

Astara

Realizado por: Juan David Ramírez Bernal

Curso académico: 2020/21
Grado Superior de Desarrollo de Aplicaciones Webs.

Índice

Introducción

Astara pretende ser una aplicación web en la que los usuarios puedan controlar sus tareas y objetivos de una forma sencilla y clara, de esa forma incrementar el rendimiento de dichas personas.

Mi objetivo con Astara ha sido crear una web a la que posteriormente le siga dando uso y siga desarrollando, es por esto que decidí no hacer una tienda web, o similares, ya que pensaba que no le iba dar uso más allá de la entrega de este proyecto.

Astara está compuesta ,en términos generales, por dos partes, una API, que sería el back-end del proyecto, la cual está desarrollando en Golang (Go), elegido este lenguaje por su eficiencia y rapidez, así como su fácil aprendizaje. La segunda parte, es un front-end desarrollado en Vue.js, una librería de JavaScript que nos permite hacer un front-end ligero y versátil.

Back-End

Como ya he mencionado anteriormente, la parte Back-End del proyecto consiste en una API desarrollada en Golang, el cual se apoya en el cliente http "Fiber" para poder controlar las peticiones que le lleguen.

Podemos encontrar numerosas webs que comparan a Fiber con Express, una librería que nos permite controlar peticiones http cuando el Back-End está desarrollado en Node.js, y es que Fiber intenta imitar a Express, con una sintaxis y metodología similar.

Con Fiber podemos aprovechar la concurrencia que nos Brinda Golang, pudiendo así responder a varias peticiones http al mismo tiempo.

Gracias a su rapidez y sencillez todos esto Golang ha ido adquiriendo mucha popularidad estos últimos años, creando así una gran comunidad de desarrollo, la cual fomenta buenas prácticas y metodologías como veremos más adelante.

Una vez instalado y configurado Golang, empecé creando el proyecto y dividiéndolo en subpartes, de esta forma, en el proyecto se encuentran carpetas dedicadas para los controladores y módulos, así como funcionalidades comunes para todos estos.

La carpeta que contiene ficheros comunes, está dividida a su vez por otras carpetas, y es que Golang nos obliga a realizar esta práctica, a la hora de declarar nuevos paquetes, y de esta forma mantener el código más organizado.

Es importante denotar que dentro de cada fichero, las variables, funciones y tipos que empiezan por letra mayúscula indican que son "exportadas" fuera de su paquete, es decir, son visibles para el resto de archivos fuera de su paquete, y las que empiezan por minúscula, solo son visibles a nivel del propio paquete.

Para la implementación de las sesiones he utilizado JWT (JSON Web Tokens). Paso ahora a hacer una breve descripción de cómo lo hago.

Cuando un usuario se intenta autenticar ante la aplicación , lo primero que hará será escribir su nombre en el login, ya que nombre y correo electrónico son únicos y con ellos se puede identificar al usuario.

La API realizará una comprobación para ver si el usuario existe o no. Si el usuario no existe mandará un error y si sí existe pasará al siguiente paso, la introducción de la contraseña por parte del usuario. Una vez introducida, se recoge la contraseña de la base de datos y se comprueba que ambas sean iguales. Si son distintas se manda un error y si son iguales, se crea un JWT con información del usuario, exactamente se guarda el Id del usuario y el rol que tiene, además de una fecha de caducidad, que será igual a la actual más diez minutos.

Con esta información se genera una Cookie y se manda al navegador, esta cookie es HttpOnly, lo que quiere decir que el usuario no tendrá acceso a ella, además de establecerse que es de SameSite, por lo que solo será útil en el mismo dominio.

Con cada petición realizada a la API, se le envía la cookie en la cabecera, así que antes de trabajar la petición se realiza una validación de los valores de la cookie. a través de un middleware. En caso de que la cookie haya sido corrompida por el usuario, se le devuelve a este un estatus 401, y se le redirige al login. Esto es gracias, en parte, al interceptor que hay en el Front-End, que pasaré a explicar más adelante.

Si la cookie es correcta, se le añaden diez minutos más a su fecha de expiración, así el usuario podrá seguir navegando por el sitio web.

