COMP 10222 - Emerging Web Technologies

Lab #5 - Payment Processing with Stripe

Due date: Sunday November 29th at 11:59pm

Worth: 14% of total grade

Marks: 100 marks total

Learning objectives

- Use a payment processing system to manage and charge users
- Use a template engine and sessions
- Create user and admin interfaces

Getting Started

You'll need to sign-up for a Stripe account. That will give you accesses to a dashboard where you can manually create things like plans, customers, and subscriptions. Use the Stripe documentation to figure out how to programmatically, using the Stripe API, create and manipulate customers, subscriptions, etc, as required for this

lab. In particular you may want to look at creating subscriptions and changing subscriptions.

Requirements

Create a subscription app using Stripe that allows customer users to create and manage subscriptions, and that allows an administrator user to manage subscriptions of all customer users.

Your application should have three monthly subscription plans:

- **Basic** - $10 per month
- **Plus** - $20 per month
- **Advanced** - $30 per month

The actual product or products that the users are subscribing to don't really matter... we can assume they are different levels of some Software-as-a-Service package (as is very common). But the plans themselves must exist in your Stripe account, and users must be able to create subscriptions to them that exist in your Stripe account. The currency can be either CDN or USA.

Application

When your application home page is loaded at the URL **localhost:3000/app** it should present the user with the option of either logging in or registering (provide the options as links). If they click on login, present the login page, and if they click on register, present the registration page.

Login Page

The login page should present the user with inputs for an e-mail address and password, as well as a submit button. If the user enters a correct e-mail address and password they should be logged into the application, otherwise the login page should be presented again with a suitable error message (e.g. "username and password not found").

Store all of the application users in either an SQLite database or a Redis database (don't worry about salting/encrypting passwords). Create a special **administrator** user with the e-mail address **admin@app.com** and the password **admin**. All other users should be considered **customers**.

Administrator user

When the administrator is logged in, they should be presented with an admin page that allows them to switch or cancel **any** customer's subscription (when a subscription is canceled, the customer should no longer be able to login, and the administrator should no longer see the customer). Implement this functionality by presenting a table of all customers containing all customer data, with clickable links in specific columns for switching and canceling a customer's subscription.

Customers

When a potential customer clicks on the register link, present them with a page that collects their full name, e-mail address, password, birth date and cell phone number. After clicking a submit button on this page, present the user with a new page that tells the user a 6-digit code will be arriving shortly at the cell phone number provided, and to enter the code when it has arrived (provide an input and submit button on this page). This will limit you to creating customers with the cell phone number you've approved for use

with Twilio, but that's OK for this lab. Use the Twilio API to send the customer a randomly generated 6-digit code (see: https://www.twilio.com/docs/sms/send-messages#send-an-sms-with-twilios-api). If the user enters this same code successfully, allow them to proceed to a subscription page, otherwise tell them the code is incorrect and re-load the page to have them enter it again.

At the next page, the customer should be allowed to select a subscription plan (basic, plus, or advanced) and to enter their payment information. If the customer enters any payment information incorrectly, allow them to keep trying until they succeed. After doing so successfully, the customer should be forwarded to the customer dashboard for logged-in customers. The customer's data should be stored in your database so that they may login to this page in the future from the login page.

The customer dashboard page should be provided at **localhost:3000/customer**, and should display all the information about that customer (full name, e-mail address, etc.). They should be able to edit all of this information except for their full name and e-mail address. The customer should also be able to switch their subscription (to one of the other two), or cancel their subscription (the customer should be logged out, forwarded to the application home page, and no longer be able to login after doing this). The customer should also have the ability to change their payment method (this action should only be allowed if they successfully input new valid payment information, i.e. there is no state in which a customer exists but that a valid payment method is not associated with them). Changes the customers make should be reflected in your database and your Stripe account.

A logged in customer should not be able to visit the login page, they should be re-directed back to the customer dashboard page.

Anyone that is not logged in should not be able to access the customer dashboard page, and they should be re-directed to the login page.

Altering subscriptions

When a subscription is canceled, either by the customer or the administrator, the customer should not be issued a refund for any remaining time during that billing period.

Style

Don't worry too much about styling, I won't be checking for this... it should be "clear, neat and readable", but beyond that, don't worry.

Authentication/authorization

Use Express Sessions to implement authentication and authorization.

Mustache

Use Mustache to present dynamic content on the pages wherever it is needed and/or possible (e.g. the admin page table should be populated using Mustache, the customer data on the customer dashboard should be populated using Mustache, error messages on login or register should be provided using Mustache, etc.).

Other notes

Don't worry about any additional error handling that isn't explicitly required as described above. You are free to use the Stripe documentation and Twilio documentation code examples more or less verbatim in your solution, but if you consult other sources, make sure to document them and do not copy code verbatim.

Submission instructions

Create a file called credentials.txt. Put your Stripe account e-mail address and password in this file, as well as your Twilio credentials, so I can access your Stripe and Twilio backend.

The main file of your server code should be called **server.js** and I should be able to run your program by running **node server.js**.

When you have completed the lab, put your source files in a folder called **lab5.zip** and submit it in the Dropbox.

Statement of Authorship

Copying others people work and claiming it as your own is a serious breach of ethics. If copied work is found, all parties involved will receive a failing grade as per departmental policy. Group work is encouraged but it is expected that you do your own work. If you do, you'll learn the material and feel better for it. Since all work submitted is assumed to be your own original work, you must include the following "Statement of Authorship" in every program file you submit for grading:

"StAuth10065: I John Doe, 123456 certify that this material is my original work. No other person's work has been used without due acknowledgement. I have not made my work available to anyone else."

- Use the exact text above. No substitutions.

- Replace John Doe with your name and the number 123456 with your student ID.
- Place this text as one line as close to the top of each document as possible.
- Include StAuth10065: at the beginning of each statement.
- Failure to include this statement will cause your work to be ineligible for grading.

Marking scheme

Stripe API	25
Twilio	10
Mustache	15
Login/Register	10
Customer	20
Admin	20
Total	/100

Labs submitted late will receive a 15% penalty per day. Labs more than 2 days late will receive a grade of zero. For example, a lab submitted at 11:01pm on the due date that was marked 80/100 would receive a mark of 80 - 15 = 65/100. A lab submitted at 11:01pm the day after the due date that was marked 65/100 would

receive a mark of 65 - 30 = 35/100. A lab submitted at 11:01pm two days after the due date would receive a mark of zero.