



# Manual Tècnic

Dan Maldonado  
Samuel Hernán  
Curs 2024 - 2025  
2n DAW

## 1. Informació General

**Nom del projecte:** SODA

**Versió del document:** 1.0

Versió 1.0 - 12/05/2025

### Autors

Dan Maldonado Peralvo

Samuel Hernan Moreno

### Propòsit del document

Descriure com funciona el sistema i els seus detalls tècnics (instal·lació, configuració, manteniment, ...). Els seus lectors necessiten tenir un coneixement expert per entendre el seu contingut.

Inclou instruccions detallades de com muntar i configurar el sistema informàtic del projecte: fitxers de configuració, captures de pantalla i explicacions detallades.

## 1.2 Taula de continguts

<b>1. Informació General</b>	<b>2</b>
1.2 Taula de continguts	3
<b>2. Arquitectura del Sistema</b>	<b>5</b>
<b>3. Entorn Tècnic</b>	<b>6</b>
3.1 Requisits de Hardware	6
3.2 Requisits de Software	7
<b>4. Configuració del Sistema</b>	<b>9</b>
4.1 Instal·lació	9
4.2 Desplegament	10
<b>5. Base de Dades</b>	<b>13</b>
5.1 Model de Dades	13
<b>6. Estructura del Codi</b>	<b>18</b>
6.1 Organització del Projecte	18
<b>7. APIs i Interfícies</b>	<b>45</b>
7.1 APIs Internes	45
POST /register	45
POST /login	46
POST /logout	46
GET /chat/conversations	47
GET /chat/:conversationId/messages	47
POST /chat/:conversationId/messages	48
POST /chat/new	49
GET /chat/users	49
GET /favorites	50
PATCH /favorites/:symbol	51
GET /status	51
POST /buy	52
POST /sell	53
GET /stocks	54
GET /news	55
GET /stocks/:symbol	56
GET /api/posts/	57
GET /api/posts/:id	58
POST /api/posts/:postId/comments	58
GET /api/posts/:postId/comments	59
PUT /api/posts/comments/:id	59
DELETE /api/posts/comments/:id	60
POST /api/posts/:postId/like	60
DELETE /api/posts/:postId/like	61
GET /api/posts/:postId/likes	61
POST /posts	62
GET /posts/:id	63

PUT /posts/:id	64
DELETE /posts/:id	65
GET /api/transactions/	65
GET /api/users/profile	66
PUT /api/users/profile	66
PUT /api/users/profile-image	67
PUT /api/users/change-password	67
GET /api/users/stocks	67
POST /api/users/addCredit	68
POST /api/users/withdrawCredit	68
<b>7.2 APIs Externes</b>	<b>69</b>
Yahoo Finance API	69
Finnhub API	70
NewsCatcher API	71
<b>7.3. Websocket</b>	<b>72</b>
subscribeToSymbols(socket, symbols)	72
checkSubscriptions(socket, symbols)	73
initializeWebSocket(server)	73
scheduleReconnect()	74
<b>10. Resolució de Problemes</b>	<b>75</b>
10.1 Problemes Comuns	75
Problema 1. Websockets	75
Problema 2. Proxy invers.	75
10.2. Procediments de diagnòstic	76
Procediment 1	76
Procediment 2.	76
Procediment 3	76
Procediment 4.	76
Procediment 5.	76
<b>11. Enllaç al GitHub</b>	<b>77</b>

## 2. Arquitectura del Sistema

- Patrons de disseny utilitzats

Model / Vista / Controlador

Stack tecnològic implementat

**Frontend:** React, NextJS

**Backend:** NodeJS, Express

**Base de dades:** MongoDB

**Memòria de cau:** Redis

**Serveis externs:**

- Connexió WebSocket amb Finnhub. (API de borsa)
- Api de Yahoo Finance. Per treure dades històriques de les accions.

## 3. Entorn Tècnic

### 3.1 Requisits de Hardware

- Servidor

Es necessita un servidor capaç d'executar contenidors Docker de forma eficient i de gestionar simultàniament serveis com ara el backend amb Node.js, la base de dades MongoDB, la memòria cau Redis, i el frontend amb Next.js.

Pot ser un servidor físic o una instància virtual (VPS o cloud), sempre que compleixi els requisits míнимs detallats a continuació.

- Capacitat d'emmagatzematge

**Mínima recomanada:** 100 GB SSD

**Recomanada per producció:** 250-500 GB SSD

L'ús de discos SSD és essencial per garantir temps de resposta ràpids, especialment en l'accés a la base de dades i en la gestió dels contenidors Docker.

- Memòria RAM recomanada

**Entorn de desenvolupament:** mínim 8 GB de RAM

**Entorn de producció:** mínim 16 GB de RAM, preferiblement 32 GB si es preveu un gran nombre d'usuaris simultanis o escenaris de càrrega elevada.

La memòria RAM es distribueix entre els serveis del backend, la base de dades, el frontend i la memòria cau.

- Processador recomanat

**Mínim:** CPU amb 4 nuclis (4 cores)

**Recomanat:** CPU amb 6 nuclis / 12 fils (cores/threads) o superior, un processador modern amb suport per a virtualització i multithreading assegura que el sistema pugui manejar diverses peticions concurrents i serveis simultanis de manera eficient.

### 3.2 Requisits de Software

- Sistema operatiu

Es pot continuar el projecte amb qualsevol sistema, ja que els softwares que tenen dependències més específiques estan dockeritzats.

- Base de dades

MongoDB. Redis per la memòria Cau

- Framework utilitzat

NodeJS amb Express.

- Llenguatges de programació

Javascript. JSX.

- Dependències i llibreries

Docker instal·lat en el sistema.

#### Frontend:

Next.js ^15.3.1 – Framework React per a aplicacions SSR/SPA.

React ^19.0.0 – Llibreria per a la construcció de la interfície d'usuari.

React DOM ^19.0.0 – Renderitzat de components React al DOM.

Axios ^1.7.9 – Client HTTP per a comunicació amb APIs.

SWR ^2.3.3 – Gestió de dades remotes amb cache i revalidació automàtica.

Chart.js ^4.4.8 – Visualització de gràfics estadístics.

React ChartJS 2 ^5.3.0 – Integració de Chart.js amb React.

Chart.js Adapter Date-fns ^3.0.0 – Adaptador de dates per a Chart.js.

Recharts ^2.15.1 – Llibreria de gràfics modular i reactiva.

Lucide ^0.475.0 – Ícones en SVG modernes.

Lucide React ^0.475.0 – Integració de Lucide amb React.

## Llibreries de desenvolupament:

Tailwind CSS ^3.4.1 – Framework CSS utilitari.

DaisyUI ^4.12.24 – Components UI per a Tailwind CSS.

PostCSS ^8 – Transformacions CSS modernes.

ESLint ^9 – Linter per a JavaScript i React.

@eslint/eslintrc ^3 – Configuració avançada d'ESLint.

eslint-config-next 15.1.7 – Configuració ESLint per a projectes Next.js.

Zustand ^5.0.3 – Gestió d'estat global lleugera per a React.

## Backend:

Express ^4.21.2 – Framework minimalist per a crear APIs HTTP.

Body-Parser ^1.20.3 – Middleware per a analitzar el cos de les peticions.

Cookie-Parser ^1.4.7 – Lectura i manipulació de cookies HTTP.

CORS ^2.8.5 – Suport per a peticions entre orígens diferents (Cross-Origin).

Dotenv ^16.4.7 – Carregador de variables d'entorn des d'arxius .env.

Mongoose ^8.10.0 – ODM per a MongoDB, facilita la interacció amb la base de dades.

Redis ^4.7.0 – Client oficial per a Redis.

ioredis ^5.6.0 – Client Redis avançat amb suport per a clústers.

JSONWebToken ^9.0.2 – Gestió d'autenticació amb tokens JWT.

BcryptJS ^3.0.0 – Encriptació i verificació de contrasenyes.

Express-Validator ^7.2.1 – Validació i sanejament de peticions HTTP.

Multer ^1.4.5-Its.2 – Middleware per a la gestió d'arxius multipart/form-data.

WS ^8.18.1 – Implementació de WebSockets per a comunicació en temps real.

Axios ^1.7.9 – Client HTTP per fer peticions a altres serveis.

Nodemon ^3.1.9 – Eina de desenvolupament per reiniciar el servidor automàticament

## 4. Configuració del Sistema

Per continuar treballant en el projecte en mode desenvolupament haurem de configurar diverses coses.

### 4.1 Instal·lació

- Pas a pas del procés d'instal·lació

Clonar el repositori:

<https://github.com/Daneteee/SODA.git>

*cd SODA/frontend*

*npm install*

*cd SODA/backend*

*npm install*

*docker compose up -d*

- Configuració de l'entorn

Cal configurar un .env en els dos directoris, tant frontend com backend.

- Variables d'entorn necessàries

#### Frontend:

```
NEXT_PUBLIC_API_URL= http://localhost:4000/api  
NEXT_PUBLIC_SERVER_URL=http://localhost:4000  
NEXT_PUBLIC_WEBSOCKET_URL=ws://localhost:4000
```

#### Backend:

```
PORT=4000  
MONGO_URI=mongodb://mongodb:27017/soda  
FINNHUB_API_KEY=api_key  
JWT_SECRET=supersecretkey123  
REDIS_HOST=redis  
REDIS_PORT=6379
```

\*S'ha de canviar localhost per les IPs i noms pels seus noms o IPs.

## 4.2 Desplegament

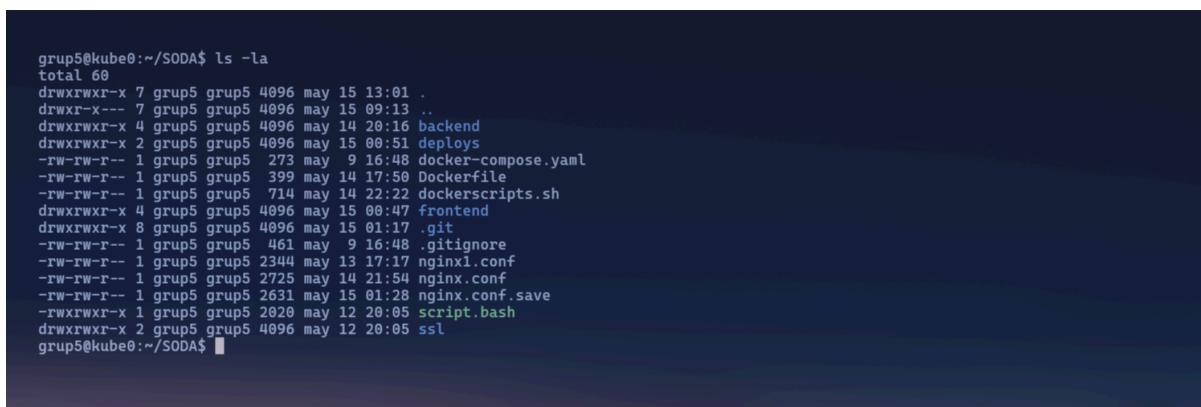
- Procediment de desplegament

El servidor de desplegament es troba a

`grup5@10.52.5.102`

Si es pretén accedir des de l'exterior es pot fer de la següent forma:

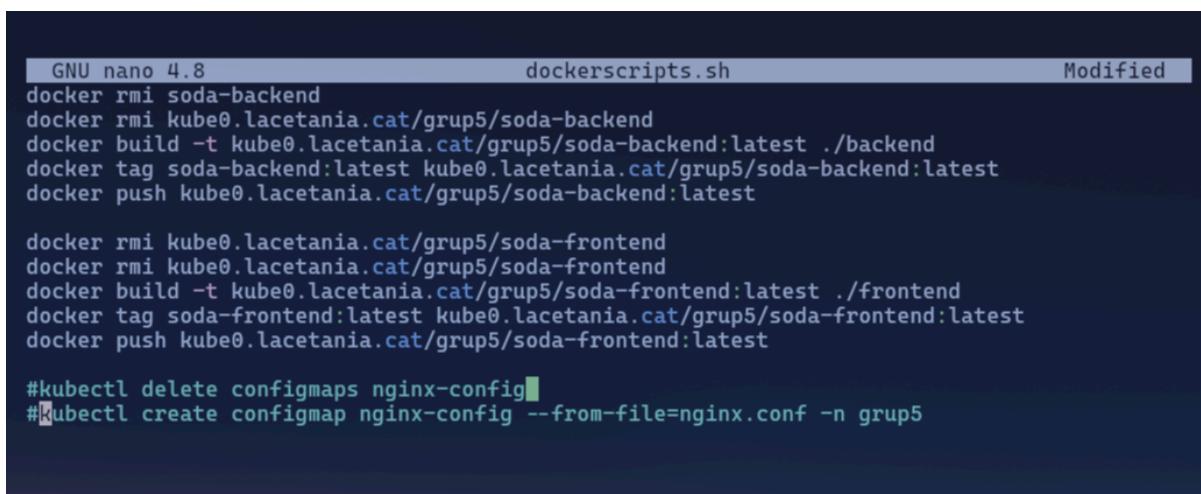
`ssh -p 2269 grup5@infla.cat`



```
grup5@kube0:~/SODA$ ls -la
total 60
drwxrwxr-x 7 grup5 grup5 4096 may 15 13:01 .
drwxr-x--- 7 grup5 grup5 4096 may 15 09:13 ..
drwxrwxr-x 4 grup5 grup5 4096 may 14 20:16 backend
drwxrwxr-x 2 grup5 grup5 4096 may 15 00:51 deploys
-rw-rw-r-- 1 grup5 grup5 273 may 9 16:48 docker-compose.yaml
-rw-rw-r-- 1 grup5 grup5 399 may 14 17:50 Dockerfile
-rw-rw-r-- 1 grup5 grup5 714 may 14 22:22 dockerscripts.sh
drwxrwxr-x 4 grup5 grup5 4096 may 15 00:47 frontend
drwxrwxr-x 8 grup5 grup5 4096 may 15 01:17 .git
-rw-rw-r-- 1 grup5 grup5 461 may 9 16:48 .gitignore
-rw-rw-r-- 1 grup5 grup5 2344 may 13 17:17 nginx1.conf
-rw-rw-r-- 1 grup5 grup5 2725 may 14 21:54 nginx.conf
-rw-rw-r-- 1 grup5 grup5 2631 may 15 01:28 nginx.conf.save
-rwxrwxr-x 1 grup5 grup5 2020 may 12 20:05 script.bash
drwxrwxr-x 2 grup5 grup5 4096 may 12 20:05 ssl
grup5@kube0:~/SODA$
```

Per posar el projecte a producció s'haurà de crear les noves imatges del software, pujar-les al registry i reiniciar els deploys. Per fer això s'ha preparat un script.

