EXOCOSMOS

MANUAL TÈCNIC

Joan Sabata Moreno
23/05/2025
Projecte de Síntesi

Exocosmos

Versió del document

1.0.0

Última data d’actualització

22/05/2025

Autors i responsables

- **Nom:** Joan Sabata Moreno
  Rol: Tècnic en desenvolupament d’aplicacions web

Historial de versions

Versió	Data	Autor	Descripció
1.0.0	22/05/2025	Joan Sabata	Creació del document tècnic

Propòsit del document

Aquest document tècnic descriu en detall la configuració, arquitectura, desplegament i manteniment del projecte **Exocosmos**, una aplicació web educativa orientada a la gestió de sistemes planetaris ficticis.

Està destinat a tècnics de sistemes, desenvolupadors i personal de manteniment amb coneixements avançats en desenvolupament web, desplegament i administració de sistemes.

L'objectiu és facilitar la comprensió del funcionament intern del sistema, així com proporcionar instruccions clares per a la seva instal·lació, configuració i operació contínua.

Índex

Índex **2**
Arquitectura del Sistema **6**
Diagrama d'arquitectura general 6
Patrons de disseny utilitzats 7
Backend 7
Frontend 7
Stack tecnològic 8
Components Principals 9
Frontend 9
Estructura general de la pàgina 9
Estils 9
Visualització 3D amb Three.js i React Three Fiber 10
Arquitectura general del sistema 3D 10
Components clau 10
Planet3D.tsx 10
SpaceScene.tsx 10
Previsualitzacions en temps real 11
Formularis amb vista prèvia reactiva 11
Formulari de planeta 11
Formulari d'estrella 12
Textures i edició visual 12
Rendiment i bloom 12
Generació de miniatures 3D (thumbnails) des del frontend 13
Procés tècnic de generació 13
Comportament específic per recurs 14
Beneficis del sistema 14
Backend 15
Gestió de pujada i emmagatzematge d'imatges 16
Endpoint de pujada 16
Estructura de carpetes 16
Requisits i validacions 17
Actualització automàtica 17
Tests automatitzats a Exocosmos 17
Estructura general 17
Setup i fixtures 17
Cobertura dels tests 18
Bonnes pràctiques aplicades 18
Base de dades 19
APIs externes 20
Entorn Tècnic **21**
Requisits de Hardware 21

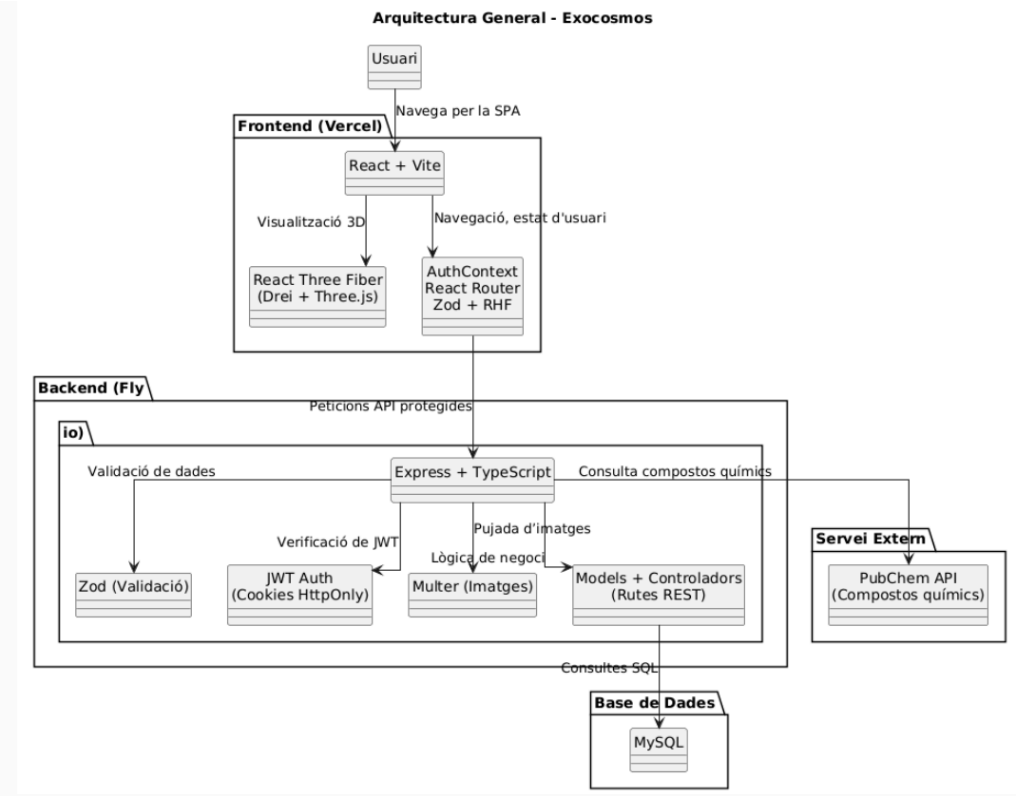
Backend 21
Client 21
Requisits de Software 21
Servidor web 21
Base de dades 21
Framework utilitzat 22
Llenguatges de programació 22
Dependències i llibreries principals 22
Frontend 22
Backend 22
Configuració del Sistema 23
Instal·lació 23
Clonació del repositori i instal·lació 23
Instal·la dependències als dos projectes: 23
Creació de la base de dades 23
Configuració de l'entorn 23
/frontend/.env: 23
/backend/.env: 24
Scripts 24
/frontend/package.json 24
/backend/package.json 24
Desplegament 25
Backend (Node) 25
Dockerfile 25
fly.toml 25
Base de Dades (MySQL 8) 26
Dockerfile (exocosmos-db) 26
fly.toml 26
Connexió entre Backend i DB 27
Frontend 27
Configuració bàsica 27
Requisits previs 27
Pasos realitzats per desplegar 28
Resultat 28
Base de dades **29**
Diagrames 29
Descripció de les taules 31
users 31
stars 31
planetary_systems 32
planet_types 32
planets 33
atmospheres 34
compounds 34

Relacions entre entitats 34
Índexs i claus 35
Claus primàries (PRIMARY KEY) 35
Claus foranes (FOREIGN KEY) 35
Índexs (INDEX, UNIQUE) 35
Script de creació 36
Estructura del Codi **37**
Organització del Projecte 37
frontend/ 37
backend/ 38
Convencions de nomenclatura 38
Patrons implementats 39
Components Principals 39
Mòduls 39
Classes principals 40
Interfícies 40
Serveis 40
APIs Internes **41**
Endpoints disponibles 41
/auth 41
/users 41
/stars 41
/planets 41
/planetary-systems 42
/compounds 42
/planet-types 42
/upload/:entity/:id/:type 42
Formats de petició/resposta 42
Exemple de petició: login 43
Exemple de resposta d'èxit 43
Exemple de creació d'un recurs (planeta) 43
Resposta esperada 44
Errors 44
APIs Externes 45
PubChem PUG-View REST API 45
Funcionament 45
Exemple de consulta 45
Configuració 45
Seguretat **46**
Mecanismes de Seguretat 46
Autenticació 46
Autorització 46
Encriptació 47
Gestió de sessions 47

Polítiques 48
Control d'accés 48
Gestió de contrasenyes 48
Protocols de seguretat 48
Logs i auditoria 48
Resolució de Problemes **49**
Problemes Comuns 49
Errors d'arrencada del servidor o de connexió a la base de dades 49
Respostes HTTP amb error 49
Errors de dades o consultes SQL 49
Errors en validació o transformació de formularis 49
Diagnòstic i testing 50
Annexos **51**
Glossari de termes 51
Documentació addicional 52

Arquitectura del Sistema

Diagrama d'arquitectura general



- **Usuari**: Accedeix a través del navegador web. Interactua amb l'aplicació React i visualitza sistemes planetaris en temps real.
- **Frontend** (desplegat a Vercel): Aplicació SPA construïda amb React + Vite. Inclou:
  - **React Three Fiber (R3F)** per a visualització 3D.
  - **AuthContext** i **React Router** per a l'estat global i navegació.
  - Validació de formularis amb **Zod** i **React Hook Form**.
- **Backend** (desplegat a Fly.io): API REST creada amb **Express + TypeScript**. Gestiona:
  - Lògica de negoci i validació de dades amb **Zod**.
  - Autenticació mitjançant **JWT** emmagatzemat en cookies **HttpOnly**.
  - Pujada d'imatges amb **Multer**.
- **Base de Dades**: MySQL conté la informació sobre usuaris, sistemes, planetes, estrelles i compostos associats.
- **Servei extern - PubChem API**: Utilitzat pel backend per verificar compostos químics reals i enriquir la informació científica dels planetes creats.

Patrons de disseny utilitzats

El projecte **Exocosmos** aplica diversos patrons de disseny que asseguren una estructura clara i mantenible tant al backend com al frontend.

Backend

- **MVC (Model-Vista-Controlador)**
  L'arquitectura del backend segueix el patró **MVC**, on:

  - El **model** (per exemple, PlanetModel.ts) s'encarrega d'executar consultes a la base de dades MySQL.

  - El **controlador** (per exemple, planetController.ts) rep les peticions HTTP (req), processa la lògica necessària i retorna la resposta (res) al client.

  - Les **rutes** defineixen els endpoints RESTful i deleguen les operacions als controladors.

- **Middlewares**
  Utilitzats per encapsular funcionalitats comunes com la **validació**, **autenticació JWT**, la **pujada d'imatges** o la **gestió d'errors**. Aquests s'apliquen de manera modular entre la petició i el controlador.

Frontend

- **Componentització**
  La UI està dividida en components reutilitzables amb responsabilitat única, com formularis, targetes i visualitzadors 3D.

- **Context API**
  S'utilitza un AuthContext global per gestionar l'estat d'usuari i autenticació, accessible des de qualsevol part de l'aplicació.

- **Hooks personalitzats**
  Hooks com useAuth() i useFetchWithAuth() encapsulen lògica d'estat i de peticions protegides, millorant la reutilització i mantenibilitat.