De esta forma, logro no guardar absolutamente nada en la base de datos que relaciones al usuario con su estado, siendo así la API, complemente Stateless.

Si un usuario malintencionado captura una cookie con el token de un usuario, tendrá escasos diez minutos para poder descifrar su contenido, para lo cual deberá de encontrar la “private key” que guarda el servidor, lo cual no es una tarea facil.

Por otro lado, la conexión de la base de datos está dividida en tres sub conexiones, cada una de ellas es un usuario distinto de la base de datos, cada uno con privilegios distintos, de esa forma añado una capa más de seguridad al sitio web.

Estas tres sub conexiones son , la primera la “Non user”, la cual se utiliza para comprobar las credenciales de los usuarios al registrarse, haría las veces de lo que comúnmente se llama “un logger”. La segunda la de “User” la cual tendrá cada usuario que se loguea, y la tercera y última, la de “Admin”, la cual tienen más permisos, pero ahora mismo no se puede obtener dicha conexión, ya que mi sitio web no dispone de una parte de administración.

Son con estas tres y solo estas tres instancias, que la web funciona. Cuando un usuario se intenta loguear o intenta realizar una operación dentro de la web, pide la instancia de la base de datos correspondiente y si no existe aún crea una, que podrá ser usada posteriormente por el resto de usuarios que la necesiten, en esencia, la base de datos utiliza el patrón de diseño Singleton.

Debido al carácter concurrente de Golang, se podría dar el caso de que dos usuarios hayan pedido una instancia a la vez, y que esta no estuviese definida aún, por lo que ambos generarían una nueva instancia, pero solo se debe de almacenar una, lo cual es un problema. Para solucionar esta situación, he implementado un sistema de bloqueo, por el cual si un usuario requiere de la creación de una instancia, esta es bloqueada para el resto de usuarios y una vez esté creada, será desbloqueada y accesible para el resto de usuarios.

Me gustaría destacar también el uso de punteros y dereferencias en la obtención de objetivos y tareas, ya que fue un problema bastante tedioso.

En la web, se pueden crear objetivos los cuales tenga a su vez dentro de ellos otras tareas, es decir, un objetivo puede tener tareas anidadas dentro de él. En este caso solo permito una anidación al primer nivel.

Para explicar mejor esto, me gustaría decir que ambos, tareas y objetivos, vienen de un “elemento” superior, un padre, llamado “target”. De aquí tareas y objetivos reciben “contenido” común a ambos.

Dicho esto, al realizar una petición para recibir los objetivos de cierto usuario, le pido a la base de datos que me devuelva, no solo los objetivos, sino todos los targets, paginados, de ese usuario, para un área en específico.

Posteriormente paso a procesar todo el contenido devuelto. Para ello creo un mapa clave valor de los “targets”, el motivo de usar un mapa (hashmap) es que la complejidad de la inserción y búsqueda es $O(1)$. Si el “target” pertenece a otro, es decir, corresponde a la anidación de otro target, entonces, se le añade al array de anidados de ese “target”. Pero esto tiene un problema y es que Golang le da un valor nulo a ese array, y en el procesado del hashmap no se puede hacer un “push” al array, así que la solución viene del uso de punteros. De ese modo, si el array es nulo, en el caso de Golang, lo llama “nil”, creo un nuevo array y asocio el array de anidación del “target” con el nuevo creado, que contendrá el “target” hijo anidado.

Debido a este proceso, el mapa se desordena, dando lugar a cambios de sitio en la posición de las tareas y objetivos en el Front-end. Para solucionar este problema, luego de anidar los “targets” se ordenan por su identificador y finalmente se convierte en array y luego en string, para poder ser enviado en forma de JSON, al cliente.

Front-End

En el Front-End he usado Vue-cli y Vue-router, además de Axios, para realizar las peticiones al servidor.

Paso ahora a comentar un poco la estructura de las carpetas y su contenido.

