**Nexus bemutató vázlat**

Az alkalmazás egy interaktív kommunikációs platform lesz, amelyben tanárom és diákok tudnak regisztrálni, posztokat létrehozni csoportokon belül, a posztokhoz kommentelni.

Technikailag összetettebb részek:

**frontend**

- Az authentikáció megoldásánál dilemma volt, hogy .server.ts kiterjesztésű fájlokat érdemes-e használni az egyes page-ekhez, amelyek session infót kezelnek – végül mindenhol kikommenteltem a próbálkozásokat, vagy nem is hoztam létre server fájlokat. Magyarázat: a Svelte képes szerver oldali renderelésre is a .server.ts kiterjesztésű fájlokban, viszont ebben a projektben külön backend van Express JS-ben, ami lekezeli a backend-oldalon a session-öket.

Egy kivétel fájl van: hooks.server.ts, ennek magyarázata a kód kommentjeiben van.

**backend**

- Authentikáció megoldása: *session* és *express-session* package-ek használatával, emellett session store megvalósítása is megtörtént *expressMySQLSession* package használatával (session-ök eltárolása az adatbázisban). Erre azért van szükség, mert enélkül a fenti session package-ek csak a memóriában tárolják el a session infókat, és a backend leállása esetén elvész minden session infó.

A jelenlegi működésben viszont valamiért elvesznek a session infók a backend leállása esetén, és mindig új bejegyzés kerül be az adatbázisba ugyanannak a felhasználónak az adataival, ha bejelentkezik – még nyomozni kell mi megy félre.

- Regisztráció probléma: az adatbázisban a user táblában a role mező enum-ként lett létrehozva, ezért csak ’ADMIN’, ’STUDENT’, ’TEACHER’ értékeket fogad el. Más értékekkel való regisztráció még nincs lekezelve, ez okozott problémát az eddigi teszteléseknél.

- Posztok listázása felhasználói csoportok szerint: komplex SQL logikát igényelt a megoldása, a lekérdezett rekordokból JSON-t állít elő a logika (MySQL 5.7-ben vezették be a JSON adattípust), az alkalmazás MySQL 8-at használ).

- a publikus posztokhoz nem csatolunk semmilyen csoportot, mert minden csoport láthatja -> felesleges lenne minden csoporthoz hozzákapcsolni, elég a visibility mező, ami jelzi, hogy PUBLIC. Ezért a publikus posztok lekérdezésére külön query van (database.js, getPublicPostsWithComments). A publikus posztok létrehozása pedig kívülről jövő paraméterként érkezik meg a frontenden bepipált érték alapján (helpers.js, addPost – itt dől el, hogy PUBLIC vagy PRIVATE lesz a poszt visibility-je).

SQL magyarázata: a getPublicPostsWithComments-ben lévő SQL lekérdezés publikus posztokat listáz, és mindegyikhez hozzácsatolja a hozzájuk tartozó kommenteket egy JSON tömbben.

Az eredmény minden egyes poszt esetén tartalmazza a poszt adatait, valamint egy comments nevű mezőt, ami JSON tömbben tartalmazza a hozzászólásokat, tehát két egymásba ágyazott JSON tömböt állítunk össze.

SELECT

Megadja, hogy mely oszlopokat szeretnénk visszakapni: a poszt adatai (id, title, content, created\_at, user\_id) + a kommentek JSON tömbje (comments).

LEFT JOIN comments c ON c.post\_id = p.id

A posts táblát bal oldali összekapcsolással (LEFT JOIN) összeköti a comments táblával a post\_id mező alapján.

Így akkor is visszakapjuk a posztot, ha nincs hozzá komment (mert a LEFT JOIN "megtartja" a postot).

WHERE p.visibility = 'PUBLIC'

Csak azokat a posztokat vesszük figyelembe, amelyek publikusak.

JSON\_ARRAYAGG(...) AS comments

A kommenteket egy JSON tömbbe aggregálja posztonként (ez a GROUP BY miatt működik). Ez azt jelenti, hogy egy posthoz egyetlen sor lesz, benne a hozzászólások tömbjével.

CASE WHEN c.id IS NOT NULL THEN JSON\_OBJECT(...) ELSE JSON\_OBJECT('content', NULL)

Ez biztosítja, hogy ha egy poszthoz nincs komment, akkor is legyen egy "üres" JSON objektum a tömbben ({ "content": null }).

GROUP BY p.id

Csoportosítjuk az eredményt postonként — így tudunk posztonként aggregálni kommenteket.

ORDER BY p.created\_at DESC

A bejegyzéseket a létrehozási dátum szerint csökkenő sorrendbe rendezzük (legújabb elöl).

- a nem publikus posztokhoz hozzá kell csatolni, hogy milyen csoportokba tartoznak. Ez a frontenden bepipált csoportok alapján dől el (csak a bejelentkezett felhasználó saját csoportjai közül lehet választani). Ezért a nem publikus posztok lekérdezésére külön query van (database.js, getPostsWithCommentsByUserGroups).

SQL magyarázata: a getPostsWithCommentsByUserGroups -ban lévő SQL lekérdezés privát posztokat listáz ki, amelyek elérhetők egy adott felhasználónak csoporttagság alapján.

Emellett hozzácsatolja a kommenteket is egy JSON tömbben minden posthoz.

A lekérdezés csak azokat a privát posztokat adja vissza, amelyek olyan csoporthoz tartoznak, ahol a megadott felhasználó (ug.user\_id = ?) is tag.

SELECT

A poszt adatait (id, title, content, created\_at, user\_id) és a kommentek JSON tömbjét (comments) kérjük le.

JOIN post\_groups pg ON p.id = pg.post\_id

A posts táblát összeköti a post\_groups táblával, hogy megtudjuk, melyik poszt melyik csoporthoz tartozik.

JOIN user\_groups ug ON pg.group\_id = ug.group\_id

A post\_groups és a user\_groups táblát összekapcsolja, hogy megtudjuk, a megadott felhasználó tagja-e annak a csoportnak, amelyikhez a poszt tartozik.

LEFT JOIN comments c ON c.post\_id = p.id

A kommenteket csatolja a posztokhoz, még akkor is, ha nincsenek (bal oldali join megtartja a posztot).

WHERE p.visibility = 'PRIVATE' AND ug.user\_id = ?

Csak a privát postokat vesszük figyelembe, és csak azokat, amelyek a megadott felhasználónak láthatók (csoporttagság alapján).

A többi megegyezik a publikus lekérdezéssel.