Stack tecnològic

El projecte **Exocosmos** s'ha desenvolupat amb un conjunt modern de tecnologies web tant per al client com per al servidor. A continuació es detalla el stack utilitzat, classificat per capes:

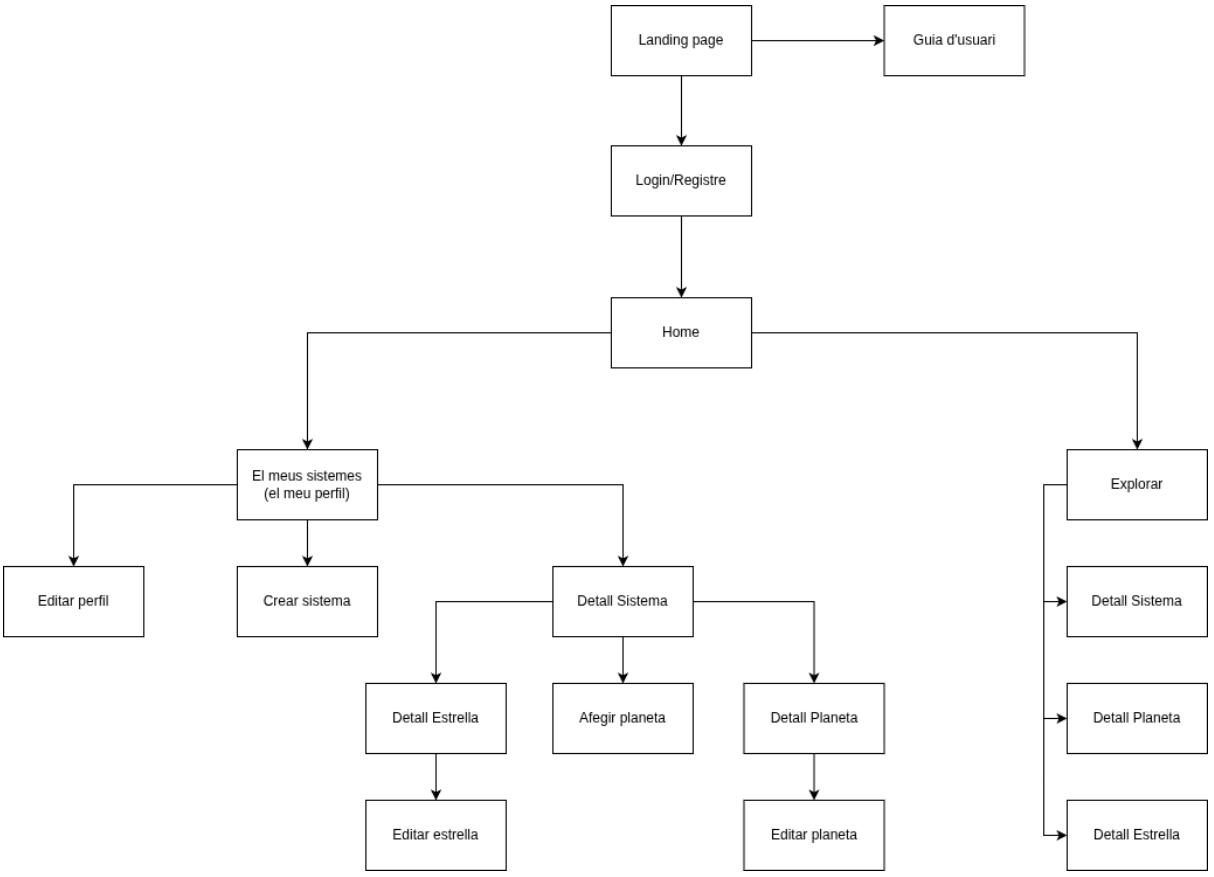
Capa	Tecnologies
Frontend	React, Vite, React Three Fiber (R3F), Drei, Tailwind CSS, v0 (UI), React Hook Form, Zod
Backend	Express, TypeScript, Zod, JWT (cookies HttpOnly), Multer, Vitest, Supertest
Base de dades	MySQL, MySQL Workbench
Desplegament	Vercel (frontend), Fly.io (backend i base de dades)
Serveis externs	PubChem API (validació de compostos químics reals)
Control de versions	Git, GitHub

Aquest stack permet oferir una aplicació moderna, segura i escalable, amb una experiència visual interactiva gràcies a la integració de tecnologies 3D dins del navegador.

Components Principals

Frontend

Estructura general de la pàgina



Estils

L'estil visual d'Exocosmos està construït amb Tailwind CSS com a sistema de disseny principal, aprofitant la seva filosofia de classes utilitàries per mantenir el codi modular, predictible i fàcil de mantenir. A més de la configuració bàsica de Tailwind, el projecte incorpora el plugin `tailwindcss-animate` per aplicar animacions predefinides i efectes visuals subtils.

Els icones són gestionats amb la llibreria Lucide React, una col·lecció moderna i lleugera d'icones SVG que s'integra fàcilment amb Tailwind. Es fan servir per reforçar la jerarquia visual, indicar accions i millorar la llegibilitat de la interfície sense afegir complexitat visual. Les classes de Tailwind permeten estilitzar-los directament amb colors, grandàries i transicions segons el context, mantenint la consistència amb la resta del disseny.

Visualització 3D amb Three.js i React Three Fiber

El frontend d'**Exocosmos** implementa una visualització tridimensional avançada per representar planetes, estrelles i sistemes planetaris. Aquest subsistema està construït sobre **React Three Fiber**, que actua com un binding de React per a la biblioteca **Three.js**, i permet visualitzar objectes astronòmics en temps real amb textures, moviment, rotació i comparació de mides basades en dades físiques reals.

Arquitectura general del sistema 3D

El punt central de tota visualització 3D és el component `SpaceScene`. Aquest encapsula el context `Canvas` de Three.js, configura l'escena, les llums, la càmera adaptativa, el fons galàctic, el post-processat de bloom i els controls orbitals. Tot el contingut 3D es renderitza com a fills d'aquest component, incloent-hi `Planet3D`, `Star3D` i `Ring`.

Components clau

Planet3D.tsx

Aquest component representa un planeta amb:

- textura de superfície (`surface_texture_url`)
- height map o mapa de relleu (`height_texture_url`)
- atmosfera opcional (`atmosphere.texture_url`)

Els radis i les dimensions visuals es calculen segons un `scaleFactor`, i el planeta pot girar segons un paràmetre de `timeScale`. La càrrega de textures es fa amb `TextureLoader`, acceptant rutes relatives i dataURLs. Si no hi ha height map, es genera un de gris per defecte. L'atmosfera es modela com una segona esfera semitransparent.

Star3D.tsx

Aquest component simula una estrella fent ús d'un shader en GLSL basat en soroll procedural (Simplex Noise). La textura no es carrega sinó que es genera en temps real segons la temperatura estimada. El color de l'estrella canvia dinàmicament amb un model de classificació espectral que assigna una tonalitat RGB coherent amb la temperatura de la massa.

SpaceScene.tsx

Aquest component s'encarrega de:

- configurar la càmera segons la mida de l'objecte principal
- afegir controls orbitals amb `OrbitControls` (activables/desactivables)
- renderitzar objectes de comparació (Sol, Terra, Júpiter) en posicions laterals
- permetre ajustar la qualitat gràfica mitjançant l'escala de píxel i l'efecte bloom

Previsualitzacions en temps real

Els components `Planet3DPreview` i `Star3DPreview` encapsulen la visualització dins d'un `SpaceScene` i s'utilitzen tant a les pàgines de detall com als formularis d'edició i creació. Aquests wrappers reben instàncies de `Planet` o `Star` i renderitzen només si els valors físics són vàlids.

A la pràctica, això permet que quan l'usuari edita una estrella o planeta, pugui veure la seva representació 3D immediatament, sense necessitat de guardar.

Formularis amb vista prèvia reactiva

Els components `LivePlanet3DPreview` i `LiveStar3DPreview` serveixen per sincronitzar el formulari amb la vista 3D. Fan servir `useFormContext()` i `useWatch()` de `react-hook-form` per detectar canvis en temps real i crear una nova instància del model `Planet` o `Star`. Aquesta instància es passa automàticament al component `Preview`, que actualitza la visualització.

Això permet que qualsevol canvi en massa, radi, textura o inclinació es reflecteixi immediatament en la escena 3D, millorant la comprensió del resultat per part de l'usuari i reduint errors.

Formulari de planeta

Quan es renderitza el formulari de creació o edició d'un planeta (`PlanetCreate` o `PlanetEdit`), es genera un esquema de validació `zod` segons el tipus de planeta. Aquest esquema conté valors límit com massa mínima, radi màxim, o si pot tenir anells. Els valors inicials s'adapten automàticament segons aquests límits.

Aquest formulari conté seccions complexes com l'atmosfera (amb composició química) o la selecció de compostos de la superfície, que són objectes nested sincronitzats amb components visuals com sliders de distribució percentual.

Els camps visuals també inclouen canvassos editables on es poden pintar les textures. Quan l'usuari aplica la textura, aquesta es converteix en un `dataURL` i s'aplica al model 3D en temps real.

Els sliders, switches i checkboxs no són components natius sinó components controlats integrats amb `react-hook-form`. Tota la UI s'actualitza a mesura que es modifiquen camps, i es reflecteix immediatament a la vista 3D gràcies a `useWatch`.

Quan s'envia el formulari, s'aplica un transformador (`handleEnrichedSubmit`) que converteix les textures i estructures en el format esperat pel backend. Es fan crides extra per pujar miniatures i regenerar el thumbnail del sistema planetari si cal.

Formulari d'estrella

El formulari d'estrella és més directe, ja que només permet modificar la massa, el radi i una descripció opcional. Els valors derivats com la temperatura, la luminositat i el tipus espectral es calculen automàticament i es mostren com a "propietats calculades".

Aquestes propietats s'utilitzen per generar la visualització en 3D amb `Star3D`, però l'usuari no les pot modificar manualment. Això assegura coherència física i una representació visual precisa.

Igual que en el cas dels planetes, es pot editar una estrella existent o crear-ne una nova, amb sincronització automàtica de la previsualització. En enviar el formulari es pot regenerar el thumbnail associat a l'estrella i al sistema que l'inclou.

Textures i edició visual

Els planetes permeten personalitzar tres capes de textura:

- **Superfície**
- **Mapa de relleu**
- **Atmosfera**

Aquestes textures es poden carregar com a fitxers o bé pintar en directe sobre un `<canvas />`. El contingut del canvas s'aplica mitjançant un `dataURL` al model 3D. Els valors es mantenen sincronitzats amb el formulari gràcies als `hidden inputs` i `setValue()`.