El proyecto consta de tres “views” que podemos encontrar en su correspondiente carpeta, estas tres “views” son la “Landing” page, la “Home” page y la página de “Login”. Dentro cada una de ellas se encuentran anidados el resto de componentes que se encuentran dentro de la carpeta “components” y que están distribuidos por su “temática”.

También podremos encontrar la carpeta “assets/” donde se almacenan estilos comunes a los distintos componentes, la carpeta “auth/” que hace de auxiliar para la autenticación del usuario, la carpeta “js/” que contiene código JavaScript personalizado, y la carpeta “router/” donde se encuentra el router de la aplicación.

Dentro de la carpeta “auth/” se definen 4 cosas, una instancia de Axios, la cual es importada por el resto de componentes que la necesiten. El uso de esta instancia es sumamente importante, ya que de otro modo, cada vez que crease una instancia de Axios, debería darle una configuración, gracias a la instancia, puedo darle una configuración a ella y repartir la instancia a los componentes.

También se define un interceptor, el cual se ejecutará antes de procesar la respuesta mandada por el servidor, de este modo, si el servidor manda una respuesta con estatus 401 el interceptor la recibirá y mandará al usuario fuera de la aplicación, a la pantalla de logueo.

Junto a estas instancias se definen dos funciones, la primera “Validate” comprueba que las respuestas del servidor entran dentro del rango 200 y la segunda “AreaCorrespond” comprueba que el área al que está intentando entrar el usuario existe para ese usuario, de otro modo será llevado al Dashboard.

También realizo una validación de los inputs de los formularios, tanto en el front-end como en el back-end. En caso de que haya algún error, uso o bien un componente que me ayuda a mostrarlo o un mensaje personalizado.

Dentro de los componentes podemos encontrar los típicos datos, métodos, vistas condicionadas, listas, etc, además de propiedades computadas, enlazado de estilos condicional, manejo de eventos de padre a hijo y de hijo a padre, con emits, y refs, manejo de eventos, eventos personalizados, etc.

También he usado transiciones, no solo a través de estilos sino a través de javascript también.

Para todo ello me he apoyado en el uso de JQuery.

Desgraciadamente no he podido implementar JQueryUI ni animaciones con fade, ya que la indexación de Vue, lo hacía bastante complicado, eliminando varios elementos a la vez, etc.

También he usado SplittingJS, que es una librería de JS que permite dividir texto en diferentes secciones, líneas, palabras, caracteres, etc, y que me ha sido de gran utilidad a la hora de dar estilos a la web.

También he creado un shortcode, a través del cual se puede acceder al perfil de manera más rápida.

Además de hecho uso del two ways databinding que nos ofrece Vue, para hacer más interacciones con el usuario.

En la sección de tareas y objetivos, existe al final de cada listado un elemento observable, el cual me ayuda a realizar la paginación de estos elementos. Gracias al uso de varios intersection observers, cuando el elemento es visto, se realiza una petición al servidor que devuelve la siguiente tanda de resultados.

Esto es muy útil en la búsqueda de los elementos, ya que en la parte superior podemos encontrar una barra de búsqueda, esta está asociada a una propiedad computada del elemento, de modo que solo muestra los elementos cuyo nombre contiene la cadena que se busca, como consecuencia de esto, el elemento observable se vuelve visible ya que viene hacia arriba, de este modo, no se ha encontrado el resultado, se harán sucesivas peticiones al servidor hasta que este devuelva todos los elementos.

Estilos

Para los estilos he usado el plugin de Vue que permite el uso de Sass. En la carpeta “assets/” podemos encontrar estilos comunes a todos los componentes, así como las media queries.

Para la elección de la paleta de colores he elegido el amarillo como color principal he elegido el amarillo, color simbólico de las estrella, como referencia al nombre de la web. También he elegido una escala de grises, que me permitiese generar dos modos, uno claro y otro oscuro.

