CSCI 130 Web Dev – Group Project Outline

Arlene Cazarez

Kenneth Willeford

Jacob Rahcal

Davis Hang

[**https://github.com/SSBMFreak/WebDev-Team-Project**](https://github.com/SSBMFreak/WebDev-Team-Project)

[**Group Project - Step 1 - Present**](https://blackboard.learn.fresnostate.edu/webapps/assignment/uploadAssignment?content_id=_1156298_1&course_id=_9270_1&assign_group_id=&mode=view)

Present your team's front-end. The front-end must use an "above the fold" design. You must use flexbox to arrange content. Include media-queries in your design so that your design looks good in any viewport (browser window; screen size). The names of the members of your team should be listed somewhere on your webpage(s). Your team will field up to 10 questions about the code. You will round-robin between team members to answer the questions. If the team-member whose turn it is to answer the question cannot answer the question, the entire team loses 20% (eg, even if one team-member wrote all of the code, everybody on the team needs to understand the code and be able to explain it). Reference the repo linked here for a code sample. Bonus: earn 20% extra for your team by including at least one inline SVG that is styled with CSS (you can find code samples for SVG in my html-css repo also).

Group Project - Step 1 - RawGit

Serve your page using Rawgit https://rawgit.com/

<https://rawgit.com/SSBMFreak/WebDev-Team-Project/master/index.html>

Group Project - Step 1 - ATF

Above the fold (ATF) design

-The navigation bar(including the nav drop-down menu), the "TOP" button, anything in the #atf container, and the background.

Group Project - Step 1 - Flexbox

Using flexbox

- We applied the flex-box format to every object above-the-fold and below-the-fold. Ex: the nav bar, the contents, the footer.

Group Project - Step 1 - Media Queries

Using media-queries

-Media queries handle different style rules for different media types/devices.

-We use @media to add breakpoint at 400px.

- If the screen width is less than 400px, the sidebars and footer image are hidden.

Group Project - Step 1 - Team Member Names

Included team member names

Arlene Cazarez

Kenneth Willeford

Jacob Rahcal

Davis Hang

Extra Credit:

Use at least one inline SVG that is styled with CSS

- We used it on our 'Top Button' image; inside the anchor <a> tag with the id name 'to-the-top' in the index.html file and '#to-the-top' in our style.css file. The SVG specify it's shape, size, and the text that's inside of it.

**Group Project - Step 2 – Present**

**\*\*\*To be handle by Arlene, David, and Jacob over the weekend\*\*\***

**\*\*\*otherwise Kenneth will get a frown face and take care of it\*\*\***

Present your team's server side code. The server side code should serve your pages and all resources. Your website must be accessible via the web (not just localhost). Create a *session* for every user that comes to your website: you may use app engine's user services for this (appengine/user) or a *cookie with a uuid*. Your webpages should visually show that state is maintained between each request from every unique user having a session. Your team will field up to 10 questions about the code. You will round-robin between team members to answer the questions. If the team-member whose turn it is to answer the question cannot answer the question, the entire team loses 20% (eg, even if one team-member wrote all of the code, everybody on the team needs to understand the code and be able to explain it). Bonus: earn 20% extra for your team by (1) passing some value through the URL and then retrieving it and (2) explaining on a page in your website how this could be helpful in maintaining state.

\*\*\*Instructions\*\*\*

|  |
| --- |
| EXAMPLES |
|  | Step 2 serving and handling |
|  | https://github.com/SSBMFreak/WebDev/blob/master/FirstHostedWebsite/main.go |
|  |  |
|  | Step 2 Cookies and State |
|  | https://github.com/SSBMFreak/WebDev/blob/master/Exercises2/uuid.go |
|  |  |
|  |  |
|  | Instructions- |
|  | Handle the CSS folder |
|  | Have two page handlers |
|  | "/" |
|  | This page executes a template "index.html" with a nil object |
|  | After executing the template if a cookie session-info exists it will print the uuid after executing the template |
|  | fmt.Fprintf(res,cookie.uuid) |
|  | "/login/" |
|  | This page will check if a cookie session-info exists |
|  | If so it will do nothing. |
|  | Otherwise it will make the cookie session-info. |
|  | Despite the name we wont be executing "login.html" |
|  |  |
|  | Regardless of what it does, it will redirect to "/" |
|  | afterwards. |

Group Project - Step 2 – Serve

Serve all pages

-what we talked about:

-example function: func ListenAndServe(addr string, handler Handler) error

-serve the following pages

“/” => index.html

“/login/” => login.html

“/register/” => register.html

-serve the following files

style.css

logo.png

footer.png

background.png

/login/’s handler will construct an arbitrary cookie with some sort of UUID.

/’s handler will execute the template, and then at the bottom of the page, add the UUID value.