Aquesta funcionalitat ofereix una experiència visual interactiva que evita haver d'utilitzar eines externes per generar textures.

Rendiment i bloom

L'efecte bloom es pot activar o desactivar amb un toggle dins la UI. El component `BloomComposer` encapsula la lògica de post-processament mitjançant `EffectComposer`, aplicant efectes només si cal. L'escala de píxel (`pixelScale`) es pot ajustar per millorar el rendiment en dispositius amb menys potència.

Els controls orbitals també es poden desactivar per optimitzar la interacció, especialment en dispositius mòbils. El sistema detecta automàticament si l'usuari està en un dispositiu petit i ajusta la disposició dels menús.

Generació de miniatures 3D (thumbnails) des del frontend

L'aplicació Exocosmos genera automàticament miniatures (thumbnails) per a planetes, estrelles i sistemes planetaris utilitzant renderització 3D. Aquestes imatges es fan servir per representar visualment els recursos en targetes, llistes o galeries, i es creen directament al navegador, sense necessitat de serveis externs ni edició manual.

La generació es basa en una renderització offscreen utilitzant una escena Three.js encapsulada dins de React Three Fiber, renderitzada a un `<canvas>` invisible i posteriorment convertida a format d'imatge.

Procés tècnic de generació

Quan es crea o edita un recurs, el frontend genera la miniatura mitjançant el mòdul `generateWebPFrom3D`, que encapsula tot el procés:

1. **Creació d'una escena 3D dedicada:**
   Es crea una escena virtual amb càmera, llums i els components 3D del recurs (`Planet3DThumbnail`, `Star3D`, etc.). La càmera es col·loca en una posició fixa per assegurar coherència visual entre recursos.

2. **Renderització oculta (offscreen):**
   La renderització es fa dins un `<div>` invisible, amb un `<Canvas>` amb `preserveDrawingBuffer` activat per capturar la imatge renderitzada. Aquest canvas no s'afegeix a la UI principal, sinó directament al DOM de manera temporal.

3. **Sincronització asíncrona:**
   Abans de capturar la imatge, el sistema espera que totes les textures i objectes estiguin carregats. Això es controla mitjançant una `Promise` resolta pels callbacks `onReady` dels components (`Planet3DThumbnail`, `Star3D`, etc.).

4. **Captura i codificació:**
   Quan tot està a punt, es renderitza l'escena i s'utilitza `canvas.toDataURL("image/webp")` per obtenir una imatge en format WebP amb fons transparent. Aquest format és compacte i adequat per a miniatures web.

5. **Conversió i pujada al backend:**
   La cadena base64 es converteix a `Blob` amb `dataURLToBlob()` i s'envia com a `FormData` mitjançant `fetch()` a l'endpoint corresponent de pujada (`/upload/planets/:id/thumbnail`, etc.). Aquesta petició inclou la cookie de sessió per validar l'autenticació.

Comportament específic per recurs

- **Planetes:**
  S'utilitza el component `Planet3DThumbnail`, que renderitza el planeta amb la seva textura de superfície, atmosfera i anells si en té. També es respecta la inclinació orbital i un lleuger aplanament equatorial (flattening). Es limita l'escala visual i la il·luminació per mantenir consistència.

- **Estrelles:**
  S'utilitza `Star3D` amb un shader que simula brillantor i color espectral segons la temperatura. Es força un radi constant per assegurar la mida visual i evitar que estrelles molt petites generin imatges inapreciables.

- **Sistemes planetaris:**
  Es renderitza una escena que conté fins a tres planetes i la seva estrella. Cada planeta es posiciona en 3D amb lleugeres variacions d'eix, mentre que l'estrella es col·loca a l'esquerra. El resultat és una miniatura representativa del sistema complet.

Beneficis del sistema

- **Totalment automatitzat:** l'usuari no ha de pujar imatges manuals.

- **Visualment coherent:** totes les miniatures tenen mida, angle i fons consistents.

- **Actualitzable:** si es modifica un recurs, la miniatura es regenera i es puja novament.

- **Independent del backend:** tota la generació es fa al navegador, reduint càrrega del servidor i facilitant proves locals.

Aquest sistema de generació de miniatures reforça l'experiència visual d'Exocosmos, mantenint coherència gràfica i professionalitat sense requerir intervenció externa ni eines de disseny.

Backend

El backend d'Exocosmos està construït amb Express i TypeScript, estructurat segons el patró Model-Vista-Controlador (MVC). Cada recurs principal —com planetes, estrelles, sistemes o compostos— disposa d'un mòdul propi que agrupa el seu **model**, **controlador**, **esquema de validació**, **rutes** i **middlewares específics**. Aquesta organització permet una separació clara de responsabilitats, facilita la lectura i simplifica el manteniment i l'evolució del sistema.

Els **models** contenen l'accés directe a la base de dades MySQL, gestionant tant les consultes bàsiques com la sincronització de relacions (per exemple, planetes amb atmosferes o compostos). Els **controladors** processen les peticions HTTP entrants, criden els models segons la lògica de negoci i retornen la resposta amb el format establert. Les rutes defineixen els endpoints RESTful disponibles i apliquen els middlewares de validació i autenticació corresponents abans d'arribar al controlador.

Els **middlewares** gestionen tasques transversals com la validació amb Zod, l'autenticació mitjançant JWT (via cookies HttpOnly), el control de propietat (checkOwnership), i la pujada d'imatges amb Multer. Aquest sistema modular assegura que cada acció del backend passi per una cadena validada i segura abans d'afectar dades sensibles.

A nivell de tipus i **validació**, s'utilitzen interfícies **TypeScript** per assegurar la coherència entre capes i schemas **Zod** per validar totes les entrades de l'API, tant al crear com al modificar recursos. Cada recurs pot tenir restriccions específiques: per exemple, un planeta ha de respectar els límits físics definits pel seu tipus (massa, radi, presència d'anells...) i aquestes condicions es verifiquen abans de permetre l'operació.

Els serveis interns (utils) encapsulen funcionalitats reutilitzables com la sincronització de compostos, la validació de cadenes, la conversió d'unitats o la comunicació amb l'API externa PubChem. Això permet mantenir el codi **DRY** i evita duplicació entre rutes com `createPlanet` o `updatePlanet`.

El sistema d'autenticació està basat en **JWT** emmagatzemats en **cookies HttpOnly**. L'autenticació es valida a cada petició protegida mitjançant middleware, i l'autorització es resol comprovant que l'usuari actual és el propietari del recurs (sense rols ni privilegis globals). A més, el backend gestiona imatges de manera segura mitjançant endpoints d'upload protegits, que associen miniatures, avatars o textures a cada entitat concreta.

Finalment, tot el backend està cobert per proves automatitzades amb **Vitest** i **Supertest**, garantint estabilitat en operacions CRUD, validació de dades i integritat de relacions. Això assegura que qualsevol canvi introduït al codi passi per un conjunt mínim de tests abans de ser desplegat a Fly.io.

Gestió de pujada i emmagatzematge d'imatges

El backend d'Exocosmos implementa un sistema centralitzat per gestionar la pujada d'imatges com **miniatures**, **textures** o **imatges de perfil**, que es desen dins del directori `public/uploads/`. Aquest directori s'exposa públicament mitjançant Express:

```
app.use('/uploads', express.static('public/uploads'));
```

Això permet accedir a les imatges des del frontend mitjançant una URL com ara:

```
https://exocosmos-backend.fly.dev/uploads/planets/thumbnail/42.webp
```

Endpoint de pujada

Totes les pujades passen per la ruta:

```
POST /upload/:entity/:id/:type
```

On:

- `:entity` pot ser `planets`, `stars`, `planetary_systems` o `users`
- `:id` és l'identificador numèric del recurs
- `:type` indica el tipus d'arxiu: `thumbnail`, `surface`, `height`, `atmosphere`, `profile_picture`

Estructura de carpetes

Les imatges es desen automàticament dins:

```
/public/uploads/
├── planets/
│   ├── thumbnail/
│   ├── surface/
│   ├── height/
│   └── atmosphere/
├── stars/thumbnail/
├── planetary_systems/thumbnail/
└── users/profile_picture/
```

El nom del fitxer correspon a l'ID del recurs amb la seva extensió original (ex. `42.webp`, `1.png`), la qual cosa permet substituir-lo fàcilment en edicions posteriors.

Requisits i validacions

- Només s'accepten imatges amb format **JPEG**, **PNG** o **WebP**
- Mida màxima per fitxer: **5 MB**
- El directori es crea automàticament si no existeix
- Si es puja un tipus no permès per l'entitat (per exemple, `surface` per a `stars`), es rebutja la petició amb un error controlat

Actualització automàtica

Després de pujar el fitxer, el backend actualitza automàticament la propietat corresponent del recurs (com `thumbnail_url`, `surface_texture_url` o `atmosphere.texture_url`) amb el camí públic generat. Això permet al frontend accedir-hi directament i renderitzar la nova imatge sense intervenció manual.

Tests automatitzats a Exocosmos

El projecte disposa d'un conjunt extens de tests automatitzats escrits amb **Vitest** i **Supertest**, enfocats a verificar el comportament de l'API REST del backend. Aquests tests abasten operacions CRUD, autenticació, validació de dades, gestió de relacions entre recursos, control d'accés i tractament d'errors.

Estructura general

Els arxius de tests es troben dins del directori `tests/`, organitzats per domini funcional (`auth`, `planet`, `compound`, etc.). A més, s'inclouen utilitats compartides, recursos de prova i configuració de base de dades per a entorns de test aïllats.

La base de dades `exocosmos_test` s'inicialitza automàticament abans d'executar els tests. Cada test es fa dins d'una transacció SQL que es cancel·la un cop finalitza, assegurant entorns consistents i idempotents.

Setup i fixtures

Abans de començar cada test, s'executa un `global-setup.ts` que crea la base de dades i insereix les dades mínimes necessàries (tipus planetaris, per exemple). El `setup.ts` gestiona les transaccions SQL per garantir aïllament entre casos.

A més, es defineixen helpers reutilitzables per:

- Registrar i iniciar sessió amb usuaris de prova.
- Crear sistemes planetaris, estrelles, planetes o compostos amb dades vàlides.
- Carregar imatges de prova (fixtures) per validar les rutes d'upload.

