

# NodeJS

Express

עבודה עם Service - Code Splitting

Users Endpoints

Cards Endpoints

# חלוקה של קוד: עקרון האחריות הבודדת SRP

Middleware  
על בסיס validateSchema

בדיקות  
תקינות

auth-service

פעולות  
בנושא  
אבטחה

user-service

פעולות  
מונגו

ראוטר

בקשה/תגובה

טיפול  
בשגיאות

# תזכורת לגבי ירושה:

```
class Person{
  firstName: string;
  constructor(firstName: string){
    this.firstName = firstName;
  }
}

class Student extends Person {
  studentId: string;
  constructor(firstName: string, studentId: string) {
    super(firstName);
    this.studentId = studentId;
  }
}
```

```
class BizCardsError extends Error {
  //props
  status: number;
  //constructor
  constructor(message: string, status: number) {
    super(message);

    this.status = status;
  }
}
```

# מחלקה שיורשת מError

## משמשת אותנו להוסיף סטטוס לשגיאות שנרצה לזרוק

```
class BizCardsError extends Error {  
  //props  
  status: number;  
  //constructor  
  constructor(message: string, status: number) {  
    super(message);  
  
    this.status = status;  
  }  
}  
  
export { BizCardsError };
```

# User Service

## פעולות בנושא משתמשים

```
import { IUser } from "../@types/user";
import { User } from "../database/model/user";
import { BizCardsError } from "../error/biz-cards-error";
import { auth } from "../auth-service";
```

```
const createUser = async (userData: IUser) => {
  const user = new User(userData);
  user.password = await auth.hashPassword(user.password);
  return user.save();
};
```

```
const validateUser = async (email: string, password: string) => {
  const user = await User.findOne({ email });
```

```
  if (!user) {
    throw new BizCardsError("Bad credentials", 401);
  }
```

אם משתמש לא קיים  
נזרוק שגיאה

```
  //check the password:
```

```
  const isValid = await auth.validatePassword(password, user.password);
```

```
  if (!isValid) {
    throw new BizCardsError("Bad credentials", 401);
  }
```

אם סיסמא לא תקינה  
נזרוק שגיאה

```
  const jwt = auth.generateJWT({ email });
```

```
  return { jwt };
```

```
};
```

```
export {createUser, validateUser}
```

# router שמשתמש בService

```
router.post("/", validateRegistration, async (req, res, next) => {  
  try {  
    const saved = await createUser(req.body as IUser);  
    res.status(201).json({ message: "Saved", user: saved });  
  } catch (err) {  
    next(err);  
  }  
});
```

לא נרצה לנהל שגיאות בראוטר עצמו  
נעביר הלאה ל ErrorHandler  
next(err)

# ניהול שגיאות במקום מרכזי

## Middleware לטיפול בשגיאות

```
import { ErrorHandler } from "express";
import { BizCardsError } from "../error/biz-cards-error";

const errorHandler: ErrorHandler = (err, req, res, next) => {

  //userService Error
  if (err instanceof BizCardsError) {
    return res.status(err.status).json({ message: err.message });
  }

  //mongoose error

  //catch-all
  return res.status(500).json({ message: "Internal Server Error" });
};

export { errorHandler };
```

# ניהול שגיאות במקום מרכזי

## Middleware לטיפול בשגיאות

```
import e, { ErrorHandler } from "express";
import { BizCardsError } from "../error/biz-cards-error";

const errorHandler: ErrorHandler = (err, req, res, next) => {
  //userService Error
  if (err instanceof BizCardsError) {
    return res.status(err.status).json({ message: err.message });
  }

  //mongoose error...
  if(err.code && err.code == 11000 && err.keyPattern && err.keyValue){
    return res.status(400).json({
      message: "Duplicate Key",
      property: err.keyValue,
      index: err.keyPattern
    });
  }

  //catchall
  return res.status(500).json({ message: "Internal Server Error" });
};

export { errorHandler };
```



# ניהול שגיאות במקום מרכזי

## Middleware לטיפול בשגיאות

```
import e, { ErrorHandler } from "express";
import { BizCardsError } from "../error/biz-cards-error";

const errorHandler: ErrorHandler = (err, req, res, next) => {
  //userService Error
  if (err instanceof BizCardsError) {
    return res.status(err.status).json({ message: err.message });
  }

  //mongoose error...
  if (err.code && err.code == 11000 && err.keyPattern && err.keyValue) {
    return res.status(400).json({
      message: "Duplicate Key - Must be Unique",
      property: err.keyValue,
      index: err.keyPattern,
    });
  }

  if (err instanceof SyntaxError) {
    return res.status(400).json({ message: "Invalid Json" });
  }

  //catchall
  return res.status(500).json({ message: "Internal Server Error", err });
};

export { errorHandler };
```

# לוגין למשתמש

```
router.post("/login", validateLogin, async (req, res, next) => {  
  try {  
    //check the request:  
    const { email, password } = req.body as ILogin;  
  
    //call the service:  
    const jwt = await validateUser(email, password);  
  
    //response  
    res.json(jwt);  
  } catch (e) {  
    next(e);  
  }  
});
```


# Protected Routes

מיועד למשתמש קיים בלבד:

שיש לו JWT

והJWT צריך להיות תקין.

```
GET http://localhost:8080/api/v1/users  
Authorization: bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ...
```