Aquest script elimina les anteriors imatges en torna a crear unes de noves i elimina i torna a crear els configmaps per nginx.



```
GNU nano 4.8                               dockerscripts.sh                               Modified
docker rmi soda-backend
docker rmi kube0.lacetania.cat/grup5/soda-backend
docker build -t kube0.lacetania.cat/grup5/soda-backend:latest ./backend
docker tag soda-backend:latest kube0.lacetania.cat/grup5/soda-backend:latest
docker push kube0.lacetania.cat/grup5/soda-backend:latest

docker rmi kube0.lacetania.cat/grup5/soda-frontend
docker rmi kube0.lacetania.cat/grup5/soda-frontend
docker build -t kube0.lacetania.cat/grup5/soda-frontend:latest ./frontend
docker tag soda-frontend:latest kube0.lacetania.cat/grup5/soda-frontend:latest
docker push kube0.lacetania.cat/grup5/soda-frontend:latest

#kubectl delete configmaps nginx-config
#kubectl create configmap nginx-config --from-file=nginx.conf -n grup5
```

Ara ja només l'administrador ha d'entrar a la carpeta /deploys

```
grup5@kube0:~/SODA/deploys$ kubectl apply -f .
persistentvolumeclaim/soda-uploads-pvc unchanged
deployment.apps/soda-backend unchanged
service/soda-backend unchanged
deployment.apps/soda-frontend configured
service/soda-frontend unchanged
deployment.apps/nginx-proxy unchanged
service/nginx-proxy unchanged
deployment.apps/mongodb unchanged
service/mongodb unchanged
persistentvolumeclaim/mongodb-pvc unchanged
deployment.apps/redis unchanged
service/redis unchanged
grup5@kube0:~/SODA/deploys$ █
```

Aquí es poden trobar tots els deploys:

```
grup5@kube0:~/SODA/deploys$ ls
backend.yaml frontend.yaml ingress.yaml mongo.yaml redis-deployment.yaml script.sh
grup5@kube0:~/SODA/deploys$ █
```

**backend.yaml:**

**PVC:**

Volume claim que s'utilitza per tenir imatges guardades de part dels usuaris de manera persistent.

**Deployment**

S'utilitza per crear un deploy amb la imatge soda-backend creada anteriorment. Es munta /app/uploads en el pvc

**Service:**

Un servei tipus ClusterIP que exposa el port 4000 de l'aplicació

**frontend.yaml:**

**Deployment:**

S'utilitza per crear un deploy amb la imatge soda-frontend creada anteriorment.

**Service:**

Servei tipus ClusterIP que expsa el port 3000 de l'aplicació

**mongo.yaml**

**PVC:**

VolumeClaim per persistència de dades

**Deployment:**

Utilitza la imatge mongo:latest

**Service:**

Exposa el port 27017 del contenidor en un Service ClusterIP

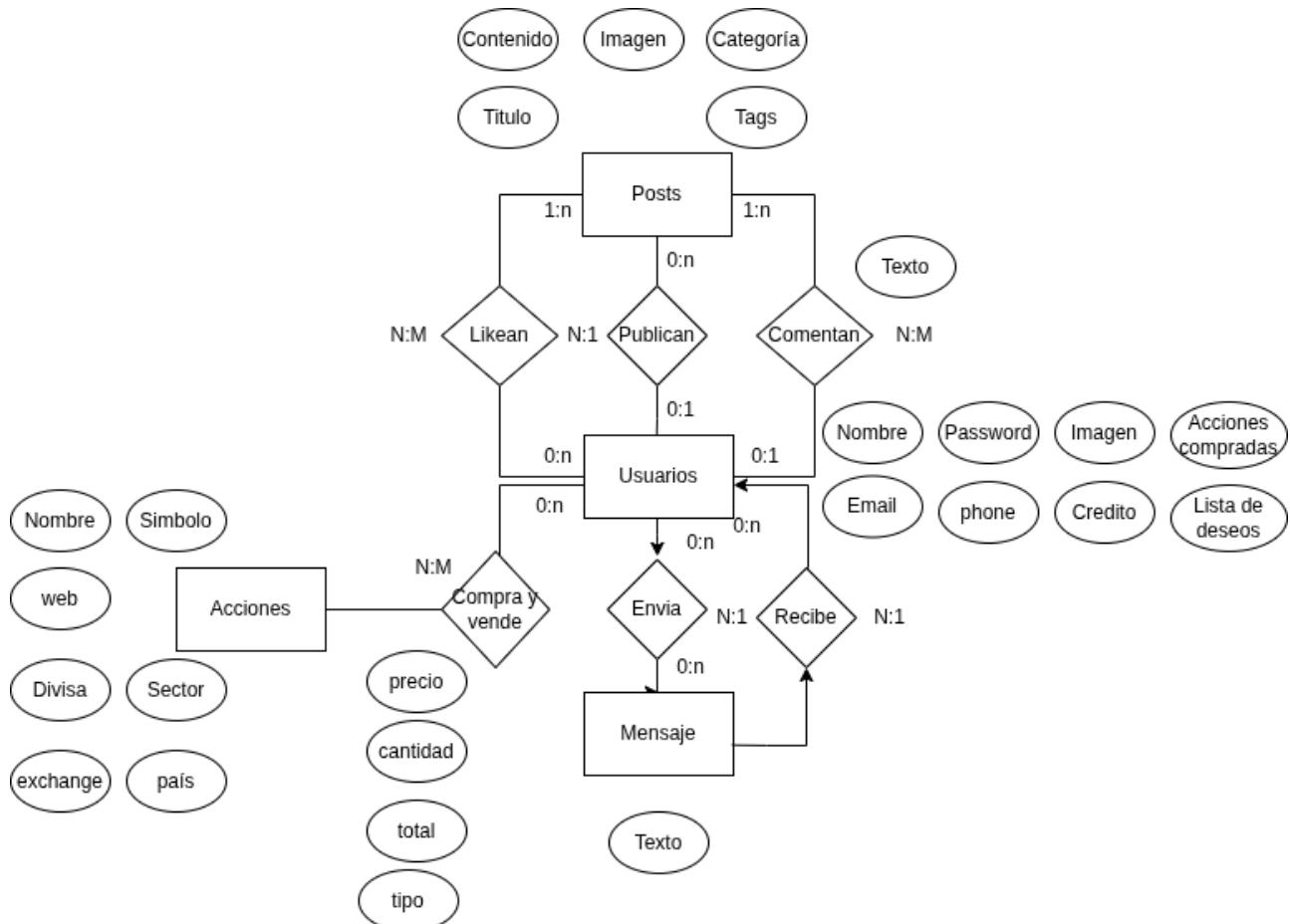
- Configuració de servidors
- Scripts necessaris
- Gestió de versions

Pel control de versions utilitzem git i GitHub, tenim una branca principal i cadascú treballa en una subbranca seva.

## 5. Base de Dades

### 5.1 Model de Dades

- Diagrama ER



## Descripció de taules

### 1. Usuari

Representa: un usuari registrat a l'aplicació.

Atributs:

name: String – Nom de l'usuari. Requerit.

email: String – Correu electrònic únic. Requerit.

password: String – Contrasenya encriptada. Requerit.

phone: String – Telèfon de l'usuari. Opcional.

profileImage: String – URL de la imatge de perfil. Opcional, per defecte una imatge genèrica.

createdAt: Date – Data de creació. Valor per defecte: Date.now.

stocks: Array<Object> – Accions que posseeix l'usuari:

symbol: String – Símbol de l'acció.

quantity: Number – Quantitat.

purchasePrice: Number – Preu de compra.

purchaseDate: Date – Data de compra.

credit: Number – Crèdit disponible per operacions. Per defecte: 50.000.

favs: Array<String> – Accions preferides de l'usuari. Opcional.

## 2. Publicació

Representa: una publicació feta per un usuari.

Atributs:

title: String – Títol de la publicació. Requerit. Màxim: 100 caràcters.

content: String – Contingut de la publicació. Requerit.

image: String – URL de la imatge (opcional).

author: ObjectId (ref: User) – Autor de la publicació. Requerit.

category: String – Categoria (enum). Valors com: general, crypto, stocks, etc.

tags: Array<String> – Etiquetes associades a la publicació.

likesCount: Number – Nombre de m'agrades. Valor inicial: 0.

commentsCount: Number – Nombre de comentaris. Valor inicial: 0.

timestamps: createdAt, updatedAt – Dates automàtiques de creació i actualització.

## 3. Comentari

Representa: un comentari fet en una publicació.

Atributs:

postId: ObjectId (ref: Post) – Publicació a la qual pertany el comentari.

author: ObjectId (ref: User) – Usuari que comenta.

content: String – Text del comentari. Requerit.

createdAt: Date – Data de creació. Per defecte: Date.now.

#### 4. M'agrada

Representa: un "m'agrada" d'un usuari sobre una publicació.

Atributs:

user: ObjectId (ref: User) – Usuari que dona el "m'agrada". Requerit.

post: ObjectId (ref: Post) – Publicació que rep el "m'agrada". Requerit.

createdAt: Date – Data en què es dona el "m'agrada".

Índex únic { user, post } per evitar duplicats.

#### 5. Conversa

Representa: una conversa entre dos o més usuaris.

Atributs:

participants: Array<ObjectId (ref: User)> – Usuaris participants. Requerit.

lastMessage: String – Últim missatge enviat. Opcional.

updatedAt: Date – Última actualització. Per defecte: Date.now.

#### 6. Missatge

Representa: un missatge enviat entre dos usuaris.

Atributs:

from: ObjectId (ref: User) – Remitent. Requerit.

to: ObjectId (ref: User) – Destinatari. Requerit.

text: String – Contingut del missatge. Requerit.

timestamp: Date – Data del missatge. Per defecte: Date.now.

## 7. Acció

Representa: una acció disponible al mercat per operar.

Atributs:

symbol: String – Símbol únic (ex: AAPL). Requerit.

name: String – Nom de l'empresa. Requerit.

sector: String – Sector de l'empresa.

industry: String – Indústria.

exchange: String – Borsa on cotitza.

country: String – País d'origen.

currency: String – Moneda.

description: String – Descripció de l'empresa.

website: String – Pàgina oficial.

logo: String – URL del logotip.

## 8. Transacció

Representa: una transacció financerca de compra o venda d'accions.

Atributs:

userId: ObjectId (ref: User) – Usuari que fa la transacció.

stock: String – Símbol de l'acció (ex: AAPL). Requerit.

type: String – Tipus d'operació (buy o sell). Requerit.

amount: Number – Quantitat d'accions. Requerit.

price: Number – Preu per acció. Requerit.

total: Number – Preu total (quantitat \* preu). Requerit.