Cobertura dels tests

Els tests cobreixen les rutes principals del backend, incloent:

- **Autenticació**: registre, login, logout, /auth/me, modificació i eliminació del compte. Es prova tant amb credencials vàlides com amb casos d'error (duplicats, camps buits, formats incorrectes…).
- **Planetes**: totes les operacions CRUD, validació segons el tipus planetari, gestió de compostos i atmosferes, control de propietat i relacions entre entitats.
- **Sistemes planetaris i estrelles**: creació vinculada, actualització (PUT/PATCH), eliminació amb control de rollback si una entitat falla, i vistes completes (`/full`).
- **Compostos**: inserció automàtica via PubChem, consulta per nom o fórmula, i limitació de mètodes (PUT, DELETE, etc.).
- **Tipus de planeta**: només lectura, amb limitacions explícites a l'escriptura.
- **Usuaris**: lectura pública limitada (sense correu electrònic), accés complet via `/full`, i control de camps accessibles.
- **Upload d'imatges**: comprovació que les imatges s'associen correctament als recursos (usuari, sistema, estrella, planeta), així com validació d'errors i accés públic a `/uploads`.

Bones pràctiques aplicades

- Cada test realitza assertions clares sobre el `status` de la resposta i el contingut rebut.
- En cas de fallada inesperada, es registra el cos de la resposta (`logIfFailed`) per facilitar el diagnòstic.
- Els recursos de prova s'autogeneren amb valors únics (`UUID`) per evitar col·lisions entre execucions.
- Es prova no només el comportament correcte, sinó també condicions límit, errors esperats, i intents d'accés indegut.

Base de dades

La base de dades d'Exocosmos està dissenyada amb **MySQL 8** i segueix una estructura relacional clara, optimitzada per mantenir la integritat referencial entre entitats i permetre consultes complexes amb un bon rendiment. El model de dades reflecteix fidelment l'organització lògica del domini: usuaris, sistemes planetaris, planetes, estrelles, tipus de planeta, atmosferes i compostos.

Cada entitat principal (com `users`, `stars`, `planets` o `planetary_systems`) disposa d'una clau primària numèrica autoincremental. Les relacions s'implementen mitjançant claus foranes amb restriccions `ON DELETE CASCADE`, assegurant que l'eliminació d'un recurs arrossega automàticament totes les entitats dependents. Per exemple, si un sistema planetari s'elimina, els planetes i atmosferes associats també desapareixen de forma controlada.

El disseny incorpora tant relacions 1:N (un usuari pot tenir molts sistemes) com N:M (un planeta pot contenir diversos compostos i viceversa). Aquestes relacions múltiples es gestionen mitjançant taules intermitjes com `planets_compounds` o `atmospheres_compounds`, que defineixen també propietats addicionals com la proporció de cada compost.

Cada taula inclou índexs per millorar el rendiment de cerca, especialment en camps habitualment filtrats o associats a JOINs, com `planet_type_id`, `user_id` o `star_id`. A més, s'han definit restriccions `UNIQUE` en camps com `username`, `email`, `planet.name` o `star.name` per evitar duplicats i garantir la coherència del sistema.

Algunes entitats com `planet_types` o `compounds` són inicialitzades amb dades predefinides mitjançant un script SQL (`/backend/sql/create-schema.sql`), que pot ser executat una sola vegada durant el desplegament inicial. Aquest script també defineix totes les claus, índexs i relacions de manera explícita.

Els models del backend accedeixen a la base de dades mitjançant la llibreria `mysql2`, i encapsulen les consultes dins de mètodes TypeScript (ex: `PlanetModel.getFullById`) que respecten el tipatge estricte i eviten SQL injection. A nivell de testing, existeix una base de dades de test separada (`exocosmos_test`) que permet executar proves automatitzades sense afectar dades reals.

Aquesta estructura de dades proporciona una base sòlida per escalar el projecte, afegir noves funcionalitats (com més capes de compostos, missions, o dades astrofísiques addicionals) i garantir una experiència consistent a nivell tant de backend com de visualització 3D en el frontend.

APIs externes

El backend d'**Exocosmos** integra un únic servei extern: **PubChem PUG-View REST API**, una API pública proporcionada pel National Center for Biotechnology Information (NCBI) que ofereix accés a informació química precisa sobre compostos identificats pel seu CID (Compound ID).

Aquest servei s'utilitza de manera automatitzada per enriquir les dades dels planetes i atmosferes amb informació científica real. Quan un usuari introdueix un CID de compost desconegut, el backend llança una petició HTTP GET cap a `https://pubchem.ncbi.nlm.nih.gov/rest/pug_view/data/compound/:CID/JSON`, extreu el nom i la fórmula molecular del compost, i els desa localment a la base de dades. Això evita duplicats, manté la coherència i estalvia feina manual a l'usuari.

La consulta es realitza mitjançant una funció auxiliar (`fetchCompoundFromPubChem`) i només s'executa si el compost no existeix prèviament a la base de dades. En cas d'error (resposta incompleta, CID inexistent, etc.), el servidor retorna un error 404 controlat, que el frontend pot gestionar adequadament.

Aquesta integració assegura que tots els compostos introduïts al sistema estiguin basats en informació científica verificable, i permet futures funcionalitats com validacions físiques més precises o visualització de fórmules.

Entorn Tècnic

Requisits de Hardware

Backend

El servidor que allotja l'API REST està desplegat a Fly.io amb la següent configuració de recursos, tant pel servidor MySQL com pel Node:

- **CPU**: 1 vCPU (shared)
- **Memòria RAM**: 1 GB

Tant el volum de la base de dades com el de la carpeta */uploads* tenen **1 GB** d'emmagatzemament.

Client

- **Navegador compatible amb WebGL**: Chrome, Firefox, Edge o Safari actualitzats
- **GPU mínima**: compatible amb WebGL 2.0 per a visualització 3D fluïda.
- **RAM**: mínim 4 GB, recomanat 8 GB per una experiència òptima
- **CPU**: processador de gamma mitjana (Intel i3 o equivalent ARM)
- **Connexió a Internet**: estable, especialment per a càrrega de textures i navegació d'escenes

Requisits de Software

Servidor web

- **Backend**: el servidor HTTP és **Express.js**, que fa de servidor web i API REST directament.
- **Frontend**: servit com fitxers estàtics HTML/CSS/JS mitjançant el CDN de **Vercel**.

Base de dades

- **Sistema**: MySQL v8
- **Gestió**: accés via MySQL Workbench durant el desenvolupament
- **Connexió**: via drivers MySQL en Node.js, usats dins els models

Framework utilitzat

- **Frontend**:
    - **React** per a la UI
    - **Vite** com a bundler i servidor de desenvolupament
- **Backend**:
    - **Express.js** com a framework principal per a l'API
    - **Zod** per a validació de dades

Llenguatges de programació

- **Frontend**: JavaScript, JSX, Typescript pels models.
- **Backend**: TypeScript (transpilat a JavaScript)
- **Estils**: CSS amb Tailwind.

Dependències i llibreries principals

Frontend

- react, react-dom, react-router-dom
- @react-three/fiber, @react-three/drei, three (visualització 3D)
- react-hook-form, zod (validació de formularis)
- tailwindcss, clsx, lucide-react, v0 (estils i icones)

Backend

- express, mysql2, jsonwebtoken, zod
- multer (gestió d'imatges)
- vitest, supertest (testing)
- dotenv (variables d'entorn)
  cors, helmet, cookie-parser (seguretat i middleware)

Configuració del Sistema

Instal·lació

Això explica com posar en marxa el projecte per desenvolupar-lo en local.

Requisits previs:

- Node.js 18+
- MySQL 8+
- Git

Clonació del repositori i instal·lació

```
git clone https://github.com/Joan289/Exocosmos.git
cd exocosmos
```

Instal·la dependències als dos projectes:

```
cd frontend
npm install
cd ../backend
npm install
```

Creació de la base de dades

Per inicialitzar l'estructura de la base de dades, importa el següent script SQL a la ruta: /backend/sql/create-schema.sql

I executar-ho a la base de dades.

```
mysql -u root -p exocosmos < backend/sql/create-schema.sql
```

Configuració de l'entorn

/frontend/.env:

```
VITE_API_URL=https://exocosmos-backend.fly.dev
```

Aquesta variable defineix l'URL base per a totes les peticions del frontend cap al backend.

/backend/.env:

```
DATABASE_URL=mysql://root:1234@localhost:3306/exocosmos
TEST_DATABASE_URL=mysql://root:1234@localhost:3306/exocosmos_test
PORT=3000
JWT_SECRET=JIUzI1...
```

- `DATABASE_URL`: cadena de connexió a la base de dades principal en entorn de desenvolupament.
- `TEST_DATABASE_URL`: base de dades separada per executar tests de manera aïllada.
- `PORT`: port en què s'executa el servidor backend (per defecte 3000).
- `JWT_SECRET`: clau privada utilitzada per generar i verificar els JSON Web Tokens (mai compartir en producció sense xifrar).

Scripts

A continuació es descriuen els scripts disponibles a cada projecte dins del repositori.

/frontend/package.json

Script	Descripció
npm run dev	Inicia el servidor de desenvolupament amb Vite.
npm run build	Compila el frontend per producció al directori dist/.
npm run preview	Serveix la build generada localment per a proves.
npm run lint	Executa ESLint per comprovar l'estil i qualitat del codi.

/backend/package.json

Script	Descripció
npm run dev	Inicia el servidor amb TypeScript en mode desenvolupament.
npm run build	Compila el codi TypeScript dins src/ i genera sortida a dist/.
npm start	Executa el servidor compilat (dist/server.js) en mode producció.
npm test	Executa els tests automatitzats amb Vitest.
npm run test:watch	Executa els tests en mode observador per a desenvolupament.

Desplegament

El backend d'Exocosmos està desplegat a **Fly.io**, utilitzant una imatge Docker personalitzada per al servidor Node.js i una instància separada de MySQL 8 per a la base de dades. A continuació es descriu el procés complet de desplegament.

Backend (Node)

El backend està estructurat per ser desplegat en dues etapes mitjançant Docker multistage:

1. **Etapa de build**
   Compila el codi TypeScript a JavaScript dins de `/dist` i copia els arxius públics (incloent textures i imatges).

2. **Etapa de runtime**
   Utilitza una imatge més lleugera de Node (`node:20-slim`) per executar el servidor compilat (`dist/server.js`). Només instal·la les dependències de producció (`npm install --omit=dev`).

