UNIVERSIDAD DE LAS FUERZAS ARMADAS – ESPE

SESSIONS EN TRACCAR

Tutor: Ing. Wilson Ramirez

Fecha: 03/09/2024

NRC: 16837

2024 - 2025

Resumen:

Este documento detalla un procedimiento sistemático para implementar y configurar Traccar, un sistema de rastreo GPS, utilizando las herramientas de Docker y Docker Compose. La implementación de estos contenedores permite un entorno aislado y reproducible, ideal para gestionar y desplegar aplicaciones de manera eficiente. Antes de iniciar, es esencial tener Docker y Docker Compose instalados en el sistema, ya que estas herramientas son fundamentales para el proceso de despliegue descrito.

El primer paso en este procedimiento es realizar un fork del repositorio oficial de Traccar que contiene la configuración de Docker y los archivos necesarios para el despliegue. Este paso es crucial porque permite a los usuarios tener su propia copia del proyecto, lo que facilita la personalización y actualización del sistema según las necesidades específicas. Al trabajar con una copia del repositorio, los cambios y configuraciones pueden realizarse de manera segura sin afectar el repositorio original, lo que es fundamental para la integridad del proyecto.

Una vez realizado el fork, el siguiente paso es crear dos redes Docker separadas para el backend y el frontend de Traccar. Estas redes específicas se configuran utilizando el comando `docker network create`, que permite definir una red con un driver de tipo bridge y un rango de direcciones IP asignado. La separación en redes independientes para el backend y el frontend no solo mejora la organización del sistema, sino que también refuerza la seguridad al limitar la comunicación entre los componentes a través de rutas controladas.

El tercer paso en el proceso es la creación de contenedores utilizando Docker Compose. Cada componente del sistema Traccar, incluyendo la base de datos, el servidor y la interfaz web, tiene su propio archivo `docker-compose.yaml`, que define la configuración y dependencias

necesarias. Docker Compose facilita el levantamiento de estos contenedores, permitiendo que múltiples servicios se desplieguen de manera coordinada. El orden de construcción de los contenedores es crítico: primero la base de datos, luego el servidor y finalmente la interfaz web, para asegurar que todos los componentes estén correctamente sincronizados.

Este enfoque estructurado y modular para la implementación de Traccar con Docker y Docker Compose proporciona una solución flexible y escalable para gestionar el sistema de rastreo GPS. La capacidad de personalizar y desplegar el sistema en diferentes entornos con un control detallado sobre cada componente es una de las principales ventajas de este método. Además, al utilizar contenedores, se garantiza que el sistema pueda ser replicado y mantenido con facilidad, lo que es esencial para su operatividad a largo plazo.

Index words: Docker, Docker Compose, Traccar, rastreo GPS, backend, frontend, configuración de redes, contenedorización, despliegue del sistema.

Abstract

This document details a systematic procedure for deploying and configuring Traccar, a GPS tracking system, using Docker and Docker Compose tools. Deploying these containers allows for an isolated and reproducible environment, ideal for managing and deploying applications efficiently. Before starting, it is essential to have Docker and Docker Compose installed on the system, as these tools are critical to the deployment process described.

The first step in this procedure is to fork the official Traccar repository containing the Docker configuration and files required for deployment. This step is crucial because it allows users to have their own copy of the project, making it easy to customize and update the system to specific needs. By working with a copy of the repository, changes and configurations can be made safely without affecting the original repository, which is critical for the integrity of the project.

Once the fork is done, the next step is to create two separate Docker networks for the Traccar backend and frontend. These specific networks are set up using the `docker network create` command, which allows you to define a network with a bridge driver and an assigned IP address range. Separating the backend and frontend into separate networks not only improves system organization, but also strengthens security by limiting communication between components through controlled paths.

The third step in the process is the creation of containers using Docker Compose. Each component of the Traccar system, including the database, server, and web interface, has its own `docker-compose.yaml` file, which defines the necessary configuration and dependencies. Docker Compose makes it easy to stand up these containers, allowing multiple services to be deployed in a coordinated manner. The order of building the containers is critical: first the database, then the server, and finally the web interface, to ensure that all components are properly synchronized.

This structured and modular approach to deploying Traccar with Docker and Docker Compose provides a flexible and scalable solution for managing the GPS tracking system. The ability to customize and deploy the system in different environments with fine-grained control over each component is one of the key advantages of this method. Additionally, by using containers, the system can be easily replicated and maintained, which is essential for its long-term operability.

Index words: Docker, Docker Compose, Traccar, GPS tracking, backend, frontend, network configuration, containerization, system deployment.

Introducción

En el desarrollo de aplicaciones que interactúan con sistemas externos a través de APIs, la gestión de la sesión es un aspecto crítico que garantiza la autenticación segura y la autorización de los usuarios. En este contexto, la API de Traccar, una plataforma de seguimiento de vehículos, proporciona un conjunto de herramientas que permite a las aplicaciones manejar sesiones de usuario de manera eficiente. Este documento explora cómo se implementa y gestiona la sesión en una aplicación que utiliza la API de Traccar, desde el proceso de autenticación inicial hasta la seguridad del token y el manejo de errores. A través de una adecuada gestión de la sesión, se asegura que las interacciones entre la aplicación y el servidor se realicen de manera segura, protegiendo tanto la información del usuario como la integridad del sistema.

Desarrollo

Login:

ChangeServerPage.jsx

```
import React from 'react';
import ElectricalServicesIcon from '@mui/icons-material/ElectricalServices';
import { makeStyles } from '@mui/styles';
import {
  Autocomplete, Button, Container, createFilterOptions, TextField,
} from '@mui/material';
import { useNavigate } from 'react-router-dom';
import { useTranslation } from '../common/components/LocalizationProvider';

const currentServer = `${window.location.protocol}//${window.location.host}`;

const officialServers = [
  currentServer,
  'https://demo.traccar.org',
```

```
  'https://demo2.traccar.org',
  'https://demo3.traccar.org',
  'https://demo4.traccar.org',
  'https://server.traccar.org',
  'http://localhost:8082',
  'http://localhost:3000',
];

const useStyles = makeStyles((theme) => ({
  icon: {
    textAlign: 'center',
    fontSize: '128px',
    color: theme.palette.neutral.main,
  },
  container: {
    textAlign: 'center',
    padding: theme.spacing(5, 3),
  },
  field: {
    margin: theme.spacing(3, 0),
  },
}));

const ChangeServerPage = () => {
  const classes = useStyles();
  const navigate = useNavigate();
  const t = useTranslation();

  const filter = createFilterOptions();

  const handleSubmit = (url) => {
    if (window.webkit && window.webkit.messageHandlers.appInterface) {

window.webkit.messageHandlers.appInterface.postMessage(`server|${url}`)
;
    } else if (window.appInterface) {
      window.appInterface.postMessage(`server|${url}`);
    } else {
      window.location.replace(url);
    }
  };

  return (
```

```
    <Container maxWidth="xs" className={classes.container}>
      <ElectricalServicesIcon className={classes.icon} />
      <Autocomplete
        freeSolo
        className={classes.field}
        options={officialServers}
                    renderInput={(params)   =>   <TextField   {...params}
label={t('settingsServer')} />}
        value={currentServer}
        onChange={(_, value) => value && handleSubmit(value)}
        filterOptions={(options, params) => {
          const filtered = filter(options, params);
                                          if     (params.inputValue     &&
!filtered.includes(params.inputValue)) {
            filtered.push(params.inputValue);
          }
          return filtered;
        }}
      />
      <Button
        onClick={() => navigate(-1)}
        color="secondary"
      >
        {t('sharedCancel')}
      </Button>
    </Container>
  );
};

export default ChangeServerPage;
```

Explicación del Código:

1. Importaciones:
    - React: Es la biblioteca base para construir interfaces de usuario en aplicaciones de una sola página (SPA).
    - ElectricalServicesIcon: Este es un ícono de Material UI que representa servicios eléctricos y se utiliza para añadir un elemento visual en la página.
    - makeStyles: Es un hook de Material UI que permite definir estilos CSS en forma de objetos de JavaScript.

- Autocomplete, Button, Container, createFilterOptions, TextField: Estos son componentes y utilidades de Material UI que ayudan a construir la interfaz de usuario de manera eficiente.
- useNavigate: Es un hook de react-router-dom utilizado para la navegación programática entre rutas dentro de la aplicación.
- useTranslation: Es un hook personalizado que probablemente se usa para manejar la localización (traducción de textos) en la aplicación.

2. Constante currentServer:
   - Esta constante almacena la URL actual del servidor, combinando el protocolo (http o https) y el host (nombre de dominio o dirección IP). Esto se utiliza para identificar el servidor en el cual se está ejecutando la aplicación.
3. Arreglo officialServers:
   - Este arreglo contiene una lista de URLs de servidores que son oficiales o predefinidos para la aplicación Traccar. Esto incluye la URL del servidor actual y varias URLs de servidores de demostración.
4. Función useStyles:
   - useStyles es una función que define y retorna un conjunto de estilos CSS personalizados para aplicar a los componentes de la interfaz. Define estilos como la alineación del ícono, el tamaño del texto, los márgenes y el relleno del contenedor.
5. Componente ChangeServerPage:
   - Este es el componente principal de la página que permite al usuario cambiar el servidor al que se conecta la aplicación.
   - classes: Se utiliza para aplicar los estilos definidos por useStyles.
   - navigate: Se usa para navegar a otras rutas dentro de la aplicación.
   - t: Es la función que se utiliza para obtener las traducciones de los textos mostrados en la interfaz.
6. Función handleSubmit:
   - handleSubmit maneja la lógica cuando un usuario selecciona o ingresa una URL de servidor. Dependiendo del entorno, puede enviar un mensaje a la interfaz de la aplicación a través de WebKit o simplemente redirigir el navegador a la nueva URL.
7. Renderizado del componente:
   - El componente renderiza un Container de tamaño máximo xs (extra pequeño), que contiene el ícono ElectricalServicesIcon, un campo de autocompletado para seleccionar o ingresar una URL de servidor, y un botón para cancelar la operación y regresar a la página anterior.
   - Autocomplete es el componente que permite al usuario seleccionar un servidor de la lista o ingresar una nueva URL. Usa la función filterOptions para filtrar las opciones basadas en la entrada del usuario.
   - Button permite al usuario cancelar la operación y regresar a la página anterior utilizando navigate(-1).

8. Exportación del Componente:
    - Finalmente, el componente ChangeServerPage se exporta para que pueda ser utilizado en otras partes de la aplicación.

LoginLayout.jsx

```
import React from 'react';
import { useMediaQuery, Paper } from '@mui/material';
import makeStyles from '@mui/styles/makeStyles';
import { useTheme } from '@mui/material/styles';
import LogoImage from './LogoImage';

const useStyles = makeStyles((theme) => ({
  root: {
    display: 'flex',
    height: '100%',
  },
  sidebar: {
    display: 'flex',
    justifyContent: 'center',
    alignItems: 'center',
    background: theme.palette.primary.main,
    paddingBottom: theme.spacing(5),
    width: theme.dimensions.sidebarWidth,
    [theme.breakpoints.down('lg')]: {
      width: theme.dimensions.sidebarWidthTablet,
    },
    [theme.breakpoints.down('sm')]: {
      width: '0px',
    },
  },
  paper: {
    display: 'flex',
    flexDirection: 'column',
    justifyContent: 'center',
    alignItems: 'center',
    flex: 1,
    boxShadow: '-2px 0px 16px rgba(0, 0, 0, 0.25)',
    [theme.breakpoints.up('lg')]: {
      padding: theme.spacing(0, 25, 0, 0),
    },
  },
  form: {
    maxWidth: theme.spacing(52),
```

```
    padding: theme.spacing(5),
    width: '100%',
  },
}));

const LoginLayout = ({ children }) => {
  const classes = useStyles();
  const theme = useTheme();

  return (
    <main className={classes.root}>
      <div className={classes.sidebar}>
            {!useMediaQuery(theme.breakpoints.down('lg')) && <LogoImage
color={theme.palette.secondary.contrastText} />}
      </div>
      <Paper className={classes.paper}>
        <form className={classes.form}>
          {children}
        </form>
      </Paper>
    </main>
  );
};

export default LoginLayout;
```

Explicación del Código:

1. Importaciones:
    - React: Importa la biblioteca base de React, que es necesaria para crear componentes de interfaz de usuario.
    - useMediaQuery, Paper: Son componentes y hooks de Material UI. useMediaQuery se utiliza para detectar el tamaño de la pantalla, y Paper es un componente de Material UI que proporciona un contenedor con un estilo de papel.
    - makeStyles: Es un hook de Material UI que permite crear estilos CSS como objetos de JavaScript.
    - useTheme: Es un hook de Material UI que proporciona acceso al tema actual de la aplicación.
    - LogoImage: Importa un componente personalizado llamado LogoImage, que probablemente renderiza el logotipo de la aplicación.
2. Constante useStyles:

- useStyles define los estilos personalizados que se aplican a los diferentes elementos del componente. Estos estilos incluyen el diseño del contenedor principal (root), la barra lateral (sidebar), el contenedor de formulario (paper), y el formulario en sí (form).
- sidebar: Ajusta el tamaño y la visibilidad de la barra lateral dependiendo del tamaño de la pantalla. Se hace más estrecha en pantallas medianas (lg) y se oculta completamente en pantallas pequeñas (sm).
- paper: Aplica un sombreado y un padding personalizado al contenedor que alberga el formulario.
- form: Establece el tamaño máximo y el padding para el formulario.

3. Componente LoginLayout:
   - LoginLayout es el componente que se encarga de la disposición del diseño de la página de inicio de sesión. Este componente es flexible y puede ser reutilizado para otras páginas con un diseño similar.
   - classes: Se utiliza para aplicar los estilos definidos por useStyles.
   - theme: Proporciona acceso al tema de Material UI, que se utiliza para adaptar los estilos a las preferencias de la aplicación, como colores y tamaños.
4. Renderizado del componente:
   - main: Es el contenedor principal que utiliza el estilo root para establecer un diseño de flexbox que cubre toda la altura de la pantalla.
   - sidebar: Muestra una barra lateral que contiene el logotipo de la aplicación, pero solo si la pantalla es lo suficientemente grande (lg o superior). En pantallas más pequeñas, esta barra lateral se oculta o se reduce en tamaño.
   - Paper: Este componente actúa como un contenedor visualmente separado que contiene el formulario de inicio de sesión. El contenedor tiene una sombra y un padding para destacarse del resto de la interfaz.
   - form: Dentro del contenedor Paper, se renderiza un formulario (form), que recibe los hijos (children) que se le pasan al componente LoginLayout. Estos hijos suelen ser los elementos de formulario como campos de texto, botones, etc.
5. Exportación del Componente:
   - Finalmente, el componente LoginLayout se exporta para que pueda ser utilizado en otras partes de la aplicación, permitiendo que cualquier contenido que se pase como children sea renderizado dentro de la estructura de diseño definida por este componente.