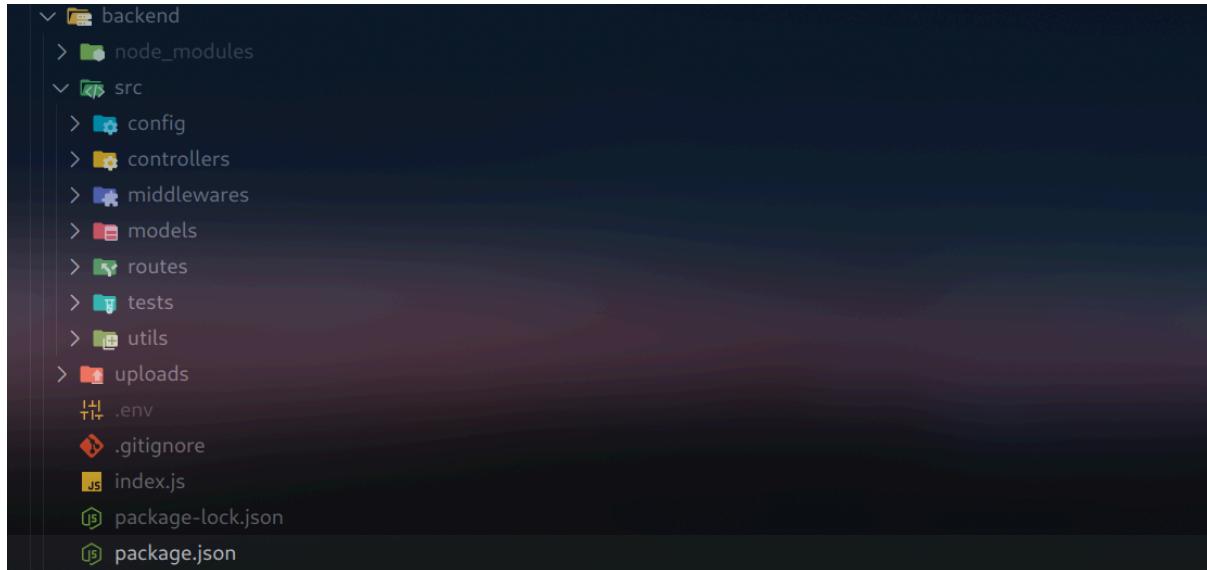
date: Date – Data de la transacció. Per defecte: Date.now

## 6. Estructura del Codi

### 6.1 Organització del Projecte

#### Estructura de directoris

Backend:



Express ressalta per la seva flexibilitat per organitzar el codi. En aquest cas s'ha organitzat el codi de manera semblant a frameworks MVC.

**src/**

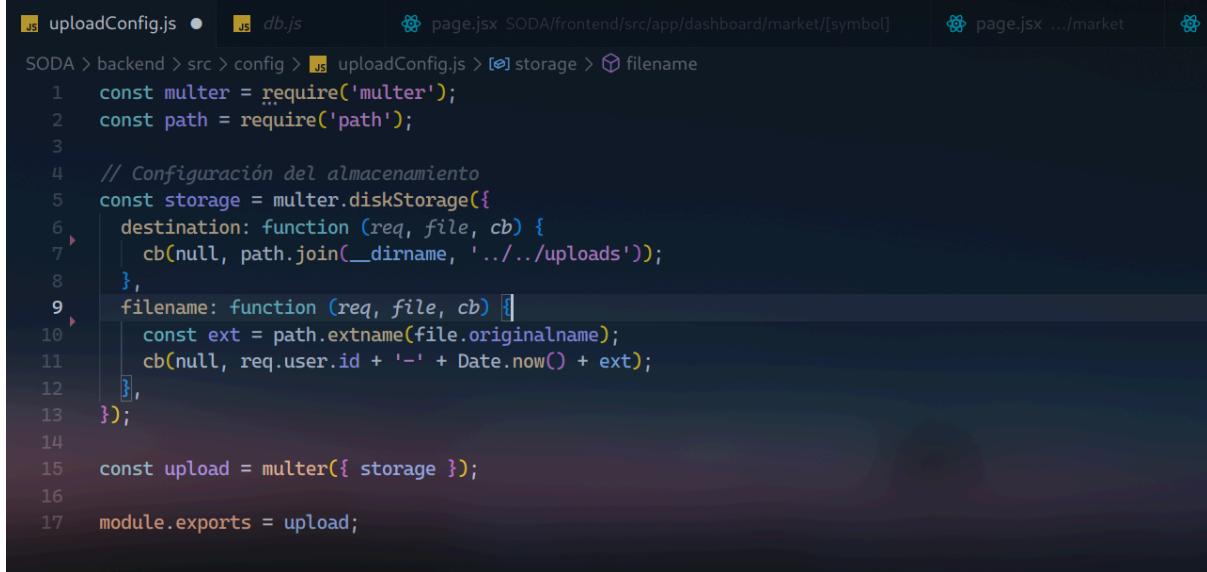
#### config/

La carpeta de config conté dos arxius de configuració per connectar la base de dades. En aquest cas s'utilitza una llibreria anomenada Mongoose, amb la qual el backend es pot connectar a Mongo fent servir el sistema de Models.

```
SODA > backend > src > config > db.js > connectDB
1  const mongoose = require('mongoose');
2
3  const connectDB = async () => {
4    try {
5      await mongoose.connect(process.env.MONGO_URI, {
6        useNewUrlParser: true,
7        useUnifiedTopology: true,
8      });
9      console.log('MongoDB Connected ... ');
10    } catch (err) {
11      console.error(err.message);
12      process.exit(1);
13    }
14  };
15
16  module.exports = connectDB;
```

### uploadConfig.js

Aquest arxiu de configuració conté les funcions storage, amb les quals es fa possible la pujada d'imatges.



The screenshot shows a code editor with several tabs at the top: 'uploadConfig.js', 'db.js', 'page.jsx SODA/frontend/src/app/dashboard/market/[symbol]', and 'page.jsx .../market'. The 'uploadConfig.js' tab is active. The code itself is as follows:

```
SODA > backend > src > config > uploadConfig.js > storage > filename
1 const multer = require('multer');
2 const path = require('path');
3
4 // Configuración del almacenamiento
5 const storage = multer.diskStorage({
6   destination: function (req, file, cb) {
7     cb(null, path.join(__dirname, '../../../../../uploads'));
8   },
9   filename: function (req, file, cb) {
10     const ext = path.extname(file.originalname);
11     cb(null, req.user.id + '-' + Date.now() + ext);
12   },
13 });
14
15 const upload = multer({ storage });
16
17 module.exports = upload;
```

### middlewares/

#### authMiddleware.js

Aquest arxiu s'usa per protegir totes les rutes que requereixen el token. El frontend envia el token amb la cookie. Amb aquest middleware es valida que el token descriptat coincideixi amb l'usuari descodificat.



The screenshot shows a code editor with several tabs at the top: 'authMiddleware.js' and '...'. The 'authMiddleware.js' tab is active. The code is as follows:

```
SODA > backend > src > middlewares > authMiddleware.js > ...
1 const jwt = require("jsonwebtoken");
2 require("dotenv").config();
3
4 const authMiddleware = (req, res, next) => {
5   const token = req.cookies.jwtToken;
6   if (!token) return res.status(401).json({ msg: "No autorizado" });
7   try {
8     const decoded = jwt.verify(token, process.env.JWT_SECRET);
9     req.user = decoded.user;
10    next();
11  } catch (err) {
12    return res.status(401).json({ msg: "Token inválido" });
13  }
14};
15
16 module.exports = authMiddleware;
```

### validateRegistration.js

Aquest middleware valida cada camp que rep del frontend del formulari. En cas que no compleixi amb els requisits retorna error.

### Possible millora:

En el cas de la contrasenya es podria posar un validador per contrasenyes més robustes, en el frontend ja està implementat, però caldria implementar-ho aquí.

```
SODA > backend > src > middlewares > validateRegistration.js > ...
1 const { body, validationResult } = require('express-validator');
2
3 const validateRegistration = [
4   body('name', 'El nombre es obligatorio').notEmpty(),
5   body('email', 'Por favor, ingresa un correo válido').isEmail(),
6   body('password', 'La contraseña debe tener al menos 6 caracteres').isLength({ min: 6 }),
7   body('phone', 'Por favor, ingresa un número de teléfono válido').optional().isMobilePhone(),
8
9   (req, res, next) => {
10     const errors = validationResult(req);
11     if (!errors.isEmpty()) {
12       console.log(errors.array())
13       return res.status(400).json({ errors: errors.array() });
14     }
15     next();
16   }
17 ];
18
19 module.exports = {validateRegistration};
20
```

## models/

### comments.js

En aquest model es guarden els diferents comentaris que els usuaris poden donar als posts.

```
SODA > backend > src > models > comment.js > ...
1 const mongoose = require('mongoose');
2
3 const CommentSchema = new mongoose.Schema({
4   postId: { type: mongoose.Schema.Types.ObjectId, ref: 'Post', required: true },
5   author: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
6   content: { type: String, required: true },
7   createdAt: { type: Date, default: Date.now },
8 };
9
10 module.exports = mongoose.model('Comment', CommentSchema);
11
```

### conversation.js

En aquest model es guarden les conversacions del xat.

```
SODA > backend > src > models > conversation.js > ...
1  const mongoose = require('mongoose');
2
3  const conversationSchema = new mongoose.Schema({
4    participants: [{ type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true }],
5    lastMessage: { type: String },
6    updatedAt: { type: Date, default: Date.now }
7  });
8
9  module.exports = mongoose.model('Conversation', conversationSchema);
10
11
12 |
```

### like.js

En el model “like” es guarden els likes que els usuaris donen el post.

```
SODA > backend > src > models > like.js > ...
1  const mongoose = require('mongoose');
2
3  const LikeSchema = new mongoose.Schema({
4    user: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
5    post: { type: mongoose.Schema.Types.ObjectId, ref: 'Post', required: true },
6    createdAt: { type: Date, default: Date.now }
7  }, { timestamps: true });
8
9
10 | LikeSchema.index({ user: 1, post: 1 }, { unique: true });
11 module.exports = mongoose.model('Like', LikeSchema);
12 |
```

### Possible millora:

Afegir un camp extra de comentari, d'aquesta manera els usuaris donaran like als comentaris.

### message.js

Model que guarda els missatges del xat.

```
SODA > backend > src > models > message.js > ...
1  const mongoose = require('mongoose');
2
3  const messageSchema = new mongoose.Schema({
4    from: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
5    to: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
6    text: { type: String, required: true },
7    timestamp: { type: Date, default: Date.now }
8  });
9
10 module.exports = mongoose.model('Message', messageSchema);
11 |
```

### post.js

En aquest model es guarden tots els atributs de cada post, també s'inclouen validacions en l'àmbit de base de dades per assegurar que tot es guarda correctament.

```
SODA > backend > src > models > post.js > ...
1  const mongoose = require('mongoose');
2
3  const postSchema = new mongoose.Schema({
4    title: { type: String, required: [true, 'El título es requerido'], trim: true, maxlength: [100, 'El título no puede tener más de 100 caracteres'] },
5    content: { type: String, required: [true, 'El contenido es requerido'], trim: true },
6    image: { type: String, default: null },
7    author: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
8    category: { type: String, enum: ['general', 'tecnología', 'deportes', 'entretenimiento', 'stocks', 'crypto', 'finanzas'], required: true },
9    tags: [{ type: String, trim: true }],
10   likesCount: { type: Number, default: 0 },
11   commentsCount: { type: Number, default: 0 }
12 }, {
13   timestamps: true
14 });
15
16 module.exports = mongoose.model('Post', postSchema);
17 |
```

### stock.js

Aquest model guarda tota la informació de les accions.

```
SODA > backend > src > models > stock.js > ...
1 const mongoose = require('mongoose');
2
3 const stockSchema = new mongoose.Schema({
4   symbol: { type: String, required: true, unique: true },
5   name: { type: String, required: true },
6   sector: String,
7   industry: String,
8   exchange: String,
9   country: String,
10  currency: String,
11  description: String,
12  website: String,
13  logo: String,
14 });
15
16 module.exports = mongoose.model('Stock', stockSchema);
```

### transaction.js

Quan es compra una acció es crea una transacció on s'associa l'usuari amb l'acció que ha comprat la quantitat i altres dades, per aquesta raó és important el model “transaction”.

```
SODA > backend > src > models > transaction.js > ...
1 const mongoose = require("mongoose");
2
3 const transactionSchema = new mongoose.Schema({
4   userId: { type: mongoose.Schema.Types.ObjectId, ref: "User", required: true },
5   stock: { type: String, required: true },
6   type: { type: String, enum: ["buy", "sell"], required: true },
7   amount: { type: Number, required: true },
8   price: { type: Number, required: true },
9   total: { type: Number, required: true },
10  date: { type: Date, default: Date.now },
11 });
12
13 module.exports = mongoose.model("Transaction", transactionSchema);
14
15 |
```

### user.js

En el model usuari hi ha diferents dades dels usuaris, també el model té un camp favs i stocks. Aquí es guarden totes les accions i accions favorites en un array.

```
SODA > backend > src > models > user.js > ...
1  const mongoose = require('mongoose');
2  const bcrypt = require('bcryptjs');
3
4  const userSchema = new mongoose.Schema({
5    name: { type: String, required: true },
6    email: { type: String, required: true, unique: true, lowercase: true, trim: true },
7    password: { type: String, required: true },
8    phone: { type: String, required: false },
9    profileImage: { type: String, required: false, default: "/uploads/default.jpg" },
10   createdAt: { type: Date, default: Date.now },
11   credit: { type: Number, default: 50000 },
12   favs: [...],
13   stocks: [...],
14 });
15 // Método para comparar contraseñas en el login
16 userSchema.methods.matchPassword = async function (enteredPassword) {
17   return await bcrypt.compare(enteredPassword, this.password);
18 };
19
20 module.exports = mongoose.model('User', userSchema);
```

### utils/

#### MarketHours.js

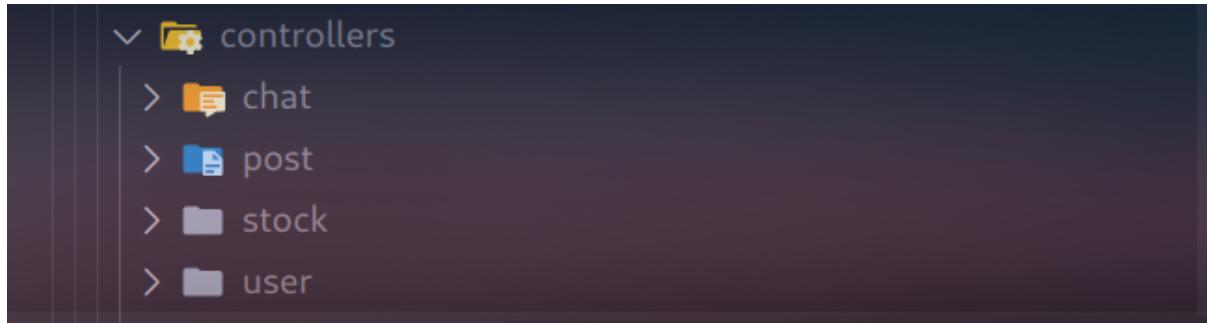
```
SODA > backend > src > utils > marketHours.js > ...
1  const isMarketOpen = () => {
2    const now = new Date();
3    const day = now.getDay();
4    const hour = now.getHours();
5    const minute = now.getMinutes();
6    const time = hour * 100 + minute;
7    if (day === 0 || day === 6) {
8      return false;
9    }
10   return time ≥ 1530 && time < 2200;
11 };
12
13 module.exports = { isMarketOpen }; |
```

#### Possibles millores:

Afegir més arxius d'utilitat per poder reaprofitar codi, i mantenir-lo més net.