Siendo mi paleta una variante de esta otra

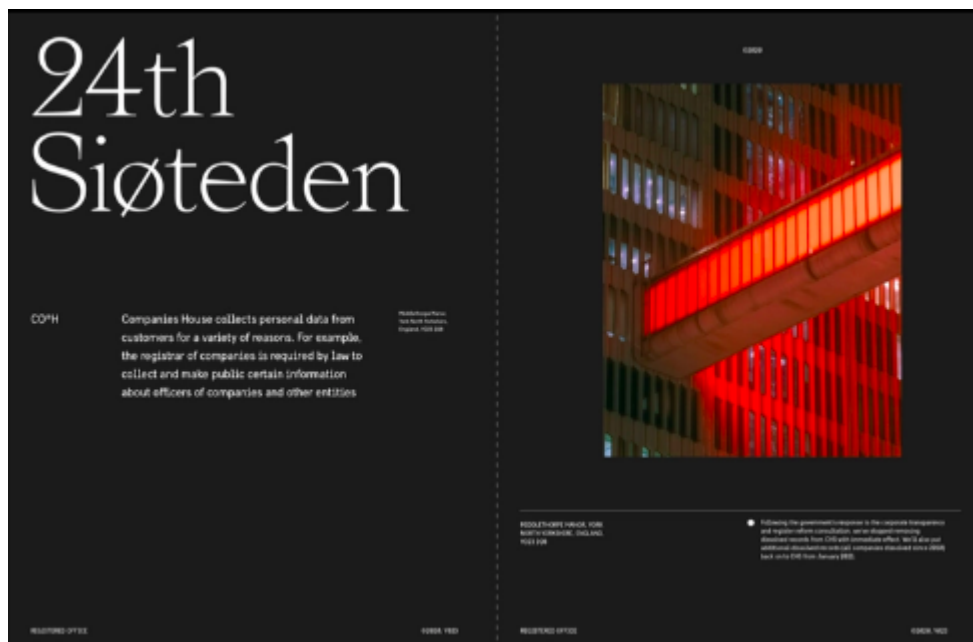


\$lightBlack = #353535

\$black = #242423

\$tertiary = #808080

\$white = #eff2f1



Para las fuentes he usado Lora, una fuente serif y la clásica Roboto, que es sans-serif, creo que ambas proporcionan un buen contraste a la vez de ser legibles. Para el título de las áreas, las descripciones de los objetivos y en los textos de la landing page, he usado Lora,

mientras que en el resto de títulos he usado Roboto, con más espaciado entre las líneas y a veces aumentando su grosor para darle más peso.

He exagerado los títulos de las áreas y sobretodo del dashboard, haciéndolos muy grandes, muy visibles, también he incrementado el tamaño de los nombres de las tareas y objetivos dándoles mayor peso visual.

Las imágenes de la landing tienen un efecto parallax, además de un un diseño “roto”.

El logo es un SVG hecho con Inkspace.

Despliegue

Compré el dominio astarapp.site y un hosting en DigitalOcean, asocie ambos y monté un servidor de aplicaciones Nginx, en una máquina ubuntu 20.02 LTS.


Dentro está montada la base de datos, la API y el frontend.

Para conectarme al servidor he usado SSH he usado la contraseña que me proporcionó DigitalOcean, que más tarde cambié y los archivos los he pasado de dos formas, a través con Git ejecutando el comando "git clone" a mi repositorio, y la siguiente forma es a través de FTP, para lo cual monté el servicio svftpd en el servidor e instalé Filezilla en mi ordenador personal.

Q Search by resource name or public IP (Ctrl+B)

Create ▾ ? 🔔

USAGE
\$1.43

 **Astara**
Web Application


→ Move Resources

Resources

Activity

Settings

DROPLETS (1)

 astara	138.68.135.5	go +1	ⓘ ⋮
---	--------------	-------	-----

DOMAINS (1)

astarapp.site	2 A / 3 NS / 1 SOA	⋮
----------------------	--------------------	---

IPv4: 138.68.135.5

juandavid@jdL: ~

juandavid@jdL:~\$ ssh root@138.68.135.5

NEW! Upgrade your Droplet for additional metrics and alerting

raphs

ccess

ower

olumes

esize

v4: 138.68.135.

