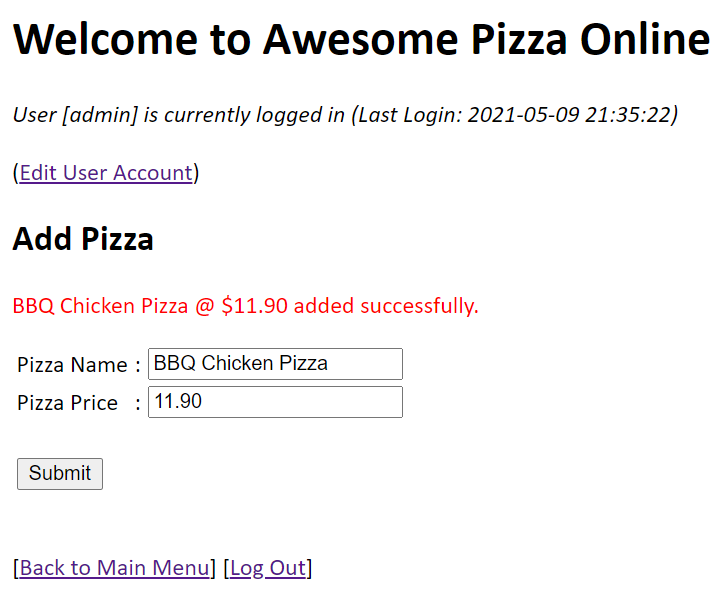
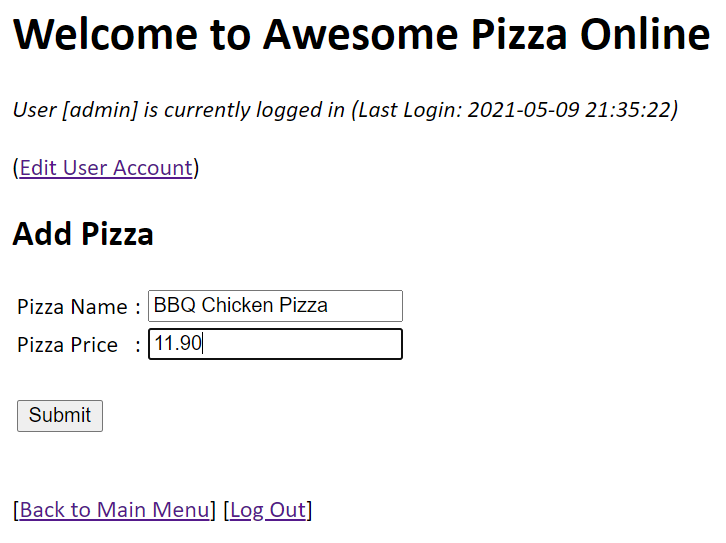
The [***Complete Order***] button will always be positioned at the next earliest order in the queue.



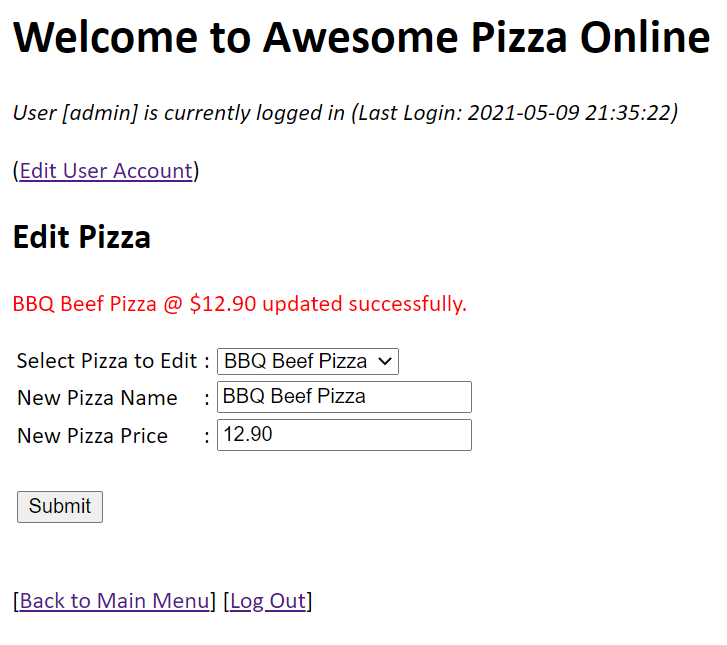
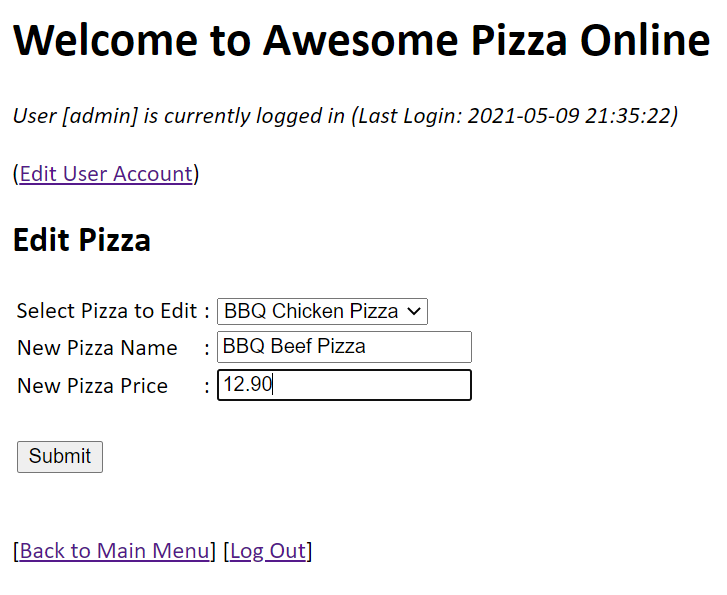
Once an order is dequeued, the pizzas ordered will be added to the pizza sales of the day. Thus in the [***View Pizza Sales of the Day***] hyperlink, the admin is able to view how many pizzas were purchased for the day and their total sales. In the page, the admin will be able to view pizzas that are currently in orders and those that were completed as well.



