Documentación técnica

Shared Server

Taller de programación II

1. Introducción

El shared server es un servidor encargado de alojar la información y lógica estable y/o común de los usuarios de diversas aplicaciones.
En líneas generales, funciona como una capa o interfaz para la base de datos SQL y proporciona comunicación con otras APIs (por ejemplo, la de pagos) además de centralizar información. Debido a la longitud de este trabajo, esto último no está explícito en la práctica.

2. Arquitectura y diseño

Este trabajo está estructurado básicamente en controladores, hay un controlador para cada set de endpoints y un controlador para cada componente/pestaña de la app web.

2.1. Principales tecnologías y paquetes utilizados

2.2. Dependencias funcionales

- Express para la creación de los endpoints y dar la estructura general a la aplicación.
- pg, pg-promise y pg-promise-simple para conexiones con la base de datos.
- request-promise y rxjs para la comunicación con apis externas.
- node-rules para el motor de reglas.

2.3. Dependencias no funcionales

- Mocha, chai, chai-http e istambul para testing.
- Winston y debug para el log.
- JSDoc para la documentación.

2.4. Tecnologías

- NodeJS y npm.
- Heroku.
- PostgresSQL.
- Angular.
- Docker.
- Travis y Coveralls para testing e integración contínua.

2.5. Simplificaciones

Dadas las dificultades surgidas de la utilización de NodeJs (principalmente, su asincronismo), se realizarón las siguientes simplificaciones al servidor:

- El error que se envía por no encontrar un dato con una cierta referencia es el mismo que se envía por no encontrarlo (404).
- EL flujo de obtener el precio de un viaje y luego pagarlo se encuentra del lado del app server en vez del shared server, pero ambos enpoints se encuentran en este segundo.
- Los datos referidos a los viajes se guardan en el app server.

La mayoría de estas simplificaciones podrían evitarse mediante el uso de *promises* (haciendo primero un pedido a la base y luego esperando su resultado), pero dada su complejidad y la falta de conocimientos previos, no fue posible.

2.6. Backend

El backend se divide en tres módulos:

- srv: se compone del log y el main del servidor.
- routes: se definen las rutas de los endpoints.
- controllers: este módulo contiene toda la funcionalidad del servidor.

2.6.1. Srv

Contiene sólo al log y al servidor. El log define los tres flujos de información (*info, warn y error*) y agrega una función básica que será llamada cada vez que ocurra un *request* desde http.
El main levanta el servidor en el puerto de entorno o en el 5000 por defecto, importa un router con las rutas de todos los endpoints y las agrega y por último define que todo el resto se corresponda con la app web. Además, agrega un parser para Json e inicializa al log.

2.6.2. Routes

Tiene a todas las rutas definidas por la API del servidor, para esto se definió un *router* del módulo *express* que se importa en todos los archivos agregándole los endpoints correspondientes.
Para modularizar el trabajo, se dividieron a los mismos en:

- BusinessUsers: servicio de gestión de usuarios de negocio.
- Rules: sistema de reglas.
- Servers: servicio de gestión de servidores, también contiene la lógica de tokens de los mismos.
- Transactions: servicio de transaciones:
- Trips: servicio de gestión de viajes.
- Users: servicio de gestión usuarios de los app servers.
- UsersCards: servicio de gestión de tarjetas de los usuarios de los app servers.
- UsersCars: servicio de gestión de autos de los usuarios de los app servers.

2.6.3. Controllers

A su vez, este módulo se divide en:

- controllerData: aquí se encuentra el controlladoras de la base de datos (que contiene dos funciones para hacer *queries* a postgres) y un parser por set de controllador-enpoint además de uno genérico usado por todo el resto.
- controllerLogic: contiene la lógica de seguridad que se necesita en todos los otros controladores. Se compone de los controladores de autorización, referencias y token.
- controllerRoutes: son las funciones que llama cada endpoint, coordinan el flujo entre la base de datos, las autorizaciónes, el parser y las llamadas http.

2.7. Frontend

El frontend se divide en:

- components: son los componentes de angular de la aplicación, definen los diferentes objetos con los que puede interactuar el usuario.
- guards: se encargan de "resguardar" la información a la que tiene acceso el usuario.
- services: son los servicios de la aplicación, sus conexiones externas a la API.

2.7.1. Components

Los componentes de este trabajo son:

- Dashboard: es el menú central para el usuario, se componen de la lista de reglas existentes. Permite abrir el componente de visualización o de creación.
- Edit-rule: es un menú que permite editar una regla del sistema.
- Home: es la pantalla de inicio de la aplicación, permite acceder al login.
- Navbar: es la barra de navegación, común a toda la app web. Contiene botones para cambiar de pantalla actual.
- Register-rule: permite crear una nueva regla.
- View-rule: permite visualizar la información completa de una dada regla.