raphs

ccess

ower

olumes

esize

etworking

ackups

apshots

ernel

istory

estroy

igs

roven

```
root@astara: ~  
root@astara:~# systemctl status nginx  
● nginx.service - A high performance web server and a reverse proxy server  
   Loaded: loaded (/lib/systemd/system/nginx.service; enabled; vendor preset: >  
   Active: active (running) since Tue 2021-06-15 18:19:34 UTC; 1 day 11h ago  
     Docs: man:nginx(8)  
  Main PID: 309575 (nginx)  
    Tasks: 2 (limit: 1136)  
   Memory: 3.7M  
   CGroup: /system.slice/nginx.service  
           └─309575 nginx: master process /usr/sbin/nginx -g daemon on; maste>  
             └─309576 nginx: worker process  
Bandwidth  
4.0 kb/s  
3.0 kb/s  
2.0 kb/s  
1.0 kb/s  
Jun 15 18:19:34 astara systemd[1]: Starting A high performance web server and a>  
Jun 15 18:19:34 astara systemd[1]: Started A high performance web server and a >  
root@astara:~#
```

```
server {  
    listen 80;  
    listen [::]:80;  
    server_name astarapp.site;  
  
    root /var/www/astara/dist;  
  
    index index.html index.htm index.nginx-debian.html;  
  
    error_page 404 /;  
  
    location / {  
        add_header Access-Control-Allow-Origin *;  
        add_header Access-Control-Allow-Credentials true always;  
    }  
  
    location /api/v1 {  
        proxy_pass http://138.68.135.5:3000;  
        proxy_set_header Host $host;  
        proxy_set_header X-Real-IP $remote_addr;  
        proxy_set_header Cookie $http_cookie;  
        proxy_set_header language es;  
        proxy_ignore_headers Set-Cookie;  
  
        add_header Access-Control-Allow-Origin *;  
        add_header Access-Control-Allow-Credentials true always;  
    }  
  
    location ~ /\.ht {  
        deny all;  
    }  
}
```

```
root@astara: ~  
root@astara:~# ls /var/www/astara/  
README.md babel.config.js dist node_modules package-lock.json package.json public src  
root@astara:~#
```

```
root@astara:/# ls ./astara/  
astara commons controllers go.mod go.sum main.go models  
root@astara:/#
```

sftp://root@138.68.135.5 - FileZilla

File Edit View Transfer Server Bookmarks Help

Host: sftp://138.68.135.5 Username: root Password: ***** Port: Quickconnect

Status: Retrieving directory listing...
Status: Listing directory /root
Status: Directory listing of "/root" successful

Local site: /home/juandavid/go/src/github.com/JuanDavid13/Astara/ Remote site: /root

Filename	Filesize	Filetype	Last modified
..			
.git		Directory	16/06/21 23:29...
api		Directory	16/06/21 11:30...
astara		Directory	16/06/21 09:47...
images		Directory	10/04/21 17:36...
.gitignore	271 B	File	13/04/21 20:57...
README.md	4,5 KB	md-file	10/04/21 17:36...
astara.sql	13,4 KB	sql-file	14/06/21 21:14...
package-lock.json	27 B	json-file	23/05/21 11:09...

4 files and 4 directories. Total size: 18,1 KB

Filename	Filesize	Filetype	Last modified	Permission	Owner/Group
..					
.cache		Directory	14/06/21 20...	drwx---	root root
.config		Directory	14/06/21 18...	drwx---	root root
.npm		Directory	14/06/21 18...	drwxr-xr-x	root root
.ssh		Directory	10/06/21 17...	drwx---	root root
go		Directory	14/06/21 20...	drwxr-xr-x	root root
snap		Directory	10/06/21 17...	drwxr-xr-x	root root
.bash_history	14,1 KB	File	15/06/21 22...	-rw----	root root
.bashrc	3,2 KB	File	14/06/21 19...	-rw-r--r	root root
.cloud-locales...	0 B	skip-file	10/06/21 18...	-rw-r--r	root root
.mysql_history	17,3 KB	File	17/06/21 08...	-rw----	root root
.profile	161 B	File	05/12/19 15...	-rw-r--r	root root

8 files and 6 directories. Total size: 129,2 MB

Server/Local file Directio Remote file Size Priority Status

