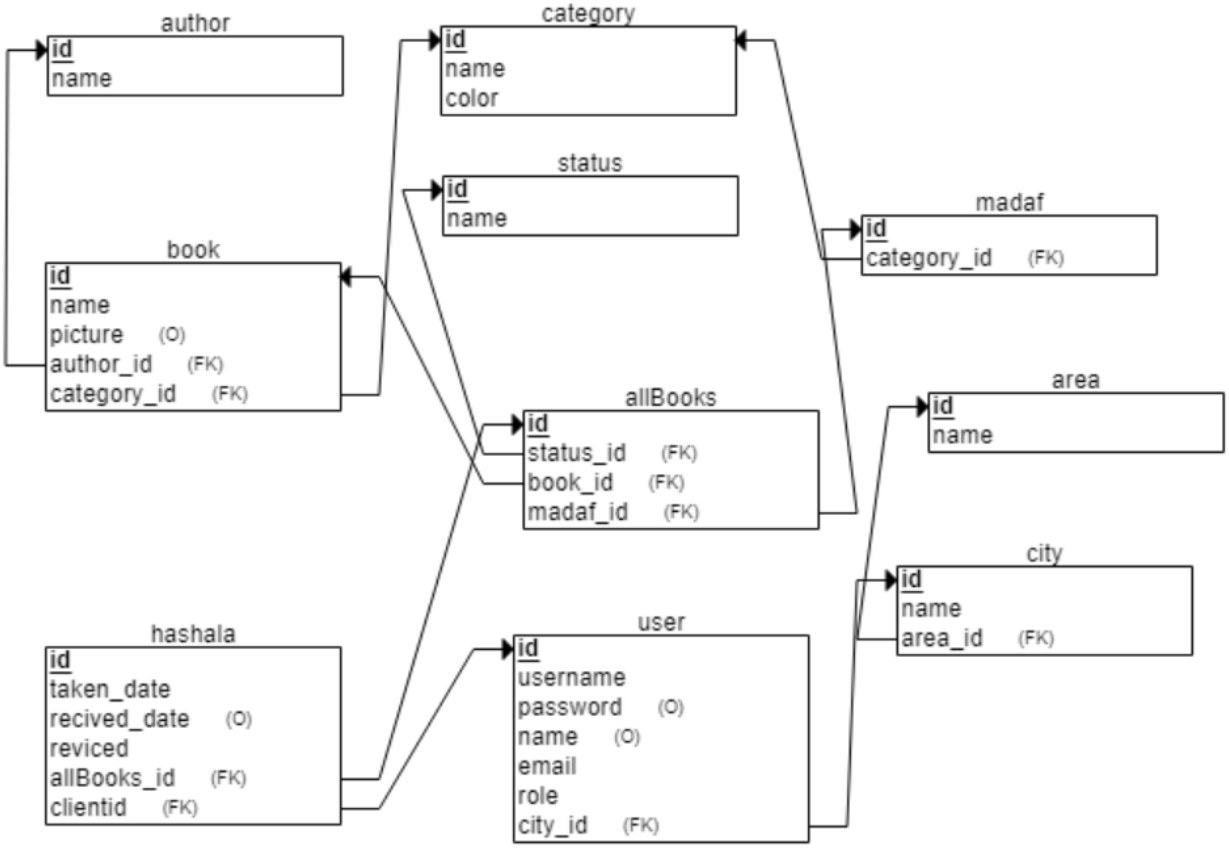
02 עבודה עם SEQUELIZE - הרחבה

בשלב זה נלמד על קשרי גומלין איך לבצע איך להגדיר, איך והיכן להגדיר את הטבלאות ועוד

