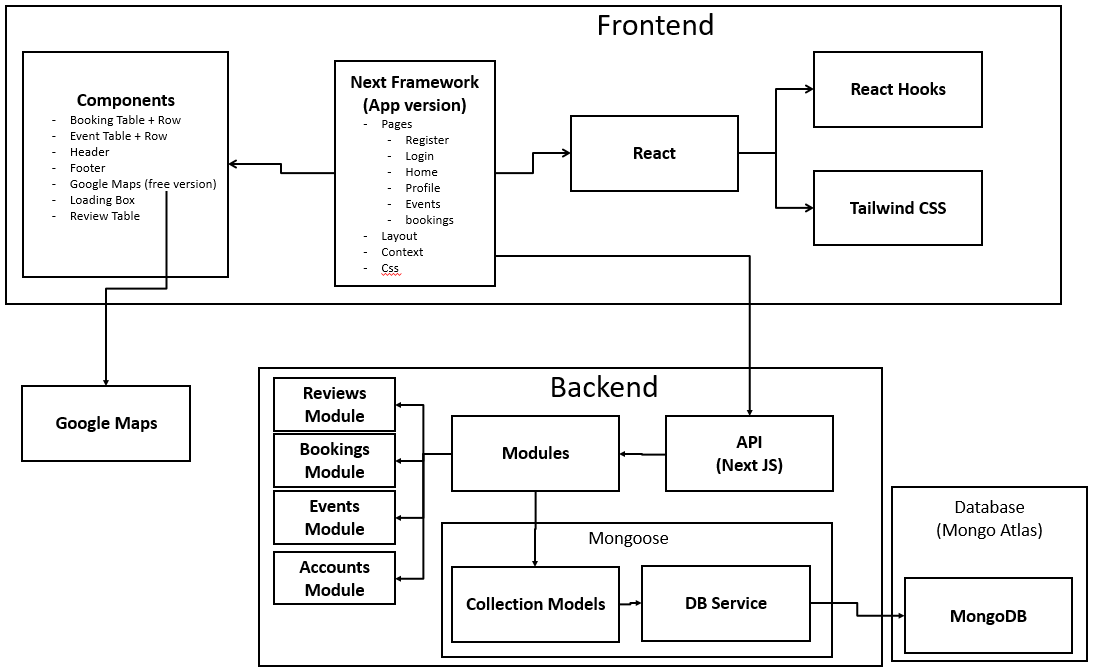
**תיק למתכנת – פרויקט TouRingo**

ארכיטקטורה של הפרויקט:



פונקציות מרכזיות:

* חיבור לבסיס נתונים: החיבור ל-MongoDB נעשה בעזרת חבילת Mongoose בתבנית עיצוב של Singleton. הקוד נמצא בקובץ src/db\_utils/dbService והחיבור ממומש במחלקת DB\_Service. הגישה לחיבור ניתנת רק על ידי הפונקציה הסטטית getInstance()

דוגמא לשימוש:

const mongoDbInstance = DB\_Service.getInstance();

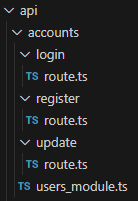
const db = await mongoDbInstance.connect();

* המשך בעמוד הבא.
* פירוט על המבנה של ה-API של השרת: כל נושא בפרויקט (כגון אירועים, משתמשים, ביקורות וכו') נמצא בתיקייה משלו על מנת שהניתוב של Next js יהיה יותר נוח.

כל פעולת טיפול בבקשה נמצאת בתת-תיקייה משלה, והפונקציות של הפעולות מול ה-DB נמצאות בקובץ ה-module של אותו נושא.

בחרנו לעבוד כך על מנת שיהיה קוד יותר "נקי" בקובץ של הבקשות, וכל הפונקציות שניגשות לשרת עבור אותו נושא יהיה באותו קובץ.

דוגמא: עבור נושא Accounts יש את התיקיות:



בכל קובץ route.ts יש את הטיפול עבור אותה בקשה (כגון ב-route.ts של register יש טיפול בבקשת ה-POST של משתמש חדש) וב-users\_module.ts יש את הטיפול של הבקשה מול ה-DB.

* אתחול בסיס הנתונים: ניתן לאתחל את בסיס הנתונים על ידי בקשת GET ל-URL:

/api/dev/createExamples

ה-collections יבנו ב-DB עם הנתונים ב-examplesData.

* בנוגע לבקשות מהשרת:
  + במידע ובקשה לא התקבלה מסיבה כלשהי יוחזר למשתמש קוד
    - 4XX אם הסיבה היא בגלל הלקוח (מידע לא תקני או חסר)
    - 5XX אם הסיבה היא פנימית (שגיאה בצד שרת או בבסיס נתונים)

ובנוסף תהיה הודעת שגיאה ב-body.message.

במידה והבקשה תקינה תוחזר תשובה עם קוד 2XX ואם אמור לחזור תוכן כלשהו הוא יחזור ב-body.result.

* + עבור בקשות מסוג GET ו-PATCH השרת מצפה לקבל את הפרמטרים ב-URL כך שהם מופרדים על ידי '/'.

דוגמא: עבור בקשה של קבלת ה-bookings של משתמש /api/bookings/get/[username] הפרמטר הוא username.

* + עבור בקשות מסוג PUT ו-POST השרת מצפה לקבל את הפרמטרים של הבקשה דרך ה-body בשדה data שיחיל מחרוזת שמייצגת JSON שהוצפן על ידי פונקציית encryptData() הנמצאת ב-utils.