Dockerfile

https://github.com/Joan289/Exocosmos/blob/main/backend/Dockerfile

Aquest fitxer es troba a la carpeta arrel de `backend/` i defineix el procés anterior. Exposa el port 3000 i configura el directori de treball a `/app`.

fly.toml

https://github.com/Joan289/Exocosmos/blob/main/backend/fly.tom

La configuració de Fly.io (`fly.toml`) defineix:

- El nom de l'aplicació: `exocosmos-backend`
- El port intern: `3000`
- Un volum muntat per a `/public/uploads`, anomenat `uploads_volume`, que manté persistents les imatges pujades
- La regió principal de desplegament (`cdg`)
- Els serveis HTTP i HTTPS exposats als ports 80 i 443 respectivament
- 1 vCPU compartida i 1 GB de RAM

Comandes principals per desplegar

```
flyctl auth login
cd backend
flyctl deploy
```

Això compilarà la imatge, la pujarà a Fly.io i iniciarà el contenidor amb la configuració especificada. És important que les variables d'entorn com `DATABASE_URL`, `PORT` i `JWT_SECRET` tenir definides via `flyctl secrets` o a l'arxiu `.env` local durant el desenvolupament.

Base de Dades (MySQL 8)

La base de dades també es desplega a Fly.io com a contenidor separat amb la imatge oficial de MySQL. Està configurada per inicialitzar automàticament l'esquema `exocosmos` amb el script SQL inclòs.

Dockerfile (exocosmos-db)

Defineix:

- La base d'imatge `mysql:8.0`
- L'ús de la variable `MYSQL_DATABASE=exocosmos`
- El script `script.sql` com a entrada inicial per crear les taules i dades base

fly.toml

Defineix:

- El nom de l'aplicació: `exocosmos-db`
- Un volum persistent `mysqldata` muntat a `/var/lib/mysql`
- La regió `cdg` igual que el backend
- La memòria i CPU idèntiques a l'API

Comandes principals per desplegar la DB

```
cd exocosmos-db
flyctl deploy
```

Una vegada desplegat, Fly inicialitzarà la base de dades amb les taules definides al `script.sql`. Si el contenidor es reinicia, el volum mantindrà persistents les dades.

Connexió entre Backend i DB

La variable `DATABASE_URL` del backend ha d'apuntar a l'host intern de Fly de la base de dades, que té la forma:

`DATABASE_URL=mysql://root:<password>@exocosmos-db.internal:3306/exocosmos`

Recorda definir aquest URL com a secret amb:

`flyctl secrets set DATABASE_URL=...`

Frontend

El frontend d'**Exocosmos** està desplegat a **Vercel**, una plataforma moderna per a aplicacions frontend estàtiques o híbrides. El projecte està construït amb **Vite**, cosa que permet una generació ràpida dels fitxers de producció i una integració molt fluïda amb el sistema de desplegament automàtic de Vercel.

Configuració bàsica

Per tal de funcionar correctament com a SPA, el projecte conté un fitxer `vercel.json` a l'arrel de `frontend/` que defineix una **regla de rewrite**. Aquesta regla assegura que totes les rutes del navegador (com `/app/profile`, `/auth`, etc.) siguin servides per l'arxiu HTML principal, permetent que **React Router** gestioni la navegació client-side:

```
{
  "rewrites": [
    { "source": "/(.*)", "destination": "/" }
  ]
}
```

Això és essencial per evitar errors 404 en rutes internes.

Requisits previs

Per desplegar-lo manualment, només cal tenir instal·lat:

- Node.js
- Vercel CLI (`npm install -g vercel`)
- Un compte a vercel.com, amb sessió iniciada des de la terminal (`vercel login`)

Pasos realitzats per desplegar

1. Entrar a la carpeta del frontend i assegurar que l'API URL està configurada correctament a l'arxiu `.env`:

   `VITE_API_URL=https://exocosmos-backend.fly.dev`

2. **Desplegar a producció** amb la CLI de Vercel:

   ```
   vercel --prod
   ```

   Durant el primer desplegament, la CLI et preguntarà:

   - El nom del projecte
   - El directori de build (`dist`)
   - Si és una SPA (cal respondre que **sí**)
   - Si vols guardar la configuració per futurs desplegaments

3. A partir d'aquí, Vercel puja automàticament els fitxers de `dist/` i genera un URL públic sobre HTTPS. El domini es pot personalitzar posteriorment des del panell de Vercel si cal.

Resultat

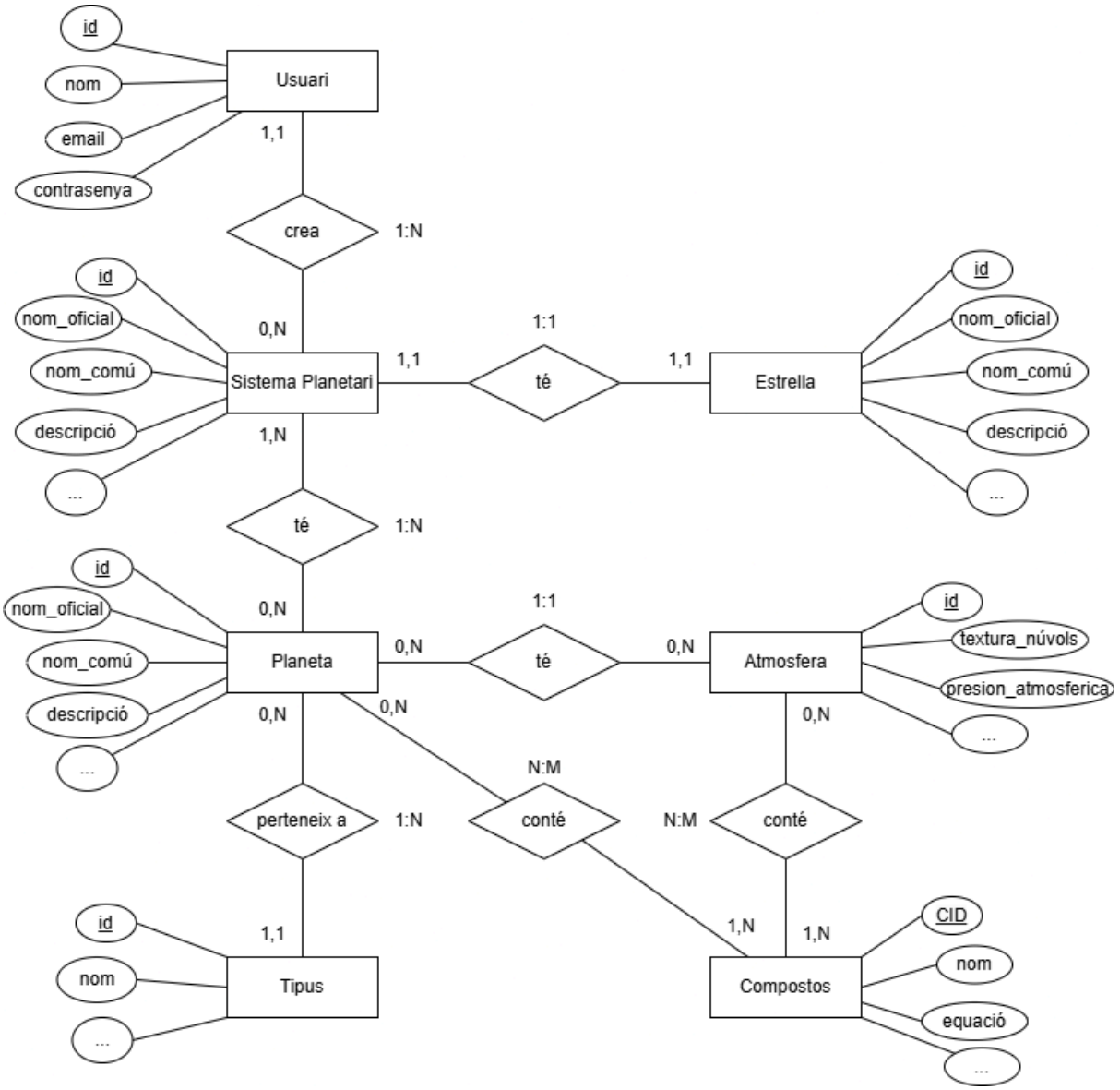
Amb aquest procés, l'aplicació frontend queda:

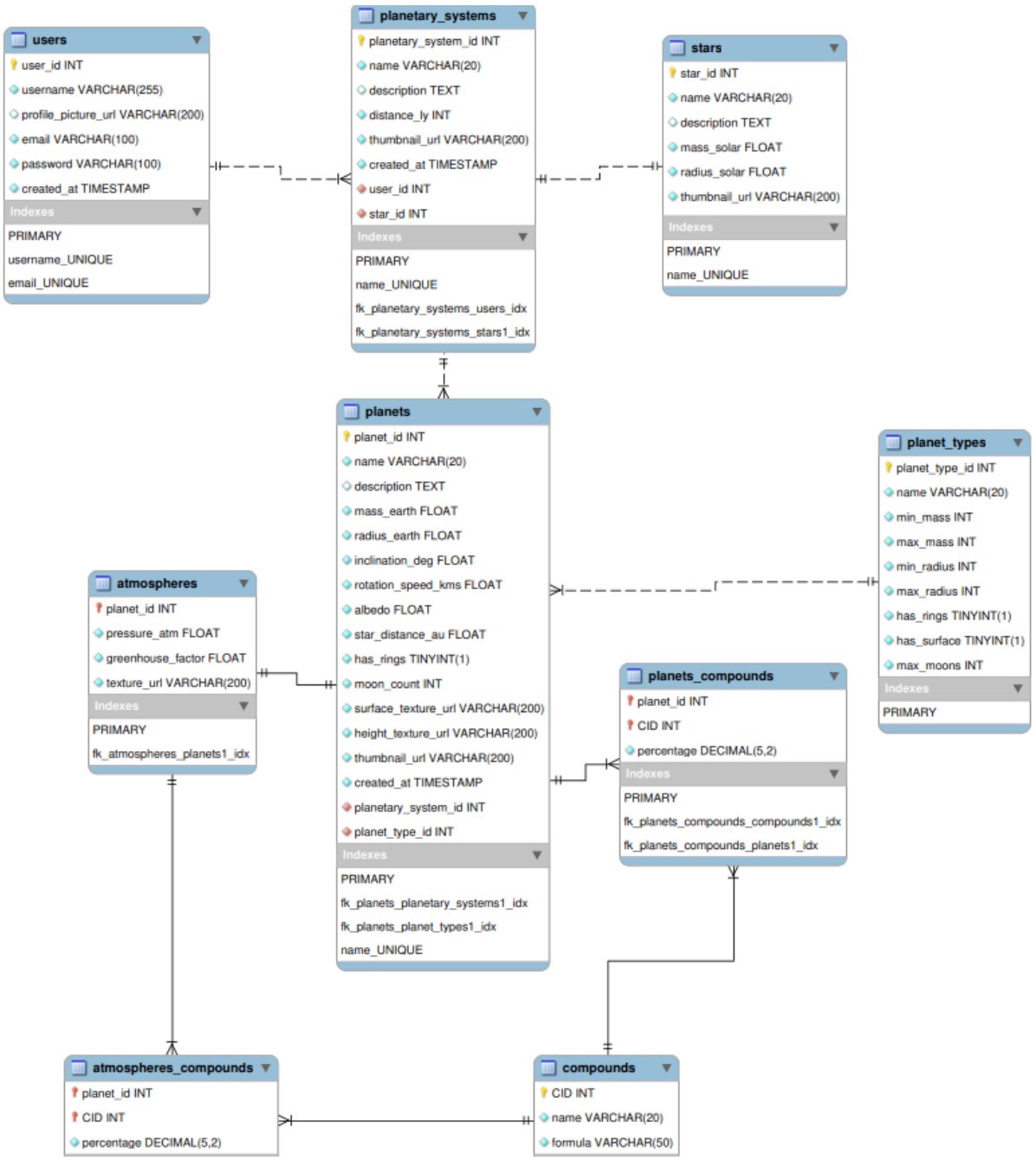
- Compilada amb `vite build`
- Servida com a SPA sota HTTPS
- Allotjada globalment a través de la CDN de Vercel