LoginPage.jsx

```
import React, { useEffect, useState } from 'react';
import dayjs from 'dayjs';
import {
    useMediaQuery, Select, MenuItem, FormControl, Button, TextField,
Link, Snackbar, IconButton, Tooltip, LinearProgress, Box,
```

```
} from '@mui/material';
import ReactCountryFlag from 'react-country-flag';
import makeStyles from '@mui/styles/makeStyles';
import CloseIcon from '@mui/icons-material/Close';
import LockOpenIcon from '@mui/icons-material/LockOpen';
import { useTheme } from '@mui/material/styles';
import { useDispatch, useSelector } from 'react-redux';
import { useNavigate } from 'react-router-dom';
import { sessionActions } from '../store';
import { useLocalization, useTranslation } from
'../common/components/LocalizationProvider';
import LoginLayout from './LoginLayout';
import usePersistedState from '../common/util/usePersistedState';
import { handleLoginTokenListeners, nativeEnvironment,
nativePostMessage } from '../common/components/NativeInterface';
import LogoImage from './LogoImage';
import { useCatch } from '../reactHelper';

const useStyles = makeStyles((theme) => ({
  options: {
    position: 'fixed',
    top: theme.spacing(2),
    right: theme.spacing(2),
    display: 'flex',
    flexDirection: 'row',
    gap: theme.spacing(1),
  },
  container: {
    display: 'flex',
    flexDirection: 'column',
    gap: theme.spacing(2),
  },
  extraContainer: {
    display: 'flex',
    flexDirection: 'row',
    justifyContent: 'center',
    gap: theme.spacing(4),
    marginTop: theme.spacing(2),
  },
  registerButton: {
    minWidth: 'unset',
  },
  link: {
```

```
    cursor: 'pointer',
  },
}));

const LoginPage = () => {
  const classes = useStyles();
  const dispatch = useDispatch();
  const navigate = useNavigate();
  const theme = useTheme();
  const t = useTranslation();

  const { languages, language, setLanguage } = useLocalization();
   const languageList = Object.entries(languages).map((values) => ({
code: values[0], country: values[1].country, name: values[1].name }));

  const [failed, setFailed] = useState(false);

  const [email, setEmail] = usePersistedState('loginEmail', '');
  const [password, setPassword] = useState('');
  const [code, setCode] = useState('');

        const registrationEnabled = useSelector((state) =>
state.session.server.registration);
         const languageEnabled = useSelector((state) =>
!state.session.server.attributes['ui.disableLoginLanguage']);
          const changeEnabled = useSelector((state) =>
!state.session.server.attributes.disableChange);
          const emailEnabled = useSelector((state) =>
state.session.server.emailEnabled);
          const openIdEnabled = useSelector((state) =>
state.session.server.openIdEnabled);
          const openIdForced = useSelector((state) =>
state.session.server.openIdEnabled                                    &&
state.session.server.openIdForce);
  const [codeEnabled, setCodeEnabled] = useState(false);

  const [announcementShown, setAnnouncementShown] = useState(false);
          const announcement = useSelector((state) =>
state.session.server.announcement);

  const generateLoginToken = async () => {
    if (nativeEnvironment) {
      let token = '';
```

```
      try {
        const expiration = dayjs().add(6, 'months').toISOString();
        const response = await fetch('/api/session/token', {
          method: 'POST',
          body: new URLSearchParams(`expiration=${expiration}`),
        });
        if (response.ok) {
          token = await response.text();
        }
      } catch (error) {
        token = '';
      }
      nativePostMessage(`login|${token}`);
    }
  };

  const handlePasswordLogin = async (event) => {
    event.preventDefault();
    setFailed(false);
    try {
                                                const query =
`email=${encodeURIComponent(email)}&password=${encodeURIComponent(passw
ord)}`;
      const response = await fetch('/api/session', {
        method: 'POST',
         body: new URLSearchParams(code.length ? `${query}&code=${code}`
: query),
      });
      if (response.ok) {
        const user = await response.json();
        generateLoginToken();
        dispatch(sessionActions.updateUser(user));
        navigate('/');
                    } else if (response.status === 401 &&
response.headers.get('WWW-Authenticate') === 'TOTP') {
        setCodeEnabled(true);
      } else {
        throw Error(await response.text());
      }
    } catch (error) {
      setFailed(true);
      setPassword('');
    }
```

```
  };

  const handleTokenLogin = useCatch(async (token) => {
                                const        response        =        await
fetch(`/api/session?token=${encodeURIComponent(token)}`);
    if (response.ok) {
      const user = await response.json();
      dispatch(sessionActions.updateUser(user));
      navigate('/');
    } else {
      throw Error(await response.text());
    }
  });

  const handleOpenIdLogin = () => {
    document.location = '/api/session/openid/auth';
  };

  useEffect(() => nativePostMessage('authentication'), []);

  useEffect(() => {
    const listener = (token) => handleTokenLogin(token);
    handleLoginTokenListeners.add(listener);
    return () => handleLoginTokenListeners.delete(listener);
  }, []);

  if (openIdForced) {
    handleOpenIdLogin();
    return (<LinearProgress />);
  }

  return (
    <LoginLayout>
      <div className={classes.options}>
        {nativeEnvironment && changeEnabled && (
          <Tooltip title={t('settingsServer')}>
            <IconButton onClick={() => navigate('/change-server')}>
              <LockOpenIcon />
            </IconButton>
          </Tooltip>
        )}
        {languageEnabled && (
          <FormControl>
```

```
                        <Select  value={language}  onChange={(e)  =>
setLanguage(e.target.value)}>
              {languageList.map((it) => (
                <MenuItem key={it.code} value={it.code}>
                  <Box component="span" sx={{ mr: 1 }}>
                    <ReactCountryFlag countryCode={it.country} svg />
                  </Box>
                  {it.name}
                </MenuItem>
              ))}
            </Select>
          </FormControl>
        )}
      </div>
      <div className={classes.container}>
            {useMediaQuery(theme.breakpoints.down('lg'))  &&  <LogoImage
color={theme.palette.primary.main} />}
        <TextField
          required
          error={failed}
          label={t('userEmail')}
          name="email"
          value={email}
          autoComplete="email"
          autoFocus={!email}
          onChange={(e) => setEmail(e.target.value)}
          helperText={failed && 'Invalid username or password'}
        />
        <TextField
          required
          error={failed}
          label={t('userPassword')}
          name="password"
          value={password}
          type="password"
          autoComplete="current-password"
          autoFocus={!!email}
          onChange={(e) => setPassword(e.target.value)}
        />
        {codeEnabled && (
          <TextField
            required
            error={failed}
```

```
            label={t('loginTotpCode')}
            name="code"
            value={code}
            type="number"
            onChange={(e) => setCode(e.target.value)}
          />
        )}
        <Button
          onClick={handlePasswordLogin}
          type="submit"
          variant="contained"
          color="secondary"
          disabled={!email || !password || (codeEnabled && !code)}
        >
          {t('loginLogin')}
        </Button>
        {openIdEnabled && (
          <Button
            onClick={() => handleOpenIdLogin()}
            variant="contained"
            color="secondary"
          >
            {t('loginOpenId')}
          </Button>
        )}
        <div className={classes.extraContainer}>
          {registrationEnabled && (
            <Link
              onClick={() => navigate('/register')}
              className={classes.link}
              underline="none"
              variant="caption"
            >
              {t('loginRegister')}
            </Link>
          )}
          {emailEnabled && (
            <Link
              onClick={() => navigate('/reset-password')}
              className={classes.link}
              underline="none"
              variant="caption"
            >
```

```
              {t('loginReset')}
            </Link>
          )}
        </div>
      </div>
      <Snackbar
        open={!!announcement && !announcementShown}
        message={announcement}
        action={(
              <IconButton size="small" color="inherit" onClick={() => setAnnouncementShown(true)}>
            <CloseIcon fontSize="small" />
          </IconButton>
        )}
      />
    </LoginLayout>
  );
};

export default LoginPage;
```

Resumen de los Componentes y Funcionalidades

1. Hooks Importados:
    - useEffect, useState: Para manejar los efectos secundarios y el estado local de los componentes.
    - useMediaQuery: Para aplicar estilos específicos según el tamaño de la pantalla.
    - useDispatch, useSelector: Para interactuar con el store de Redux.
    - useNavigate: Para la navegación entre rutas.
    - useTheme: Para acceder al tema personalizado de Material-UI.
2. Estilos:
    - El uso de makeStyles de Material-UI define los estilos específicos para los elementos de la página de inicio de sesión.
    - Las clases generadas (options, container, extraContainer, etc.) se aplican a diferentes partes de la UI.
3. Login Page:
    - Muestra un formulario de inicio de sesión donde los usuarios pueden ingresar su correo electrónico y contraseña. Si se requiere un código TOTP (para autenticación de dos factores), se mostrará un campo adicional.

- Incluye la opción de autenticación mediante OpenID si está habilitada.
- También se maneja la opción de cambiar de servidor y seleccionar el idioma para la interfaz.

4. Funciones Clave:
   - handlePasswordLogin: Envía los datos del formulario al servidor para autenticación mediante correo electrónico y contraseña.
   - handleTokenLogin: Se utiliza para manejar la autenticación mediante un token.
   - handleOpenIdLogin: Redirige a la página de autenticación de OpenID.
   - generateLoginToken: Genera un token de inicio de sesión si la aplicación se ejecuta en un entorno nativo (por ejemplo, una aplicación móvil).
5. Efectos Secundarios:
   - Se ejecutan efectos secundarios al montar el componente para solicitar autenticación y manejar tokens de inicio de sesión.
6. UI Adicional:
   - Snackbar para mostrar anuncios importantes.
   - Banderas de país para la selección de idiomas.
   - Botones para registro, restablecimiento de contraseña, y autenticación OpenID.

LogoImage.jsx

```
import React from 'react';
import { useTheme, useMediaQuery } from '@mui/material';
import { useSelector } from 'react-redux';
import { makeStyles } from '@mui/styles';
import Logo from '../resources/images/logo.svg?react';

const useStyles = makeStyles((theme) => ({
  image: {
    alignSelf: 'center',
    maxWidth: '240px',
    maxHeight: '120px',
    width: 'auto',
    height: 'auto',
    margin: theme.spacing(2),
  },
}));

const LogoImage = ({ color }) => {
  const theme = useTheme();
  const classes = useStyles();
```

```
  const expanded = !useMediaQuery(theme.breakpoints.down('lg'));

              const       logo       =       useSelector((state)       =>
state.session.server.attributes?.logo);
           const      logoInverted      =      useSelector((state)      =>
state.session.server.attributes?.logoInverted);

  if (logo) {
    if (expanded && logoInverted) {
        return <img className={classes.image} src={logoInverted} alt=""
/>;
    }
    return <img className={classes.image} src={logo} alt="" />;
  }
  return <Logo className={classes.image} style={{ color }} />;
};

export default LogoImage;
```

Explicación del Código

1. Importaciones:
    - React: Biblioteca principal para la creación de componentes en React.
    - useTheme, useMediaQuery: Hooks de Material-UI para acceder al tema y gestionar consultas de medios (media queries).
    - useSelector: Hook de react-redux para acceder al estado global.
    - makeStyles: Hook de Material-UI para crear estilos personalizados.
    - Logo: Un archivo SVG que se usará como logotipo predeterminado si no hay logotipo personalizado disponible.
2. Estilos:
    - useStyles: Se usa para definir estilos personalizados para el componente LogoImage. Aquí, el estilo image asegura que la imagen del logotipo tenga un tamaño máximo y se centre en su contenedor.
3. Componente LogoImage:
    - Propiedades:
        - color: Permite personalizar el color del logotipo si se usa el archivo SVG predeterminado.
    - Hooks:

        - useTheme: Obtiene el tema actual de Material-UI.
        - useStyles: Aplica los estilos definidos.
        - useMediaQuery: Verifica si la pantalla es de tamaño grande (lg) o más pequeña.
    - Acceso al Estado:
        - useSelector se utiliza para acceder a los atributos logo y logoInverted del estado global, que podrían ser URLs de logotipos personalizados almacenados en el estado de Redux.
4. Lógica de Renderizado:
    - Si logo está disponible en el estado global, el componente:
        - Verifica el tamaño de la pantalla (expanded) y si logoInverted está disponible. Si ambos son verdaderos, muestra el logotipo invertido.
        - Si no se cumple la condición anterior, muestra el logotipo normal.
    - Si logo no está disponible en el estado global, utiliza el archivo SVG predeterminado (Logo) y aplica el color proporcionado a través de la prop color.

Flujo del Componente

1. Estilos:
    - Se definen y aplican a través de makeStyles.
2. Renderizado Condicional:
    - Primero, verifica si hay un logotipo disponible en el estado global (logo).
    - Luego, determina si se debe mostrar el logotipo invertido en función del tamaño de la pantalla y si logoInverted está definido.
    - Si no hay logotipo personalizado, usa el archivo SVG proporcionado (Logo) y aplica el color dado a través de las propiedades de estilo en línea.

RegisterPage.jsx

```
import React, { useState } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import {
  Button, TextField, Typography, Snackbar, IconButton,
} from '@mui/material';
import makeStyles from '@mui/styles/makeStyles';
import { useNavigate } from 'react-router-dom';
import ArrowBackIcon from '@mui/icons-material/ArrowBack';
import LoginLayout from './LoginLayout';
```

```
import { useTranslation } from '../common/components/LocalizationProvider';
import { snackBarDurationShortMs } from '../common/util/duration';
import { useCatch, useEffectAsync } from '../reactHelper';
import { sessionActions } from '../store';

const useStyles = makeStyles((theme) => ({
  container: {
    display: 'flex',
    flexDirection: 'column',
    gap: theme.spacing(2),
  },
  header: {
    display: 'flex',
    alignItems: 'center',
  },
  title: {
    fontSize: theme.spacing(3),
    fontWeight: 500,
    marginLeft: theme.spacing(1),
    textTransform: 'uppercase',
  },
}));

const RegisterPage = () => {
  const classes = useStyles();
  const dispatch = useDispatch();
  const navigate = useNavigate();
  const t = useTranslation();

  const server = useSelector((state) => state.session.server);
  const totpForce = useSelector((state) => state.session.server.attributes.totpForce);

  const [name, setName] = useState('');
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [totpKey, setTotpKey] = useState(null);
  const [snackbarOpen, setSnackbarOpen] = useState(false);

  useEffectAsync(async () => {
    if (totpForce) {
```

```
      const response = await fetch('/api/users/totp', { method: 'POST'
});
      if (response.ok) {
        setTotpKey(await response.text());
      } else {
        throw Error(await response.text());
      }
    }
  }, [totpForce, setTotpKey]);

  const handleSubmit = useCatch(async (event) => {
    event.preventDefault();
    const response = await fetch('/api/users', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ name, email, password, totpKey }),
    });
    if (response.ok) {
      setSnackbarOpen(true);
    } else {
      throw Error(await response.text());
    }
  });

  return (
    <LoginLayout>
      <div className={classes.container}>
        <div className={classes.header}>
          {!server.newServer && (
                          <IconButton  color="primary"  onClick={()  =>
navigate('/login')}>
              <ArrowBackIcon />
            </IconButton>
          )}
          <Typography className={classes.title} color="primary">
            {t('loginRegister')}
          </Typography>
        </div>
        <TextField
          required
          label={t('sharedName')}
          name="name"
          value={name}
```

```
          autoComplete="name"
          autoFocus
          onChange={(event) => setName(event.target.value)}
        />
        <TextField
          required
          type="email"
          label={t('userEmail')}
          name="email"
          value={email}
          autoComplete="email"
          onChange={(event) => setEmail(event.target.value)}
        />
        <TextField
          required
          label={t('userPassword')}
          name="password"
          value={password}
          type="password"
          autoComplete="current-password"
          onChange={(event) => setPassword(event.target.value)}
        />
        {totpForce && (
          <TextField
            required
            label={t('loginTotpKey')}
            name="totpKey"
            value={totpKey || ''}
            InputProps={{
              readOnly: true,
            }}
          />
        )}
        <Button
          variant="contained"
          color="secondary"
          onClick={handleSubmit}
          type="submit"
                disabled={!name || !password || !(server.newServer ||
/(.+)@(.+)\.(.{2,})/.test(email))}
          fullWidth
        >
          {t('loginRegister')}
```

```
        </Button>
      </div>
      <Snackbar
        open={snackbarOpen}
        onClose={() => {
            dispatch(sessionActions.updateServer({ ...server, newServer:
false }));
          navigate('/login');
        }}
        autoHideDuration={snackBarDurationShortMs}
        message={t('loginCreated')}
      />
    </LoginLayout>
  );
};

export default RegisterPage;
```

1. Importaciones:
    - React, useState: Importa React y el hook useState para manejar el estado local.
    - useDispatch, useSelector: Hooks de react-redux para interactuar con el estado global y despachar acciones.
    - Button, TextField, Typography, Snackbar, IconButton: Componentes de Material-UI para crear la interfaz de usuario.
    - makeStyles: Hook de Material-UI para crear estilos personalizados.
    - useNavigate: Hook de react-router-dom para la navegación programática.
    - ArrowBackIcon: Icono de Material-UI para el botón de retroceso.
    - LoginLayout: Componente que envuelve la interfaz de usuario, probablemente proporciona un diseño común para las páginas de inicio de sesión y registro.
    - useTranslation: Hook para la internacionalización y traducción de textos.
    - snackBarDurationShortMs: Duración de la Snackbar en milisegundos.
    - useCatch, useEffectAsync: Hooks personalizados para manejar errores y efectos asíncronos.
    - sessionActions: Acciones de Redux relacionadas con la sesión.
2. Estilos:
    - useStyles: Define los estilos personalizados para el componente, como el diseño del contenedor y la cabecera.

