**MEGA TW – doar backend**

**!!!DISCLAIMER!!!**

sarcină = task; utilizator = user; proiect = project; permisiune = permission; starea = status;

S-a folosit romgleză în document ... *îmi cer scuze!*

**BACKEND – server**

*1. O solicitare vine*

*2. Trece prin middleware (de exemplu, autentificare).*

*3. Ajunge la router, care decide unde să-l trimită.*

*4. Controlerul îl prelucrează (de exemplu, se autentifică utilizatorul).*

*5. Dacă este necesar, interacționează cu baza de date (modele).*

*6. Un răspuns este trimis înapoi către front-end.*

*1️. Utilizatorul trimite o cerere de conectare (e-mail + parolă).*

*2️. Middleware-ul procesează cererea (CORS, parsare JSON).*

*3️. Routerul îl direcționează către controlerul corect (auth-controller.mjs).*

*4️. Controlerul se ocupă de logica (verifică parola, creează token).*

*5️. Baza de date stochează/preluează datele utilizatorului (modele/User.mjs).*

*6️. Răspunsul este trimis înapoi (succes sau eșec)*

* middleware – procesează request-urile înainte de a ajunge la controllere
  + **19)** assigned-task-middleware.mjs
    - Acest fișier asigură că numai utilizatorul atribuit poate actualiza o sarcină.
    - Găsește task-ul în baza de date → Căută un task în care:
      * Id-ul se potrivește cu req.params.tid (ID-ul task-ului).
      * assignedToId se potrivește cu req.params.uid (ID-ul utilizatorului atribuit).
    - Verifică dacă utilizatorul este alocat sarcinii:
      * Dacă sarcina este găsită, apelează next(), permițând cererii să continue.
      * Dacă sarcina nu este găsită, returnează o eroare 403 Forbidden.
    - Scopul principal: Acest middleware restricționează actualizările sarcinilor, astfel încât numai utilizatorul atribuit sarcinii o poate modifica.
  + **17)** auth-middleware.mjs
    - Impune autentificarea prin verificarea dacă un utilizator este conectat înainte de a accesa rutele protejate.
    - Verifică dacă există un token de autorizare:
      * Dacă nu este furnizat niciun token în anteturile cererii, acesta returnează o eroare 401 „Unauthorized”.
    - Găsește utilizatorul pe baza token-ului:
      * Caută un utilizator în baza de date al cărui token se potrivește cu cel din req.headers.authorization.
      * Dacă nu se găsește niciun utilizator care se potrivește, returnează o eroare 401 „Unauthorized”.
    - Atașează utilizatorul la cerere (req.user):
      * Dacă utilizatorul este găsit, acesta îl atașează la req.user, permițând cererii să continue.
    - Apelează next() pentru a continua procesarea cererii dacă autentificarea are succes.
    - Scopul principal: protejează rutele asigurându-se că numai utilizatorii autentificați le pot accesa.
  + **21)** generic-error-middlewware.mjs
    - Tratează erori neașteptate în backend.
    - Înregistrează eroarea → Imprimă eroarea pe consolă utilizând console.warn(err).
    - Trimite un răspuns de 500 de „Internal Server Error”→ Returnează { status: 'server error' } în format JSON.
    - Scopul principal: Acest middleware prinde și tratează erorile la nivel global, prevenind blocarea serverului din cauza erorilor negestionate.
  + **16)** index.mjs
    - Acest fișier organizează și exportă toate funcțiile middleware utilizate în backend.
    - Importă funcții middleware individuale:
      * genericError → Gestionează erorile din aplicație.
      * auth → Se asigură că utilizatorii sunt autentificați înainte de a accesa rutele protejate.
      * getPermMiddleware → Verifică dacă un utilizator are permisiunile necesare pentru o sarcină sau proiect.
      * assignedTaskMiddleware → Se asigură că numai utilizatorii alocați pot actualiza starea sarcinii.
      * getUserTypeMiddleware → Determină tipul de utilizator (de exemplu, administrator sau utilizator obișnuit).
    - Le grupează într-un singur obiect → Ușurează importarea și aplicarea middleware-ului în app.mjs.
    - Scopul principal: Centralizează funcțiile middleware, facilitând aplicarea autentificărilor, a permisiunilor și a gestionării erorilor în backend.
  + **18)** perm-middleware.mjs
    - Verifică dacă un utilizator are permisiunile necesare pentru a accesa sau modifica o resursă.
    - Primește parametri (resourceIdKey, drepturi) → Stabilește ce resursă (sarcină sau proiect) dorește să acceseze utilizatorul și ce permisiuni sunt necesare.
    - Găsește înregistrarea de permisiuni a utilizatorului → Căută în modelul de permisiuni pentru a verifica dacă utilizatorul are acces la resursa solicitată.
    - Validează permisiunile necesare:
      * Dacă utilizatorul are toate drepturile necesare, cererea continuă (next()).
      * Dacă utilizatorul nu are drepturi, returnează o eroare 403 „Forbidden”.
    - Scopul principal: Acest middleware impune controlul accesului, asigurându-se că utilizatorii pot modifica doar sarcini sau proiecte pentru care au permisiunea.
  + **20)** user-type-middleware.mjs
    - Acest fișier restricționează accesul pe baza rolurilor utilizatorului (de exemplu, administrator vs. utilizator obișnuit).
    - Primește un tip de utilizator (type) ca parametru → Specifică ce rol este permis.
    - Verifică dacă utilizatorul conectat corespunde tipului necesar:
      * Dacă tipul utilizatorului se potrivește cu rolul solicitat, cererea continuă (next()).
      * Dacă nu, returnează o eroare 403 „Forbidden”.
    - Scopul principal: Acest middleware asigură că numai utilizatorii cu rolul corect (de exemplu, administratori) pot accesa anumite rute.
* models – ***BAZE DE DATE, TABELE etc.***

