xd media

Realizado por Mario Davó e Ignacio Alavés

Índice

Introducción

El proyecto consiste en hacer nuestra propia versión de “X”, anteriormente conocida como Twitter, donde utilizando Golang y encriptando la información entre cliente y servidor según los métodos descritos en clase, los usuarios clientes podrán llevar a cabo publicaciones, dar “me gusta” a las publicaciones de otros usuarios, seguirlos y mantener conversaciones en privado.

Puesta en marcha

Para llevar a cabo la práctica, esta se ha llevado a cabo en Linux con nuestros portátiles personales.

Software necesario

Base de datos

- Docker
- Docker Compose

Backend y Logs

- Go

Frontend

- NodeJS
- npm/bun

Ejecución

Para iniciar la base de datos, sitúate en `/db`

```
docker-compose up -d
docker exec -i db psql -U admin -d xdmedia < ./seed.sql # Seedear la base de datos con datos de ejemplo

# Utilitarios
docker exec -it db psql -U admin -d xdmedia # Acceder al contenedor
```

Existe un Makefile en el directorio fuente, con él se puede poner en marcha el servidor de logs y el backend. Estos son los comandos del Makefile explicados. Ejecutar desde `/`

```
make logger     # Compila el sistema de logs
make logger_run # Compila y ejecuta el sistema de logs
make server     # Compila el servidor backend
make server_run # Compila y ejecuta el servidor backend
```

Para ejecutar el cliente, ir al directorio `/cliente`. Nosotros hemos utilizado BunJS.

```
bun i
bun dev
# ó
npm i
npm run dev
```

Funcionalidades

Arquitectura cliente/servidor

Servidor: Llevado a cabo en Go por Mario Davó. Cliente: Llevado a cabo en Typescript

Mecanismo de autentificación (gestión de contraseñas, identidades y sesión)

Se ha implementado la seguridad mediante TLS.

```go
func run() error {
  var err error
  addr := fmt.Sprintf(":%d", SERVER_PORT)

  db, err = connectDB()
  if err != nil {
    Error(err.Error())
    Error("Unsuccessful database connection attempt")
    os.Exit(1)
  }
  defer db.Close()

  mux := http.NewServeMux()
  addRoutes(mux)
  handler := corsMiddleware(mux)

  cfg := &tls.Config{
    MinVersion:               tls.VersionTLS13,
    CurvePreferences:         []tls.CurveID{tls.CurveP521, tls.CurveP384, tls.CurveP256},
    PreferServerCipherSuites: true,
    InsecureSkipVerify:       true,
    CipherSuites: []uint16{
      tls.TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,
      tls.TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,
      tls.TLS_RSA_WITH_AES_256_GCM_SHA384,
      tls.TLS_RSA_WITH_AES_256_CBC_SHA,
    },
  }
  srv := &http.Server{
    Addr:         addr,
    Handler:      handler,
    TLSConfig:    cfg,
    TLSNextProto: make(map[string]func(*http.Server, *tls.Conn, http.Handler), 0),
  }
  Info("listening on addr %s", addr)
  Fatal(srv.ListenAndServeTLS("crypto/localhost.pem", "crypto/localhost-key.pem").Error())
  return nil
}

func main() {
  if err := run(); err != nil {
    Fatal(err.Error())
  }
}
```

Usando la implementación de la libraría estándar (`HandleFunc`) ligamos las rutas a las funciones que las manejan a través de la función `authMiddleware()`

```go
func addRoutes(mux *http.ServeMux) {
  mux.HandleFunc("POST /signup", signupHandler)
  mux.HandleFunc("POST /signin", signinHandler)
  mux.HandleFunc("POST /users/{user_id}/follow", authMiddleware(userFollowHandler))
  mux.HandleFunc("DELETE /users/{user_id}/follow", authMiddleware(userUnfollowHandler))
  mux.HandleFunc("POST /posts", authMiddleware(createPostHandler))
  mux.HandleFunc("GET /posts", getPostsHandler)
  mux.HandleFunc("GET /users/{user_id}", getUserHandler)
  mux.HandleFunc("GET /posts/{post_id}", getPostHandler)
  mux.HandleFunc("POST /posts/{post_id}/like", authMiddleware(likePostHandler))
  mux.HandleFunc("DELETE /posts/{post_id}/like", authMiddleware(deleteLikePostHandler))
  mux.HandleFunc("GET /users/{user_id}/like", authMiddleware(userLikesHandler))
  mux.HandleFunc("POST /users/{user_id}/chat", authMiddleware(sendMessageHandler))
```

```
14    mux.HandleFunc("GET /users/{user_id}/chat", authMiddleware(retrieveMessagesHandler))
15  }
```