3. Componente RegisterPage:
    - Hooks de Estado:
        - name, email, password, totpKey, snackbarOpen: Estados locales para manejar los campos del formulario y la visibilidad de la Snackbar.
    - Hooks de Redux:
        - server: Accede a la información del servidor desde el estado global.
        - totpForce: Verifica si la autenticación de dos factores (TOTP) es obligatoria.
    - Hooks Adicionales:
        - useEffectAsync: Maneja la recuperación de la clave TOTP si totpForce es verdadero.
        - useCatch: Maneja errores durante el envío del formulario.
    - Funciones:
        - handleSubmit: Envía los datos del formulario al servidor cuando el usuario hace clic en el botón de registro. Muestra una Snackbar en caso de éxito.
    - Renderizado:
        - Muestra un formulario de registro con campos para el nombre, correo electrónico, contraseña y, si es necesario, una clave TOTP.
        - Incluye un botón de retroceso que navega a la página de inicio de sesión si el servidor no es nuevo.
        - Muestra una Snackbar al confirmar el registro exitoso, que cierra el mensaje y navega a la página de inicio de sesión.

Flujo del Componente

1. Estilos:
    - Aplicados a través de makeStyles para el contenedor, la cabecera y el título.
2. Efectos y Manejo de Errores:
    - useEffectAsync recupera la clave TOTP si se requiere.
    - useCatch maneja errores durante el envío del formulario.
3. Formulario de Registro:
    - Campos para nombre, correo electrónico, y contraseña.
    - Campo adicional para la clave TOTP si es necesario.
4. Botón de Envío:
    - Deshabilitado si faltan campos obligatorios o el correo electrónico es inválido.
    - Envía los datos del formulario al servidor y muestra una Snackbar en caso de éxito.
5. Snackbar:

- Muestra un mensaje de éxito y cierra la Snackbar después de la duración especificada, actualiza el estado del servidor y navega a la página de inicio de sesión.

ResetPasswordPage.jsx

```
import React, { useState } from 'react';
import {
  Button, TextField, Typography, Snackbar, IconButton,
} from '@mui/material';
import makeStyles from '@mui/styles/makeStyles';
import { useNavigate } from 'react-router-dom';
import ArrowBackIcon from '@mui/icons-material/ArrowBack';
import LoginLayout from './LoginLayout';
import { useTranslation } from '../common/components/LocalizationProvider';
import useQuery from '../common/util/useQuery';
import { snackBarDurationShortMs } from '../common/util/duration';
import { useCatch } from '../reactHelper';

const useStyles = makeStyles((theme) => ({
  container: {
    display: 'flex',
    flexDirection: 'column',
    gap: theme.spacing(2),
  },
  header: {
    display: 'flex',
    alignItems: 'center',
  },
  title: {
    fontSize: theme.spacing(3),
    fontWeight: 500,
    marginLeft: theme.spacing(1),
    textTransform: 'uppercase',
  },
}));

const ResetPasswordPage = () => {
  const classes = useStyles();
  const navigate = useNavigate();
  const t = useTranslation();
  const query = useQuery();

  const token = query.get('passwordReset');
```

```
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [snackbarOpen, setSnackbarOpen] = useState(false);

  const handleSubmit = useCatch(async (event) => {
    event.preventDefault();
    let response;
    if (!token) {
      response = await fetch('/api/password/reset', {
        method: 'POST',
                                                    body:       new
URLSearchParams(`email=${encodeURIComponent(email)}`),
      });
    } else {
      response = await fetch('/api/password/update', {
        method: 'POST',
                                                    body:       new
URLSearchParams(`token=${encodeURIComponent(token)}&password=${encodeUR
IComponent(password)}`),
      });
    }
    if (response.ok) {
      setSnackbarOpen(true);
    } else {
      throw Error(await response.text());
    }
  });

  return (
    <LoginLayout>
      <div className={classes.container}>
        <div className={classes.header}>
                          <IconButton   color="primary"   onClick={()   =>
navigate('/login')}>
            <ArrowBackIcon />
          </IconButton>
          <Typography className={classes.title} color="primary">
            {t('loginReset')}
          </Typography>
        </div>
        {!token ? (
          <TextField
```

```
            required
            type="email"
            label={t('userEmail')}
            name="email"
            value={email}
            autoComplete="email"
            onChange={(event) => setEmail(event.target.value)}
          />
        ) : (
          <TextField
            required
            label={t('userPassword')}
            name="password"
            value={password}
            type="password"
            autoComplete="current-password"
            onChange={(event) => setPassword(event.target.value)}
          />
        )}
        <Button
          variant="contained"
          color="secondary"
          type="submit"
          onClick={handleSubmit}
          disabled={!/(.+)@(.+)\.(.{2,})/.test(email) && !password}
          fullWidth
        >
          {t('loginReset')}
        </Button>
      </div>
      <Snackbar
        open={snackbarOpen}
        onClose={() => navigate('/login')}
        autoHideDuration={snackBarDurationShortMs}
                        message={!token   ?   t('loginResetSuccess')    :
t('loginUpdateSuccess')}
      />
    </LoginLayout>
  );
};

export default ResetPasswordPage;
```

Descripción del Código

1. Importaciones y Dependencias:
    - React, useState para manejar el estado.
    - Componentes de @mui/material como Button, TextField, Typography, Snackbar, e IconButton.
    - makeStyles para los estilos.
    - useNavigate de react-router-dom para la navegación.
    - ArrowBackIcon para un botón de retroceso.
    - LoginLayout para el diseño de la página.
    - useTranslation para la traducción.
    - useQuery para obtener parámetros de consulta.
    - snackBarDurationShortMs para la duración del Snackbar.
    - useCatch para manejar errores.
2. Uso de makeStyles:
    - Define los estilos para la página, incluyendo el contenedor, el encabezado, y el título.
3. Componentes de Estado:
    - email, password, y snackbarOpen para manejar los valores del formulario y el estado del Snackbar.
4. Manejo del Envío del Formulario:
    - handleSubmit gestiona el envío del formulario para restablecer la contraseña o actualizarla dependiendo de si el token está presente o no.
5. Renderización Condicional:
    - Muestra un campo para el correo electrónico si no hay token y un campo para la contraseña si hay un token.
6. Snackbar:
    - Muestra un mensaje después de que la solicitud de restablecimiento o actualización de contraseña se complete exitosamente.

Mejoras y Recomendaciones

1. Validación de Campo:
    - La condición en disabled del botón de envío es algo confusa. La expresión actual !/(.+)@(.+)\.(.{2,})/.test(email) && !password deshabilita el botón si el correo electrónico no es válido y la contraseña está vacía. Considera simplificarlo a !email || !(/(.+)@(.+)\.(.{2,})/.test(email)) || (token && !password) para hacer que el botón se habilite adecuadamente.

2. Mensajes de Error:
    - El uso de throw Error(await response.text()) puede hacer que el usuario vea mensajes de error no amigables. Considera manejar errores de manera que se muestren mensajes claros al usuario.
3. Estilos:
    - Asegúrate de que los estilos proporcionados por makeStyles sean consistentes con el resto de tu aplicación y considera usar sx si estás usando la última versión de Material-UI.
4. Uso de useQuery:
    - Verifica que useQuery funcione correctamente para obtener parámetros de consulta. Asegúrate de que esté manejado de manera eficiente.
5. Accesibilidad:
    - Considera añadir atributos aria-label o aria-labelledby a los elementos para mejorar la accesibilidad.
6. Navegación:
    - Tras el éxito en el restablecimiento o actualización de la contraseña, podrías proporcionar una opción para redirigir a la página de inicio de sesión o al panel de usuario en lugar de solo redirigir a /login.

UserConnetionsPage.jsx

```
import React from 'react';
import { useParams } from 'react-router-dom';
import {
  Accordion,
  AccordionSummary,
  AccordionDetails,
  Typography,
  Container,
} from '@mui/material';
import ExpandMoreIcon from '@mui/icons-material/ExpandMore';
import LinkField from '../common/components/LinkField';
import { useTranslation } from '../common/components/LocalizationProvider';
import SettingsMenu from './components/SettingsMenu';
import { formatNotificationTitle } from '../common/util/formatter';
import PageLayout from '../common/components/PageLayout';
import useSettingsStyles from './common/useSettingsStyles';

const nectionsPage = () => {
  const classes = useSettingsStyles();
  const t = useTranslation();
```

```
  const { id } = useParams();

  return (
    <PageLayout
      menu={<SettingsMenu />}
                        breadcrumbs={['settingsTitle',    'settingsUser',
'sharedConnections']}
    >
      <Container maxWidth="xs" className={classes.container}>
        <Accordion defaultExpanded>
          <AccordionSummary expandIcon={<ExpandMoreIcon />}>
            <Typography variant="subtitle1">
              {t('sharedConnections')}
            </Typography>
          </AccordionSummary>
          <AccordionDetails className={classes.details}>
            <LinkField
              endpointAll="/api/devices?all=true"
              endpointLinked={`/api/devices?userId=${id}`}
              baseId={id}
              keyBase="userId"
              keyLink="deviceId"
              titleGetter={(it) => `${it.name} (${it.uniqueId})`}
              label={t('deviceTitle')}
            />
            <LinkField
              endpointAll="/api/groups?all=true"
              endpointLinked={`/api/groups?userId=${id}`}
              baseId={id}
              keyBase="userId"
              keyLink="groupId"
              label={t('settingsGroups')}
            />
            <LinkField
              endpointAll="/api/geofences?all=true"
              endpointLinked={`/api/geofences?userId=${id}`}
              baseId={id}
              keyBase="userId"
              keyLink="geofenceId"
              label={t('sharedGeofences')}
            />
            <LinkField
```

```
              endpointAll="/api/notifications?all=true"
              endpointLinked={`/api/notifications?userId=${id}`}
              baseId={id}
              keyBase="userId"
              keyLink="notificationId"
                    titleGetter={(it) => formatNotificationTitle(t, it,
true)}
              label={t('sharedNotifications')}
            />
            <LinkField
              endpointAll="/api/calendars?all=true"
              endpointLinked={`/api/calendars?userId=${id}`}
              baseId={id}
              keyBase="userId"
              keyLink="calendarId"
              label={t('sharedCalendars')}
            />
            <LinkField
              endpointAll="/api/users?all=true"
              endpointLinked={`/api/users?userId=${id}`}
              baseId={id}
              keyBase="userId"
              keyLink="managedUserId"
              label={t('settingsUsers')}
            />
            <LinkField
              endpointAll="/api/attributes/computed?all=true"
              endpointLinked={`/api/attributes/computed?userId=${id}`}
              baseId={id}
              keyBase="userId"
              keyLink="attributeId"
              titleGetter={(it) => it.description}
              label={t('sharedComputedAttributes')}
            />
            <LinkField
              endpointAll="/api/drivers?all=true"
              endpointLinked={`/api/drivers?userId=${id}`}
              baseId={id}
              keyBase="userId"
              keyLink="driverId"
              titleGetter={(it) => `${it.name} (${it.uniqueId})`}
              label={t('sharedDrivers')}
            />
```

```
            <LinkField
              endpointAll="/api/commands?all=true"
              endpointLinked={`/api/commands?userId=${id}`}
              baseId={id}
              keyBase="userId"
              keyLink="commandId"
              titleGetter={(it) => it.description}
              label={t('sharedSavedCommands')}
            />
            <LinkField
              endpointAll="/api/maintenance?all=true"
              endpointLinked={`/api/maintenance?userId=${id}`}
              baseId={id}
              keyBase="userId"
              keyLink="maintenanceId"
              label={t('sharedMaintenance')}
            />
          </AccordionDetails>
        </Accordion>
      </Container>
    </PageLayout>
  );
};

export default nectionsPage;
```

Descripción del Código

1. Importaciones y Dependencias:
    - React y useParams para manejar los parámetros de la URL.
    - Componentes de Material-UI como Accordion, AccordionSummary, AccordionDetails, Typography, y Container.
    - ExpandMoreIcon para el ícono de expansión en el acordeón.
    - LinkField para mostrar los enlaces a diferentes recursos.
    - useTranslation para manejar la traducción.
    - SettingsMenu para el menú de configuración.
    - formatNotificationTitle para formatear títulos de notificaciones.
    - PageLayout para la estructura de la página.
    - useSettingsStyles para estilos personalizados.

2. Uso de useParams:
    - Obtiene el parámetro id de la URL.
3. Estructura de la Página:
    - Usa un PageLayout que incluye un menú y una jerarquía de migas de pan.
    - Dentro del contenedor, un Accordion muestra una lista de LinkField que están relacionados con el id del usuario.
4. LinkField:
    - Se usa para crear enlaces a diferentes tipos de recursos como dispositivos, grupos, geocercas, notificaciones, calendarios, usuarios, atributos computados, conductores, comandos y mantenimiento.

Mejoras y Recomendaciones

1. Nombre del Componente:
    - El nombre del componente nectionsPage parece estar incompleto. Debería ser ConnectionsPage para seguir las convenciones de nombres en React (PascalCase).
2. Optimización de LinkField:
    - Si LinkField es un componente que se usa con frecuencia y tiene muchas propiedades, considera extraer la lógica repetitiva en un array o función para evitar la repetición de código.
3. Manejo de Claves de Elementos:
    - Asegúrate de que las claves para los elementos en el AccordionDetails sean únicas. Aunque en este caso no parece haber un problema, ten en cuenta que las claves deben ser únicas para cada instancia en listas.
4. Accesibilidad:
    - Considera añadir atributos aria-label para mejorar la accesibilidad de los componentes interactivos.
5. Uso de Tipografía:
    - El AccordionSummary usa Typography con variant="subtitle1". Asegúrate de que esta variante sea la más adecuada para el diseño general de la página.
6. Optimización del Código:
    - Puedes usar un array para iterar sobre los tipos de datos en LinkField en lugar de definir cada uno por separado. Esto hará que el código sea más mantenible.