## controllers/

Hi ha diferents controladors per totes les funcionalitats. Les funcionalitats del xat, de la part de posts. En el controlador stock hi ha totes les funcionalitats relacionades amb les accions. I a user les relacionades amb el model Usuari.



## chat/

### getChatUsers

Recupera totes les converses de l'usuari autenticat. Obté la llista de xats ordenats per data d'actualització, incloent-hi el darrer missatge i els detalls de l'altre participant (excloent-ne l'usuari actual). Utilitza populate per obtenir només els camps necessaris de l'altre usuari (nom i imatge de perfil).

### getChatMessages

Obté l'historial de missatges de conversa específica. Verifica que l'usuari tingui accés a la conversa i formata els missatges incloent-hi un flag `isMe` per identificar els missatges enviats per l'usuari actual. Els missatges s'ordenen cronològicament.

### sendMessage

Gestionar l'enviament de missatges nous en una conversa. Valida que el missatge no estigui buit i que l'usuari tingui accés a la conversa. Actualitza l'últim missatge i timestamp de la conversa, i torna el missatge formatat amb el vostre ID i marca temporal.

### **createConversation**

Crea una conversa nova entre dos usuaris. Verifica que l'usuari de destinació existeixi i que no hi hagi una conversa prèvia entre els dos usuaris. Si ja hi ha una conversa, torna el vostre ID. En cas contrari, creeu una conversa nova i retorneu el vostre ID.

### **getAllUsers**

Recupera la llista de tots els usuaris de la plataforma, excloent-ne l'usuari actual. Per seguretat, exclou informació sensible com a contrasenyes i emails dels resultats. Útil per iniciar noves converses.

## **post/**

### **post/commentController**

#### **createComment**

Gestiona la creació d'un nou comentari en una publicació. Verifica l'existència de la publicació, crea el comentari amb l'autor actual i actualitza el comptador de comentaris. Retorna el comentari creat amb les dades de l'autor.

#### **getCommentsByPost**

Recupera tots els comentaris d'una publicació específica, ordenats per data de creació descendente. Inclou les dades dels autors i actualitza el comptador de comentaris de la publicació. Retorna la llista completa de comentaris amb el seu recompte.

#### **updateComment**

Permet modificar el contingut d'un comentari existent. Verifica que l'usuari sigui l'autor original del comentari abans de permetre l'actualització. Retorna el comentari actualitzat amb les dades de l'autor.

### **deleteComment**

Gestiona l'eliminació d'un comentari. Verifica l'autoria, elimina els "likes" associats al comentari, actualitza el comptador de comentaris de la publicació i elimina el comentari. Inclou verificacions de seguretat per assegurar que només l'autor pot eliminar el seu propi comentari.

### **post/likeController**

#### **likePost**

Gestiona l'addició d'un "m'agrada" a una publicació. Verifica si la publicació existeix i si l'usuari ja ha donat like. Crea un nou registre de like i incrementa el comptador de likes de la publicació. Inclou validacions per evitar likes duplicats.

#### **unlikePost**

S'encarrega d'eliminar el "m'agrada" d'una publicació. Comprova l'existència de la publicació i del like previ. Elimina el registre de like i decrementa el comptador de likes de la publicació. Garanteix que només es pot eliminar un like existent.

### **getPostLikes**

Recupera tots els "m'agrada" d'una publicació específica. Retorna la llista d'usuaris que han donat like, incloent informació bàsica de cada usuari (nom, email, imatge de perfil) i el recompte total de likes.

### **post/postController**

#### **createPost**

Gestiona la creació d'una nova publicació. Valida els camps obligatoris (títol i contingut), processa la pujada d'imatges si n'hi ha, i crea un nou post amb les etiquetes normalitzades. Inclou logging detallat per facilitar el debugging.

### **getPosts**

Recupera totes les publicacions ordenades per data de creació descendent. Utilitza populate per incloure les dades bàsiques de l'autor (nom, email, imatge de perfil). Retorna el recompte total i la llista de posts.

### **getPostById**

Obté una publicació específica mitjançant el seu ID. Inclou les dades de l'autor mitjançant populate. Verifica l'existència del post i retorna un error 404 si no es troba.

### **updatePost**

Permet modificar una publicació existent. Verifica que l'usuari sigui l'autor original. Gestiona:

- Actualització d'imatges (elimina l'antiga si es puja una nova)
- Modificació de títol, contingut, categoria i etiquetes
- Manté els camps existents si no es proporcionen nous valors

### **deletePost**

S'encarrega d'eliminar una publicació i tots els seus elements relacionats:

- Verifica l'autoria del post
- Elimina la imatge associada del servidor
- Elimina tots els comentaris relacionats
- Elimina tots els likes associats
- Elimina la publicació

### **getPostsByUser**

Recupera totes les publicacions d'un usuari específic. Ordena per data de creació descendente i inclou les dades bàsiques de l'autor. Retorna el recompte i la llista de posts.

### **posts/buyController**

#### **buyStock**

Gestiona tot el procés de compra d'accions. Primer verifica si el mercat està obert (3:30 PM - 10:00 PM CET). Comprova que l'usuari proporcioni el símbol de l'acció, la quantitat i el preu de compra, i valida que la quantitat sigui positiva.

Busca l'acció a la base de dades i verifica que l'usuari existeixi i tingui prou crèdit per la compra. Si l'usuari ja té accions d'aquesta empresa, calcula el nou preu mitjà i actualitza la quantitat. Si són accions noves, crea una nova entrada al portafolis amb tota la informació de l'acció (nom, sector, país, etc.).

Després de descomptar el cost del crèdit de l'usuari, crea una transacció amb tots els detalls (tipus "buy", quantitat, preu, cost total) i la guarda a la base de dades.

Si tot va bé, retorna un missatge d'èxit amb les dades actualitzades de l'usuari i la transacció. Si hi ha algun error, retorna un missatge d'error apropiat.

### **posts/marketStatusController**

#### **getMarketStatus**

Funció simple que verifica l'estat actual del mercat. Utilitza l'ajudant `isMarketOpen` del mòdul `marketHours` per determinar si el mercat està obert o tancat.

Retorna un objecte JSON amb una propietat `isOpen` que indica l'estat del mercat (true/false). Si hi ha algun error durant la verificació, retorna un error 500 amb el missatge corresponent.

L'estat del mercat es determina segons l'horari estàndard de la borsa de Nova York (NYSE):

- Horari de mercat: 3:30 PM - 10:00 PM CET
- Dies hàbils: dilluns a divendres
- Tancat: Caps de setmana i festius dels Estats Units

### **post/sellStock**

#### **sellStock**

Funció que gestiona la venda d'accions dins la plataforma. El procés comença verificant que el mercat estigui obert (horari 9:30 AM - 4:00 PM ET) i que l'usuari hagi proporcionat tota la informació necessària: símbol de l'acció, quantitat a vendre i preu de venda.

Abans de processar la venda, el sistema verifica l'existència de l'acció a la base de dades i comprova que l'usuari tingui prou accions disponibles per vendre. Un cop confirmades aquestes condicions, calcula l'ingrés total multiplicant el preu de venda per la quantitat.

El sistema actualitza automàticament el portafolis de l'usuari, eliminant l'entrada si es venen totes les accions o actualitzant la quantitat si és una venda parcial. També s'actualitza el crèdit de l'usuari sumant l'ingrés de la venda.

### **stock/stockController**

#### **initializeWebSocket**

Gestiona la connexió WebSocket amb Finnhub. Configura:

- Servidor WebSocket local
- Connexió amb Finnhub
- Sistema de reconnexió automàtica
- Heartbeat per verificar l'estat
- Gestió de subscripcions a símbols

#### **getStocksList**

Recupera la llista completa d'accions amb cache:

- Primer intenta llegir de Redis
- Si no existeix, obté dades de la BD
- Afegeix preus històrics de Yahoo
- Guarda en cache per 5 minuts
- getStockHistoricalData

Obté dades històriques d'una acció:

- Utilitza Yahoo Finance API
- Implementa cache amb TTL variable
- Suporta diferents intervals (1d, 1wk, 1mo, 1y)
- Formata les dades amb OHLCV

### **stockDetail**

Endpoint que retorna l'historial d'una acció:

- Rep símbol, interval i rang
- Utilitza getStockHistoricalData
- Gestiona errors de l'API

### **getAllStocks**

Endpoint que retorna totes les accions:

- Utilitza getStocksList
- Retorna dades en cache si disponibles
- Inclou preus actualitzats

### **getNews**

Obté notícies relacionades amb una acció:

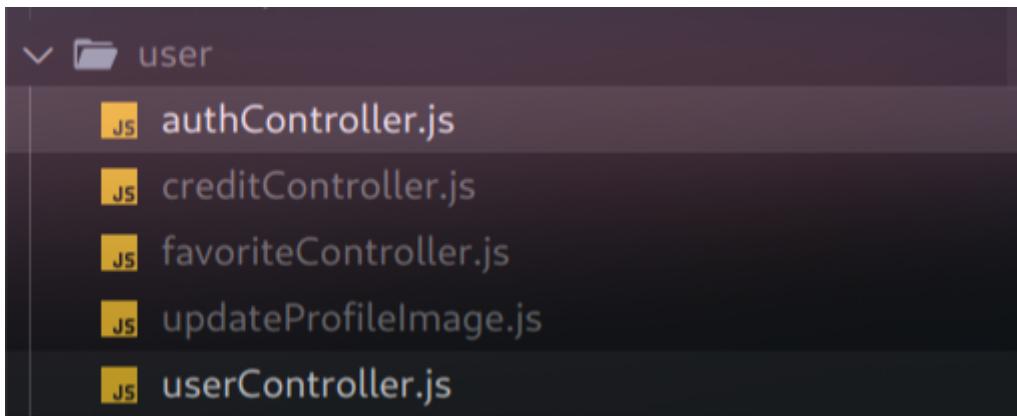
- Utilitza NewsAPI
- Filtra per mes actual
- Retorna 10 notícies més populars
- Timeout de 10 segons

### **flushCache**

Utilitat per desenvolupament:

- Neteja tota la cache de Redis
- Útil per testing i debug

## user/



### authController

#### registerUser

Rep les dades d'un nou usuari (nom, email, contrasenya, telèfon). Primer verifica que l'email no existeixi ja a la base de dades. Si és únic, encripta la contrasenya amb bcrypt, crea un nou usuari amb una imatge de perfil per defecte i el guarda. Després genera un token JWT amb l'ID de l'usuari, el guarda en una cookie segura i el retorna.

#### loginUser

Rep email i contrasenya. Busca l'usuari per email i verifica que la contrasenya coincideixi amb el hash emmagatzemat utilitzant bcrypt. Si les credencials són correctes, genera un token JWT, el guarda en una cookie configurada per cross-origin i el retorna juntament amb un missatge de confirmació.

#### logoutUser

Funció simple que elimina la cookie 'authToken' i retorna un missatge de confirmació de tancament de sessió. No requereix paràmetres més enllà de la que aquest per identificar la cookie a eliminar.

### user/creditController

#### addCredit

Gestiona l'addició de crèdit al compte d'un usuari. Verifica que la quantitat sigui positiva i que no superi el límit màxim establert (1.000.000). Si les validacions són correctes, actualitza el saldo de l'usuari a la base de dades. Inclou validacions per usuari no trobat i errors del servidor.

#### withdrawCredit

Gestiona la retirada de crèdit del compte d'un usuari. Realitza les següents comprovacions:

- La quantitat ha de ser positiva
- L'usuari ha d'existeix a la base de dades
- L'usuari ha de tenir saldo suficient

Si totes les validacions són correctes, resta la quantitat del crèdit de l'usuari i desa els canvis. Retorna el nou saldo actualitzat.