Para la autenticación empleamos tokens JWT y la librería Pinia junto con VueJS para crear *stores* donde guardar valores de forma persistente.

Transporte de red seguro entre cliente y servidor

Se ha implementado transporte seguro TLS entre cliente y servidor, además de HTTPS para la conexión con el cliente.

Almacenamiento seguro

cifrado en descanso

Sistema de gestión de contenido general

(público)El usuario puede ver posts de otros usuarios

Sistema de comunicación privado (cifrado) entre usuarios

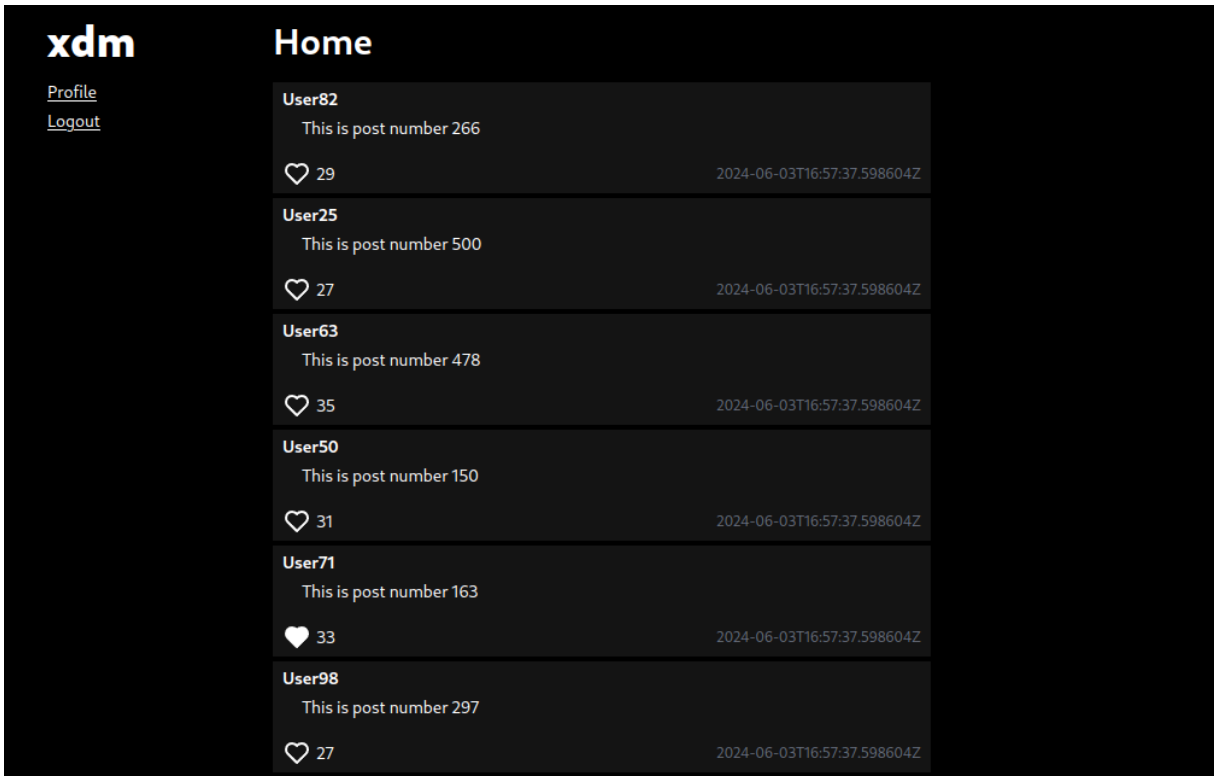
Se ha implementado un chat donde cada usuario posee su clave pública y clave privada.