UserPage.jsx

```
import React, { useEffect, useState } from 'react';
import { useNavigate, useParams } from 'react-router-dom';
import {
  Accordion,
  AccordionSummary,
```

```
  AccordionDetails,
  Typography,
  FormControl,
  InputLabel,
  Select,
  MenuItem,
  FormControlLabel,
  Checkbox,
  FormGroup,
  TextField,
  Button,
  InputAdornment,
  IconButton,
  OutlinedInput,
} from '@mui/material';
import ExpandMoreIcon from '@mui/icons-material/ExpandMore';
import DeleteForeverIcon from '@mui/icons-material/DeleteForever';
import CachedIcon from '@mui/icons-material/Cached';
import CloseIcon from '@mui/icons-material/Close';
import { useDispatch, useSelector } from 'react-redux';
import dayjs from 'dayjs';
import EditItemView from './components/EditItemView';
import EditAttributesAccordion from
'./components/EditAttributesAccordion';
import { useTranslation } from
'../common/components/LocalizationProvider';
import useUserAttributes from '../common/attributes/useUserAttributes';
import { sessionActions } from '../store';
import SelectField from '../common/components/SelectField';
import SettingsMenu from './components/SettingsMenu';
import useCommonUserAttributes from
'../common/attributes/useCommonUserAttributes';
import { useAdministrator, useRestriction, useManager } from
'../common/util/permissions';
import useQuery from '../common/util/useQuery';
import { useCatch } from '../reactHelper';
import useMapStyles from '../map/core/useMapStyles';
import { map } from '../map/core/MapView';
import useSettingsStyles from './common/useSettingsStyles';

const UserPage = () => {
  const classes = useSettingsStyles();
  const navigate = useNavigate();
```

```
  const dispatch = useDispatch();
  const t = useTranslation();

  const admin = useAdministrator();
  const manager = useManager();
  const fixedEmail = useRestriction('fixedEmail');

  const currentUser = useSelector((state) => state.session.user);
        const    registrationEnabled    =    useSelector((state)    =>
state.session.server.registration);
          const     openIdForced     =     useSelector((state)     =>
state.session.server.openIdForce);
           const      totpEnable      =      useSelector((state)      =>
state.session.server.attributes.totpEnable);
            const      totpForce      =      useSelector((state)      =>
state.session.server.attributes.totpForce);

  const mapStyles = useMapStyles();
  const commonUserAttributes = useCommonUserAttributes(t);
  const userAttributes = useUserAttributes(t);

  const { id } = useParams();
   const [item, setItem] = useState(id === currentUser.id.toString() ?
currentUser : null);

  const [deleteEmail, setDeleteEmail] = useState();
  const [deleteFailed, setDeleteFailed] = useState(false);

  const handleDelete = useCatch(async () => {
    if (deleteEmail === currentUser.email) {
      setDeleteFailed(false);
        const response = await fetch(`/api/users/${currentUser.id}`, {
method: 'DELETE' });
      if (response.ok) {
        navigate('/login');
        dispatch(sessionActions.updateUser(null));
      } else {
        throw Error(await response.text());
      }
    } else {
      setDeleteFailed(true);
    }
  });
```

```
  const handleGenerateTotp = useCatch(async () => {
    const response = await fetch('/api/users/totp', { method: 'POST' });
    if (response.ok) {
      setItem({ ...item, totpKey: await response.text() });
    } else {
      throw Error(await response.text());
    }
  });

  const query = useQuery();
  const [queryHandled, setQueryHandled] = useState(false);
  const attribute = query.get('attribute');

  useEffect(() => {
    if (!queryHandled && item && attribute) {
      if (!item.attributes.hasOwnProperty('attribute')) {
        const updatedAttributes = { ...item.attributes };
        updatedAttributes[attribute] = '';
        setItem({ ...item, attributes: updatedAttributes });
      }
      setQueryHandled(true);
    }
  }, [item, queryHandled, setQueryHandled, attribute]);

  const onItemSaved = (result) => {
    if (result.id === currentUser.id) {
      dispatch(sessionActions.updateUser(result));
    }
  };

  const validate = () => item && item.name && item.email && (item.id || item.password) && (admin || !totpForce || item.totpKey);

  return (
    <EditItemView
      endpoint="users"
      item={item}
      setItem={setItem}
      defaultItem={admin ? { deviceLimit: -1 } : {}}
      validate={validate}
      onItemSaved={onItemSaved}
```

```
      menu={<SettingsMenu />}
      breadcrumbs={['settingsTitle', 'settingsUser']}
    >
      {item && (
        <>
          <Accordion defaultExpanded={!attribute}>
            <AccordionSummary expandIcon={<ExpandMoreIcon />}>
              <Typography variant="subtitle1">
                {t('sharedRequired')}
              </Typography>
            </AccordionSummary>
            <AccordionDetails className={classes.details}>
              <TextField
                value={item.name || ''}
                            onChange={(e) => setItem({ ...item, name:
e.target.value })}
                label={t('sharedName')}
              />
              <TextField
                value={item.email || ''}
                           onChange={(e) => setItem({ ...item, email:
e.target.value })}
                label={t('userEmail')}
                disabled={fixedEmail && item.id === currentUser.id}
              />
              {!openIdForced && (
                <TextField
                  type="password"
                         onChange={(e) => setItem({ ...item, password:
e.target.value })}
                  label={t('userPassword')}
                />
              )}
              {totpEnable && (
                <FormControl>
                  <InputLabel>{t('loginTotpKey')}</InputLabel>
                  <OutlinedInput
                    readOnly
                    label={t('loginTotpKey')}
                    value={item.totpKey || ''}
                    endAdornment={(
                      <InputAdornment position="end">
```

```
                                    <IconButton size="small" edge="end"
onClick={handleGenerateTotp}>
                                      <CachedIcon fontSize="small" />
                                    </IconButton>
                                     <IconButton size="small" edge="end" onClick={()
=> setItem({ ...item, totpKey: null })}>
                                      <CloseIcon fontSize="small" />
                                    </IconButton>
                                  </InputAdornment>
                                )}
                              />
                            </FormControl>
                          )}
                        </AccordionDetails>
                      </Accordion>
                      <Accordion>
                        <AccordionSummary expandIcon={<ExpandMoreIcon />}>
                          <Typography variant="subtitle1">
                            {t('sharedPreferences')}
                          </Typography>
                        </AccordionSummary>
                        <AccordionDetails className={classes.details}>
                          <TextField
                            value={item.phone || ''}
                                      onChange={(e) => setItem({ ...item, phone:
e.target.value })}
                            label={t('sharedPhone')}
                          />
                          <FormControl>
                            <InputLabel>{t('mapDefault')}</InputLabel>
                            <Select
                              label={t('mapDefault')}
                              value={item.map || 'locationIqStreets'}
                                        onChange={(e) => setItem({ ...item, map:
e.target.value })}
                            >
                                                {mapStyles.filter((style) =>
style.available).map((style) => (
                                <MenuItem key={style.id} value={style.id}>
                                                                    <Typography
component="span">{style.title}</Typography>
                                </MenuItem>
                              ))}
```

```
                </Select>
              </FormControl>
              <FormControl>

<InputLabel>{t('settingsCoordinateFormat')}</InputLabel>
                <Select
                  label={t('settingsCoordinateFormat')}
                  value={item.coordinateFormat || 'dd'}
                   onChange={(e) => setItem({ ...item, coordinateFormat:
e.target.value })}
                >
                                                                    <MenuItem
value="dd">{t('sharedDecimalDegrees')}</MenuItem>
                                                                    <MenuItem
value="ddm">{t('sharedDegreesDecimalMinutes')}</MenuItem>
                                                                    <MenuItem
value="dms">{t('sharedDegreesMinutesSeconds')}</MenuItem>
                </Select>
              </FormControl>
              <FormControl>
                <InputLabel>{t('settingsSpeedUnit')}</InputLabel>
                <Select
                  label={t('settingsSpeedUnit')}
                   value={(item.attributes && item.attributes.speedUnit)
|| 'kn'}
                      onChange={(e) => setItem({ ...item, attributes: {
...item.attributes, speedUnit: e.target.value } })}
                >
                  <MenuItem value="kn">{t('sharedKn')}</MenuItem>
                  <MenuItem value="kmh">{t('sharedKmh')}</MenuItem>
                  <MenuItem value="mph">{t('sharedMph')}</MenuItem>
                </Select>
              </FormControl>
              <FormControl>
                <InputLabel>{t('settingsDistanceUnit')}</InputLabel>
                <Select
                  label={t('settingsDistanceUnit')}
                                                value={(item.attributes   &&
item.attributes.distanceUnit) || 'km'}
                      onChange={(e) => setItem({ ...item, attributes: {
...item.attributes, distanceUnit: e.target.value } })}
                >
                  <MenuItem value="km">{t('sharedKm')}</MenuItem>
```

```
                <MenuItem value="mi">{t('sharedMi')}</MenuItem>
                <MenuItem value="nmi">{t('sharedNmi')}</MenuItem>
              </Select>
            </FormControl>
            <FormControl>
              <InputLabel>{t('settingsAltitudeUnit')}</InputLabel>
              <Select
                label={t('settingsAltitudeUnit')}
                                        value={(item.attributes   &&
item.attributes.altitudeUnit) || 'm'}
                    onChange={(e) => setItem({ ...item, attributes: {
...item.attributes, altitudeUnit: e.target.value } })}
              >
                <MenuItem value="m">{t('sharedMeters')}</MenuItem>
                <MenuItem value="ft">{t('sharedFeet')}</MenuItem>
              </Select>
            </FormControl>
            <FormControl>
              <InputLabel>{t('settingsVolumeUnit')}</InputLabel>
              <Select
                label={t('settingsVolumeUnit')}
                                        value={(item.attributes   &&
item.attributes.volumeUnit) || 'ltr'}
                    onChange={(e) => setItem({ ...item, attributes: {
...item.attributes, volumeUnit: e.target.value } })}
              >
                <MenuItem value="ltr">{t('sharedLiter')}</MenuItem>
                                                        <MenuItem
value="usGal">{t('sharedUsGallon')}</MenuItem>
                                                        <MenuItem
value="impGal">{t('sharedImpGallon')}</MenuItem>
              </Select>
            </FormControl>
            <SelectField
              value={item.attributes && item.attributes.timezone}
                   onChange={(e) => setItem({ ...item, attributes: {
...item.attributes, timezone: e.target.value } })}
              endpoint="/api/server/timezones"
              keyGetter={(it) => it}
              titleGetter={(it) => it}
              label={t('sharedTimezone')}
            />
            <TextField
```

```
                value={item.poiLayer || ''}
                        onChange={(e) => setItem({ ...item, poiLayer:
e.target.value })}
                label={t('mapPoiLayer')}
              />
            </AccordionDetails>
          </Accordion>
          <Accordion>
            <AccordionSummary expandIcon={<ExpandMoreIcon />}>
              <Typography variant="subtitle1">
                {t('sharedLocation')}
              </Typography>
            </AccordionSummary>
            <AccordionDetails className={classes.details}>
              <TextField
                type="number"
                value={item.latitude || 0}
                        onChange={(e) => setItem({ ...item, latitude:
Number(e.target.value) })}
                label={t('positionLatitude')}
              />
              <TextField
                type="number"
                value={item.longitude || 0}
                       onChange={(e) => setItem({ ...item, longitude:
Number(e.target.value) })}
                label={t('positionLongitude')}
              />
              <TextField
                type="number"
                value={item.zoom || 0}
                           onChange={(e) => setItem({ ...item, zoom:
Number(e.target.value) })}
                label={t('serverZoom')}
              />
              <Button
                variant="outlined"
                color="primary"
                onClick={() => {
                  const { lng, lat } = map.getCenter();
                  setItem({
                    ...item,
                    latitude: Number(lat.toFixed(6)),
```

```
                    longitude: Number(lng.toFixed(6)),
                    zoom: Number(map.getZoom().toFixed(1)),
                  });
                }}
              >
                {t('mapCurrentLocation')}
              </Button>
            </AccordionDetails>
          </Accordion>
          <Accordion>
            <AccordionSummary expandIcon={<ExpandMoreIcon />}>
              <Typography variant="subtitle1">
                {t('sharedPermissions')}
              </Typography>
            </AccordionSummary>
            <AccordionDetails className={classes.details}>
              <TextField
                label={t('userExpirationTime')}
                type="date"
                                        value={(item.expirationTime   &&
dayjs(item.expirationTime).locale('en').format('YYYY-MM-DD'))          ||
'2099-01-01'}
                    onChange={(e) => setItem({ ...item, expirationTime:
dayjs(e.target.value, 'YYYY-MM-DD').locale('en').format() })}
                disabled={!manager}
              />
              <TextField
                type="number"
                value={item.deviceLimit || 0}
                      onChange={(e) => setItem({ ...item, deviceLimit:
Number(e.target.value) })}
                label={t('userDeviceLimit')}
                disabled={!admin}
              />
              <TextField
                type="number"
                value={item.userLimit || 0}
                        onChange={(e) => setItem({ ...item, userLimit:
Number(e.target.value) })}
                label={t('userUserLimit')}
                disabled={!admin}
              />
              <FormGroup>
```

```
                <FormControlLabel
                            control={<Checkbox  checked={item.disabled}
onChange={(e) => setItem({ ...item, disabled: e.target.checked })} />}
                  label={t('sharedDisabled')}
                  disabled={!manager}
                />
                <FormControlLabel
                        control={<Checkbox  checked={item.administrator}
onChange={(e) => setItem({ ...item, administrator: e.target.checked })}
/>}
                  label={t('userAdmin')}
                  disabled={!admin}
                />
                <FormControlLabel
                            control={<Checkbox  checked={item.readonly}
onChange={(e) => setItem({ ...item, readonly: e.target.checked })} />}
                  label={t('serverReadonly')}
                  disabled={!manager}
                />
                <FormControlLabel
                       control={<Checkbox  checked={item.deviceReadonly}
onChange={(e)  =>  setItem({  ...item,  deviceReadonly:  e.target.checked
})} />}
                  label={t('userDeviceReadonly')}
                  disabled={!manager}
                />
                <FormControlLabel
                        control={<Checkbox  checked={item.limitCommands}
onChange={(e) => setItem({ ...item, limitCommands: e.target.checked })}
/>}
                  label={t('userLimitCommands')}
                  disabled={!manager}
                />
                <FormControlLabel
                       control={<Checkbox  checked={item.disableReports}
onChange={(e)  =>  setItem({  ...item,  disableReports:  e.target.checked
})} />}
                  label={t('userDisableReports')}
                  disabled={!manager}
                />
                <FormControlLabel
```

```
                          control={<Checkbox checked={item.fixedEmail}
onChange={(e) => setItem({ ...item, fixedEmail: e.target.checked })}
/>}
                  label={t('userFixedEmail')}
                  disabled={!manager}
                />
              </FormGroup>
            </AccordionDetails>
          </Accordion>
          <EditAttributesAccordion
            attribute={attribute}
            attributes={item.attributes}
                    setAttributes={(attributes) => setItem({ ...item,
attributes })}
              definitions={{ ...commonUserAttributes, ...userAttributes
}}
            focusAttribute={attribute}
          />
                {registrationEnabled && item.id === currentUser.id &&
!manager && (
            <Accordion>
              <AccordionSummary expandIcon={<ExpandMoreIcon />}>
                <Typography variant="subtitle1" color="error">
                  {t('userDeleteAccount')}
                </Typography>
              </AccordionSummary>
              <AccordionDetails className={classes.details}>
                <TextField
                  value={deleteEmail}
                  onChange={(e) => setDeleteEmail(e.target.value)}
                  label={t('userEmail')}
                  error={deleteFailed}
                />
                <Button
                  variant="outlined"
                  color="error"
                  onClick={handleDelete}
                  startIcon={<DeleteForeverIcon />}
                >
                  {t('userDeleteAccount')}
                </Button>
              </AccordionDetails>
            </Accordion>
```

```
          )}
        </>
      )}
    </EditItemView>
  );
};

export default UserPage;
```

Componente UserPage

Este componente se encarga de la gestión y edición de la información de usuario en una aplicación React. Utiliza varios módulos y librerías para manejar el estado, las solicitudes a la API, y la interfaz de usuario. Aquí está un resumen de sus partes principales:

1. Importaciones y Setup:
    - Se importan varias librerías y componentes necesarios, como React, useState, useEffect, useNavigate, useParams, y varios componentes de Material-UI.
    - Se importan funciones y utilidades específicas para manejar el estado y la autorización, así como componentes específicos de la aplicación como EditItemView y SettingsMenu.
2. Estado y Variables:
    - currentUser, registrationEnabled, openIdForced, totpEnable, y totpForce se obtienen del estado global usando useSelector.
    - item es el estado local que almacena los datos del usuario actual o el usuario especificado en la URL.
3. Funciones de Manejo:
    - handleDelete: Elimina la cuenta del usuario si el email proporcionado coincide con el email del usuario actual.
    - handleGenerateTotp: Genera una nueva clave TOTP (One Time Password) para el usuario.
    - onItemSaved: Actualiza el estado global si el usuario actual es el que se ha guardado.
    - validate: Verifica si todos los campos requeridos están completos.
4. Efectos Secundarios:
    - useEffect: Maneja la actualización de atributos del usuario si el parámetro attribute está presente en la URL.
5. Renderizado:
    - Utiliza el componente EditItemView para renderizar el formulario de edición.