### user/favoriteController

**toggleFavorite:** Aquesta funció s'encarrega d'afegir o eliminar un valor dels favorits de l'usuari segons si ja existeix o no dins la seva llista. Rep el símbol del valor per paràmetre i comprova primer si és vàlid. Després busca l'usuari a la base de dades utilitzant l'identificador proporcionat pel middleware d'autenticació. Si el valor no es troba als favorits de l'usuari, consulta el model Stock per obtenir tota la informació del valor i l'afegeix complet a la llista de favorits. Si ja hi és, l'elimina de la llista. Finalment, guarda els canvis i retorna la nova llista de favorits. Si hi ha algun error en el procés, retorna l'error corresponent amb el codi adequat.

**getFavorites:** Aquesta funció s'encarrega de recuperar la llista actual de valors favorits de l'usuari autenticat. Fa una consulta a la base de dades pel seu identificador i retorna només el camp favs. Si l'usuari no existeix, retorna un error 404, i si hi ha algun error intern, retorna un error 500.

### users/updateProfileImage

**updateProfileImage:** Aquesta funció s'encarrega d'actualitzar la imatge de perfil de l'usuari autenticat. Comprova primer si s'ha pujat un fitxer a través de la petició; si no n'hi ha, retorna un error 400. Si hi ha una imatge, construeix la ruta relativa on s'ha desat (per exemple, /uploads/nom\_de\_l'arxiu) i actualitza el camp profileImage de l'usuari a la base de dades utilitzant el seu identificador. Retorna l'usuari actualitzat (sense la contrasenya) i un missatge d'èxit. Si l'usuari no existeix, retorna un error 404, i si hi ha algun problema intern durant el procés, retorna un error 500.

### user/userController

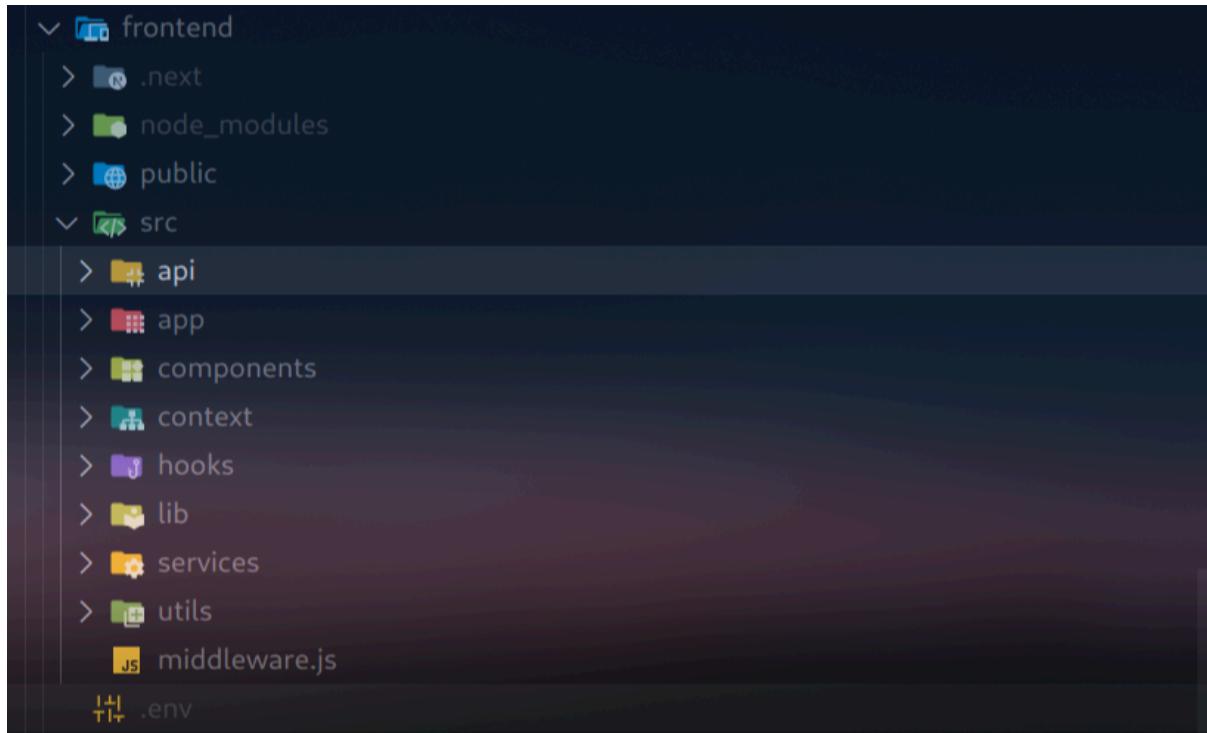
**getUserProfile:** Aquesta funció recupera el perfil complet de l'usuari autenticat, excepte la contrasenya. Comprova que hi hagi un usuari actiu a la petició, consulta la base de dades pel seu identificador i retorna l'objecte usuari. Si no es troba o hi ha un error, retorna el codi i missatge d'error corresponent.

**updateUserProfile:** Aquesta funció permet actualitzar dades bàsiques del perfil de l'usuari, com el nom, el correu, el telèfon o la imatge de perfil. Llegeix només els camps que s'han enviat a la petició i actualitza l'usuari a la base de dades amb aquests nous valors. Retorna l'usuari actualitzat (sense la contrasenya) o un error si no existeix o si hi ha algun problema durant l'operació.

**getUserStocks:** Aquesta funció obté només el camp stocks de l'usuari autenticat. Serveix per consultar quines accions o valors té actualment associats. Comprova que l'usuari estigui autenticat, accedeix a la base de dades pel seu identificador i retorna el camp stocks. Si l'usuari no existeix o hi ha un error, respon amb el codi d'error adequat.

**changePassword:** Aquesta funció permet a l'usuari canviar la seva contrasenya. Comprova que els camps currentPassword i newPassword s'hagin proporcionat. Recupera l'usuari de la base de dades, verifica que la contrasenya actual sigui correcta comparant-la amb l'encriptada, genera un nou hash amb la nova contrasenya i l'actualitza. Si tot és correcte, confirma el canvi; en cas contrari, retorna l'error corresponent.

Frontend:



api/

**stockApi**

**fetchStockData**

- Funció: `fetchStockData(symbol, interval = '5m', range = '1d')`
- Aquesta funció obté l'històric de preus d'una acció específica des del backend, utilitzant com a paràmetres:
  - `symbol`: el símbol de l'accio (ex: AAPL, TSLA, etc.)
  - `interval`: interval de temps entre dades (ex: cada 5 minuts)
  - `range`: rang de temps a consultar (ex: 1 dia)

Què fa:

- Construeix la URL amb els paràmetres.
- Fa una petició GET al backend.
- Si la resposta no és correcta, llança un error.
- Si tot va bé, retorna les dades en format JSON.

### fetchUserData

Funció: fetchUserData()

Aquesta funció recupera informació de l'usuari autenticat i les seves accions des del backend.

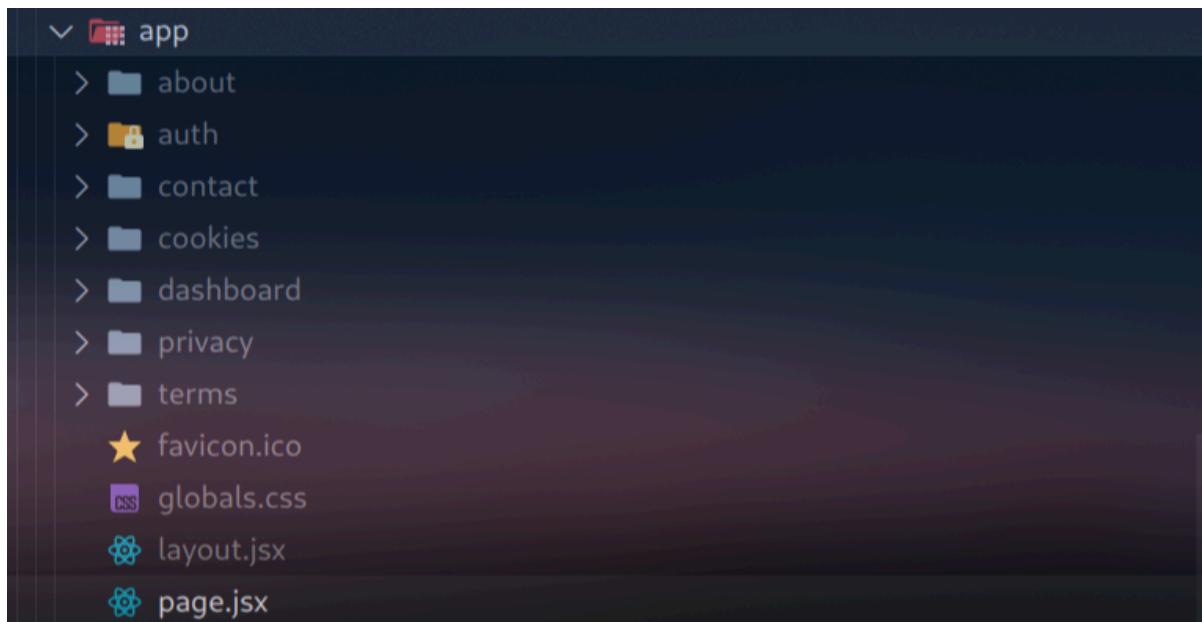
Què fa:

- Fa una petició GET a /user/profile per obtenir les dades bàsiques de l'usuari (com el crèdit disponible).
- Si hi ha error, el llança.
- Després fa una segona petició GET a /user/stocks per obtenir la llista d'accions que té l'usuari.

Retorna un objecte amb:

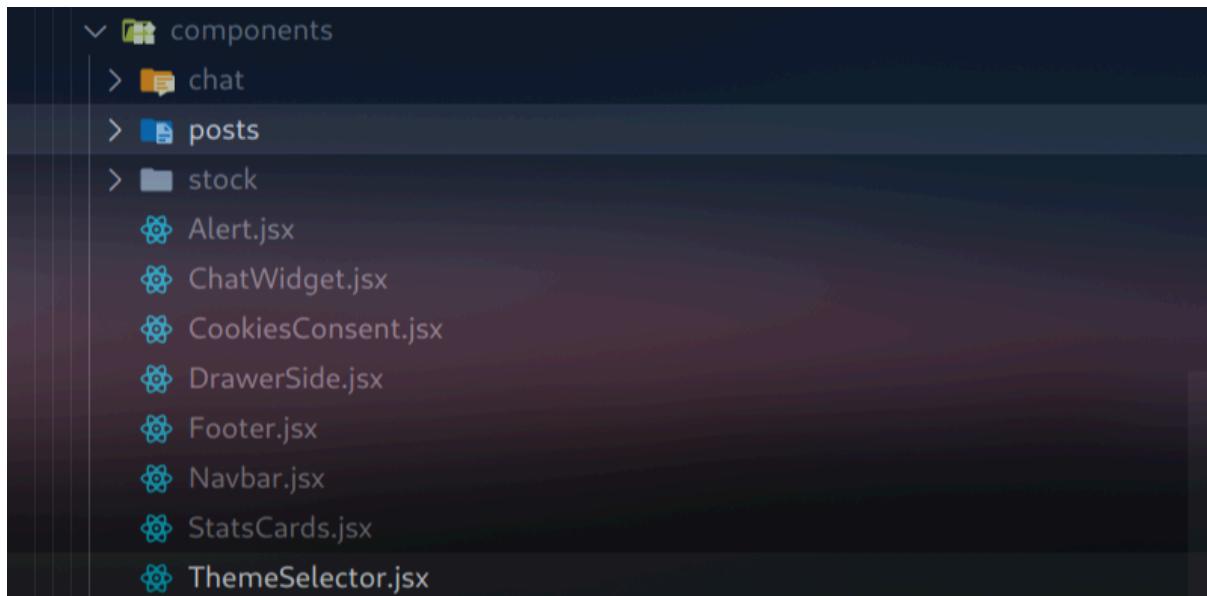
- credit: el crèdit disponible de l'usuari.
- stocks: la llista d'accions (si no n'hi ha, retorna un array buit).

### /app



En aquest directori podem trobar totes les rutes que té l'aplicació, cada un d'aquests directoris té un arxiu page.jsx que és l'arxiu que NextJS utilitza per mostrar com arrel, com l'index.html en un servidor normal. L'altre arxiu diferent és layout.jsx que conté l'estruatura de cada pàgina.

## /components



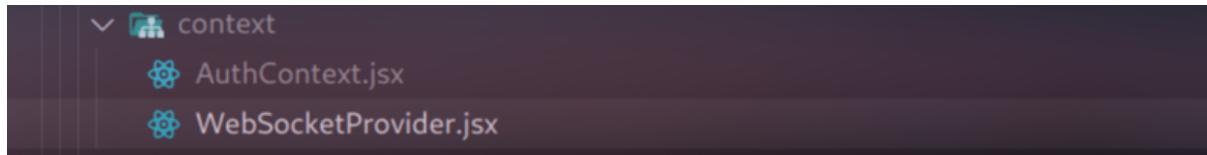
Després tenim /components on es pot observar els diferents components de l'aplicatiu. Està separat per aplicacions, on les que no tenen directori són els components del directori arrel.

Cada component té les seves funcions i la seva lògica per funcionar dependent de la funcionalitat que tinguin, alguns components tenen molta lògica dins i altres està més modularitzat.

Propostes:

S'hauria de treballar a terminar de modularitzar i reciclar codi. Per tal que els components hi hagi la menor quantitat de lògica repetida possible.