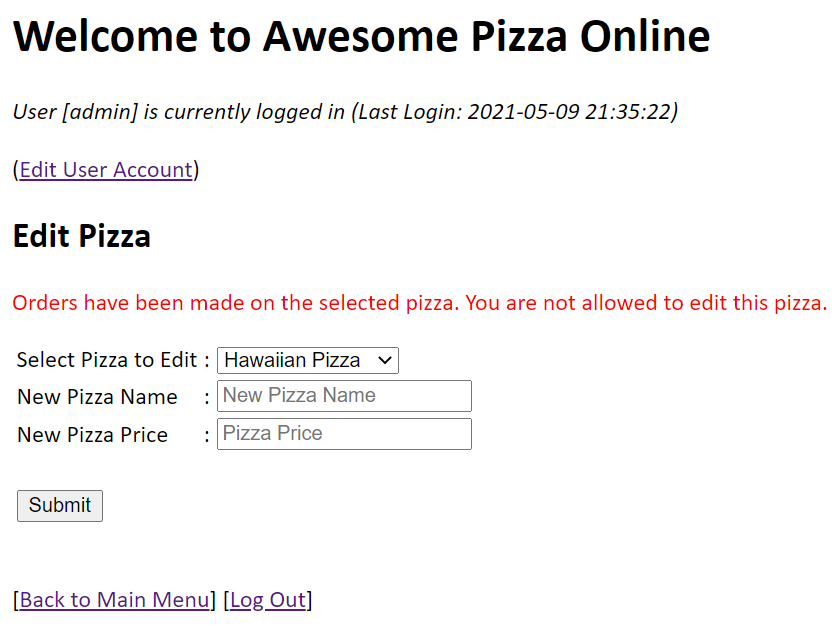
Next, the admin is able to add more pizza types to the menu through the [***Add Pizza***] hyperlink in the Admin Menu. During application runtime, I have chosen to initialize 5 standard pizzas to the menu. The standard price of each pizza is set at $10.90 in the .env file to facilitate the start of the application.



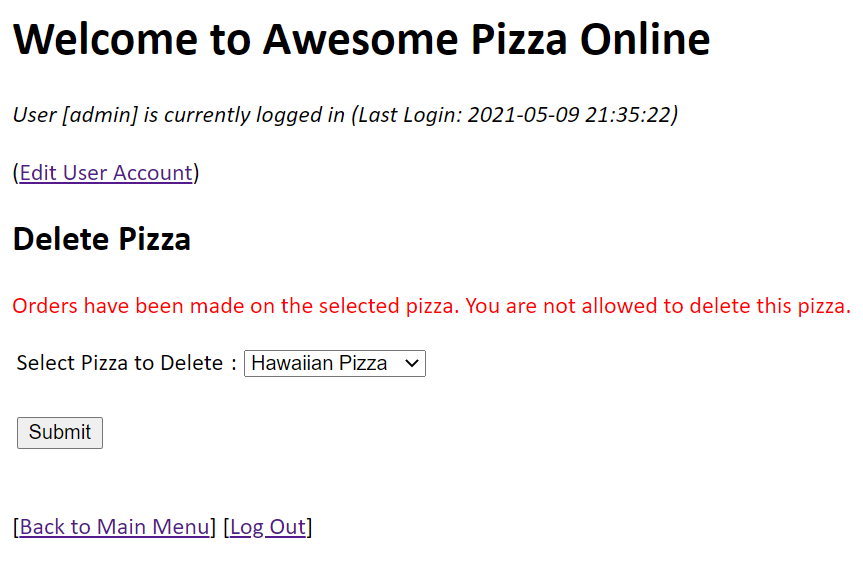
The [***Edit Pizza***] hyperlink in the Admin Menu allows the admin to edit a pizza which can be selected in a dropdown list. Admin can choose to edit just the pizza name or pizza price as well as none.