- Contiene varias secciones (Accordion) para diferentes tipos de información, como datos requeridos, preferencias, ubicación y permisos:
  - Datos Requeridos: Nombre, email, contraseña, clave TOTP.
  - Preferencias: Teléfono, estilo de mapa, formato de coordenadas, unidades de velocidad, distancia, altitud y volumen.
  - Ubicación: Latitud, longitud, zoom y botón para establecer la ubicación actual del mapa.
  - Permisos: Fecha de expiración, límites de dispositivos y usuarios, y varias opciones de permisos.
- Eliminación de Cuenta: Opción para eliminar la cuenta si el usuario está habilitado para hacerlo.

6. Componentes Adicionales:
   - EditAttributesAccordion: Permite editar atributos adicionales del usuario.
   - SettingsMenu: Menú de configuración.

UsersPage.jsx

```
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import {
      Table,   TableRow,   TableCell,   TableHead,   TableBody,   Switch,
TableFooter, FormControlLabel,
} from '@mui/material';
import LoginIcon from '@mui/icons-material/Login';
import LinkIcon from '@mui/icons-material/Link';
import { useCatch, useEffectAsync } from '../reactHelper';
import { formatBoolean, formatTime } from '../common/util/formatter';
import            {            useTranslation            }            from
'../common/components/LocalizationProvider';
import PageLayout from '../common/components/PageLayout';
import SettingsMenu from './components/SettingsMenu';
import CollectionFab from './components/CollectionFab';
import CollectionActions from './components/CollectionActions';
import TableShimmer from '../common/components/TableShimmer';
import { useManager } from '../common/util/permissions';
import       SearchHeader,       {       filterByKeyword       }       from
'./components/SearchHeader';
import useSettingsStyles from './common/useSettingsStyles';

const UsersPage = () => {
  const classes = useSettingsStyles();
  const navigate = useNavigate();
```

```
  const t = useTranslation();

  const manager = useManager();

  const [timestamp, setTimestamp] = useState(Date.now());
  const [items, setItems] = useState([]);
  const [searchKeyword, setSearchKeyword] = useState('');
  const [loading, setLoading] = useState(false);
  const [temporary, setTemporary] = useState(false);

  const handleLogin = useCatch(async (userId) => {
    const response = await fetch(`/api/session/${userId}`);
    if (response.ok) {
      window.location.replace('/');
    } else {
      throw Error(await response.text());
    }
  });

  const actionLogin = {
    key: 'login',
    title: t('loginLogin'),
    icon: <LoginIcon fontSize="small" />,
    handler: handleLogin,
  };

  const actionConnections = {
    key: 'connections',
    title: t('sharedConnections'),
    icon: <LinkIcon fontSize="small" />,
                                    handler:          (userId)          =>
navigate(`/settings/user/${userId}/connections`),
  };

  useEffectAsync(async () => {
    setLoading(true);
    try {
      const response = await fetch('/api/users');
      if (response.ok) {
        setItems(await response.json());
      } else {
        throw Error(await response.text());
      }
```

```
    } finally {
      setLoading(false);
    }
  }, [timestamp]);

  return (
    <PageLayout menu={<SettingsMenu />} breadcrumbs={['settingsTitle',
'settingsUsers']}>
                              <SearchHeader     keyword={searchKeyword}
setKeyword={setSearchKeyword} />
      <Table className={classes.table}>
        <TableHead>
          <TableRow>
            <TableCell>{t('sharedName')}</TableCell>
            <TableCell>{t('userEmail')}</TableCell>
            <TableCell>{t('userAdmin')}</TableCell>
            <TableCell>{t('sharedDisabled')}</TableCell>
            <TableCell>{t('userExpirationTime')}</TableCell>
            <TableCell className={classes.columnAction} />
          </TableRow>
        </TableHead>
        <TableBody>
                    {!loading  ?  items.filter((u)  =>  temporary  ||
!u.temporary).filter(filterByKeyword(searchKeyword)).map((item) => (
            <TableRow key={item.id}>
              <TableCell>{item.name}</TableCell>
              <TableCell>{item.email}</TableCell>
                          <TableCell>{formatBoolean(item.administrator,
t)}</TableCell>
              <TableCell>{formatBoolean(item.disabled, t)}</TableCell>
                            <TableCell>{formatTime(item.expirationTime,
'date')}</TableCell>
                           <TableCell  className={classes.columnAction}
padding="none">
                <CollectionActions
                  itemId={item.id}
                  editPath="/settings/user"
                  endpoint="users"
                  setTimestamp={setTimestamp}
                               customActions={manager  ?  [actionLogin,
actionConnections] : [actionConnections]}
                />
              </TableCell>
```

```
            </TableRow>
          )) : (<TableShimmer columns={6} endAction />)}
        </TableBody>
        <TableFooter>
          <TableRow>
            <TableCell colSpan={6} align="right">
              <FormControlLabel
                control={(
                  <Switch
                    value={temporary}
                    onChange={(e) => setTemporary(e.target.checked)}
                    size="small"
                  />
                )}
                label={t('userTemporary')}
                labelPlacement="start"
              />
            </TableCell>
          </TableRow>
        </TableFooter>
      </Table>
      <CollectionFab editPath="/settings/user" />
    </PageLayout>
  );
};

export default UsersPage;
```

Explicación del Componente UsersPage

El componente UsersPage es una página de React que muestra una lista de usuarios y permite realizar acciones sobre ellos. Utiliza Material-UI para el diseño y la interfaz de usuario, y maneja el estado y las solicitudes de datos con React y hooks personalizados. Aquí está un resumen de su funcionalidad y estructura:

1. Importaciones

- Librerías de React y Material-UI: Para manejar el estado, la navegación, y los componentes de la interfaz.
- Componentes personalizados y utilidades:
    - PageLayout, SettingsMenu, CollectionFab, CollectionActions, TableShimmer, SearchHeader para la estructura y funcionalidad.
    - useCatch, useEffectAsync para manejo de errores y efectos asíncronos.

- formatBoolean, formatTime para formatear datos.
- useTranslation para traducción de textos.
- useManager para verificar permisos.

2. Estado y Funciones

- Estado Local:
  - timestamp: Para forzar la actualización de datos.
  - items: Almacena los datos de usuarios obtenidos de la API.
  - searchKeyword: Palabra clave para filtrar los usuarios en la tabla.
  - loading: Indica si los datos se están cargando.
  - temporary: Booleano para filtrar usuarios temporales.
- Funciones:
  - handleLogin: Maneja el inicio de sesión del usuario, redirigiendo a la página principal si la operación es exitosa.
  - actionLogin y actionConnections: Definen acciones que se pueden realizar en cada usuario de la lista, como iniciar sesión o ver conexiones.

3. Uso de useEffectAsync

- Efecto Asíncrono:
  - Se carga la lista de usuarios desde la API cuando el componente se monta o cuando timestamp cambia.
  - Actualiza el estado items con los datos obtenidos o maneja errores si la solicitud falla.

4. Renderizado

- Estructura del Componente:
  - Encabezado de Página: Usa PageLayout para envolver el contenido y proporcionar un menú y migas de pan.
  - Encabezado de Búsqueda: Usa SearchHeader para permitir la búsqueda de usuarios.
  - Tabla de Usuarios:
    - Encabezado de la Tabla: Muestra las columnas para el nombre, email, administrador, estado desactivado y tiempo de expiración.
    - Cuerpo de la Tabla: Muestra los usuarios filtrados por palabra clave y si son temporales, si el filtro temporary está activado. Usa TableShimmer para mostrar un efecto de carga mientras se obtienen los datos.
    - Pie de Tabla: Incluye un Switch para filtrar usuarios temporales.
  - Botón Flotante: CollectionFab proporciona un botón para agregar o editar usuarios.

5. Acciones en la Tabla

- Acciones Personalizadas:

- CollectionActions permite realizar acciones específicas sobre cada usuario, como iniciar sesión (actionLogin) o ver conexiones (actionConnections).

Session.jsx

```
import { createSlice } from '@reduxjs/toolkit';

const { reducer, actions } = createSlice({
  name: 'session',
  initialState: {
    server: null,
    user: null,
    socket: null,
    includeLogs: false,
    logs: [],
    positions: {},
    history: {},
  },
  reducers: {
    updateServer(state, action) {
      state.server = action.payload;
    },
    updateUser(state, action) {
      state.user = action.payload;
    },
    updateSocket(state, action) {
      state.socket = action.payload;
    },
    enableLogs(state, action) {
      state.includeLogs = action.payload;
      if (!action.payload) {
        state.logs = [];
      }
    },
    updateLogs(state, action) {
      state.logs.push(...action.payload);
    },
    updatePositions(state, action) {
          const liveRoutes = state.user.attributes.mapLiveRoutes ||
state.server.attributes.mapLiveRoutes || 'none';
                                    const liveRoutesLimit =
state.user.attributes['web.liveRouteLength'] ||
state.server.attributes['web.liveRouteLength'] || 10;
      action.payload.forEach((position) => {
```

```
        state.positions[position.deviceId] = position;
        if (liveRoutes !== 'none') {
          const route = state.history[position.deviceId] || [];
          const last = route.at(-1);
            if (!last || (last[0] !== position.longitude && last[1] !==
position.latitude)) {
                state.history[position.deviceId] = [...route.slice(1 -
liveRoutesLimit), [position.longitude, position.latitude]];
          }
        } else {
          state.history = {};
        }
      });
    },
  },
});

export { actions as sessionActions };
export { reducer as sessionReducer };
```

Explicación del Slice de Redux para la Sesión

El código que has proporcionado define un slice de Redux usando createSlice de @reduxjs/toolkit. Este slice se encarga de manejar el estado relacionado con la sesión de usuario, incluyendo el servidor, el usuario, el socket de conexión, los logs, las posiciones y el historial de rutas. A continuación, se detalla cada parte del código:

1. Definición del Slice

- name: Define el nombre del slice. En este caso, se llama 'session'.
- initialState: Especifica el estado inicial del slice. Este estado contiene:
    - server: Información sobre el servidor.
    - user: Información sobre el usuario.
    - socket: La conexión del socket.
    - includeLogs: Un booleano que indica si los logs deben ser incluidos.
    - logs: Un array para almacenar los logs.
    - positions: Un objeto para almacenar las posiciones de los dispositivos.
    - history: Un objeto que guarda el historial de rutas para cada dispositivo.

2. Reducers

Los reducers son funciones que modifican el estado basado en las acciones que se envían. Aquí están los reducers definidos en este slice:

- updateServer(state, action): Actualiza el estado del servidor con el payload de la acción.
- updateUser(state, action): Actualiza el estado del usuario con el payload de la acción.
- updateSocket(state, action): Actualiza el estado del socket con el payload de la acción.
- enableLogs(state, action): Habilita o deshabilita los logs basado en el payload. Si los logs son deshabilitados (false), también limpia el array de logs.
- updateLogs(state, action): Agrega los nuevos logs al array de logs.
- updatePositions(state, action): Actualiza las posiciones de los dispositivos y el historial de rutas:
    - Actualiza la posición del dispositivo en state.positions.
    - Si liveRoutes no es 'none', actualiza el historial de rutas para cada dispositivo. Si la ruta es demasiado larga, mantiene solo las últimas liveRoutesLimit posiciones.

3. Exportaciones

- sessionActions: Exporta las acciones generadas por createSlice. Estas acciones se usan para despachar cambios en el estado desde los componentes.
- sessionReducer: Exporta el reducer generado por createSlice. Este reducer se debe añadir al store de Redux para manejar el estado relacionado con la sesión.

Uso

Este slice puede ser integrado en el store de Redux para gestionar el estado de la sesión en una aplicación. Los componentes pueden despachar acciones para actualizar el estado de la sesión y suscribirse a los cambios en el estado usando useSelector de react-redux.

CacheGraph.java

```
/*
 * Copyright 2023 Anton Tananaev (anton@traccar.org)
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
  *  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
```

```java
package org.traccar.session.cache;

import org.traccar.model.BaseModel;

import java.util.HashMap;
import java.util.Map;
import java.util.Set;
import java.util.stream.Stream;

public class CacheGraph {

    private final Map<CacheKey, CacheNode> roots = new HashMap<>();
        private final WeakValueMap<CacheKey, CacheNode> nodes = new
WeakValueMap<>();

    void addObject(BaseModel value) {
        CacheKey key = new CacheKey(value);
        CacheNode node = new CacheNode(value);
        roots.put(key, node);
        nodes.put(key, node);
    }

    void removeObject(Class<? extends BaseModel> clazz, long id) {
        CacheKey key = new CacheKey(clazz, id);
        CacheNode node = nodes.remove(key);
        if (node != null) {
                                node.getAllLinks(false).forEach(child ->
child.getLinks(key.clazz(), true).remove(node));
        }
        roots.remove(key);
    }

    @SuppressWarnings("unchecked")
    <T extends BaseModel> T getObject(Class<T> clazz, long id) {
        CacheNode node = nodes.get(new CacheKey(clazz, id));
        return node != null ? (T) node.getValue() : null;
    }

    <T extends BaseModel> Stream<T> getObjects(
            Class<? extends BaseModel> fromClass, long fromId,
               Class<T> clazz, Set<Class<? extends BaseModel>> proxies,
boolean forward) {
```

```
                CacheNode rootNode = nodes.get(new CacheKey(fromClass,
fromId));
        if (rootNode != null) {
            return getObjectStream(rootNode, clazz, proxies, forward);
        } else {
            return Stream.empty();
        }
    }

    @SuppressWarnings("unchecked")
    private <T extends BaseModel> Stream<T> getObjectStream(
                CacheNode rootNode, Class<T> clazz, Set<Class<? extends
BaseModel>> proxies, boolean forward) {

        if (proxies.contains(clazz)) {
            return Stream.empty();
        }

        var directSteam = rootNode.getLinks(clazz, forward).stream()
                .map(node -> (T) node.getValue());

        var proxyStream = proxies.stream()
                    .flatMap(proxyClass -> rootNode.getLinks(proxyClass,
forward).stream()
                            .flatMap(node -> getObjectStream(node, clazz,
proxies, forward)));

        return Stream.concat(directSteam, proxyStream);
    }

    void updateObject(BaseModel value) {
        CacheNode node = nodes.get(new CacheKey(value));
        if (node != null) {
            node.setValue(value);
        }
    }

    boolean addLink(
            Class<? extends BaseModel> fromClazz, long fromId,
            BaseModel toValue) {
        boolean stop = true;
                CacheNode fromNode = nodes.get(new CacheKey(fromClazz,
fromId));
```

```
        if (fromNode != null) {
            CacheKey toKey = new CacheKey(toValue);
            CacheNode toNode = nodes.get(toKey);
            if (toNode == null) {
                stop = false;
                toNode = new CacheNode(toValue);
                nodes.put(toKey, toNode);
            }
            fromNode.getLinks(toValue.getClass(), true).add(toNode);
            toNode.getLinks(fromClazz, false).add(fromNode);
        }
        return stop;
    }

    void removeLink(
            Class<? extends BaseModel> fromClazz, long fromId,
            Class<? extends BaseModel> toClazz, long toId) {
               CacheNode  fromNode  =  nodes.get(new  CacheKey(fromClazz,
fromId));
        if (fromNode != null) {
            CacheNode toNode = nodes.get(new CacheKey(toClazz, toId));
            if (toNode != null) {
                fromNode.getLinks(toClazz, true).remove(toNode);
                toNode.getLinks(fromClazz, false).remove(fromNode);
            }
        }
    }

    @Override
    public String toString() {
        StringBuilder stringBuilder = new StringBuilder();
        for (CacheNode node : roots.values()) {
            printNode(stringBuilder, node, "");
        }
        return stringBuilder.toString().trim();
    }

    private void printNode(StringBuilder stringBuilder, CacheNode node,
String indentation) {
        stringBuilder
                .append('\n')
                .append(indentation)
                .append(node.getValue().getClass().getSimpleName())
```

```
.append('(').append(node.getValue().getId()).append(')');
                                node.getAllLinks(true).forEach(child    ->
printNode(stringBuilder, child, indentation + "   "));
    }

}
```

1. Estructura de la Clase

- roots: Un HashMap que guarda los nodos raíz del grafo de caché. La clave es un CacheKey y el valor es un CacheNode.
- nodes: Un WeakValueMap que almacena nodos de caché, usando claves de tipo CacheKey. Esta estructura es útil para evitar fugas de memoria, ya que los nodos serán eliminados automáticamente cuando no haya más referencias a ellos.