*!!!Nu prea contează ordinea numerotării aici – sunt doar tabele și index.mjs inițializează BD-ul cu toate tabelele și dependințele dintre ele*

* + **8)** index.mjs
    - Inițializează baza de date, definește modele (tabele) și stabilește relații folosind Sequelize.
    - Creează o instanță Sequelize → Se conectează la o bază de date SQLite (db.sqlite).
    - Inițializează modelele (tabelele) → Definește user, project, task și permission. *(NUMEȘTE-LE VERE ÎN ROMÂNĂ ȘI LASĂ-MĂ ÎN PACE CĂ M-AI OMORÂT)*
    - Definește relațiile dintre modele:
      * Un user poate avea multe projects.
      * Un project aparține unui user.
      * Un project poate avea multe task-uri.
      * Un task aparține unui project.
      * Un project și un task pot avea fiecare un permission.
      * Un task aparține unui user.
    - Sincronizează baza de date (sequelize.sync()) → Se asigură că tabelele se potrivesc cu definițiile modelului.
    - Exportă modele → Deci ele pot fi folosite în întregul backend.
    - Scopul principal: Configurează și gestionează structura bazei de date, asigurându-se că modelele și relațiile sunt definite corect.
  + **11)** permission.mjs
    - Definește tabela permission, care gestionează controlul accesului pentru sarcini și proiecte.
    - Creează un model (o tabelă) Sequelize numit(ă) „permission” → Reprezintă permisiunile în baza de date.
    - Definește câmpurile pentru permisiuni:
      * forResource → ID-ul sarcinii/proiectului căruia i se aplică această permisiune (obligatoriu).
      * forUser → ID-ul utilizatorului căruia i se aplică această permisiune (obligatoriu).
      * type → Specifică dacă permisiunea se aplică unei sarcini sau unui proiect (obligatoriu, ENUM).
      * rights → Stochează lista de permisiuni acordate, stocată ca o matrice JSON (obligatorie, implicit la []).
    - Utilizează Sequelize DataTypes:
      * INTEGER → Pentru ID-uri de resursă și utilizator.
      * ENUM(task, project) → Se asigură că permisiunile se aplică numai sarcinilor sau proiectelor.
      * JSON → Stochează o listă de drepturi ca matrice.
    - Exportă modelul → Deci poate fi folosit pentru verificări de control al accesului în backend.
    - Scopul principal: Controlează accesul utilizatorilor la sarcini și proiecte, specificând ce acțiuni pot efectua.
  + **9)** project.mjs
    - Creează un model (o tabelă) Sequelize numit(ă) „project” → Reprezintă proiectele în baza de date.
    - Definește câmpurile pentru proiect:
      * name → Numele proiectului (obligatoriu).
      * description → O descriere text mai lungă (opțional).
    - Utilizează Sequelize DataTypes:
      * STRING → Pentru numele proiectelor.
      * TEXT → Pentru descrieri mai lungi.
    - Exportă modelul → Deci poate fi folosit în întregul backend.
    - Scopul principal: Definește structura proiectelor din baza de date, specificând câmpurile obligatorii și opționale.
  + **10)** task.mjs
    - Creează o tabelă Sequelize numită „task” → Reprezintă sarcini în baza de date.
    - Definește câmpurile pentru sarcină:
      * title → Titlul sarcinii (obligatoriu).
      * description → O descriere text mai lungă (opțional, dar marcată incorect ca fiind necesar).
      * status → Starea curentă a sarcinii (implicit la „open”).
    - Utilizează Sequelize DataTypes:
      * STRING → Pentru titlu și stare.
      * TEXT → Pentru o descriere detaliată.
    - Exportă modelul → Deci poate fi folosit în întregul backend.
    - Scopul principal: Definește structura sarcinilor din baza de date, asigurându-se că fiecare sarcină are un titlu, o descriere și un statut.
  + **7)** user.mjs.
    - Definește în Sequelize o tabelă „user” → Reprezintă utilizatorii din baza de date.
    - Câmpurile din model:
      * e-mail → Stochează e-mailul utilizatorului (obligatoriu).
      * passwordHash → Stochează parola codificată (obligatoriu).
      * token → Stochează token-ul de autentificare (opțional).
      * type → Definește rolul utilizatorului (admin sau regular).
    - Utilizează Sequelize DataTypes:
      * STRING → Folosit pentru stocarea câmpurilor de text, cum ar fi e-mail și indicativ.
      * ENUM('admin', 'regular') → Restrânge câmpul de tip la două valori posibile.
    - Scopul principal: Definește structura bazei de date pentru utilizatori, inclusiv câmpuri legate de autentificare, cum ar fi e-mailul, hashul parolei și token-ul.
* *node\_modules (nu ne interesează)*
* routers – decid unde (la ce controller) să meargă request-ul
  + controllers – decid ce să se întâmple cu request-ul
    - **6)** auth-controller.mjs
      * Gestionează autentificarea utilizatorilor, inclusiv login, logout și registration.
      * login(req, res, next) → Gestionează autentificarea utilizatorului:
        + Găsește utilizatorul în baza de date prin e-mail.
        + Verifică parola folosind bcrypt.
        + Dacă este valid, generează un token JWT, îl salvează și îl returnează utilizatorului.
        + Dacă este invalid, returnează o eroare 401 neautorizată.
      * logout(req, res, next) → Gestionează deconectarea utilizatorului:
        + Găsește utilizatorul după token său.
        + Elimină token-ul din baza de date.
        + Returnează un mesaj de succes.
      * register(req, res, next) → Înregistrează un utilizator nou:
        + Hash parola folosind bcrypt.
        + Creează un utilizator nou în baza de date.
        + Returnează utilizatorul nou creat.
      * Scopul principal: oferă o logică de autentificare pentru autentificare, deconectare și înregistrarea utilizatorilor.
    - **5)** index.mjs
      * Acest fișier organizează și exportă toate controlerele API, care se ocupă de logica de afaceri pentru diferite părți ale aplicației.
      * Importă controlere individuale:
        + auth-controller.mjs → Gestionează autentificarea (conectare, deconectare).
        + project-controller.mjs → Gestionează operațiunile legate de proiect.
        + task-controller.mjs → Gestionează sarcinile din cadrul proiectelor.
        + user-controller.mjs → Se ocupă de acțiunile legate de utilizator.
      * Le grupează într-un singur obiect → Facilitează accesul la controlerele din routere.
      * Exportă controlerele → Permite ruterelor să apeleze anumite funcții ale controlerului.
      * Scopul principal: Acționează ca un hub centralizat pentru logica afacerii, păstrând controlerele modulare și organizate.
    - **13)** project-controller.mjs
      * Se ocupă de acțiuni legate de proiect, cum ar fi preluarea, crearea, actualizarea și ștergerea proiectelor.
      * getAllProjects(req, res, next) → Preia toate proiectele pentru un utilizator.
        + Acceptă filtrarea, paginarea și sortarea pe baza parametrilor de interogare.
        + Include permisiuni pentru a verifica dacă utilizatorul are acces la proiecte.
      * getOneOwnedProject(req, res, next) → Preluează un singur proiect după ID-ul său.
        + Dacă proiectul există, acesta returnează proiectul.
        + Dacă nu, returnează o eroare 404 Project not found.
      * createOwnedProject(req, res, next) → Creează un nou proiect pentru utilizator.
        + Stochează proiectul în baza de date.
        + Atribuie automat creatorului permisiuni complete (read, write).
      * updateOwnedProject(req, res, next) → Actualizează un proiect existent.
        + Găsește proiectul și își actualizează detaliile.
        + Returnează o eroare 404 Project not found dacă proiectul nu există.
      * deleteOwnedProject(req, res, next) → Șterge un proiect.
        + Găsește și elimină proiectul din baza de date.
        + Dacă are succes, returnează un răspuns 204 „No Content”
        + Dacă proiectul nu există, returnează o eroare 404 „Project not found.”
      * Scopul principal: Acest controller gestionează toate operațiunile legate de proiecte, asigurându-se că utilizatorii pot crea, actualiza și șterge proiecte în timp ce verifică permisiunile.
    - **14)** task-controller.mjs
      * Se ocupă de toate operațiunile legate de task-uri, cum ar fi preluarea, crearea, actualizarea, ștergerea și atribuirea de task-uri.