Aquest mètode és simple, ràpid i no requereix repositori Git connectat, ideal per projectes petits o desplegaments puntuals.

Base de dades

Diagrames




users
user_id INT
username VARCHAR(255)
profile_picture_url VARCHAR(200)
email VARCHAR(100)
password VARCHAR(100)
created_at TIMESTAMP
Indexes
PRIMARY
username_UNIQUE
email_UNIQUE
planetary_systems
planetary_system_id INT
name VARCHAR(20)
description TEXT
distance_ly INT
thumbnail_url VARCHAR(200)
created_at TIMESTAMP
user_id INT
star_id INT
Indexes
PRIMARY
name_UNIQUE
fk_planetary_systems_users_idx
fk_planetary_systems_stars1_idx
stars
star_id INT
name VARCHAR(20)
description TEXT
mass_solar FLOAT
radius_solar FLOAT
thumbnail_url VARCHAR(200)
Indexes
PRIMARY
name_UNIQUE
planets
planet_id INT
name VARCHAR(20)
description TEXT
mass_earth FLOAT
radius_earth FLOAT
inclination_deg FLOAT
rotation_speed_kms FLOAT
albedo FLOAT
star_distance_au FLOAT
has_rings TINYINT(1)
moon_count INT
surface_texture_url VARCHAR(200)
height_texture_url VARCHAR(200)
thumbnail_url VARCHAR(200)
created_at TIMESTAMP
planetary_system_id INT
planet_type_id INT
Indexes
PRIMARY
fk_planets_planetary_systems1_idx
fk_planets_planet_types1_idx
name_UNIQUE
planet_types
planet_type_id INT
name VARCHAR(20)
min_mass INT
max_mass INT
min_radius INT
max_radius INT
has_rings TINYINT(1)
has_surface TINYINT(1)
max_moons INT
Indexes
PRIMARY
atmospheres
planet_id INT
pressure_atm FLOAT
greenhouse_factor FLOAT
texture_url VARCHAR(200)
Indexes
PRIMARY
fk_atmospheres_planets1_idx
planets_compounds
planet_id INT
CID INT
percentage DECIMAL(5,2)
Indexes
PRIMARY
fk_planets_compounds_compounds1_idx
fk_planets_compounds_planets1_idx
atmospheres_compounds
planet_id INT
CID INT
percentage DECIMAL(5,2)
compounds
CID INT
name VARCHAR(20)
formula VARCHAR(50)

Descripció de les taules

users

Conté la informació dels usuaris registrats.

Columna	Tipus	Descripció
`user_id`	INT (PK)	Identificador únic de l'usuari
`username`	VARCHAR(50)	Nom d'usuari (únic)
`email`	VARCHAR(100)	Correu electrònic (únic)
`password`	VARCHAR(100)	Hash de la contrasenya
`profile_picture_url`	VARCHAR(200)	URL opcional de la imatge de perfil
`created_at`	TIMESTAMP	Data de creació automàtica

stars

Estrelles disponibles per associar a sistemes planetaris.

Columna	Tipus	Descripció
`star_id`	INT (PK)	Identificador únic de l'estrella
`name`	VARCHAR(20)	Nom de l'estrella (únic)
`description`	TEXT	Descripció opcional
`mass_solar`	NUMERIC	Massa en unitats solars
`radius_solar`	NUMERIC	Radi en unitats solars
`thumbnail_url`	VARCHAR(200)	URL de la imatge

planetary_systems

Sistemes creats pels usuaris, que contenen estrelles i planetes.

Columna	Tipus	Descripció
`planetary_system_id`	INT (PK)	Identificador del sistema
`name`	VARCHAR(20)	Nom del sistema (únic)
`description`	TEXT	Descripció opcional
`distance_ly`	INT	Distància en anys llum
`thumbnail_url`	VARCHAR(200)	Imatge representativa
`user_id`	INT (FK)	Propietari del sistema
`star_id`	INT (FK)	Estrella principal del sistema (única)

planet_types

Tipus de planetes predefinits, amb límits físics i característiques comunes.

Columna	Tipus	Descripció
`planet_type_id`	INT (PK)	Identificador
`name`	VARCHAR(20)	Nom del tipus
`min_mass`, `max_mass`	INT	Rangs de massa admissibles
`min_radius`, `max_radius`	INT	Rangs de radi admissibles
`has_rings`	TINYINT	Indicador de presència d'anells
`has_surface`	TINYINT	Indicador de superfície sòlida
`max_moons`	INT	Nombre màxim de llunes

planets

Planetes definits dins de sistemes, amb propietats físiques i visuals.

Columna	Tipus	Descripció
planet_id	INT (PK)	Identificador
name	VARCHAR(20)	Nom del planeta (únic)
description	TEXT	Descripció opcional
mass_earth	NUMERIC	Massa en unitats terrestres
radius_earth	NUMERIC	Radi en unitats terrestres
inclination_deg	NUMERIC	Inclinació orbital
rotation_speed_kms	NUMERIC	Velocitat de rotació (km/s)
albedo	NUMERIC	Reflectivitat
star_distance_au	NUMERIC	Distància a l'estrella (unitats astronòmiques)
has_rings	TINYINT	Té anells?
moon_count	INT	Nombre de llunes
surface_texture_url	VARCHAR(200)	Textura de superfície
height_texture_url	VARCHAR(200)	Textura d’alçada (opcional)
thumbnail_url	VARCHAR(200)	Imatge representativa
planetary_system_id	INT (FK)	Sistema al qual pertany
planet_type_id	INT (FK)	Tipus de planeta

atmospheres

Informació d'atmosfera per a planetes individuals.

Columna	Tipus	Descripció
`planet_id`	INT (PK, FK)	Referència directa al planeta
`pressure_atm`	NUMERIC	Pressió atmosfèrica
`greenhouse_factor`	NUMERIC	Factor d'efecte hivernacle
`texture_url`	VARCHAR(200)	Imatge de la textura atmosfèrica

compounds

Compostos químics utilitzats per atmosferes i superfícies.

Columna	Tipus	Descripció
`CID`	INT (PK)	Identificador oficial (PubChem)
`name`	VARCHAR(100)	Nom del compost
`formula`	VARCHAR(50)	Fórmula química

Relacions entre entitats

La base de dades segueix un **model relacional** amb diverses **claus foranes** que garanteixen la integritat referencial. A continuació es resumeixen les relacions principals:

- **`users` ↔ `planetary_systems`**
  Un usuari pot crear molts sistemes planetaris (`1:N`).
  Si un usuari s'elimina, els sistemes associats també s'eliminen automàticament (`ON DELETE CASCADE`).

- **`stars` ↔ `planetary_systems`**
  Cada sistema planetari té una estrella assignada de manera única (`1:1`).
  Una estrella no pot ser compartida per diversos sistemes.

- **`planetary_systems` ↔ `planets`**
  Cada sistema pot contenir múltiples planetes (`1:N`).
  Si s'elimina un sistema, els planetes associats també s'eliminen.

- **planets** ↔ **planet_types**
  Cada planeta pertany a un tipus (N:1), que defineix els seus límits físics.

- **planets** ↔ **atmospheres**
  Relació 1:1. Cada planeta pot tenir com a màxim una atmosfera associada.

- **planets** ↔ **compounds** (taula intermitja planets_compounds)
  Relació N:M. Un planeta pot tenir diversos compostos, i un compost pot aparèixer en molts planetes.

- **atmospheres** ↔ **compounds** (taula atmospheres_compounds)
  Relació N:M. Igual que amb els planetes, però dins l'atmosfera.

- **planet_types** ↔ **compounds** (taula planet_types_compounds)
  Especifica quins compostos són habituals en cada tipus de planeta, i si s'apliquen a la superfície (planet) o a l'atmosfera.

Índexs i claus

S'han definit índexs i claus per garantir el rendiment i la coherència de les dades.

Claus primàries (PRIMARY KEY)

- Cada taula té una clau primària única: user_id, planet_id, planet_type_id
- Les taules intermitjes tenen claus compostes (planet_id, CID) per evitar duplicats.

Claus foranes (FOREIGN KEY)

- Enllacen entitats com planetary_systems.user_id → users.user_id
- S'aplica ON DELETE CASCADE per mantenir la coherència i eliminar dades relacionades automàticament.

Índexs (INDEX, UNIQUE)

- UNIQUE INDEX sobre username, email, star.name, system.name, planet.name per evitar duplicats.
- INDEX addicionals en columnes de cerca i claus foranes (planet_type_id, user_id, etc.) per millorar el rendiment en consultes JOIN.

Script de creació

La base de dades del projecte Exocosmos s'inicialitza mitjançant un script SQL que crea totes les taules, relacions i dades base (com compostos químics i tipus de planeta).