2. Métodos

- addObject(BaseModel value): Añade un objeto al grafo de caché. Crea un nuevo CacheNode para el objeto y lo añade tanto a roots como a nodes.
- removeObject(Class<? extends BaseModel> clazz, long id): Elimina un objeto del grafo de caché, usando su clase e ID. También elimina los enlaces relacionados del nodo.
- <T extends BaseModel> T getObject(Class<T> clazz, long id): Recupera un objeto del grafo de caché dado su tipo de clase e ID. Devuelve el objeto si existe, o null si no se encuentra.
- <T extends BaseModel> Stream<T> getObjects(...): Recupera un flujo de objetos relacionados a partir de un nodo raíz dado su clase y ID. También toma en cuenta las clases de proxy y la dirección de la búsqueda.
- <T extends BaseModel> Stream<T> getObjectStream(...): Recupera un flujo de objetos a partir de un nodo raíz, evitando ciclos usando un conjunto de clases proxy.
- updateObject(BaseModel value): Actualiza un objeto en el grafo de caché si ya existe, ajustando su valor.
- boolean addLink(...): Añade un enlace entre dos nodos en el grafo. Si el nodo destino no existe, se crea uno nuevo. Devuelve true si se detiene la creación de enlaces (lo que puede ocurrir si el nodo ya está en el grafo), o false si se crea un nuevo nodo.
- void removeLink(...): Elimina un enlace entre dos nodos del grafo.
- @Override public String toString(): Devuelve una representación en cadena del grafo, mostrando todos los nodos y sus enlaces.
- private void printNode(StringBuilder stringBuilder, CacheNode node, String indentation): Método auxiliar para imprimir la representación del grafo en formato de texto.

Consideraciones Adicionales

- CacheKey: Se utiliza para identificar nodos en el grafo. Se asume que esta clase implementa la comparación adecuada para los objetos CacheKey y maneja el hash de manera eficiente.
- CacheNode: Representa un nodo en el grafo de caché. Los métodos como getLinks y getValue se utilizan para gestionar las conexiones y el valor del nodo.
- WeakValueMap: Se asume que esta clase implementa un mapa con valores de tipo WeakReference, lo que significa que los valores serán eliminados si no hay referencias fuertes a ellos.

CacheKey.java

```java
package org.traccar.session.cache;

import org.traccar.model.BaseModel;

record CacheKey(Class<? extends BaseModel> clazz, long id) {
    CacheKey(BaseModel object) {
        this(object.getClass(), object.getId());
    }
}
```

Componentes del Record

- record CacheKey(...): La palabra clave record se usa para definir un record en Java. Los records son una forma concisa de crear clases inmutables que se utilizan principalmente para transportar datos. Un record automáticamente proporciona un constructor, métodos getter, equals(), hashCode(), y toString().
- Class<? extends BaseModel> clazz: Este es un campo del record que almacena la clase del objeto base (de tipo BaseModel). Utiliza generics para permitir cualquier subclase de BaseModel.
- long id: Este es otro campo del record que almacena el identificador del objeto.
- CacheKey(BaseModel object): Este es un constructor adicional que permite crear un CacheKey a partir de un objeto de tipo BaseModel. Utiliza el constructor primario para inicializar los campos clazz y id del record.

CacheManager.java

```java
/*
 * Copyright 2022 - 2024 Anton Tananaev (anton@traccar.org)
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
```

```
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.traccar.session.cache;

import jakarta.inject.Inject;
import jakarta.inject.Singleton;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.traccar.broadcast.BroadcastInterface;
import org.traccar.broadcast.BroadcastService;
import org.traccar.config.Config;
import org.traccar.model.Attribute;
import org.traccar.model.BaseModel;
import org.traccar.model.Calendar;
import org.traccar.model.Device;
import org.traccar.model.Driver;
import org.traccar.model.Geofence;
import org.traccar.model.Group;
import org.traccar.model.GroupedModel;
import org.traccar.model.Maintenance;
import org.traccar.model.Notification;
import org.traccar.model.ObjectOperation;
import org.traccar.model.Permission;
import org.traccar.model.Position;
import org.traccar.model.Schedulable;
import org.traccar.model.Server;
import org.traccar.model.User;
import org.traccar.storage.Storage;
import org.traccar.storage.StorageException;
import org.traccar.storage.query.Columns;
import org.traccar.storage.query.Condition;
import org.traccar.storage.query.Request;

import java.util.HashMap;
import java.util.HashSet;
```

```
import java.util.Map;
import java.util.Set;
import java.util.concurrent.locks.ReadWriteLock;
import java.util.concurrent.locks.ReentrantReadWriteLock;
import java.util.stream.Collectors;

@Singleton
public class CacheManager implements BroadcastInterface {

                private     static     final     Logger     LOGGER     =
LoggerFactory.getLogger(CacheManager.class);

          private   static   final   Set<Class<?   extends   BaseModel>>
GROUPED_CLASSES =
                Set.of(Attribute.class, Driver.class, Geofence.class,
Maintenance.class, Notification.class);

    private final Config config;
    private final Storage storage;
    private final BroadcastService broadcastService;

    private final ReadWriteLock lock = new ReentrantReadWriteLock();

    private final CacheGraph graph = new CacheGraph();

    private Server server;
         private   final   Map<Long,   Position>   devicePositions   =   new
HashMap<>();
      private final Map<Long, HashSet<Object>> deviceReferences = new
HashMap<>();

    @Inject
           public   CacheManager(Config   config,   Storage   storage,
BroadcastService broadcastService) throws StorageException {
        this.config = config;
        this.storage = storage;
        this.broadcastService = broadcastService;
             server  =  storage.getObject(Server.class,  new  Request(new
Columns.All()));
        broadcastService.registerListener(this);
    }

    @Override
```

```
    public String toString() {
        return graph.toString();
    }

    public Config getConfig() {
        return config;
    }

    public <T extends BaseModel> T getObject(Class<T> clazz, long id) {
        try {
            lock.readLock().lock();
            return graph.getObject(clazz, id);
        } finally {
            lock.readLock().unlock();
        }
    }

    public <T extends BaseModel> Set<T> getDeviceObjects(long deviceId,
Class<T> clazz) {
        try {
            lock.readLock().lock();
                return graph.getObjects(Device.class, deviceId, clazz,
Set.of(Group.class), true)
                    .collect(Collectors.toUnmodifiableSet());
        } finally {
            lock.readLock().unlock();
        }
    }

    public Position getPosition(long deviceId) {
        try {
            lock.readLock().lock();
            return devicePositions.get(deviceId);
        } finally {
            lock.readLock().unlock();
        }
    }

    public Server getServer() {
        try {
            lock.readLock().lock();
            return server;
        } finally {
```

```
            lock.readLock().unlock();
        }
    }

      public Set<User> getNotificationUsers(long notificationId, long
deviceId) {
        try {
            lock.readLock().lock();
                    Set<User> deviceUsers = getDeviceObjects(deviceId,
User.class);
             return graph.getObjects(Notification.class, notificationId,
User.class, Set.of(), false)
                    .filter(deviceUsers::contains)
                    .collect(Collectors.toUnmodifiableSet());
        } finally {
            lock.readLock().unlock();
        }
    }

    public Set<Notification> getDeviceNotifications(long deviceId) {
        try {
            lock.readLock().lock();
                 var direct = graph.getObjects(Device.class, deviceId,
Notification.class, Set.of(Group.class), true)
                    .map(BaseModel::getId)
                    .collect(Collectors.toUnmodifiableSet());
                       return graph.getObjects(Device.class, deviceId,
Notification.class, Set.of(Group.class, User.class), true)
                     .filter(notification -> notification.getAlways() ||
direct.contains(notification.getId()))
                    .collect(Collectors.toUnmodifiableSet());
        } finally {
            lock.readLock().unlock();
        }
    }

    public void addDevice(long deviceId, Object key) throws Exception {
        try {
            lock.writeLock().lock();
             var references = deviceReferences.computeIfAbsent(deviceId,
k -> new HashSet<>());
            if (references.isEmpty()) {
```

```
                    Device device = storage.getObject(Device.class, new
Request(
                            new Columns.All(), new Condition.Equals("id",
deviceId)));
                graph.addObject(device);
                initializeCache(device);
                if (device.getPositionId() > 0) {
                                        devicePositions.put(deviceId,
storage.getObject(Position.class, new Request(
                                                new Columns.All(), new
Condition.Equals("id", device.getPositionId()))));
                }
            }
            references.add(key);
               LOGGER.debug("Cache add device {} references {} key {}",
deviceId, references.size(), key);
        } finally {
            lock.writeLock().unlock();
        }
    }

    public void removeDevice(long deviceId, Object key) {
        try {
            lock.writeLock().lock();
             var references = deviceReferences.computeIfAbsent(deviceId,
k -> new HashSet<>());
            references.remove(key);
            if (references.isEmpty()) {
                graph.removeObject(Device.class, deviceId);
                devicePositions.remove(deviceId);
                deviceReferences.remove(deviceId);
            }
             LOGGER.debug("Cache remove device {} references {} key {}",
deviceId, references.size(), key);
        } finally {
            lock.writeLock().unlock();
        }
    }

    public void updatePosition(Position position) {
        try {
            lock.writeLock().lock();
            if (deviceReferences.containsKey(position.getDeviceId())) {
```

```
                devicePositions.put(position.getDeviceId(), position);
            }
        } finally {
            lock.writeLock().unlock();
        }
    }

    @Override
    public <T extends BaseModel> void invalidateObject(
                boolean local, Class<T> clazz, long id, ObjectOperation
operation) throws Exception {
        if (local) {
                    broadcastService.invalidateObject(true, clazz, id,
operation);
        }

        if (operation == ObjectOperation.DELETE) {
            graph.removeObject(clazz, id);
        }
        if (operation != ObjectOperation.UPDATE) {
            return;
        }

        if (clazz.equals(Server.class)) {
                server = storage.getObject(Server.class, new Request(new
Columns.All()));
            return;
        }

                var after = storage.getObject(clazz, new Request(new
Columns.All(), new Condition.Equals("id", id)));
        if (after == null) {
            return;
        }
        var before = getObject(after.getClass(), after.getId());
        if (before == null) {
            return;
        }

        if (after instanceof GroupedModel) {
            long beforeGroupId = ((GroupedModel) before).getGroupId();
            long afterGroupId = ((GroupedModel) after).getGroupId();
            if (beforeGroupId != afterGroupId) {
```

```
                if (beforeGroupId > 0) {
                        invalidatePermission(clazz, id, Group.class,
beforeGroupId, false);
                }
                if (afterGroupId > 0) {
                        invalidatePermission(clazz, id, Group.class,
afterGroupId, true);
                }
            }
        } else if (after instanceof Schedulable) {
                            long beforeCalendarId = ((Schedulable)
before).getCalendarId();
                             long afterCalendarId = ((Schedulable)
after).getCalendarId();
            if (beforeCalendarId != afterCalendarId) {
                if (beforeCalendarId > 0) {
                        invalidatePermission(clazz, id, Calendar.class,
beforeCalendarId, false);
                }
                if (afterCalendarId > 0) {
                        invalidatePermission(clazz, id, Calendar.class,
afterCalendarId, true);
                }
            }
            // TODO handle notification always change
        }

        graph.updateObject(after);
    }

    @Override
        public <T1 extends BaseModel, T2 extends BaseModel> void
invalidatePermission(
                 boolean local, Class<T1> clazz1, long id1, Class<T2>
clazz2, long id2, boolean link) throws Exception {
        if (local) {
               broadcastService.invalidatePermission(true, clazz1, id1,
clazz2, id2, link);
        }

                                if    (clazz1.equals(User.class)    &&
GroupedModel.class.isAssignableFrom(clazz2)) {
            invalidatePermission(clazz2, id2, clazz1, id1, link);
```

```
        } else {
            invalidatePermission(clazz1, id1, clazz2, id2, link);
        }
    }

    private <T1 extends BaseModel, T2 extends BaseModel> void
invalidatePermission(
            Class<T1> fromClass, long fromId, Class<T2> toClass, long
toId, boolean link) throws Exception {

                                        boolean groupLink =
GroupedModel.class.isAssignableFrom(fromClass)                    &&
toClass.equals(Group.class);
                                    boolean calendarLink =
Schedulable.class.isAssignableFrom(fromClass)                     &&
toClass.equals(Calendar.class);
                boolean userLink = fromClass.equals(User.class) &&
toClass.equals(Notification.class);

                                    boolean groupedLinks =
GroupedModel.class.isAssignableFrom(fromClass)
                            && (GROUPED_CLASSES.contains(toClass) ||
toClass.equals(User.class));

        if (!groupLink && !calendarLink && !userLink && !groupedLinks)
{
            return;
        }

        if (link) {
            BaseModel object = storage.getObject(toClass, new Request(
                        new Columns.All(), new Condition.Equals("id",
toId)));
            if (!graph.addLink(fromClass, fromId, object)) {
                initializeCache(object);
            }
        } else {
            graph.removeLink(fromClass, fromId, toClass, toId);
        }
    }

    private void initializeCache(BaseModel object) throws Exception {
        if (object instanceof User) {
```

```
                                    for (Permission permission :
storage.getPermissions(User.class, Notification.class)) {
                if (permission.getOwnerId() == object.getId()) {
                    invalidatePermission(
                                            permission.getOwnerClass(),
permission.getOwnerId(),
                                        permission.getPropertyClass(),
permission.getPropertyId(), true);
                }
            }
        } else {
            if (object instanceof GroupedModel groupedModel) {
                long groupId = groupedModel.getGroupId();
                if (groupId > 0) {
                                invalidatePermission(object.getClass(),
object.getId(), Group.class, groupId, true);
                }

                                    for (Permission permission :
storage.getPermissions(User.class, object.getClass())) {
                    if (permission.getPropertyId() == object.getId()) {
                        invalidatePermission(
                                        object.getClass(), object.getId(),
User.class, permission.getOwnerId(), true);
                    }
                }

                            for (Class<? extends BaseModel> clazz :
GROUPED_CLASSES) {
                                        for (Permission permission :
storage.getPermissions(object.getClass(), clazz)) {
                        if (permission.getOwnerId() == object.getId())
{
                            invalidatePermission(
                                        object.getClass(), object.getId(),
clazz, permission.getPropertyId(), true);
                        }
                    }
                }
            }

            if (object instanceof Schedulable schedulable) {
                long calendarId = schedulable.getCalendarId();
```

```
                if (calendarId > 0) {
                                invalidatePermission(object.getClass(),
object.getId(), Calendar.class, calendarId, true);
                }
            }
        }
    }

}
```

Funciones Clave

1. Inicialización y Configuración:
    - Constructor (CacheManager): Configura el CacheManager inyectando las dependencias necesarias (Config, Storage, BroadcastService) y obteniendo el objeto Server de la base de datos. También registra el CacheManager como un listener en el BroadcastService.
2. Obtención de Objetos:
    - getObject(Class<T> clazz, long id): Recupera un objeto específico de la caché basado en su clase y ID.
    - getDeviceObjects(long deviceId, Class<T> clazz): Obtiene un conjunto de objetos relacionados con un dispositivo específico.
3. Gestión de Posiciones:
    - getPosition(long deviceId): Obtiene la posición actual de un dispositivo desde la caché.
    - updatePosition(Position position): Actualiza la posición de un dispositivo en la caché si el dispositivo está registrado.
4. Gestión de Dispositivos:
    - addDevice(long deviceId, Object key): Agrega un dispositivo a la caché y lo inicializa si es necesario. También gestiona las referencias asociadas al dispositivo.
    - removeDevice(long deviceId, Object key): Elimina un dispositivo de la caché y limpia las referencias asociadas si ya no hay otros objetos referenciando al dispositivo.
5. Actualización y Eliminación de Objetos:
    - invalidateObject(boolean local, Class<T> clazz, long id, ObjectOperation operation): Maneja la invalidación de objetos en la caché. Dependiendo de la operación (actualización o eliminación), actualiza o elimina el objeto de la caché y maneja las referencias y permisos asociados.
6. Invalidación de Permisos:

- invalidatePermission(boolean local, Class<T1> clazz1, long id1, Class<T2> clazz2, long id2, boolean link): Actualiza los permisos relacionados con los objetos en la caché, añadiendo o eliminando enlaces de permisos según sea necesario.