דוגמא: בקשת update של accounts מצפה לפרמטרים

interface UpdateBodyParam{

    username: string;

    updatedAccount: Account;

}

וניגשת אליהן בצורה הבאה:

        const reqBody = await request.json();

        const data = decryptData(reqBody.data) as UpdateBodyParam;

        const username = (!data) ? null : data.username;

        const account = (!data) ? null : data.updatedAccount;

* פירוט תהליך טיפול בבקשת login בשרת על ידי שליחת בקשת PUT ל-URL

/api/accounts/login

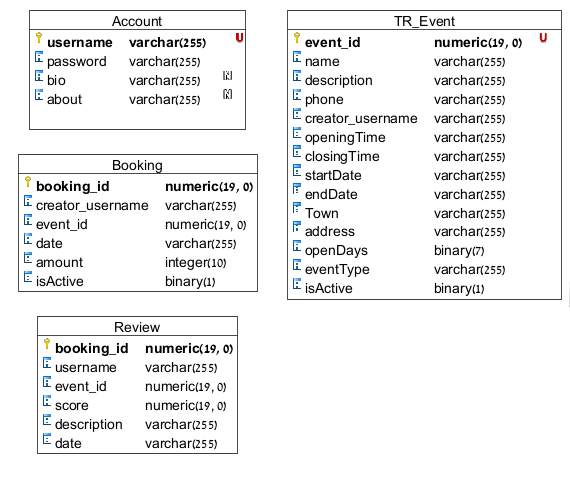
* + השרת מצפה לקבל בקשה מסוג PUT עם הפרמטרים בצורך המתוארת מעל.
    - בחרנו לעשות את הבקשה הזו מסוג PUT בגלל סיבות אבטחת מידע.
  + במידע והפרמטרים לא תקינים יוחזר למשתמש תשובה עם קוד 400.
  + נקראת הפונקציה verifyAccount() מ-accounts\_module שבודקת האם נמצא משתמש עם אותו username ו-password.
    - במידה וכן יוחזר לבקשה תשובה עם קוד 200 ואובייקט של המשתמש ב-body.result, הסיסמה האמתית לא תהיה באובייקט אלה תוחלף ב-"#######".
    - במידה ולא קיים משתמש כזה תוחזר תשובה עם קוד 401 והודעה body.message= "Username or password incorrect"
* תהליך הצגת האירועים למשתמש:
  + בעת טעינת העמוד נשלחת בקשת GET לשרת (/api/events/getAll) ואם התשובה תקינה האירועים שהתקבלו ישמרו ב-state של events.
    - עד שהאירועים יתקבלו מהשרת מוצג למשתמש LoadingBox.
  + כל האירועים שבוטלו (isActive=false) יוסרו מהרשימה.
  + רשימת האירועים נשלחת ל-EventTable וכל אירוע מוצג ב-EventRow.
  + במידע והשתמש לחץ על אירוע, אירוע זה נשמר ב-selectedEvent על ידי קריאה ל-dispatch של ה-MainContext.
  + כאשר משתמש נכנס לעמוד של אירוע יש שני מקרים:
    - אם הוא הגיע על ידי לחיצה על EventRow המידע של האירוע יילקח מ-selectedEvent ויובאו הביקורות של אותו אירוע על ידי אובייקט זה.
    - אם הוא הגיע ידנית על ידי עריכת ה-url תשלח בקשת GET מהשרת על מנת לקבל את הנתונים של האירוע.
      * אם האירוע לא קיים המשתמש יופנה ל-404.
      * אם האירוע כן קיים תשלח בקשת GET עבור קבלת הביקורות של האירוע.
* המשך בעמוד הבא.

הספריות החיצוניות שנעשה בהן שימוש:

* + - Next js: לבניית הפרויקט, ספציפית ב-Next js מסוג App.
    - Mongoose – כל התקשורת עם ה-MongoDB atlas נעשתה בעזרת חבילה זו, כולל בניית ה-schemes והמודלים של ה-collections.
    - crypto-js – נעשה שימוש בחבילה זו על מנת להצפין את המידע שעובר לשרת בבקשות ה-POST וה-PUT, ולפענח אותן בצד של השרת.
    - Tailwind – לצורך עיצוב הצד לקוח.
    - Eslint – לצורך זיהוי בעיות בקוד, אכיפת סגנון כתיבה עקבי, ווידוי עמידה בסטנדרטים הנפוצים בשוק (כגון אלה של גוגל).

פירוט על הבסיס נתונים: השתמשנו ב-MongoDB של Mongo Atlas. ה-collections הן:

* + היות והשתמשנו ב-nosql, אין קשרים ולכן לצורך נוחות לא נרמלנו את הטבלאות.



* המשך בעמוד הבא

סביבת עבודה: VS-Code עם ה-extensions שהוצגו בקורס.

שימוש ב-ChatGPT: נעזרנו בכלי זה על מנת ללמוד MongoDB, וקבלת דוגמאות לשימוש ב-Mongoose. הפרופטים היו:

* + - Mongoose, how to create schema and models
    - Mongoose, how to create singleton connection
    - Mongoose, given collection 'Account {username: String, password: String} and a connection, give example for simple CRUD

מקורות שנעשה בהן שימוש במהלך הפיתוח:

* + - <https://mongoosejs.com/docs/>
    - <https://nextjs.org/docs>
    - <https://react.dev/>
    - <https://www.typescriptlang.org/docs/>
    - <https://tailwindcss.com/>
    - המצגות של הקורס.