בכל בקשה - ישלחו Header של Authorization  
בפורמט <JWT> bearer

# התמודדות עם JWT

```
import { RequestHandler, Request } from "express";
import { BizCardsError } from "../error/biz-cards-error";
import { auth } from "../service/auth-service";

const extractToken = (req: Request) => {
  const authHeader = req.header("Authorization"); // "bearer aslkfdjasfl2ejroi2ejwroi32jerf"

  if (
    authHeader &&
    authHeader.length > 7 &&
    authHeader.toLowerCase().startsWith("bearer ")
  ) {
    return authHeader.substring(7);
  }
  throw new BizCardsError("token is missing in Authorization header", 400);
};
```

```
const validateToken: RequestHandler = (req, res, next) => {
  const token = extractToken(req); // aslkfdjasfl2ejroi2ejwroi32jerf
  const { email } = auth.verifyJWT(token);
  //use the email...

  next();
};
```

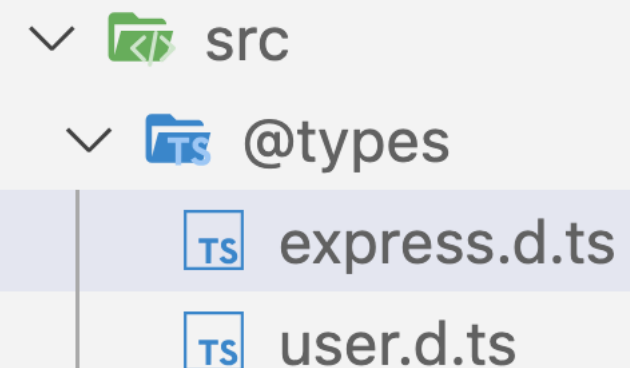
נבדוק שיש JWT  
אחרת נזרוק שגיאה

נבדוק שJWT תקין  
אחרת נזרוק שגיאה

```
export { validateToken };
```

# הוספת מפתח לטיפוס Request של הספרייה Express Typescript

```
import { Request } from "express";  
// add user to express request  
  
import { IJWTPayload } from "../user";  
  
declare global {  
  namespace Express {  
    interface Request {  
      user?: IJWTPayload;  
    }  
  }  
}
```



```
src  
└── @types  
    ├── express.d.ts  
    └── user.d.ts
```

# JWT לבדיקה של Middleware

```
import { RequestHandler, Request } from "express";
import { BizCardsError } from "../error/biz-cards-error";
import { auth } from "../service/auth-service";

const extractToken = (req: Request) => {
  const authHeader = req.header("Authorization"); // "bearer aslkfdjasfl2ejroi2ejwroi32jerf"

  if (
    authHeader &&
    authHeader.length > 7 &&
    authHeader.toLowerCase().startsWith("bearer ")
  ) {
    return authHeader.substring(7);
  }
  throw new BizCardsError("token is missing in Authorization header", 400);
};

const validateToken: RequestHandler = (req, res, next) => {
  const token = extractToken(req);

  const { email } = auth.verifyJWT(token); // בדיקה של JWT

  req.user = { email }; // הוספה של כתובת המייל לבקשה

  next();
};

export { validateToken };
```

# הרשמה והתחברות:

## הרשמה:

```
const user = new User(body)
user.password = bcrypt(password)
user.save()
```

## התחברות:

```
const {email password} = body
בודקים שהסיסמא תואמת
מנפיקים JWT
```

## כל נתיב אחר

הלקוח מזין JWT בAuthorization Header  
שמספק מידע על הלקוח

# בדיקה - האם admin

```
import { RequestHandler, Request } from "express";
import { BizCardsError } from "../error/biz-cards-error";
import { auth } from "../service/auth-service";
import { User } from "../database/model/user";

const extractToken = (req: Request) => {
  const authHeader = req.header("Authorization"); // "bearer aslkfdjasfl2ejroi2ejwroi32jerf"

  if (
    authHeader &&
    authHeader.length > 7 &&
    authHeader.toLowerCase().startsWith("bearer ")
  ) {
    return authHeader.substring(7);
  }
  throw new BizCardsError("token is missing in Authorization header", 400);
};

const isAdmin: RequestHandler = async (req, res, next) => {
  const token = extractToken(req); //
  const { email } = auth.verifyJWT(token);

  const user = await User.findOne({ email });

  const isAdmin = user?.isAdmin;
  if (isAdmin) {
    return next();
  }

  return res.status(401).json({ "message": "Must be admin" });
};

export { isAdmin };
```



# בתוך JWT

יש רק email

# שיעורי בית:

צרו בפרוייקט תיקיה בשם public

בתיקיה אפשר ליצור קבצים עם סיומת .html או .css או .js או .png

הפרוייקט יאפשר החזרה של קבצים סטטיים מהתיקיה public

אם קובץ לא קיים בתיקיה וגם הנתיב לא מוגדר ב-routes שלכם הלקוח יקבל 404  
אחרת הלקוח יקבל את הקובץ המבוקש.

בפרוייקט של שיעורי הבית הקודמים -

צרו error-handler

צרו Service

כך שהRouter יטפל רק בהחזרת תגובה ללקוח ולא בטיפול בשגיאות או בשמירה של המשתמש