7. Inicialización de Caché:
    - initializeCache(BaseModel object): Inicializa la caché para un objeto recién agregado, incluyendo el manejo de permisos y relaciones asociadas.

CacheNode.java

```
package org.traccar.session.cache;

import org.traccar.model.BaseModel;

import java.util.HashSet;
import java.util.Map;
import java.util.Set;
import java.util.concurrent.ConcurrentHashMap;
import java.util.stream.Stream;

public class CacheNode {

    private BaseModel value;

    private final Map<Class<? extends BaseModel>, Set<CacheNode>> links
= new ConcurrentHashMap<>();
    private final Map<Class<? extends BaseModel>, Set<CacheNode>>
backlinks = new ConcurrentHashMap<>();

    public CacheNode(BaseModel value) {
        this.value = value;
    }

    public BaseModel getValue() {
        return value;
    }

    public void setValue(BaseModel value) {
        this.value = value;
    }

    public Set<CacheNode> getLinks(Class<? extends BaseModel> clazz,
boolean forward) {
        var map = forward ? links : backlinks;
```

```
        return map.computeIfAbsent(clazz, k -> new HashSet<>());
    }

    public Stream<CacheNode> getAllLinks(boolean forward) {
        var map = forward ? links : backlinks;
        return map.values().stream().flatMap(Set::stream);
    }

}
```

Componentes Principales

1. Atributos:
    - BaseModel value: El objeto asociado con este nodo. Este objeto es una instancia de BaseModel y se almacena en el nodo.
    - Map<Class<? extends BaseModel>, Set<CacheNode>> links: Un mapa que almacena enlaces desde este nodo hacia otros nodos. La clave es la clase de modelo a la que el enlace está dirigido, y el valor es un conjunto de nodos que están conectados.
    - Map<Class<? extends BaseModel>, Set<CacheNode>> backlinks: Un mapa que almacena enlaces que apuntan de vuelta a este nodo desde otros nodos. Al igual que links, la clave es la clase de modelo y el valor es un conjunto de nodos que están conectados de vuelta a este nodo.
2. Constructor:
    - CacheNode(BaseModel value): Inicializa un nuevo nodo con un objeto BaseModel. Los mapas links y backlinks se inicializan como instancias de ConcurrentHashMap.
3. Métodos:
    - BaseModel getValue(): Devuelve el objeto BaseModel almacenado en el nodo.
    - void setValue(BaseModel value): Establece el objeto BaseModel almacenado en el nodo.
    - Set<CacheNode> getLinks(Class<? extends BaseModel> clazz, boolean forward): Devuelve el conjunto de nodos vinculados a este nodo para una clase específica. Si forward es true, devuelve los nodos a los que este nodo está vinculado; si false, devuelve los nodos que están vinculados de vuelta a este nodo.
    - Stream<CacheNode> getAllLinks(boolean forward): Devuelve un flujo de todos los nodos vinculados a este nodo. Si forward es true, devuelve todos los nodos a los que este nodo está vinculado; si false, devuelve todos los nodos que están vinculados de vuelta a este nodo.

Uso de la Clase CacheNode

- Enlaces y Backlinks: La clase CacheNode permite gestionar tanto los enlaces directos a otros nodos (links) como los enlaces que apuntan de vuelta al nodo actual (backlinks). Esto facilita la navegación y la gestión de relaciones en un grafo de objetos en caché.
- ConcurrentHashMap: Utiliza ConcurrentHashMap para links y backlinks para asegurar que las operaciones en estos mapas sean seguras en un entorno multihilo, permitiendo accesos concurrentes sin problemas.
- Flujos de Datos: Los métodos que devuelven flujos (getAllLinks) permiten una fácil manipulación y procesamiento de los enlaces en el grafo, utilizando las capacidades de la API de Streams de Java.

WeekValueMap.java

```
/*
 * Copyright 2023 Anton Tananaev (anton@traccar.org)
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
  *  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.traccar.session.cache;

import java.lang.ref.WeakReference;
import java.util.HashMap;
import java.util.Map;

public class WeakValueMap<K, V> {

    private final Map<K, WeakReference<V>> map = new HashMap<>();

    public void put(K key, V value) {
        map.put(key, new WeakReference<>(value));
    }

    public V get(K key) {
```

```
        WeakReference<V> weakReference = map.get(key);
        return (weakReference != null) ? weakReference.get() : null;
    }

    public V remove(K key) {
        WeakReference<V> weakReference = map.remove(key);
        return (weakReference != null) ? weakReference.get() : null;
    }

    private void clean() {
            map.entrySet().removeIf(entry -> entry.getValue().get() ==
null);
    }

}
```

Componentes de la Clase WeakValueMap

1. Atributo:
    - Map<K, WeakReference<V>> map: Un mapa que almacena pares clave-valor. Las claves son de tipo K y los valores son referencias débiles (WeakReference) de tipo V. El uso de WeakReference permite que los valores sean recolectados si ya no hay referencias fuertes a ellos, ayudando a evitar fugas de memoria.
2. Métodos:
    - void put(K key, V value):
        - Descripción: Inserta un par clave-valor en el mapa. La clave es key y el valor es value, que se envuelve en una WeakReference.
        - Uso: Utilizado para añadir o actualizar una entrada en el mapa.
    - V get(K key):
        - Descripción: Recupera el valor asociado con la clave proporcionada. Devuelve el valor si está disponible, o null si el valor ha sido recolectado por el recolector de basura o la clave no existe.
        - Uso: Se utiliza para obtener el valor asociado a una clave específica.
    - V remove(K key):
        - Descripción: Elimina la entrada asociada con la clave proporcionada. Devuelve el valor si estaba presente antes de la eliminación, o null si no había ningún valor asociado a esa clave.
        - Uso: Se utiliza para eliminar una entrada del mapa.
    - private void clean():

- Descripción: Elimina las entradas del mapa cuyos valores han sido recolectados por el recolector de basura (es decir, las referencias débiles a estos valores son null).
- Uso: Normalmente, este método se invocaría en un momento apropiado para limpiar el mapa de entradas obsoletas, aunque en esta implementación no se llama explícitamente.

ConnectionKey.java

```
package org.traccar.session;

import io.netty.channel.Channel;

import java.net.SocketAddress;

public record ConnectionKey(SocketAddress localAddress, SocketAddress
remoteAddress) {
    public ConnectionKey(Channel channel, SocketAddress remoteAddress)
{
        this(channel.localAddress(), remoteAddress);
    }
}
```

Componentes del record

1. Campos (Componentes del Record):
    - SocketAddress localAddress: La dirección local del socket, que representa la dirección del servidor o la máquina local.
    - SocketAddress remoteAddress: La dirección remota del socket, que representa la dirección del cliente o de la máquina remota con la que se está comunicando.
2. Constructor Primario:
    - public ConnectionKey(SocketAddress localAddress, SocketAddress remoteAddress): Este es el constructor principal del record, que toma dos parámetros (localAddress y remoteAddress) y los asigna a los campos del record.
3. Constructor Secundario:
    - public ConnectionKey(Channel channel, SocketAddress remoteAddress): Este constructor adicional permite crear un ConnectionKey usando un objeto Channel y una dirección remota. En este constructor:
        - channel.localAddress(): Se obtiene la dirección local del canal (un canal de Netty que generalmente representa una conexión de red).
        - remoteAddress: La dirección remota proporcionada como argumento al constructor.