## /context



Un context a React és un mecanisme que permet compartir dades i funcions entre diferents components sense haver de passar-les com a props manualment per tots els nivells del component.

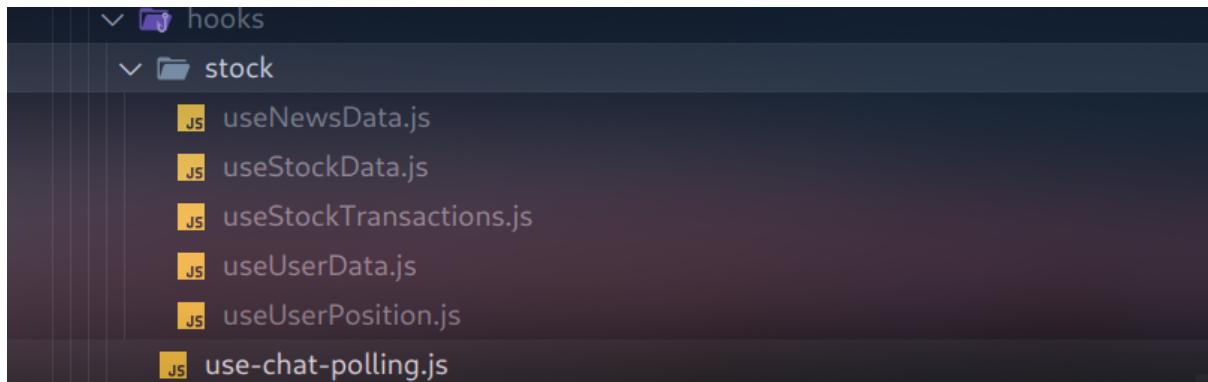
### Context d'Autenticació:

És un espai on es guarda i comparteix la informació de l'usuari que estàloguejat a l'aplicació, juntament amb funcions per entrar (login), sortir (logout) i comprovar si està autenticat. Així, qualsevol component pot saber qui està connectat i actuar en conseqüència sense haver de passar aquesta informació de component en component.

### Context de WebSocket:

És un espai on es gestiona la connexió en temps real amb un servidor via WebSocket. Guarda l'estat de la connexió, les dades rebudes i controla quan cal tornar a connectar-se si es perd la connexió. Això permet que diferents parts de l'aplicació puguin utilitzar dades en directe i l'estat de la connexió sense haver de configurar la connexió en cada component.

## /hooks



En els arxius hooks és on es gestionen la major part d'estats dels components com són useEffect, useState.

### **useNewsData:**

Aquest hook personalitzat s'encarrega de carregar i gestionar les notícies relacionades amb un símbol borsari quan l'usuari decideix obrir la secció de notícies (newsOpen). Manté l'estat de les notícies i si estan en procés de càrrega, fent una petició a l'API corresponent cada vegada que el símbol o l'estat d'obertura canviem, i actualitza la informació o gestiona errors segons correspongui.

### **useStockData:**

Aquest hook personalitzat gestiona tota la informació d'una acció concreta i l'usuari relacionat. Carrega inicialment les dades històriques de l'acció i les dades de l'usuari (crèdit i accions que té), sincronitzant la càrrega per saber quan ha acabat. A més, escolta dades en temps real a través d'un WebSocket per actualitzar el preu, volum i canvis de l'acció de forma instantània. Tot això permet tenir sempre l'estat actualitzat i controlar quan la informació està carregada o no.

### **useStockTransactions:**

Aquest hook useStockTransactions gestiona la lògica per comprar i vendre accions d'una acció concreta. Manté l'estat de la quantitat d'euros (amount) i el nombre d'accions (shares) que l'usuari vol operar, i sincronitza aquests dos valors segons el preu actual de l'acció. També inclou funcions per fer peticions al backend per comprar o vendre accions, i després d'aquestes operacions actualitza les dades de l'usuari cridant loadUserData. A més, té una funció per vendre un percentatge concret d'accions, facilitant la selecció ràpida d'imports.

Aquest hook centralitza tota la gestió i actualització de les transaccions d'una manera simple i reutilitzable.

**useUserData:**

S'encarrega de carregar i gestionar les dades principals de l'usuari: el crèdit disponible, les accions que té i el nombre total de transaccions que ha fet. Quan es monta el component que l'utilitza, fa tres peticions al backend per obtenir aquestes dades i les guarda en estats locals. També controla l'estat de càrrega per saber quan ha acabat de carregar tota la informació. En cas d'error, mostra el missatge per consola i també desactiva la càrrega. Així se centralitza la gestió i actualització de les dades de l'usuari de forma fàcil i reutilitzable.

**useUserPosition:**

Calcula la posició que té un usuari en una acció concreta, usant les dades de l'estoc i les accions de l'usuari). Calcula el valor total de la posició actual, el rendiment absolut i percentual respecte a la compra, les accions que té, el preu d'adquisició, el percentatge que ocupa aquesta acció dins del seu portafolis i el preu actual de l'acció. Si no hi ha dades o l'usuari no té accions d'aquell símbol, retorna valors a zero. Tot això es recalcula automàticament quan canvia el preu de l'acció, les accions de l'usuari o el símbol.

**useChatPolling:**

Gestiona l'estat i la sincronització d'un xat en temps real fent peticions periòdiques per obtenir les converses i els missatges actualitzats mentre el xat està obert. Utilitza referències per evitar re-renderitzacions innecessàries i permet buscar usuaris, iniciar nous xats i enviar missatges amb actualitzacions optimistes a la UI, tot controlant errors i l'estat de càrrega de manera eficient.

**/services**

En aquest arxiu hi ha només les funcions necessàries per a la gestió de posts.

Millors:

En un futur caldria afegir més arxius per separar encara més les funcionalitats i tenir-ho tot millor ordenat.

**createPost:**

Aquesta funció envia una sol·licitud POST per crear un nou post. Utilitza multipart/form-data perquè es poden enviar fitxers juntament amb les dades del formulari. El token d'autenticació s'obté de localStorage i s'envia en la capçalera Authorization perquè el servidor pugui validar la petició.

**getPosts:**

Aquesta funció fa una sol·licitud GET per recuperar tots els posts disponibles. El token d'autenticació també s'envia a través de la capçalera Authorization per assegurar que només els usuaris autenticats puguin accedir a aquesta informació.

**getPostById:**

Aquesta funció sol·licita un post específic pel seu identificador. Fa una petició GET passant l'ID a l'URL i envia el token per autenticació, cosa que permet obtenir el detall d'un post concret només a usuaris autoritzats.

**updatePost:**

Aquesta funció actualitza un post existent mitjançant una petició PUT. També utilitza multipart/form-data perquè es puguin modificar fitxers o imatges juntament amb les dades del post. El token s'envia a la capçalera per garantir que l'usuari tingui permisos per modificar el post.

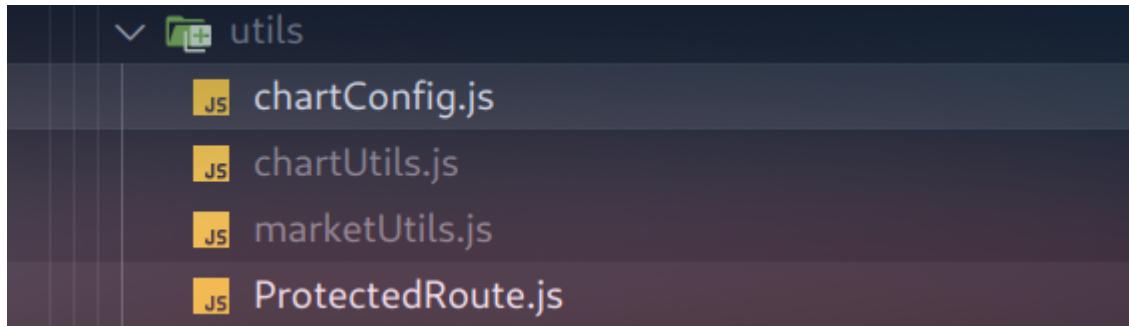
**deletePost:**

Aquesta funció elimina un post especificat pel seu ID mitjançant una petició DELETE. El token d'autenticació es passa en la capçalera per assegurar que només l'usuari autoritzat pugui eliminar el post.

**getPostsByUser:**

Aquesta funció recupera tots els posts publicats per un usuari concret, passant el seu ID a l'URL. La petició GET inclou el token a la capçalera perquè només usuaris autenticats puguin veure aquests posts.

## /utils



En aquest directori es pot trobar diferents arxius d'utilitats que s'utilitzen al llarg del programa.

### **chartConfig**

#### **getChartData:**

Aquesta funció extreu dues llistes de dades d'un objecte stock: una amb els horaris de cada registre convertits a format de temps (hores i minuts), i una altra amb els preus de tancament (close). Si no existeix l'historial, retorna arrays buides.

#### **chartOptions:**

Aquesta constant defineix la configuració general del gràfic. El gràfic és responsiu, no manté una proporció fixa, i desactiva la llegenda. També configura que les eines d'ajuda mostrin informació quan s'hi passa el cursor. A més, personalitza els eixos, com el color i la posició, i l'aspecte de la línia i punts del gràfic.

#### **getChartConfig:**

Aquesta funció crea l'objecte de configuració específic per al gràfic, amb les etiquetes (temps) i les dades (preus). També estableix el color de la línia i els fons segons si el canvi de preu és positiu (verd) o negatiu (vermell), fent així que el gràfic sigui visualment més clar segons la tendència del valor.

### chartUtils

#### **getInterval:**

Aquesta funció rep un període (period) i retorna la unitat de temps adequada per fer les consultes de dades (intervals). Per exemple, per un dia ("1D") retorna "5m" (5 minuts), per una setmana ("1W") retorna "1h" (1 hora), i per períodes més llargs com un any o cinc anys, retorna intervals diaris o setmanals. Si el període no coincideix amb cap cas, retorna "5m" per defecte.

#### **getRange:**

Aquesta funció també rep un període i retorna el rang de dades corresponent en format abreujat. Per exemple, per un dia retorna "1d", per una setmana "5d" (5 dies), per un mes "1mo" (1 mes), i així successivament fins a cinc anys "5y". Si el període no és reconegut, retorna "1d" per defecte.

### /marketutils

#### **isMarketClosed:**

Aquesta funció comprova si el mercat està tancat segons l'hora actual. Considera que el mercat està tancat els caps de setmana (diumenge i dissabte) i fora de l'horari laboral entre les 15:30 i les 22:00 (hora ET). Retorna true si el mercat està tancat i false si està obert.

#### **MarketClosedAlert:**

Aquest és un component React que mostra una alerta visual quan el mercat està tancat. Conté una icona d'avertència i un missatge que indica que el mercat està tancat i l'horari oficial d'obertura (de dilluns a divendres, de 15:30 a 22:00 ET).

## /middleware



Aquest middleware serveix per controlar l'accés a certes rutes de la teva aplicació Next.js, comprovant si l'usuari està autenticat mitjançant un token JWT emmagatzemat en una cookie.

Primer, s'obté el token des de la cookie anomenada `jwtToken`. Si el token existeix, es tracta de descodificar la seva part del mig (el payload) que està codificat en base64. Dins del payload, es busca la propietat `exp` que indica el temps d'expiració del token en segons des de l'època Unix.

Seguidament, es comprova si el token és vàlid comparant la data actual amb la data d'expiració del token. Si el token està caducat o no es pot descodificar correctament, es considera invàlid.

Després, el middleware verifica si la ruta a la qual s'accedeix comença per `/dashboard`. Si és així, i no hi ha un token vàlid, l'usuari és redirigit a la pàgina de login perquè s'autentiqui.

També, si l'usuari intenta accedir a la pàgina de login (en aquest cas `/auth/login`) però ja té un token vàlid, se'l redirigeix directament al dashboard per evitar que entri de nou al login.

Finalment, si no es compleix cap de les condicions anteriors, el middleware permet continuar amb la petició sense redireccions. Això assegura que només els usuaris autenticats puguin accedir a les rutes protegides i que els usuaris ja autenticats no puguin veure la pàgina de login.

- Convencions de nomenclatura

Per tot el codi s'ha utilitzat camelCase tant per frontend com backend, ja que es una bona pràctica escriure JavaScript d'aquesta forma.

- Patrons implementats

S'ha implementat un patró Model Controlador. On des de l'api del backend gestionem models i controladors i des del frontend la part visual.

## 7. APIs i Interfícies

### 7.1 APIs Internes

POST /register

Descripció: Crea un nou usuari a la base de dades.

Body (JSON):

```
{  
  "name": "Nombre del usuario",  
  "email": "usuario@example.com",  
  "password": "contraseñaSegura123",  
  "phone": "123456789"  
}
```

**Respostes:**

```
{  
  "msg": "Usuario registrado",  
  "token": "JWT_GENERADO"  
}
```

### POST /login

- Descripció: Inicia sessió amb email i contrasenya vàlids
- Body (JSON):

```
{  
  "email": "usuario@example.com",  
  "password": "contraseñaSegura123"  
}
```

### Resposta

```
{  
  "msg": "Usuario logeado",  
  "token": "JWT_GENERADO"  
}
```

### POST /logout

- Descripció: Tanca sessió eliminant la cookie d'autenticació.
- Requereix autenticació prèvia (cookie authToken)
- Respostes:

```
{  
  "msg": "Cierre de sesión exitoso"  
}
```

## src/routes/chatRoutes.js

GET /chat/conversations

Descripció: Obté totes les converses de l'usuari autenticat.