(fmt.Fprintf(res,cookie.uuid))

-basically alot of handle func, handle files

\*\*Look at Kenneth’s FirstHostedWebsite/main.go

<https://github.com/SSBMFreak/WebDev/blob/master/FirstHostedWebsite/main.go>

Group Project - Step 2 - Serves All Resources

Serves All Resources

-what we talked about

-handling the css, png files

Group Project - Step 2 - Via Web

Accessible via the web

-what we talked about:

-func ServeFile(w ResponseWriter, r \*Request, name string)

Group Project - Step 2 - Session

Creates a session

-what we talked about:

-creating a cooking with a UUID

-don’t make a new cooking if one already exist

Group Project - Step 2 - Maintains State

Visually showing state is maintained

-will demonstrate

**Group Project - Step 3 – Present**

**\*\*\*To be handle by Kenneth\*\*\***

Present your team's server side code. The server side code should allow a user to "sign-up" for an account on your website. User names must be unique within your website. Use AJAX to let the user know if a user-name is already taken. Store user information in memcache and datastore. When you retrieve user information, try to retrieve it from memcache first. If it's not in memcache, retrieve it from datastore and also then store it in memcache. Allow a user to update their information. Your team will field up to 10 questions about the code. You will round-robin between team members to answer the questions. If the team-member whose turn it is to answer the question cannot answer the question, the entire team loses 20% (eg, even if one team-member wrote all of the code, everybody on the team needs to understand the code and be able to explain it). Bonus: earn 10% extra for your team by validating email addresses on the client-side using HTML. bonus.Bonus: earn 20% more extra and then also validating the email address on the server-side.

**Group Project - Step 3 - Signup**

user can signup

-authentication

-look at func register in main.go

**Group Project - Step 3 - User Names Unique**

user names are unique

-uuid

-look at func registerRequest in API.go

**Group Project - Step 3 - AJAX**

uses AJAX to notify a user if a user name is taken

**AJAX** (Asynchronous JavaScript and XML)

-use of XMLHTTPRequest object to communicate with server-side scripts

-make requests to server w/o reloading page

-receive and work w/ data from the server

-if you look at the script under the login.html file, the userCheck function uses AJAX to check if the username exist or not; it interact with userNameExists in the API.go file.

**Group Project - Step 3 - Datastore**

user info is stored in datastore

**datastore**

-our primary database in app engine

-nosql database

-less flexible

-not as powerful in what it can do with queries

-but if you don’t need that power, it’s a good solution

\*\*Datastore Viewer

**Group Project - Step 3 - Memcache**

user info is stored in memcache

**memcache** is used for storing data, it is much quicker to retrieve data from memcache than say datastore; however memcache is volatile (unpredictable); if that data is not in memcache then you get it from the datastore and save it back to memcache.

-a big distributed map, across all of your machine

-very fast, storing things in memory

-improve data retrieval

-cache database queries in memcache

-use memcache for sessions

-instead of storing user data in cookie, store it in memcache

**Group Project - Step 3 - Mem & DS**

retrieves user info from memcache first, then datastore and puts into memcache

-look at dataStoreMemcache.go

-JSON (JavaScript Object Notation)

-Marshalling- converting custom object to a representation accepted by the client

-Unmarshalling- creation of custom objects from XML, JSON and the like

**Group Project - Step 3 - Update**

User can update their info

-look at func edituser in main.go

Side note\*\*

A context carries a deadline, a cancellation signal, and other values across API boundaries.

Every incoming context to a server should create a Context

**Group Project - Step 4 - Present**

Present your team's server side code. The server side code should allow a user to upload files to google cloud storage (GCS). The user should also be able to see all of their files by name, they should be able to see their file contents in a browser, and they should also be able to download their files. You should limit the types of files that a user can upload. Your team will field up to 10 questions about the code. You will round-robin between team members to answer the questions. If the team-member whose turn it is to answer the question cannot answer the question, the entire team loses 20% (eg, even if one team-member wrote all of the code, everybody on the team needs to understand the code and be able to explain it). Bonus: earn 5% extra for your team by providing some type of processing on text files. bonus.Bonus: earn 20% extra for your team by letting a user know if another user has stored the exact same file. bonus.BonusMax: earn 25% extra for your team by processing images to be two different sizes.

Group Project - Step 4 - GCS

user can upload files to GCS

-Once logged in on website, under ‘Edit info’, under ‘Manage Images’ at bottom of Edit info page, user can choose file (jpg or png) to upload.

-look at Step 5 dataStoreMemcache.go

-type Blogdata struct

Group Project - Step 4 - List Files

user can see all of their files by file name

-Manage Images

Group Project - Step 4 - Display File Contents

user can see file contents in browser

-Blog post

Group Project - Step 4 - Download

user can download files

-from Manage Images

Group Project - Step 4 - File Filter

file types to upload are limited

-png and jpg

Bonus: earn 20% extra for your team by letting a user know if another user has stored the exact same file.

-look at func createFileGCS in Step 5, cloud.go

**Group Project - Step 5 - Present**

Present your server side code. Integrate some external service into your website, eg, like the giphy example, or github's markdown to html api, or something else. Your team will field up to 10 questions about the code. You will round-robin between team members to answer the questions. If the team-member whose turn it is to answer the question cannot answer the question, the entire team loses 20% (eg, even if one team-member wrote all of the code, everybody on the team needs to understand the code and be able to explain it). Bonus: earn 50% extra for your team by bringing some new piece of Go code to share with the class which is integrated into your website.

Group Project - Step 5 - API

external service integrated