ConnetionManager.java

```
/*
 * Copyright 2015 - 2024 Anton Tananaev (anton@traccar.org)
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
  *  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.traccar.session;

import io.netty.channel.Channel;
import io.netty.util.Timeout;
import io.netty.util.Timer;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.traccar.Protocol;
import org.traccar.broadcast.BroadcastInterface;
import org.traccar.broadcast.BroadcastService;
import org.traccar.config.Config;
import org.traccar.config.Keys;
import org.traccar.database.DeviceLookupService;
import org.traccar.database.NotificationManager;
import org.traccar.model.BaseModel;
import org.traccar.model.Device;
import org.traccar.model.Event;
import org.traccar.model.LogRecord;
import org.traccar.model.Position;
import org.traccar.model.User;
import org.traccar.session.cache.CacheManager;
import org.traccar.storage.Storage;
import org.traccar.storage.StorageException;
import org.traccar.storage.query.Columns;
import org.traccar.storage.query.Condition;
```

```java
import org.traccar.storage.query.Request;

import jakarta.inject.Inject;
import jakarta.inject.Singleton;
import java.net.InetSocketAddress;
import java.net.SocketAddress;
import java.util.Arrays;
import java.util.Collections;
import java.util.Date;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Objects;
import java.util.Set;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.TimeUnit;
import java.util.stream.Collectors;

@Singleton
public class ConnectionManager implements BroadcastInterface {

                private static final Logger LOGGER =
LoggerFactory.getLogger(ConnectionManager.class);

    private final long deviceTimeout;
    private final boolean showUnknownDevices;

      private final Map<Long, DeviceSession> sessionsByDeviceId = new
ConcurrentHashMap<>();
       private final Map<ConnectionKey, Map<String, DeviceSession>>
sessionsByEndpoint = new ConcurrentHashMap<>();
     private final Map<ConnectionKey, String> unknownByEndpoint = new
ConcurrentHashMap<>();

    private final Config config;
    private final CacheManager cacheManager;
    private final Storage storage;
    private final NotificationManager notificationManager;
    private final Timer timer;
    private final BroadcastService broadcastService;
    private final DeviceLookupService deviceLookupService;
```

```
        private final Map<Long, Set<UpdateListener>> listeners = new
HashMap<>();
    private final Map<Long, Set<Long>> userDevices = new HashMap<>();
    private final Map<Long, Set<Long>> deviceUsers = new HashMap<>();

            private final Map<Long, Timeout> timeouts = new
ConcurrentHashMap<>();

    @Inject
    public ConnectionManager(
            Config config, CacheManager cacheManager, Storage storage,
                 NotificationManager notificationManager, Timer timer,
BroadcastService broadcastService,
            DeviceLookupService deviceLookupService) {
        this.config = config;
        this.cacheManager = cacheManager;
        this.storage = storage;
        this.notificationManager = notificationManager;
        this.timer = timer;
        this.broadcastService = broadcastService;
        this.deviceLookupService = deviceLookupService;
        deviceTimeout = config.getLong(Keys.STATUS_TIMEOUT);
                                                showUnknownDevices      =
config.getBoolean(Keys.WEB_SHOW_UNKNOWN_DEVICES);
        broadcastService.registerListener(this);
    }

    public DeviceSession getDeviceSession(long deviceId) {
        return sessionsByDeviceId.get(deviceId);
    }

    public DeviceSession getDeviceSession(
                    Protocol protocol, Channel channel, SocketAddress
remoteAddress,
            String... uniqueIds) throws Exception {

             ConnectionKey connectionKey = new ConnectionKey(channel,
remoteAddress);
                     Map<String, DeviceSession> endpointSessions =
sessionsByEndpoint.getOrDefault(
                connectionKey, new ConcurrentHashMap<>());
```

```
                                                    uniqueIds       =
Arrays.stream(uniqueIds).filter(Objects::nonNull).toArray(String[]::new
);
        if (uniqueIds.length > 0) {
            for (String uniqueId : uniqueIds) {
                                            DeviceSession   deviceSession   =
endpointSessions.get(uniqueId);
                if (deviceSession != null) {
                    return deviceSession;
                }
            }
        } else {
                                                                    return
endpointSessions.values().stream().findAny().orElse(null);
        }

        Device device = deviceLookupService.lookup(uniqueIds);

        String firstUniqueId = uniqueIds[0];
                                        if     (device     ==     null     &&
config.getBoolean(Keys.DATABASE_REGISTER_UNKNOWN)) {
                                                                        if
(firstUniqueId.matches(config.getString(Keys.DATABASE_REGISTER_UNKNOWN_
REGEX))) {
                device = addUnknownDevice(firstUniqueId);
            }
        }

        if (device != null) {
            unknownByEndpoint.remove(connectionKey);
            device.checkDisabled();

                                            DeviceSession    oldSession    =
sessionsByDeviceId.remove(device.getId());
            if (oldSession != null) {
                        Map<String, DeviceSession> oldEndpointSessions =
sessionsByEndpoint.get(oldSession.getConnectionKey());
                                    if   (oldEndpointSessions   !=   null   &&
oldEndpointSessions.size() > 1) {
                    oldEndpointSessions.remove(device.getUniqueId());
                } else {

sessionsByEndpoint.remove(oldSession.getConnectionKey());
```

```
                }
            }

            DeviceSession deviceSession = new DeviceSession(
                                    device.getId(), device.getUniqueId(),
device.getModel(), protocol, channel, remoteAddress);
            endpointSessions.put(device.getUniqueId(), deviceSession);
            sessionsByEndpoint.put(connectionKey, endpointSessions);
            sessionsByDeviceId.put(device.getId(), deviceSession);

            if (oldSession == null) {
                cacheManager.addDevice(device.getId(), connectionKey);
            }

            return deviceSession;
        } else {
            unknownByEndpoint.put(connectionKey, firstUniqueId);
                  LOGGER.warn("Unknown device - " + String.join(" ",
uniqueIds)
                                          + " (" + ((InetSocketAddress)
remoteAddress).getHostString() + ")");
            return null;
        }
    }

    private Device addUnknownDevice(String uniqueId) {
        Device device = new Device();
        device.setName(uniqueId);
        device.setUniqueId(uniqueId);

device.setCategory(config.getString(Keys.DATABASE_REGISTER_UNKNOWN_DEFA
ULT_CATEGORY));

                                          long     defaultGroupId     =
config.getLong(Keys.DATABASE_REGISTER_UNKNOWN_DEFAULT_GROUP_ID);
        if (defaultGroupId != 0) {
            device.setGroupId(defaultGroupId);
        }

        try {
                 device.setId(storage.addObject(device, new Request(new
Columns.Exclude("id"))));
            LOGGER.info("Automatically registered " + uniqueId);
```

```
            return device;
        } catch (StorageException e) {
            LOGGER.warn("Automatic registration failed", e);
            return null;
        }
    }

          public  void  deviceDisconnected(Channel  channel,  boolean
supportsOffline) {
        SocketAddress remoteAddress = channel.remoteAddress();
        if (remoteAddress != null) {
              ConnectionKey connectionKey = new ConnectionKey(channel,
remoteAddress);
                      Map<String,  DeviceSession>  endpointSessions  =
sessionsByEndpoint.remove(connectionKey);
            if (endpointSessions != null) {
                                  for  (DeviceSession  deviceSession  :
endpointSessions.values()) {
                    if (supportsOffline) {
                                updateDevice(deviceSession.getDeviceId(),
Device.STATUS_OFFLINE, null);
                    }

sessionsByDeviceId.remove(deviceSession.getDeviceId());

cacheManager.removeDevice(deviceSession.getDeviceId(), connectionKey);
                }
            }
            unknownByEndpoint.remove(connectionKey);
        }
    }

    public void deviceUnknown(long deviceId) {
        updateDevice(deviceId, Device.STATUS_UNKNOWN, null);
        removeDeviceSession(deviceId);
    }

    private void removeDeviceSession(long deviceId) {
                                  DeviceSession    deviceSession    =
sessionsByDeviceId.remove(deviceId);
        if (deviceSession != null) {
                                     ConnectionKey   connectionKey   =
deviceSession.getConnectionKey();
```

```
            cacheManager.removeDevice(deviceId, connectionKey);
                 sessionsByEndpoint.computeIfPresent(connectionKey, (e,
sessions) -> {
                sessions.remove(deviceSession.getUniqueId());
                return sessions.isEmpty() ? null : sessions;
            });
        }
    }

    public void updateDevice(long deviceId, String status, Date time) {
        Device device = cacheManager.getObject(Device.class, deviceId);
        if (device == null) {
            try {
                device = storage.getObject(Device.class, new Request(
                            new Columns.All(), new Condition.Equals("id",
deviceId)));
            } catch (StorageException e) {
                LOGGER.warn("Failed to get device", e);
            }
            if (device == null) {
                return;
            }
        }

        String oldStatus = device.getStatus();
        device.setStatus(status);

        if (!status.equals(oldStatus)) {
            String eventType;
            Map<Event, Position> events = new HashMap<>();
            eventType = switch (status) {
                case Device.STATUS_ONLINE -> Event.TYPE_DEVICE_ONLINE;
                                        case   Device.STATUS_UNKNOWN   ->
Event.TYPE_DEVICE_UNKNOWN;
                default -> Event.TYPE_DEVICE_OFFLINE;
            };
            events.put(new Event(eventType, deviceId), null);
            notificationManager.updateEvents(events);
        }

        if (time != null) {
            device.setLastUpdate(time);
        }
```

```
        Timeout timeout = timeouts.remove(deviceId);
        if (timeout != null) {
            timeout.cancel();
        }

        if (status.equals(Device.STATUS_ONLINE)) {
            timeouts.put(deviceId, timer.newTimeout(timeout1 -> {
                if (!timeout1.isCancelled()) {
                    deviceUnknown(deviceId);
                }
            }, deviceTimeout, TimeUnit.SECONDS));
        }

        try {
            storage.updateObject(device, new Request(
                    new Columns.Include("status", "lastUpdate"),
                    new Condition.Equals("id", deviceId)));
        } catch (StorageException e) {
            LOGGER.warn("Update device status error", e);
        }

        updateDevice(true, device);
    }

    public synchronized void sendKeepalive() {
        for (Set<UpdateListener> userListeners : listeners.values()) {
            for (UpdateListener listener : userListeners) {
                listener.onKeepalive();
            }
        }
    }

    @Override
    public synchronized void updateDevice(boolean local, Device device)
{
        if (local) {
            broadcastService.updateDevice(true, device);
        } else if (Device.STATUS_ONLINE.equals(device.getStatus())) {
            timeouts.remove(device.getId());
            removeDeviceSession(device.getId());
        }
```

```
            for (long userId : deviceUsers.getOrDefault(device.getId(),
Collections.emptySet())) {
                if (listeners.containsKey(userId)) {
                    for (UpdateListener listener : listeners.get(userId)) {
                        listener.onUpdateDevice(device);
                    }
                }
            }
        }

        @Override
          public synchronized void updatePosition(boolean local, Position
position) {
            if (local) {
                broadcastService.updatePosition(true, position);
            }
                                                for     (long     userId     :
deviceUsers.getOrDefault(position.getDeviceId(),
Collections.emptySet())) {
                if (listeners.containsKey(userId)) {
                    for (UpdateListener listener : listeners.get(userId)) {
                        listener.onUpdatePosition(position);
                    }
                }
            }
        }

        @Override
         public synchronized void updateEvent(boolean local, long userId,
Event event) {
            if (local) {
                broadcastService.updateEvent(true, userId, event);
            }
            if (listeners.containsKey(userId)) {
                for (UpdateListener listener : listeners.get(userId)) {
                    listener.onUpdateEvent(event);
                }
            }
        }

        @Override
         public synchronized <T1 extends BaseModel, T2 extends BaseModel>
void invalidatePermission(
```

```
                boolean local, Class<T1> clazz1, long id1, Class<T2>
clazz2, long id2, boolean link) {
                    if (link && clazz1.equals(User.class) &&
clazz2.equals(Device.class)) {
            if (listeners.containsKey(id1)) {
                userDevices.get(id1).add(id2);
                deviceUsers.put(id2, new HashSet<>(List.of(id1)));
            }
        }
    }

    public synchronized void updateLog(LogRecord record) {
                                        var sessions =
sessionsByEndpoint.getOrDefault(record.getConnectionKey(), Map.of());
        if (sessions.isEmpty()) {
                                    String unknownUniqueId =
unknownByEndpoint.get(record.getConnectionKey());
            if (unknownUniqueId != null && showUnknownDevices) {
                record.setUniqueId(unknownUniqueId);
                listeners.values().stream()
                        .flatMap(Set::stream)
                                            .forEach((listener) ->
listener.onUpdateLog(record));
            }
        } else {
            var firstEntry = sessions.entrySet().iterator().next();
            record.setUniqueId(firstEntry.getKey());
            record.setDeviceId(firstEntry.getValue().getDeviceId());
                                        for (long userId :
deviceUsers.getOrDefault(record.getDeviceId(), Set.of())) {
                                    for (UpdateListener listener :
listeners.getOrDefault(userId, Set.of())) {
                    listener.onUpdateLog(record);
                }
            }
        }
    }

    public interface UpdateListener {
        void onKeepalive();
        void onUpdateDevice(Device device);
        void onUpdatePosition(Position position);
        void onUpdateEvent(Event event);
```

```
        void onUpdateLog(LogRecord record);
    }

      public synchronized void addListener(long userId, UpdateListener
listener) throws StorageException {
        var set = listeners.get(userId);
        if (set == null) {
            set = new HashSet<>();
            listeners.put(userId, set);

            var devices = storage.getObjects(Device.class, new Request(
                                        new  Columns.Include("id"),  new
Condition.Permission(User.class, userId, Device.class)));
                                                userDevices.put(userId,
devices.stream().map(BaseModel::getId).collect(Collectors.toSet()));
                                            devices.forEach(device    ->
deviceUsers.computeIfAbsent(device.getId(),       id       ->       new
HashSet<>()).add(userId));
        }
        set.add(listener);
    }

    public synchronized void removeListener(long userId, UpdateListener
listener) {
        var set = listeners.get(userId);
        set.remove(listener);
        if (set.isEmpty()) {
            listeners.remove(userId);

                        userDevices.remove(userId).forEach(deviceId  ->
deviceUsers.computeIfPresent(deviceId, (x, userIds) -> {
                userIds.remove(userId);
                return userIds.isEmpty() ? null : userIds;
            }));
        }
    }

}
```

Descripción General:

1. Gestión de Sesiones de Dispositivos:

- sessionsByDeviceId: Rastrea las sesiones de dispositivos por ID de dispositivo.
- sessionsByEndpoint: Mapea los puntos finales de conexión (definidos por ConnectionKey) a sesiones de dispositivos.
- unknownByEndpoint: Lleva un registro de los dispositivos desconocidos por sus puntos finales de conexión.

2. Configuración y Dependencias:
    - Configuración: deviceTimeout, showUnknownDevices.
    - Dependencias Inyectadas:
        - Config: Configuraciones del sistema.
        - CacheManager: Servicios de caché para dispositivos.
        - Storage: Interfaz para operaciones de almacenamiento en base de datos.
        - NotificationManager: Manejo de notificaciones.
        - Timer: Para programar tiempos de espera.
        - BroadcastService: Para transmitir actualizaciones.
        - DeviceLookupService: Servicio para buscar dispositivos en la base de datos.
3. Métodos de Manejo de Sesiones:
    - getDeviceSession: Recupera o crea una DeviceSession para un protocolo, canal y IDs únicos dados.
    - deviceDisconnected: Maneja la desconexión de dispositivos, actualiza el estado y elimina sesiones.
    - deviceUnknown: Marca un dispositivo como desconocido y elimina su sesión.
    - updateDevice: Actualiza el estado del dispositivo, maneja tiempos de espera y activa notificaciones si cambia el estado.
4. Transmisión y Notificaciones:
    - sendKeepalive: Envía señales de mantenimiento a los oyentes.
    - updateDevice: Actualiza a los oyentes y transmite información del dispositivo.
    - updatePosition: Actualiza a los oyentes y transmite información de la posición.
    - updateEvent: Actualiza a los oyentes y transmite información de eventos.
    - updateLog: Actualiza a los oyentes y maneja los registros de logs.
5. Gestión de Oyentes:
    - addListener: Registra un nuevo oyente para un usuario.
    - removeListener: Desregistra un oyente para un usuario.
6. Métodos Auxiliares:

- addUnknownDevice: Registra automáticamente dispositivos con IDs únicos desconocidos.
- removeDeviceSession: Elimina una sesión de dispositivo y limpia las asignaciones relacionadas.

DeviceSession.java

```
/*
 * Copyright 2016 - 2024 Anton Tananaev (anton@traccar.org)
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
  *  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.traccar.session;

import io.netty.channel.Channel;
import io.netty.handler.codec.http.HttpRequestDecoder;
import org.traccar.BasePipelineFactory;
import org.traccar.Protocol;
import org.traccar.model.Command;

import java.net.SocketAddress;
import java.util.HashMap;
import java.util.Map;

public class DeviceSession {

    private final long deviceId;
    private final String uniqueId;
    private final String model;
    private final Protocol protocol;
    private final Channel channel;
    private final SocketAddress remoteAddress;
```

```
    public DeviceSession(
            long deviceId, String uniqueId, String model,
                    Protocol protocol, Channel channel, SocketAddress
remoteAddress) {
        this.deviceId = deviceId;
        this.uniqueId = uniqueId;
        this.model = model;
        this.protocol = protocol;
        this.channel = channel;
        this.remoteAddress = remoteAddress;
    }

    public long getDeviceId() {
        return deviceId;
    }

    public String getUniqueId() {
        return uniqueId;
    }

    public String getModel() {
        return model;
    }

    public Channel getChannel() {
        return channel;
    }

    public ConnectionKey getConnectionKey() {
        return new ConnectionKey(channel, remoteAddress);
    }

    public boolean supportsLiveCommands() {
            return  BasePipelineFactory.getHandler(channel.pipeline(),
HttpRequestDecoder.class) == null;
    }

    public void sendCommand(Command command) {
        protocol.sendDataCommand(channel, remoteAddress, command);
    }

    public static final String KEY_TIMEZONE = "timezone";
```

```
    private final Map<String, Object> locals = new HashMap<>();

    public boolean contains(String key) {
        return locals.containsKey(key);
    }

    public void set(String key, Object value) {
        if (value != null) {
            locals.put(key, value);
        } else {
            locals.remove(key);
        }
    }

    @SuppressWarnings("unchecked")
    public <T> T get(String key) {
        return (T) locals.get(key);
    }

}
```

Componentes Clave:

1. Atributos:
    - deviceId: Identificador único del dispositivo.
    - uniqueId: ID único asociado al dispositivo.
    - model: Modelo del dispositivo.
    - protocol: Protocolo utilizado para la comunicación con el dispositivo.
    - channel: Canal de comunicación Netty asociado con la sesión del dispositivo.
    - remoteAddress: Dirección remota desde la cual se está conectando el dispositivo.
2. Constructor:
    - DeviceSession(long deviceId, String uniqueId, String model, Protocol protocol, Channel channel, SocketAddress remoteAddress): Inicializa una nueva instancia de DeviceSession con los parámetros proporcionados.
3. Métodos:
    - getDeviceId(): Devuelve el ID del dispositivo.
    - getUniqueId(): Devuelve el ID único del dispositivo.
    - getModel(): Devuelve el modelo del dispositivo.

- getChannel(): Devuelve el canal de comunicación asociado con la sesión.
- getConnectionKey(): Crea una clave de conexión utilizando el canal y la dirección remota.
- supportsLiveCommands(): Verifica si la sesión admite comandos en vivo (verifica si el canal no tiene un HttpRequestDecoder en su pipeline).
- sendCommand(Command command): Envía un comando al dispositivo utilizando el protocolo especificado.
- contains(String key): Verifica si un valor está presente en el mapa de variables locales.
- set(String key, Object value): Establece un valor en el mapa de variables locales. Si el valor es null, se elimina la entrada del mapa.
- get(String key): Obtiene un valor del mapa de variables locales.

4. Constantes:
   - KEY_TIMEZONE: Clave para almacenar la zona horaria en el mapa de variables locales.
5. Variables Locales:
   - locals: Mapa que almacena variables locales asociadas con la sesión del dispositivo.

Conclusiones

En general, los códigos revisados reflejan un enfoque integral y eficiente para la gestión de sesiones de dispositivos y la comunicación en aplicaciones basadas en redes. La implementación aborda de manera efectiva la administración de sesiones, el manejo de dispositivos y la actualización de estados, garantizando una gestión fluida de la comunicación y las interacciones entre dispositivos y usuarios. La capacidad de manejar estados y eventos en tiempo real, junto con la integración de mecanismos para registrar y gestionar dispositivos desconocidos, asegura una alta disponibilidad y adaptabilidad en diversos escenarios operativos.

El diseño modular y bien estructurado de las clases permite una escalabilidad adecuada y una respuesta rápida ante cambios en la red o en el estado de los dispositivos. Además, la gestión eficiente de sesiones y la sincronización de datos entre componentes aseguran una operación estable y coherente. Este enfoque proporciona una base sólida para aplicaciones que requieren un manejo robusto y flexible de dispositivos, optimizando la comunicación y la gestión de eventos en tiempo real.

La gestión de la sesión en una aplicación que utiliza la API de Traccar es un componente fundamental para garantizar un acceso seguro y eficiente a los recursos del sistema. El uso adecuado del proceso de autenticación, almacenamiento seguro del token de sesión, y manejo de errores asegura que los usuarios puedan interactuar con la aplicación de forma continua y segura. Además, implementar mecanismos para cerrar sesión de manera adecuada y proteger el token contra vulnerabilidades refuerza la seguridad de la aplicación. En conclusión, una correcta gestión de la sesión no solo garantiza una experiencia de usuario fluida, sino que también protege los datos y la integridad de la aplicación.