Aquest script:
https://github.com/Joan289/Exocosmos/blob/main/backend/exocosmos-db/script.sql

- Defineix l'esquema exocosmos
- Crea totes les taules relacionades amb usuaris, sistemes, planetes i compostos
- Aplica claus primàries, foranes i índexs únics
- Inclou dades inicials per a planet_types i compounds
- Està preparat per executar-se una única vegada sobre una base de dades buida

Estructura del Codi

Organització del Projecte

El repositori conté dues carpetes principals:

```
/Exocosmos
├── backend/
├── frontend/
```

frontend/

`components/`: components reutilitzables i visuals

- `3d/`: components basats en react-three-fiber
- `cards/`: Targetes per mostrar planetes, estrelles i sistemes
- `forms/`: Formularis d'edició i creació
- `planet/`: Components específics de planetes (dades, gràfiques)
- `/ui`: Elements d'interfície genèrics

`constants/`: constants globals (opcions de filtre, etiquetes, etc.)

`context/`: context global d'autenticació (`AuthContext`)

`hooks/`: hooks personalitzats (com `useFetchWithAuth`)

`models/`: classes amb lògica i càlculs derivats dels recursos

`pages/`: vistes per cada ruta (`/menu`, `/profile`, etc.)

`routes/`: definició central de rutes amb `AppRouter`

`schemas/`: validació de formularis amb `Zod`

`services/`: interacció amb la API

`utils/`: funcions d'utilitat (fetch, parseig d'errors, pujades)

backend/

`controllers/`: lògica de les peticions i respostes de cara recurs

`models/`: accés a base de dades i mètodes CRUD

`routes/`: definició de rutes agrupades per recurs

`middlewares/`: gestió d'errors, validació, auth, pujades

`interfaces/`: definicions TypeScript per a models i dades

`schemas/`: validació d'inputs amb Zod

`utils/`: funcions auxiliars (autenticació, validació, PubChem...)

`tests/`: proves automatitzades amb Vitest i fixtures

`init/`: scripts d'inicialització de la base de dades

L'arxiu principal és server.ts, que monta l'app i exposa la API.

Convencions de nomenclatura

Components React: `PascalCase` (ex: `StarCard.jsx`)

Hooks: prefix `use` (ex: `useAuth.jsx`)

Variables i funcions: `camelCase` (ex: `getPlanetData`)

Constants: `SCREAMING_SNAKE_CASE` (ex: `API_BASE_URL`)

Models i entitats: singular i `PascalCase` (ex: `Planet.ts`)

Schemas Zod: acabats en `Schema.js` o `Schema.ts`

Carpetes: `kebab-case` al backend o `camelCase` al frontend.

Patrons implementats

El projecte Exocosmos implementa diversos patrons de disseny que organitzen el codi de manera clara i escalable. Al backend s'utilitza el patró **Model-Vista-Controlador** (MVC), amb una separació definida entre models de dades, controladors que gestionen la lògica de negoci i rutes que exposen l'API REST. Al frontend, la gestió d'estat global es resol amb el **Context API**, seguint el patró Provider/Consumer, mentre que la reutilització de lògica es fa mitjançant **hooks personalitzats** com `useAuth` o `useFetchWithAuth`, que encapsulen comportaments comuns. Tant en backend com en frontend, el codi es manté modular, agrupant components, pàgines i serveis per funcionalitats concretes, fet que facilita la lectura, el manteniment i la futura ampliació del projecte.

Components Principals

El projecte segueix una arquitectura modular i escalable. Cada recurs principal (planetes, estrelles, sistemes...) està compost per esquemes de validació, models de dades, controladors, rutes, middlewares específics i tipus TypeScript. Això permet una separació clara de responsabilitats i facilita l'evolució i manteniment del codi.

Mòduls

Cada recurs disposa del seu propi mòdul complet amb:

- **Schema Zod** (`schemas/planet.ts`): defineix la validació estricta per crear, modificar, filtrar i recuperar planetes. Inclou esquemes compostos i personalitzats com atmosferes i compostos.

- **Model** (`models/planet.ts`): encapsula tota la lògica d'accés a la base de dades. Realitza consultes SQL, sincronitza relacions amb compostos i atmosferes, i resol entitats relacionades.

- **Controlador** (`controllers/planet.ts`): gestiona les peticions HTTP. Extén un controlador base generic i afegeix funcionalitat personalitzada com `getFull`.

- **Ruta** (`routes/planet.ts`): defineix els endpoints REST per a cada acció (GET, POST, PUT, PATCH, DELETE), protegits per middlewares com `authenticate`, `checkOwnership` i validacions.

- **Middlewares específics** (`middlewares/validate-planet-against-type.ts`): garanteixen que un planeta sigui coherent amb les restriccions del seu tipus (anells, superfície, etc.).

Classes principals

El backend fa ús de classes com `PlanetModel` per accedir a la base de dades amb estructures consistents i reutilitzables. A més, `PlanetController` implementa la lògica de negoci derivada de `BaseController`, permetent accions CRUD genèriques i personalitzades.

Interfícies

Les interfícies TypeScript (a `interfaces/`) defineixen estructures estàtiques per les dades gestionades (`Planet`, `PlanetFull`, `PlanetModelType`...) i pels sistemes auxiliars (`QueryConfig`, `QueryOptions`, etc.). Aquestes asseguren coherència i tipatge fort entre totes les capes.

Serveis

La sincronització de dades relacionades, com compostos i atmosferes, es delega a serveis interns (`utils/planet-sync.ts`). Això permet encapsular complexitat, mantenir DRY i reduir duplicació entre `create`, `update` i `patch`.

APIs Internes

El backend exposa una API RESTful organitzada per recursos. Cada entitat té el seu propi *router* d'Express, controladors especialitzats i esquemes de validació mitjançant Zod. Totes les rutes protegides requereixen autenticació mitjançant cookies amb JWT i passen per middlewares com `authenticate`, `validate`, `checkOwnership` o `methodNotAllowed`.

Endpoints disponibles

A continuació es descriuen els principals endpoints agrupats per recurs:

/auth

- `POST /register` – Registre d'usuari nou
- `POST /login` – Inici de sessió (crea cookie)
- `GET /me` – Retorna dades de l'usuari autenticat
- `PATCH /me` – Edició parcial del propi perfil
  `DELETE /me` – Elimina l'usuari actual
- `POST /logout` – Tanca la sessió

/users

- `GET /` – Llista d'usuaris (filtrable)
- `GET /:id` – Usuari per ID
- `GET /:id/full` – Inclou sistemes, planetes, estrelles associades

/stars

- `GET /` – Llista d'estrelles
- `GET /:id` – Dades bàsiques d'una estrella
- `GET /:id/full` – Inclou sistemes relacionats
- `PUT /:id` – Actualització completa
- `PATCH /:id` – Edició parcial

/planets

- `GET /` – Llista de planetes amb filtres
- `GET /:id` – Dades bàsiques del planeta
- `GET /:id/full` – Inclou sistema, estrella i usuari

- `POST /` – Crea planeta (amb compostos i atmosfera)
  `PUT /:id` – Actualització completa
- `PATCH /:id` – Edició parcial
- `DELETE /:id` – Elimina el planeta

/planetary-systems

- `GET /` – Llista de sistemes planetaris
- `GET /:id` – Dades bàsiques del sistema
- `GET /:id/full` – Inclou estrella i usuari
- `POST /` – Crea sistema planetari
- `PUT /:id`, `PATCH /:id` – Modificació (segons permisos)
- `DELETE /:id` – Elimina sistema

/compounds

- `GET /` – Llista de compostos químics
- `GET /:id` – Compost per ID
- `POST /` – Crea nou compost (opcional)

/planet-types

- `GET /` – Llista de tipus de planeta
- `GET /:id` – Tipus concret per ID

/upload/:entity/:id/:type

- `POST` – Upload d'imatges (avatar, thumbnail, textures...)

Formats de petició/resposta

Totes les comunicacions entre client i servidor segueixen el format JSON estàndard. Les peticions POST, PUT i PATCH utilitzen el header:

```
Content-Type: application/json
```

i el cos (body) de la petició conté dades estructurades segons els schemas validats per Zod.

Exemple de petició: login

```
POST /auth/login
Content-Type: application/json

{
  "email": "joan@gmail.com",
  "password": "Contrasenya123!"
}
```

Exemple de resposta d'èxit

```
{
  "status": "success",
  "message": "Logged in successfully",
  "data": {
    "user_id": 1,
    "username": "joan",
    "email": "joan@gmail.com",
    "profile_picture_url": null,
    "created_at": "2025-05-20T10:26:39.000Z"
  }
}
```

Exemple de creació d'un recurs (planeta)

```
POST /planets
Content-Type: application/json

{
  "name": "TestPlanet",
  "mass_earth": 1,
  "radius_earth": 1,
  ...
}
```

Resposta esperada

```
{
  "status": "success",
  "data": {
    "planet_id": 42,
    "name": "TestPlanet",
    ...
  }
}
```

Errors

Les respostes d'error segueixen el format següent:

json
CopiarEditar

```
{
  "status": "error",
  "message": "Validation failed: 'name' is required"
}
```

Codis d'estat típics:

- `200 OK` — Operació correcta
- `201 Created` — Recurs creat amb èxit
- `400 Bad Request` — Dades incorrectes o incompletes
- `401 Unauthorized` — Token/JWT no vàlid o falta d'autenticació
- `403 Forbidden` — Intent de modificar recurs sense permís (ownership)
- `404 Not Found` — Recurs inexistent
- `500 Internal Server Error` — Error desconegut al servidor

APIs Externes

Aquest projecte fa ús del servei públic **PubChem** per obtenir informació química sobre compostos. Aquesta integració permet que els usuaris introdueixin compostos pel seu identificador únic (CID) i que el sistema recuperi automàticament el nom i la fórmula química corresponents per emmagatzemar-los a la base de dades.

PubChem PUG-View REST API

URL base: https://pubchem.ncbi.nlm.nih.gov/rest/pug_view/data/compound/:CID/JSON

Aquesta API permet consultar informació detallada d'un compost a partir del seu CID.