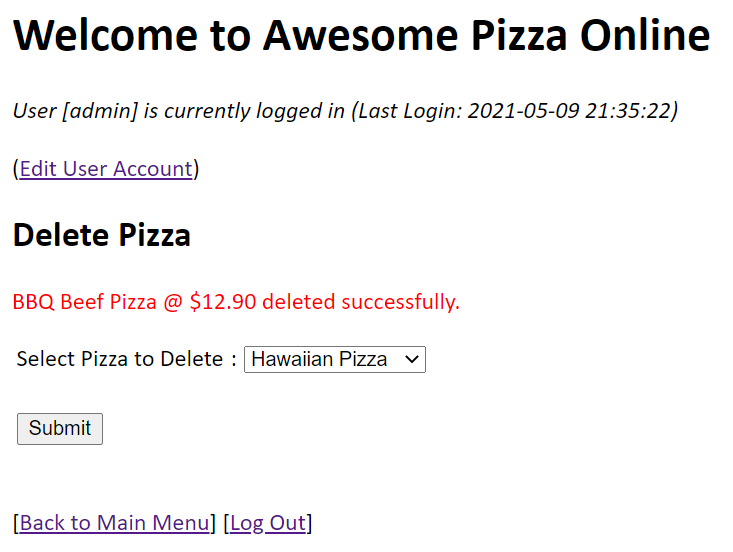
Note that if an order with the selected pizza has been created, admin is not allowed to edit this pizza.



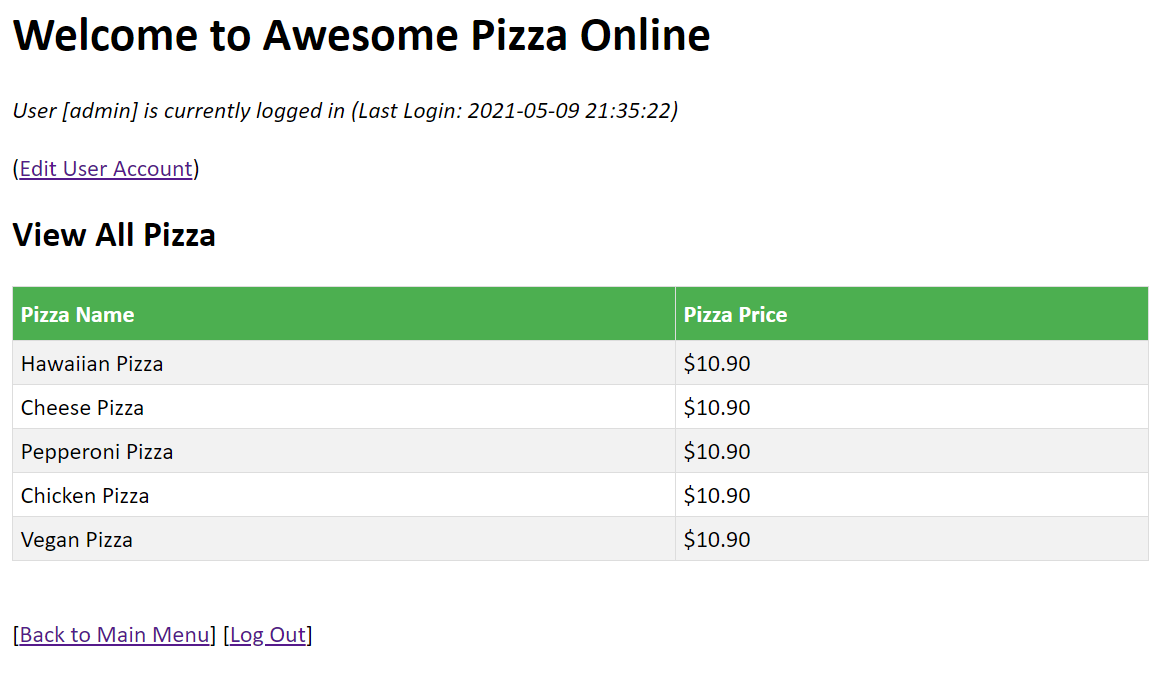
The [***Delete Pizza***] hyperlink in the Admin Menu allows the admin to delete a pizza. As with [***Edit Pizza***], the selected pizza cannot be deleted if an order exists.



Otherwise, a message will be displayed to indicate that the pizza has been deleted.



Next, admin can choose to view all pizza by clicking on the [***View All Pizza***] hyperlink in the Admin Menu.



Lastly, an admin can delete user account (except itself) by selecting the [***Delete User***] hyperlink in the Admin Menu.