      * getAllTasksForProject(req, res, next) → Preia toate task-urile pentru un proiect dat.
        + Filtrează sarcinile după ID-ul proiectului (pid).
        + Include detaliile utilizatorului atribuite și datele de permisiuni.
        + Acceptă paginarea și filtrarea dacă sunt furnizați parametrii de interogare.
      * getOneTaskForProject(req, res, next) → Preluează o singură sarcină după ID-ul său într-un proiect.
        + Dacă sarcina există, aceasta returnează sarcina.
        + Dacă nu, returnează o eroare 404 „Task not found”.
      * createOwnedTaskForProject(req, res, next) → Creează o sarcină nouă în cadrul unui proiect.
        + Salvează sarcina în baza de date.
        + Atribuie automat creatorului permisiuni de citire și scriere pentru sarcină.
      * updateOwnedTaskForProject(req, res, next) → Actualizează o sarcină existentă într-un proiect.
        + Găsește sarcina și își actualizează detaliile.
        + Dacă sarcina nu există, returnează o eroare 404 „Task not found”.
      * deleteOwnedTaskForProject(req, res, next) → Șterge o sarcină dintr-un proiect.
        + Găsește sarcina și o elimină din baza de date.
        + Dacă are succes, returnează un răspuns 204 „No content”.
        + Dacă sarcina nu există, returnează o eroare 404 „Task not found”.
      * assignTaskToUser(req, res, next) → Atribuie o sarcină unui utilizator.
        + Verifică dacă sarcina și utilizatorul există.
        + Actualizează câmpul assignedToId al task-ului pentru utilizatorul specificat.
        + Dacă task-ul sau utilizatorul nu există, returnează o eroare 404 „Not Found”.
      * updateAssignedTaskStatus(req, res, next) → Actualizează statusul unei sarcini atribuite.
        + Găsește sarcina și își actualizează câmpul de status.
        + Dacă sarcina nu există, returnează o eroare 404 „Task not found”.
      * Scopul principal: Acest controller gestionează toate operațiunile legate de sarcini, asigurându-se că utilizatorii pot crea, actualiza, atribui și urmări sarcini în cadrul proiectelor.
    - **15)** user-controller.mjs
      * Se ocupă de operațiuni legate de utilizator, cum ar fi preluarea unui profil de utilizator și sugerarea utilizatorilor pe baza căutării prin e-mail.
      * getUserProfile(req, res, next) → Preia profilul unui utilizator.
        + Găsește utilizatorul după ID-ul de utilizator (uid).
        + Dacă utilizatorul există, returnează datele utilizatorului.
        + Dacă nu, returnează o eroare 404 „User not found”.
      * suggestUser(req, res, next) → Sugerează utilizatori pe baza unei potriviri parțiale de e-mail.
        + Caută utilizatori a căror e-mail conține termenul de căutare furnizat.
        + Returnează până la 5 utilizatori (pentru a limita rezultatele căutării).
        + Returnează doar id și e-mail.
      * Scopul principal: Acest controller preia informații despre utilizator și sugerează utilizatorii pe baza căutărilor prin e-mail, făcându-l util pentru funcții precum selecția utilizatorului cu completare automată.
  + **21)** admin-router.mjs
    - Acest fișier restricționează accesul la rutele numai pentru administrator.
    - Creează un router Express (adminRouter) → Gestionează rutele legate de administrator.
    - Aplică middleware-ul getUserTypeMiddleware('admin') → Se asigură că numai utilizatorii cu rol de administrator pot accesa aceste rute.
    - Exportă routerul → Deci poate fi folosit în app.mjs.
    - Scopul principal: Acest router protejează rutele specifice administratorului, permițând doar utilizatorilor admin să le acceseze.
  + **12)** api-router.mjs
    - Definește principalele API routes pentru gestionarea utilizatorilor, proiectelor și sarcinilor.
    - Creează un router Express (apiRouter) → Gestionează solicitările către /api/....
    - Aplică middleware de autentificare (middleware.auth) → Se asigură că numai utilizatorii conectați pot accesa aceste rute.
    - Definește endpoints-urile legate de proiecte (project):
      * Obțineți toate proiectele utilizatorului → GET /users/:uid/projects
      * Obțineți un singur proiect → GET /users/:uid/projects/:pid