Funcionament

Quan un compost no existeix a la base de dades local, el sistema l'intenta obtenir automàticament des de PubChem mitjançant la funció `fetchCompoundFromPubChem(CID)`. Aquesta fa una petició `GET` a l'endpoint anterior i extreu:

- El nom del compost (`RecordTitle`)
- La fórmula molecular (del bloc "Names and Identifiers > Molecular Formula")

Si no es troba el compost o si la resposta és incompleta, es llença un error 404 controlat.

Aquest procés s'utilitza tant en la creació manual de compostos com de manera indirecta quan es creen planetes que especifiquen compostos per CID.

Exemple de consulta

```
const compound = await fetchCompoundFromPubChem(23987);
```

Resposta esperada:

```
{
  "CID": 23987,
  "name": "Helium",
  "formula": "He"
}
```

Configuració

No es necessita clau d'API ni registre previ per utilitzar PubChem. Es tracta d'un servei públic gratuït, però es recomana limitar la freqüència de peticions per evitar bloquejos. Per tant no cal incloure claus a variables d'entorn ni cap configuració especial de seguretat.

Seguretat

Mecanismes de Seguretat

Autenticació

L'aplicació implementa un sistema d'autenticació basat en cookies HttpOnly que contenen un token JWT. Aquest token és generat en el backend durant el procés de login i inclou com a mínim l'identificador i el correu electrònic de l'usuari. La cookie és:

- HttpOnly, evitant que el client accedeixi al token mitjançant JavaScript.
- Secure, de manera que només es transmet per connexions HTTPS.
- SameSite configurat com a "None" per permetre la comunicació entre frontend i backend allotjats en orígens diferents.
- Té una expiració de 24 hores (1 dia).

Durant el procés de login (`POST /auth/login`), si l'usuari proporciona credencials vàlides, el backend retorna el token dins d'una cookie. En cada petició posterior a un endpoint protegit, el client envia automàticament aquesta cookie. El backend valida la sessió mitjançant el middleware d'autenticació, que comprova i descodifica el token.

En cas de logout (`POST /auth/logout`), la cookie és eliminada mitjançant `res.clearCookie()` al backend i el frontend neteja l'estat d'autenticació local i redirigeix a `/auth`.

Autorització

L'autorització es basa exclusivament en la propietat dels recursos. No hi ha rols diferenciats (com admin o editor). El middleware `checkOwnership` comprova si l'usuari autenticat és el propietari del recurs que intenta modificar o eliminar.

Aquest mecanisme admet dues formes de comprovació:

- Directa: el recurs conté un `user_id` directament (com en `planetary_systems`).
- Indirecta: el recurs té una relació amb un altre recurs que conté el `user_id`, com ara `planets`, que estan vinculats a sistemes planetaris.

Si l'usuari no és propietari del recurs, es retorna un error 403. Si el recurs no existeix, es retorna un 404.

Encriptació

Les contrasenyes dels usuaris són encriptades amb bcrypt abans de ser emmagatzemades. El cost de hash utilitzat és 10. La comparació durant el login també es fa mitjançant bcrypt.

El token JWT es signa amb una clau secreta definida a `process.env.JWT_SECRET`. Si aquesta variable no està definida (per exemple, en entorns de desenvolupament), s'utilitza un valor per defecte ("supersecret"), només per a testing.

No hi ha altres camps sensibles xifrats explícitament (com el correu o noms).

Gestió de sessions

Les sessions d'usuari són mantingudes mitjançant la cookie HttpOnly que conté el JWT. El frontend no accedeix mai al token ni l'emmagatzema en localStorage o sessionStorage.

Quan l'usuari accedeix a una ruta privada, el frontend fa una petició a `/auth/me` per validar la sessió. Si el token és vàlid, es retorna l'usuari i es marca com a autenticat en el context global de React (`AuthContext`). Si no, es redirigeix a la pantalla de login.

El sistema no implementa cap mecanisme de "refresh token" ni blacklist per revocar tokens. Un cop la cookie expira, l'usuari ha de tornar a iniciar sessió.

L'estat d'autenticació es gestiona globalment mitjançant un context de React, que exposa les funcions `login()`, `logout()`, `checkAuthStatus()` i els valors `isAuthenticated`, `user` i `isLoading`. Aquest context s'inicialitza en tota l'aplicació (inclosos `/auth` i `/`) per poder accedir als mecanismes d'autenticació des de qualsevol lloc.

Polítiques

Control d'accés

Actualment, l'accés als recursos es controla mitjançant autenticació basada en JWT i el middleware de comprovació de propietat (`checkOwnership`). No s'han definit rols ni permisos granulars per a diferents tipus d'usuaris (com administradors, editors, etc.). Tots els usuaris tenen el mateix nivell d'accés i només poden modificar recursos que han creat.

Gestió de contrasenyes

Les contrasenyes es processen al backend mitjançant `bcrypt` amb un cost de 10. No es desen mai en text pla. En actualitzar la contrasenya, es torna a encriptar abans de desar-la. No hi ha polítiques de caducitat, complexitat mínima ni mecanismes de recuperació o reset per correu electrònic.

Protocols de seguretat

Les cookies d'autenticació tenen les opcions `HttpOnly`, `Secure` i `SameSite=None` activades. El projecte està pensat per a funcionar sobre HTTPS. No s'han definit protocols específics per a seguretat en el desplegament, CORS restringit, o protecció contra atacs comuns (XSS, CSRF) més enllà del que ofereix Express i les opcions de cookie.

Logs i auditoria

No hi ha un sistema de logging o auditoria implementat actualment. Els errors es gestionen mitjançant el middleware d'error personalitzat (`AppError`), però no s'enregistren de forma persistent (fitxer o base de dades). Es recomana integrar un sistema de logging (com `winston`, `pino` o similar) i definir una política de retenció per a operacions sensibles.

Resolució de Problemes

Problemes Comuns

En aquesta secció es recullen les incidències més habituals durant el desenvolupament o execució del projecte Exocosmos, així com pautes per diagnosticar-les i solucionar-les eficaçment.

Errors d'arrencada del servidor o de connexió a la base de dades

Verifica que el fitxer `.env` estigui present i correctament configurat, especialment les variables `DATABASE_URL`, `JWT_SECRET`, `PORT` i `TEST_DATABASE_URL` si s'estan executant tests. També és recomanable comprovar si la base de dades està activa i escoltant al port correcte.

Respostes HTTP amb error

En cas de rebre codis d'error del backend:

- `400`: Les dades enviades no passen la validació Zod, revisar formats, camps obligatoris o estructures nested.
- `401`: El token JWT és invàlid, ha expirat o no s'ha enviat; revisar autenticació i cookies.
- `403`: L'usuari autenticat intenta accedir o modificar un recurs que no li pertany; comprovar ownership.
- `404`: El recurs no existeix o l'ID no és vàlid; comprovar si l'element ha estat eliminat.
- `500`: Error intern del servidor; revisar el stack trace complet a consola per identificar l'origen.

Errors de dades o consultes SQL

Si es sospita d'un problema amb consultes a la base de dades, activa logs SQL temporals dins dels models (`console.log(query, values)`) per veure exactament què s'està executant. També és útil revisar que no hi hagi conflictes de claus úniques o problemes de foreign keys.

Errors en validació o transformació de formularis

Si l'enviament d'un formulari falla, comprova la sincronització entre el formulari react-hook-form i els models del backend. També assegura't que les dades transformades abans de fer el submit (per exemple, textures en base64) es generen correctament.

Diagnòstic i testing

- Executa `npm run test` al backend per detectar errors automàticament.
- Utilitza `npm run test:watch` per rebre feedback immediat durant el desenvolupament.
- Fes servir clients REST (com REST Client a VSCode o Postman) per simular peticions manualment.
- Compila el backend amb `tsc --watch` per detectar errors de tipus o imports trencats en temps real.

Aquestes estratègies cobreixen la majoria d'incidències que poden aparèixer tant en entorn local com en producció, i permeten diagnosticar i solucionar problemes sense dependre de tercers.

Annexos

Glossari de termes

Terme	Definició
SPA (Single Page Application)	Aplicació web que carrega una sola pàgina HTML i actualitza dinàmicament el contingut sense recarregar la pàgina.
JWT (JSON Web Token)	Format estàndard per a tokens d'autenticació, signat digitalment. En aquest projecte, s'emmagatzema en cookies HttpOnly.
Cookie HttpOnly	Cookie que no pot ser accedida des de JavaScript, augmentant la seguretat contra atacs XSS.
REST API	Estil arquitectònic d'API que utilitza mètodes HTTP per gestionar recursos remots.
CRUD	Operacions bàsiques sobre dades: Create, Read, Update, Delete.
Zod	Llibreria per a validació d'esquemes en TypeScript.
Multer	Middleware per gestionar pujades d'arxius a Node.js.
Three.js	Llibreria JavaScript per crear gràfics 3D dins del navegador utilitzant WebGL. Permet dibuixar objectes, llums, càmeres i animacions 3D.
React Three Fiber (R3F)	Binding que permet utilitzar Three.js amb la sintaxi de React, creant escenes 3D declarativament.
WebGL	Tecnologia que permet renderitzar gràfics 2D i 3D dins de navegadors web, aprofitant la potència de la targeta gràfica (GPU).
Canvas	Element HTML que permet dibuixar gràfics i imatges personalitzades mitjançant JavaScript.
OrbitControls	Controls que permeten girar la càmera al voltant d'un objecte 3D amb el ratolí o el dit (p. ex. en mòbil).

Post-processat	Conjunt d'efectes visuals aplicats després del renderitzat 3D, com desenfocament o il·luminació avançada.
Bloom	Efecte visual que simula com una llum molt intensa es "desborda" lleugerament, creant un halo o resplendor.
Shader	Petits programes que determinen com es pinta un objecte 3D. Funcionen dins la GPU i poden controlar la forma, el color, les ombres o efectes especials. Exemples: fer que una estrella sembli que brilla o que una superfície tingui textura rugosa.

Documentació addicional

Aquest apartat recull fonts externes, tutorials i enllaços útils consultats o recomanats per entendre o ampliar els coneixements relacionats amb les tecnologies emprades a *Exocosmos*.

Three.js – Documentació oficial: Guia completa per crear escenes 3D al navegador.

https://threejs.org/docs/

React Three Fiber (R3F): Com integrar Three.js en entorns React de manera declarativa.

https://r3f.docs.pmnd.rs/getting-started/introduction

PubChem PUG REST API: Documentació oficial per consultar dades químiques públiques.

https://pubchemdocs.ncbi.nlm.nih.gov/pug-rest