(Opcional) Sistema de registro de eventos (logging), para mejorar la trazabilidad (remoto)

El backend hace logs tanto a su terminal como a un servidor remoto dedicado a logging.

```
1   func sendLog(message string) {
2     addr := fmt.Sprintf(":%d", LOGGER_PORT)
3     conn, err := net.Dial("tcp", addr)
4     if err != nil {
5       log.Println("Failed to connect to logger server:", err)
6       os.Exit(1)
7       return
8     }
9     defer conn.Close()
10    if _, err := conn.Write([]byte(message)); err != nil {
11      log.Println("Failed to send log message:", err)
12    }
13  }
```

Capturas


xdm
Profile
Logout
Home
User82
This is post number 266
29
2024-06-03T16:57:37.598604Z
User25
This is post number 500
27
2024-06-03T16:57:37.598604Z
User63
This is post number 478
35
2024-06-03T16:57:37.598604Z
User50
This is post number 150
31
2024-06-03T16:57:37.598604Z
User71
This is post number 163
33
2024-06-03T16:57:37.598604Z
User98
This is post number 297
27
2024-06-03T16:57:37.598604Z

Aspectos adicionales

Aspectos de diseño y seguridad que hayan quedado fuera de la implementación

Planificación

Cronología aproximada

- **Enero y Febrero**

Familiarización con Golang, proyectos y diseño del proyecto

- **Marzo**

Primera versión realizada del proyecto. Cambio a Docker y a Linux entre otros motivos para facilitar su ejecución/testeo mediante makefile.

- **Abril**

Segunda versión realizada, se cumplen los puntos obligatorios en el back-end. Experiencia del usuario a través de una TUI.

- **Mayo**

Se lleva a cabo la experiencia front-end a través de TypeScript. Se retocan los últimos cambios a la práctica y se trabaja en funcionalidades adicionales. Se comienza a llevar a cabo la memoria de la práctica.

- **Junio**

Toques finales a la práctica y se finaliza la memoria.

División del trabajo

Mario Davó
Encargado principal del apartado back-end (servidor)

Ignacio Alavés
Diseñador de la experiencia del usuario y encargado principal del front-end + apoyo al back-end.

Ambos nos apoyamos mutuamente para llevar a cabo las tareas, cogíamos lo que más le gustaba a cada uno, a Ignacio el diseño y la experiencia del usuario, mientras que Mario fue el aportador principal al apartado de back-end, apoyado periódicamente por Ignacio.

Expectativas de seguridad y privacidad

Límites de seguridad (en qué condiciones el proyecto es seguro y en cuáles no)

Condiciones seguras

- Concurrencia y Gestión de Memoria

Las goroutines reducen el riesgo de problemas con la gestión de memoria

- Manejo de errores

Manejo explícito de errores

- Lenguaje Tipado Estáticamente

Reduce errores de tipo y posibles vulnerabilidades a ser explotadas por atacantes

- Bibliotecas Estándar Seguras

Operaciones criptográficas, manejo de solicitudes HTTP, implementación de funcionalidades seguras sin depender de bibliotecas externas.

Condiciones no seguras

- Aspecto 1
- Aspecto 2

Conclusión

¿ Qué se ha realizado?

Fuimos capaces de llevar a cabo todos los apartados básicos del proyecto más algún que otro apartado opcional como el de logging. Aunque no puntuara para la nota además llevamos a cabo un apartado gráfico mediante TypeScript ya que facilitaría nuestro trabajo a la hora de testear el funcionamiento del código a lo largo de su implementación.

Aspectos positivos y relevantes del proyecto

- Hemos aprendido ampliamente las diversas tecnologías propuestas por la asignatura para asegurar la protección de los datos y la encriptación según explicado por el profesor y las clases de asignatura.
- El proyecto es funcional y cumple con los objetivos inicialmente establecidos
- Hemos aprendido a utilizar los algoritmos en un entorno práctico

Posibles trabajos futuros

Si hubiéramos tenido más tiempo para elaborar la práctica y profundizar el temario nos hubiera gustado llevar a cabo más de los puntos opcionales ya que se veían muy interesantes y hubieran ayudado a completar la funcionalidad general de la práctica.