Requereix autenticació:  (Token JWT)

**Resposta:**

```
[  
  {  
    "id": "665000ca9e191934d6a7e2c1",  
    "user": {  
      "_id": "664ffffd19e191934d6a7e2b4",  
      "name": "María López",  
      "profileImage": "/uploads/profiles/maria.jpg"  
    },  
    "lastMessage": "Hola, ¿cómo estás?",  
    "timestamp": "2025-05-22T13:40:19.123Z"  
  }  
]
```

GET /chat/:conversationId/messages

Descripció: Obté tots els missatges d'una conversa específica.

Requereix autenticació.

Parametres

- conversationId: ID de la conversació

### Resposta

```
[  
  {  
    "id": "665002af9e191934d6a7e2f7",  
    "text": "Hola, ¿cómo estás?",  
    "timestamp": "2025-05-22T13:50:10.456Z",  
    "isMe": false  
  },  
  {  
    "id": "665002c19e191934d6a7e2f8",  
    "text": "Todo bien, ¿y tú?",  
    "timestamp": "2025-05-22T13:50:20.789Z",  
    "isMe": true  
  }  
]
```

POST /chat/:conversationId/messages

- Descripció: Envia un missatge a una conversa existent.
- Requereix autenticació
- Paràmetres d'URL:
- conversationId: ID de la conversa

```
{  
  "text": "¿Nos vemos mañana?"  
}
```

### Resposta:

```
{  
  "id": "665003ff9e191934d6a7e2f9",  
  "text": "¿Nos vemos mañana?",  
  "timestamp": "2025-05-22T13:55:30.321Z",  
  "isMe": true  
}
```

### POST /chat/new

Descripció: Crea una conversa nova amb un altre usuari.  
Requereix autenticació.

```
{  
  "userId": "664ffffd19e191934d6a7e2b4"  
}
```

### Respostes:

```
{  
  "id": "665005aa9e191934d6a7e2fa",  
  "message": "Conversación creada con éxito"  
}
```

### GET /chat/users

Descripció: Obté tots els usuaris disponibles per iniciar conversa, excepte l'usuari actual.  
Requereix autenticació

### Resposta

```
[  
  {  
    "_id": "664ffffd19e191934d6a7e2b4",  
    "name": "María López",  
    "profileImage": "/uploads/profiles/maria.jpg",  
    "createdAt": "2025-05-01T10:00:00.000Z",  
    "updatedAt": "2025-05-20T15:30:00.000Z",  
    "__v": 0  
  },  
  {  
    "_id": "664ffffd19e191934d6a7e2b5",  
    "name": "Carlos Pérez",  
    "profileImage": "/uploads/profiles/carlos.jpg",  
    "createdAt": "2025-05-02T12:15:00.000Z",  
    "updatedAt": "2025-05-21T11:45:00.000Z",  
    "__v": 0  
  }  
]
```

### GET /favorites

Descripció: Obté tots els favorits de l'usuari autenticat.

Requereix autenticació

Resposta

```
{  
  "favs": [  
    {  
      "symbol": "AAPL",  
      "name": "Apple Inc.",  
      "sector": "Technology",  
      "industry": "Consumer Electronics",  
      "exchange": "NASDAQ",  
      "country": "USA",  
      "currency": "USD",  
      "description": "Apple Inc. designs, manufactures, and markets  
smartphones, personal computers...",  
      "website": "https://www.apple.com",  
      "logo": "https://logo.clearbit.com/apple.com"  
    }, ...  
  ]  
}
```

### PATCH /favorites/:symbol

Descripció: Afegeix o elimina un símbol de l'array de favorits de l'usuari autenticat.  
Requereix autenticació

Paràmetres d'URL:

symbol: símbol de l'estoc (per exemple: AAPL, TSLA)

### Resposta:

```
{  
  "favs": [  
    {  
      "symbol": "TSLA",  
      "name": "Tesla, Inc.",  
      "sector": "Consumer Cyclical",  
      "industry": "Automobiles",  
      "exchange": "NASDAQ",  
      "country": "USA",  
      "currency": "USD",  
      "description": "Tesla designs, develops, manufactures, and sells electric vehicles and energy storage products.",  
      "website": "https://www.tesla.com",  
      "logo": "https://logo.clearbit.com/tesla.com"  
    }  
  ]  
}
```

### Rutes de Mercat

#### GET /status

Descripció: Obté l'estat actual del mercat (obert o tancat).  
Autenticació: No requerida.

```
{  
  "marketStatus": "open",  
  "message": "El mercado está abierto"  
}
```

POST /buy

Descripció: Permet comprar accions.  
Requereix autenticació

```
{  
    "symbol": "AAPL",  
    "quantity": 10,  
    "buyPrice": 150.50  
}
```

### Resposta

```
{  
    "message": "Compra realizada exitosamente",  
    "user": { /* usuario actualizado */ },  
    "transaction": { /* detalle de la transacción */ }  
}
```

POST /sell

Descripció: Permet vendre accions.

Requereix autenticació

```
{  
  "symbol": "AAPL",  
  "quantity": 5,  
  "sellPrice": 152.00  
}
```

### Resposta

```
{  
  "message": "Venta realizada exitosamente",  
  "user": { /* usuario actualizado */ },  
  "transaction": { /* detalle de la transacción */ }  
}
```

GET /stocks

Descripció: Llista totes les accions disponibles amb info bàsica i històric escorcollat.

### Resposta

```
{  
    "AAPL": {  
        "name": "Apple Inc.",  
        "sector": "Technology",  
        "industry": "...",  
        "exchange": "...",  
        "country": "...",  
        "currency": "...",  
        "description": "...",  
        "website": "...",  
        "logo": "...",  
        "firstPriceToday": 123.45,  
        "lastYahooPrice": 124.00  
    }  
}
```

GET /news

Descripció: Obté notícies financeres rellevants al símbol enviat com a query parameter per al mes actual, ordenades per popularitat (màxim 10 notícies).

**Paràmetres de query:**

symbol (string, obligatori): símbol de l'estoc per filtrar notícies (exemple: AAPL, TSLA).

**Resposta**

```
{  
  "articles": [  
    {  
      "source": { "id": null, "name": "ExampleSource" },  
      "author": "Autor de la noticia",  
      "title": "Título de la noticia",  
      "description": "Descripción breve",  
      "url": "https://url-de-la-noticia.com",  
      "urlToImage": "https://url-de-imagen.jpg",  
      "publishedAt": "2025-05-01T12:00:00Z",  
      "content": "Contenido completo o parcial de la noticia"  
    },  
    {  
      "source": { "id": "bbc", "name": "BBC News" },  
      "author": "BBC",  
      "title": "Otra noticia relacionada",  
      "description": "Descripción breve",  
      "url": "https://bbc.com/otra-noticia",  
      "urlToImage": null,  
      "publishedAt": "2025-05-02T09:30:00Z",  
      "content": "Contenido completo o parcial"  
    }  
    // hasta 10 artículos  
  ]  
}
```

GET /stocks/:symbol

Descripció: Obté les dades completes d'una acció específica pel símbol.  
Requereix autenticació

Paràmetres d'URL:

- symbol: símbol de l'acció (per exemple: AAPL, TSLA)

**Resposta:**

```
{  
  "symbol": "AAPL",  
  "name": "Apple Inc.",  
  "sector": "Technology",  
  "industry": "Consumer Electronics",  
  "exchange": "NASDAQ",  
  "country": "USA",  
  "currency": "USD",  
  "description": "Apple Inc. designs, manufactures, and markets  
smartphones, personal computers...",  
  "website": "https://www.apple.com",  
  "logo": "https://logo.clearbit.com/apple.com"  
}
```

GET /api/posts/

Descripció: Retorna una llista amb tots els posts.

Resposta exitosa

```
{  
  "success": true,  
  "count": 2,  
  "data": [  
    {  
      "_id": "postId",  
      "title": "Título",  
      "image": "url_imagen",  
      "author": { "_id": "userId", "name": "Usuario" },  
      "likesCount": 3,  
      "commentsCount": 5,  
      "createdAt": "2025-05-20T00:00:00.000Z"  
    }  
  ]  
}
```

GET /api/posts/:id

Descripció: Obté un post pel seu ID.

```
{  
  "data": [  
    {  
      "_id": "postId",  
      "title": "Título",  
      "image": "url_imagen",  
      "author": { "_id": "userId", "name": "Usuario" },  
      "likesCount": 3,  
      "commentsCount": 5,  
      "createdAt": "2025-05-20T00:00:00.000Z"  
    }  
  ]}
```

## Comentaris

POST /api/posts/:postId/comments

Descripció: Crea un comentari en un post.

```
{  
  "content": "Buen post, espero que sea el ultimo!"  
}
```

Resposta:

```
{  
  "success": true,  
  "data": {  
    "_id": "commentId",  
    "content": "Buen post, espero que sea el ultimo!",  
    "author": {  
      "_id": "userId",  
      "name": "Juan",  
      "email": "juan@email.com",  
      "profileImage": "imagen.jpg"  
    },  
    "postId": "postId",  
    "createdAt": "2025-05-21T00:00:00.000Z"  
  }  
}
```

GET /api/posts/:postId/comments

Descripció: Llista els comentaris d'un post

Resposta

```
{  
  "success": true,  
  "data": {  
    "_id": "commentId",  
    "content": "Buen post, espero que sea el ultimo!",  
    "author": {  
      "_id": "userId",  
      "name": "Juan",  
      "email": "juan@email.com",  
      "profileImage": "imagen.jpg"  
    },  
    "postId": "postId",  
    "createdAt": "2025-05-21T00:00:00.000Z"  
  }  
}
```

PUT /api/posts/comments/:id

Descripció: Actualitza el comentari si ets el propietari del comentari.

```
{  
  "content": "Comentario editado"  
}
```

DELETE /api/posts/comments/:id

Descripció: Elimina un comentari si ets el propietari

**Resposta:**

```
{  
    success: true,  
    message: 'Comentario eliminado correctamente'  
}
```

**Endpoints de Likes**

POST /api/posts/:postId/like

Descripció: Afegeix un like al post

Resposta

```
{  
    "success": true,  
    "message": "Like agregado correctamente"  
}
```

DELETE /api/posts/:postId/like

Descripció: Elimina el like del post.

**Resposta:**

```
{  
    success: true,  
    message: 'Like eliminado correctamente'  
}
```

GET /api/posts/:postId/likes

Obté els likes d'un post

**Resposta**

```
{  
    success: true,  
    count: likes.length,  
    data: likes  
}
```

POST /posts

Descripció: Crear un nou post

Requereix autenticació.

title (string, obligatori): Títol del post.

content (string, obligatori): Contingut del post.

category (string, opcional): Categoria del post.

tags (string, opcional): Llista de tags separada per comes.

image (arxiu, opcional): Imatge per al post.

### Resposta

```
{  
  "success": true,  
  "data": {  
    "_id": "postId",  
    "title": "Título del post",  
    "content": "Contenido del post",  
    "image": "/uploads/imagen.jpg",  
    "author": "userId",  
    "category": "Categoría",  
    "tags": ["tag1", "tag2"],  
    "createdAt": "...",  
    "updatedAt": "..."  
  }  
}
```

GET /posts/:id

Descripció: Obtenir un post pel ID.

**Parametres:**

id (string): ID del post.

**Resposta**

```
{  
  "success": true,  
  "data": {  
    "_id": "postId",  
    "title": "Título del post",  
    "content": "Contenido del post",  
    "image": "/uploads/imagen.jpg",  
    "author": {  
      "_id": "userId",  
      "name": "Nombre autor",  
      "email": "email@ejemplo.com",  
      "profileImage": "url-imagen"  
    },  
    "category": "Categoría",  
    "tags": ["tag1", "tag2"],  
    "createdAt": "...",  
    "updatedAt": "..."  
  }  
}
```

PUT /posts/:id

Actualizar un post existente.

Requereix autenticació.

### Parametres

id (string): ID del post.

FormData (multipart/form-data):

title (string, opcional): Nuevo título.

content (string, opcional): Nuevo contenido.

category (string, opcional): Nueva categoría.

tags (string, opcional): Nuevos tags separados por coma.

image (archivo, opcional): Nueva imagen (si se sube, elimina la anterior).

### Resposta

```
{  
  "success": true,  
  "data": {  
    "_id": "postId",  
    "title": "Título actualizado",  
    "content": "Contenido actualizado",  
    "image": "/uploads/nueva-imagen.jpg",  
    "author": "userId",  
    "category": "Categoría",  
    "tags": ["tag1", "tag2"],  
    "createdAt": "...",  
    "updatedAt": "..."  
  }  
}
```

DELETE /posts/:id

Descripció: Eliminar un post si ets l'autor.  
Requereix autenticació

id (string): ID del post.

### Resposta

```
{  
  "success": true,  
  "message": "Post eliminado correctamente"  
}
```

GET /api/transactions/

Descripció: Obté totes les transactions fetes per l'usuari que fa la petició.  
Requereix autenticació.