      * Creați un proiect → POST /users/:uid/projects
      * Actualizați un proiect (necesită permisiunea de scriere) → PUT /users/:uid/projects/:pid
      * Ștergeți un proiect (necesită permisiunea de scriere) → DELETE /users/:uid/projects/:pid
    - Definește endpoints-urile legate de sarcini (task):
      * Obțineți toate sarcinile pentru un proiect → GET /users/:uid/projects/:pid/tasks
      * Obțineți o singură sarcină → GET /users/:uid/projects/:pid/tasks/:tid
      * Creați o sarcină (necesită permisiunea de scriere) → POST /users/:uid/projects/:pid/tasks
      * Actualizați o sarcină (necesită permisiunea de scriere) → PUT /users/:uid/projects/:pid/tasks/:tid
      * Ștergeți o sarcină (necesită permisiunea de scriere) → DELETE /users/:uid/projects/:pid/tasks/:tid
      * Atribuiți o sarcină unui utilizator → POST /users/:uid/projects/:pid/tasks/:tid/assignments
      * Actualizați starea sarcinii (pentru utilizatorii alocați) → PUT /users/:uid/projects/:pid/tasks/:tid/status
    - Definește endpoints-urile legate de utilizator:
      * Obține profilul utilizator → GET /users/:uid/profile
      * Sugerați un utilizator pe baza e-mailului → GET /users/suggestions
    - Scopul principal: gestionează endpoints-urile API pentru operațiunile utilizator, proiecte și sarcini, asigurând un control adecvat al accesului prin middleware.
  + **4)** auth-router.mjs
    - Definește punctele finale API legate de autentificare folosind Express.
    - Acțiuni cheie:
      * Creează un nou router Express → authRouter se ocupă de rutele de autentificare.
      * Definește rutele de autentificare:
        + POST /login → Apelează controllers.auth.login pentru a gestiona autentificarea utilizatorului.
        + POST /logout → Apelează controllers.auth.logout pentru a gestiona deconectarea utilizatorului.
      * Exportă routerul → Permite app.mjs să înregistreze aceste rute sub /auth.
  + Scopul principal: Setează API endpoints-urile legate de autentificare pentru autentificare și deconectare.
  + **3)** index.mjs
    - Organizează și exportă toate API routes pentru un acces ușor în app.mjs.
    - Acțiuni cheie:
      * Importă route handlers → admin-router.mjs, api-router.mjs, auth-router.mjs.
      * Le grupează într-un singur obiect → Permite importarea și utilizarea ușoară în app.mjs.
      * Oferă modularitate → Menține definițiile rutelor organizate și separate.
  + Scopul principal: Acționează ca un hub centralizat pentru rutare, facilitând gestionarea API endpoints-urilor.
* **2)** app.mjs
  + Inițializează și configurează aplicația Express care alimentează backend-ul.
  + Acțiuni cheie:
    - Creează o aplicație Express → Inițializează cadrul de backend.
    - Activează CORS → Permite solicitări de la <http://localhost:3000>.
    - Analizează solicitările JSON → Se asigură că serverul poate gestiona datele JSON.
    - Înregistrează rutele API →
      * /auth → Se ocupă de autentificare (autentificare, înscriere etc.).
      * /api → Gestionează funcționalitățile generale ale aplicației (proiecte, sarcini etc.).
    - Se aplică middleware de tratare a erorilor → Captează și procesează erori.
    - Scopul principal: setează middleware și rute, asigurându-se că backend-ul Express este pregătit să gestioneze solicitările API.
* data-preload.mjs
* db.sqlite
* *package-lock.json (nu ne interesează)*
* *package.json (maxim pentru script-uri, dar în rest nu ne interesează)*
* **1)** server.mjs
  + Inițializează și pornește serverul backend folosind Node.js și Express.
  + Acțiuni cheie:
    - Încarcă variabile de mediu → Utilizează dotenv pentru a citi configurația dintr-un fișier .env.
    - Definește portul serverului → Utilizează process.env.PORT sau implicit la 8080.
    - Creează un server HTTP → Înfășoară aplicația Express (app.mjs) într-un server http Node.js.
    - Începe să asculte cereri → Rulează serverul și înregistrează portul activ.
  + Scopul principal: servește ca punct de intrare pentru backend, instalând și lansând serverul Express.