2.7.2. Guards

Contiene un único guard que evitar que el usuario pueda volver a hacer un login una vez que ha ingresado en el sistema, que se haga un logout sin haber ingresa o que una persona externa acceda a las pantallas de visualización, creación y edición de reglas.

2.7.3. Services

Contiene un único servicio que se encarga de comunicarse con la API del servidor haciendo los pedidos pertinentes. También guarda la sesión una vez iniciada y borra los datos al hacer un logout.

3. Modelo de datos en postgres

Se cuentan con cuatro tablas distintas en la base de datos de postgres, las siguientes son:

- srvusers: contiene la información de todos los usuarios que utilizarán al servidor, esto es, los usuarios de negocio y los app servers.
- users: contiene la información de los usuarios de los app servers.
- trips: contiene la información de los viajes realizados por los usuarios de los app servers.
- rules: guarda las rules (junto con todas sus versiones anteriores) utilizadas para calcular el costo de un viaje.

3.1. srvusers

Los datos de esta tabla tienen la siguiente forma:

- id: numérico y único, es autoincrementado por postgres.
- _ref: una referencia del objeto, se utiliza para asegurarse que la versión que se modifica es siempre la más actual, texto.
- rol: define si el usuario es un usuario de negocios o un servidor, texto.
- token: token de autentificación del usuario, texto.
- tokenexp: es la fecha de caducidad del token, timestamp.
- data: contienen toda la información propia del usuario, diferenciada para usuario de negocio y servidor, jsonb.
  - usuario de negocio: contiene el nombre de usuario, contraseña, nombre, apellido y los roles del mismo. Todos los campos son texto salvo los roles que es un arreglo de strings.
  - servidor: contiene su fecha de creación, el nombre de su creador, su nombre y la fecha de su última conexión. Las fechas se guardan como timestamp, el resto, como texto.

3.2. users

Los datos de esta tabla tienen la siguiente forma:

- id: numérico y único, es autoincrementado por postgres.
- _ref: una referencia del objeto, se utiliza para asegurarse que la versión que se modifica es siempre la más actual, texto.
- driver: determina si el usuario es un pasajero o un conductor, texto.
- username: el nombre de usuario, texto.
- password: la contraseña, texto.
- facebookId: el identificador de facebook del usuario (el mail), texto.
- facebookToken: el token de acceso vía facebook del usuario, texto.
- firstname: el nombre del usuario, texto.
- lastname: el apellido del usuario, texto.
- country: el país dónde recide el usuario, texto.
- email: el email del usuario, texto.

- birthdate: la fecha de cumpleaños del usuario, timestamp.
- car: todos los datos del auto del usuario, jsonb.
  - brand: marca del auto, texto.
  - model: modelo del auto, texto.
  - color: color del auto, texto.
  - plate: patente del auto, texto.
  - year: año de fabricación del auto, texto.
  - status: descripción del estado del auto, texto.
  - radio: preferencia musical del conductor, texto.
  - airconditioner: define si el auto tienen o no aire acondicionado, boolean.
  - _ref: una referencia del objeto, se utiliza para asegurarse que la versión que se modifica es siempre la más actual, texto.
- card: todos los datos de la tarjeta del usuario, jsonb.
  - ccvv: el número de seguridad de la tarjeta, texto.
  - expirationmonth: el número de mes cuando expira la tarjeta, texto.
  - expirationyear: el número de año cuando expira la tarjeta, texto.
  - method: por defecto es siempre çard", valor que requiere la api de pagos, texto.
  - number: número de tarjeta, texto.
  - type: tipo de tarjeta, texto.
- balance: el balance actual del usuario, sólo se considera lo que le falta pagar, integer.

3.3. trips

Los datos de esta tabla tienen la siguiente forma:

- driver: id del driver, texto.
- passenger: id del pasajero, texto.
- start y stop: inicio y fin del recorrido. Ambos son un jsonb con clave "address" que contiene otro json con los datos:
  - street: calle y altura, texto.
  - localización:
    - lat: latitud, número.
    - lon: longitud, número.

3.4. rules

Los datos de esta tabla tienen la siguiente forma:

- id: numérico y único, es autoincrementado por postgres.
- _ref: una referencia del objeto, se utiliza para asegurarse que la versión que se modifica es siempre la más actual, texto.
- commits: información de todos los cambios a la regla, incluida la versión actual, jsonb.
  - _ref: es la referencia al momento de realizar el commit, se utiliza como id del mismo, texto.
  - message: un mensaje sobre el cambio, texto.
  - blob: el cuerpo de la regla a ejecutar, texto.
  - timestamp: la fecha del cambio, timestamp.
- active: determina si la regla está o no activa, booleano.