### Resposta

```
{  
  "transactions": [  
    {  
      "_id": "transactionId",  
      "userId": "userId",  
      "type": "buy",  
      "symbol": "AAPL",  
      "quantity": 10,  
      "price": 150.25,  
      "date": "2025-05-21T00:00:00.000Z"  
    }  
  ]  
}
```

GET /api/users/profile

Descripció: Obté les dades del perfil autenticat

### Resposta

```
{  
  "_id": "6647c92fc0327d001e18e349",  
  "name": "Samu",  
  "email": "samu@example.com",  
  "phone": "123456789",  
  "profileImage": "https://...jpg",  
  "stocks": [],  
  "createdAt": "2024-05-01T12:00:00.000Z",  
  ...  
}
```

PUT /api/users/profile

Descripción: Actualizar perfil del usuario.

```
{  
  "name": "Nuevo nombre",  
  "email": "nuevo@email.com",  
  "phone": "111222333",  
  "profileImage": "https://...jpg"  
}
```

### Resposta

```
{  
  "msg": "Perfil actualizado correctamente",  
  "user": {  
    "_id": "...",  
    "name": "Nuevo nombre",  
    "email": "nuevo@email.com",  
    ...  
  }  
}
```

PUT /api/users/profile-image

Descripció: Actualitza la imatge de perfil de l'usuari.

**Resposta**

```
{ "msg": "Imagen de perfil actualizada correctamente", "imageUrl":  
"https://..." }
```

PUT /api/users/change-password

Descripció: Canvia la password del usuari

```
{  
  "currentPassword": "contraseñaActual",  
  "newPassword": "nuevaContraseñaSegura"  
}
```

**Resposta**

```
{ "msg": "Contraseña actualizada correctamente" }
```

GET /api/users/stocks

Descripción: Obtener las acciones (stocks) del usuario autenticado.

```
{  
  "stocks": [  
    { "symbol": "AAPL", "quantity": 5 },  
    { "symbol": "GOOG", "quantity": 3 }  
  ]  
}
```

POST /api/users/addCredit

Descripció: Afageix credit al usuari.

```
{ "amount": 100 }
```

Resposta

```
{ "msg": "Crédito añadido correctamente", "balance": 300 }
```

POST /api/users/withdrawCredit

Descripció: Retira salari de la cartera.

Requereix autenticació.

```
{ "amount": 50 }
```

Resposta

```
{ "msg": "Crédito retirado correctamente", "balance": 250 }
```

## 7.2 APIs Externes

### Serveis de tercers utilitzats

El sistema utilitza diverses APIs externes per obtenir dades financeres i notícies en temps real:

#### Yahoo Finance API

Utilitzada per recuperar informació general de mercat, incloent dades històriques, preus actuals, gràfics i mètriques fonamentals d'actius financers. S'utilitza majoritàriament per consultes puntuals o dades no en temps real.

#### Exemple resposta:

```
{  
    "chart": {  
        "result": [  
            {  
                "meta": {  
                    "currency": "", "symbol": "", "exchangeName": "", "fullExchangeName": "", "instrumentType": "",  
                    "firstTradeDate": 0, "regularMarketTime": 0, "hasPrePostMarketData": false, "gmtOffset": 0,  
                    "timezone": "", "exchangeTimezoneName": "", "regularMarketPrice": 0,  
                    "fiftyTwoWeekHigh": 0, "fiftyTwoWeekLow": 0, "regularMarketDayHigh": 0,  
                    "regularMarketDayLow": 0, "regularMarketVolume": 0,  
                    "longName": "", "shortName": "", "chartPreviousClose": 0, "priceHint": 0,  
                    "currentTradingPeriod": {  
                        "pre": { "timezone": "", "end": 0, "start": 0, "gmtOffset": 0 },  
                        "regular": { "timezone": "", "end": 0, "start": 0, "gmtOffset": 0 },  
                        "post": { "timezone": "", "end": 0, "start": 0, "gmtOffset": 0 }  
                    },  
                    "dataGranularity": "", "range": "", "validRanges": []  
                },  
                "timestamp": [],  
                "indicators": {  
                    "quote": [  
                        { "low": [], "close": [], "high": [], "open": [], "volume": [] }  
                    ],  
                    "adjclose": [  
                        { "adjclose": [] }  
                    ]  
                }  
            }  
        ]  
    }  
}
```

```
],
  "error": null
}
}
```

### Finnhub API

Utilitzada per rebre dades de mercat en temps real a través de WebSocket. Proporciona informació de transaccions (trade) per símbol, incloent preu, volum, hora i més. També permet consultar dades fonamentals i notícies.

Petició per SYMBOL.

```
{
  "c": 0,    // current price
  "d": 0,    // change
  "dp": 0,   // percent change
  "h": 0,    // high of the day
  "l": 0,    // low of the day
  "o": 0,    // open of the day
  "pc": 0,   // previous close
  "t": 0     // timestamp
}
```

### NewsCatcher API

Utilitzada per obtenir notícies relacionades amb finances, economia i empreses específiques. Permet fer cerques per paraula clau, idioma i rang de dates, i filtrar resultats rellevants per l'usuari.

#### Exemple:

[https://newsapi.org/v2/everything?q=\\${symbol}+financial&from=\\${firstDay}&to=\\${lastDay}&sortBy=popularity&pageSize=10&apiKey=\\${apiKey}](https://newsapi.org/v2/everything?q=${symbol}+financial&from=${firstDay}&to=${lastDay}&sortBy=popularity&pageSize=10&apiKey=${apiKey})

#### Retorna:

```
{"status":"ok",
"totalResults":179,
"articles":[{
  "source":{"id":"business-insider",
  "name":"Business Insider"},
  "author":"wedwards@businessinsider.com","title":"9 stocks that
  investing legends like Warren Buffett are betting at least 20% of
  their portfolios on",
  "description":"Warren Buffett is stepping down. Here's the one
  stock he's betting 28% of his portfolio
  on.", "url":"https://www.businessinsider.com/warren-buffett-stocks-
  top-investors-are-betting-on-whale-watch-2025-5", "urlToImage":"htt
  ps://i.insider.com/6818f2683fe8d3928364f07e?width=921&format=jpeg"
  , "publishedAt":"2025-05-06T08:00:01Z", "content":"Warren Buffett,
  perhaps the greatest investor of all time, announced on Saturday
  that he would step down as CEO of Berkshire Hathaway at the end of
  the year.\r\nGiven the occasion, we thought it would ... [+2779
  chars]"},
```

## 7.3. Websocket

### WS /websocket

subscribeToSymbols(socket, symbols)

#### Descripció

Subscripció a una llista de símbols de borsa mitjançant una connexió WebSocket activa. Aquesta funció elimina qualsevol subscripció anterior i registra els nous símbols.

#### Paràmetres

socket (WebSocket): Connexió WebSocket cap a Finnhub.

symbols (string[]): Llista de símbols a subscriure.

#### Details

Comprova que el socket estigui obert abans d'enviar missatges.

Envia un missatge de tipus subscribe per cada símbol.

Manté el seguiment dels símbols subscrits mitjançant un conjunt (Set).

Marca la subscripció com activa (finnhubSubscriptionActive = true).

checkSubscriptions(socket, symbols)

#### Descripció

Comprova si hi ha una subscripció activa a Finnhub. Si no n'hi ha, torna a subscriure tots els símbols.

#### Paràmetres

socket (WebSocket): Connexió WebSocket.  
symbols (string[]): Llista de símbols.

Retorn

true si s'ha tornat a subscriure.  
false si no calia fer-ho.

initializeWebSocket(server)

#### Descripció

Inicialitza el servidor WebSocket i la connexió amb Finnhub per rebre dades de borsa en temps real.

#### Paràmetres

server (http.Server): Instància de servidor HTTP sobre la qual es muntarà el WebSocket.

#### Details de funcionament

Connexió a Redis

Intenta connectar-se a Redis abans d'inicialitzar res més.

Inicialització de WebSocket del servidor

Es crea un servidor WebSocket local en la ruta /websocket.

Obtenció de símbols

Es recupera la llista de símbols a subscriure mitjançant getStocksList.

Connexió a Finnhub (connectToFinnhub)

Estableix la connexió amb el WebSocket públic de Finnhub.

**Inclou, per tenir un sistema robust:**

Connexió i subscripció: Connexió amb timeout. Subscripció automàtica a símbols. Ping cada 30 s per mantenir-la viva. Comprovació de subsrcipcions cada 2 min.

Recepció de dades: Es processen missatges de tipus trade (preu, volum, hora, símbol) i es reenvien a tots els clients locals. Es gestionen missatges pong per verificar la connexió.

Errors i reconnexió: En cas d'error o tancament, es programa una reconnexió. La freqüència s'adapta segons si el mercat està obert.

Heartbeats: Cada 5 min es verifica l'estat de la connexió. Si cau, es reconnecta o es torna a subscriure.

Connexions de clients: S'accepten nous clients locals. Es gestionen errors i desconnexions individuals.

Tancament segur: En rebre SIGTERM, es tanquen connexions i intervals de forma controlada.

**Retorna**

Retorna la instància del servidor WebSocket creat (wss).

scheduleReconnect()

**Descripció**

Planifica una nova connexió a Finnhub en cas de desconnexió. La freqüència de reconnexió varia en funció de l'horari de mercat.

**Details**

Durant l'horari de mercat (15:00 - 22:00 UTC aprox.), es reconnecta ràpidament (cada 5 segons).

Fora de l'horari de mercat, la reconnexió és menys freqüent (cada 5 minuts), excepte prop de l'obertura del mercat.

## 10. Resolució de Problemes

### 10.1 Problemes Comuns

#### Problema 1. Websockets

Si el frontend no aconseguir connectar amb el websocket del backend, es rebrà un error per consola.

Probablement, si això no funciona és pel fet que no pot iniciar la connexió amb el backend, el que vol dir que no és problema de l'API Externa de Finnhub.

#### Solucions:

Revisar el .env del frontend, revisar que el host i el port siguin els correctes.

Veure la consola del backend amb:

kubectl logs soda-backend

Reiniciar el deploy.

#### Problema 2. Proxy invers.

El proxy invers quan es reinicia un deploy perd el servidor i pot arribar a donar un error 404.

#### **Solucions.**

Degut a no poder contactar amb un dels dos servidors (frontend o backend). En aquests casos l'administrador podria reiniciar el deploy.

Una altra opció seria configurar-ho perquè no passi això.

Mirar els logs.

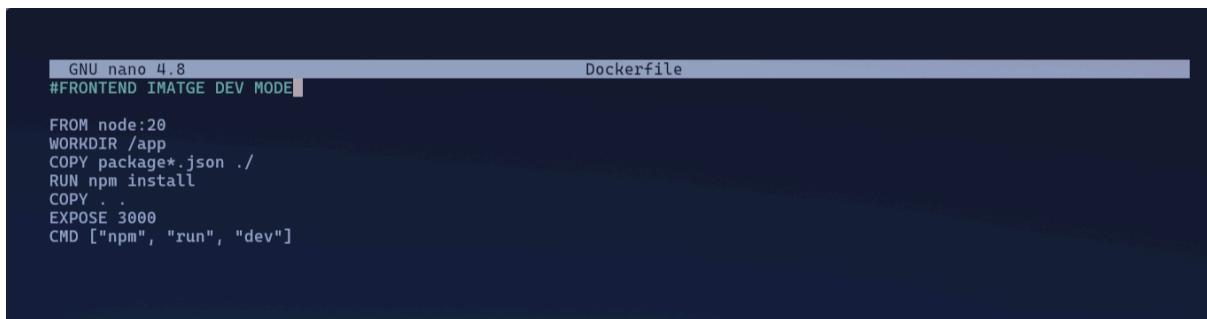
## 10.2. Procediments de diagnòstic

Per localitzar problemes és important distingir d'on ve el problema, si és nivell de codi de programació o en l'àmbit de l'entorn del desplegament.

### Procediment 1

A vegades podria passar que el codi en un entorn de desenvolupament no donés cap problema, però a l'hora de desplegar-ho sí.

Per això en aquests casos s'ha fet un Dockerfile preparat per tenir una imatge del frontend en mode desenvolupament.



```
GNU nano 4.8 Dockerfile
#FRONTEND IMATGE DEV MODE

FROM node:20
WORKDIR /app
COPY package*.json .
RUN npm install
COPY .
EXPOSE 3000
CMD ["npm", "run", "dev"]
```

Es pujaria la imatge i es reiniciaria el deploy. D'aquesta manera en el navegador es podrien visualitzar els diferents errors que dona Next i poder saber d'on ve el problema.

### Procediment 2.

En cas que sigui un problema de codi, és molt important mirar els logs del navegador, realment allà és on es veu si és problema del frontend, a l'hora de fer la petició o del backend que està donant una resposta incorrecta.

### Procediment 3

Agafar una eina com ThunderClient, o un Postman i provar l'API sense utilitzar navegador per comprovar si es rep la resposta desitjada

### Procediment 4.

Esborrar la memòria cau i les dades guardades. És probable que hi hagi algun problema relacionada amb la memòria cau, i més si s'estan realitzant canvis constantment. Esborrar la memòria cau pot solucionar molts problemes de desenvolupament.

### Procediment 5.

Visualitzar tots els logs que hi ha al programa i afegir-ne de més per trobar els problemes.

## 11. Enllaç al GitHub

<https://github.com/Daneteee/SODA.git>