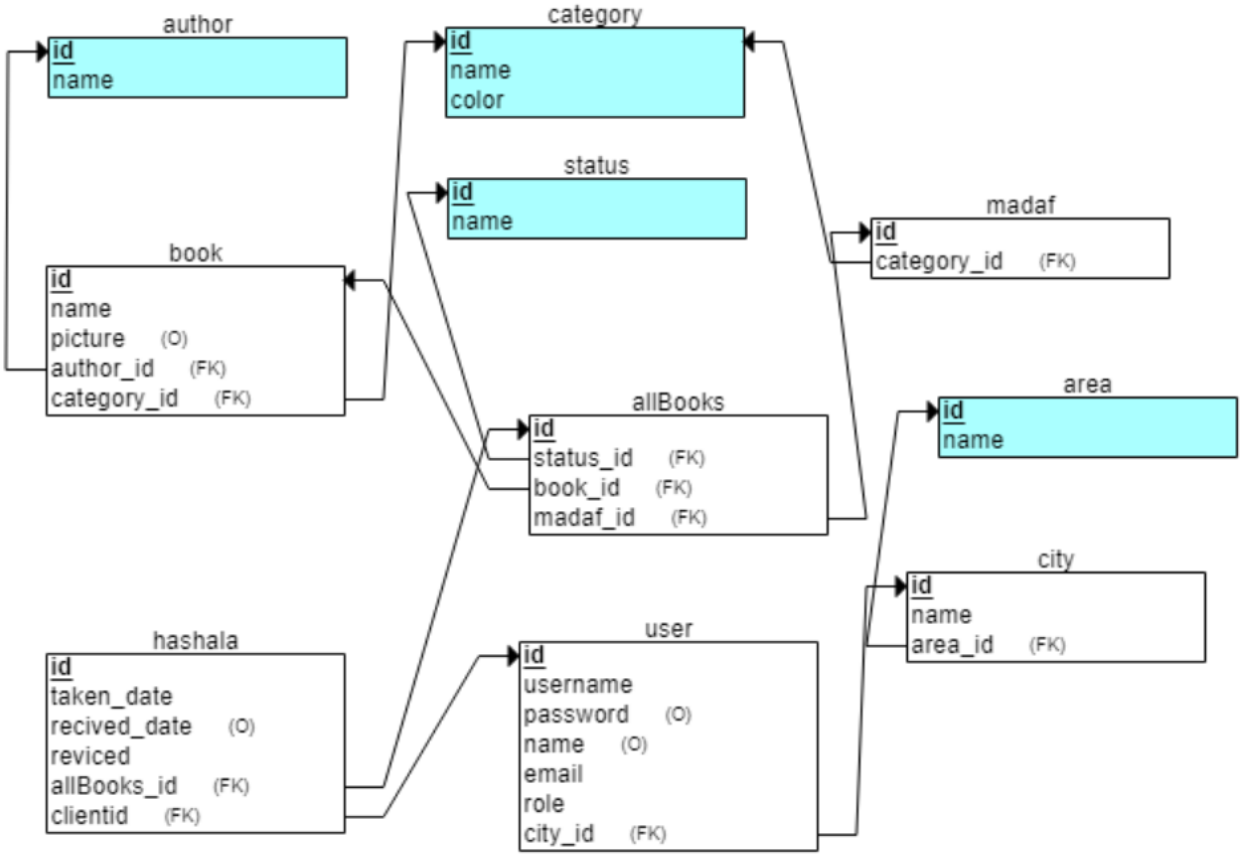
הפרוייקט אותו ננסה להטמיע זה פרוייקט ספרייה
מבנה מסד הנתונים של הפרוייקט



הטבלאות:

Author
Status
Category
Area

הן טבלאות ללא קשרים ולכן נתחיל איתן



1. יצירת מודל

נתחיל עם category
,תחת תקיית models ניצור קובץ category.js

ונתחיל עם יצירת האובייקט
זו הגדרה ראשית לאובייקט המהווה טבלה במסד הנתונים

```
module.exports = (sequelize, DataTypes) => {
    const Category = sequelize.define(

    );

    return Category;
};
```

הפונקציה define מקבלת 3 פרמטרים

1. שם הטבלה / המודל
2. מבנה השדות

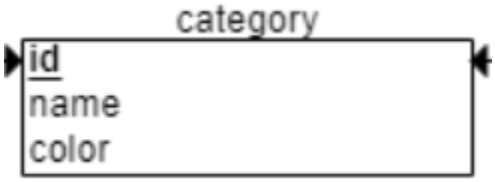
3. הגדרות נוספות

שם הטבלה / המודל

```
module.exports = (sequelize, DataTypes) => {
    const Category = sequelize.define(
        "category"

    );

    return Category;
 };
```

מבנה השדות

לטבלת קטגוריות יש 2 שדות name, color + שדה של מפתח ראשי (שכרגע לא נגדיר אותו כי הוא מוגדר אוטומטית)




```
module.exports = (sequelize, DataTypes) => {
    const Category = sequelize.define(
        "category",
        {
            name:{
                type:DataTypes.STRING
            },
            color:{
                type:DataTypes.STRING
            }
        }

    );

    return Category;
 };
```

קיימים הרבה סוגים, הסוגים הדיפולטיים הם:

DataType	What it Means in MySQL
STRING	VARCHAR(255)
TEXT	TEXT
BOOLEAN	TINYINT(1)
INTEGER	INTEGER
FLOAT	FLOAT
STRING(1234)	VARCHAR(1234)
DATE	DATE

מאפייני שדה

allowNull - האם לאפשר ריק
defaultValue - ערך ברירת מחדל
unique: - מתאים לשם משתמש - השדה ללא כפילויות
primaryKey - מפתח ראשי
autoIncrement - מספור רץ

```
            name:{
                type:DataTypes.STRING,
                allowNull: false,
                unique:true,
            },
            color:{
                type:DataTypes.STRING,
                defaultValue:'red'
            }
```

2 עדכון המודל שיצרנו באובייקט מסד הנתונים

```
db.categories  = require('./category')(sequelize, DataTypes)
```

כך נראה הקובץ

```
EXPLORER
SERVER
> ex1
v ex2
  > config
  > controllers
  > migrations
  v models
    JS category.js
    JS index.js
    JS note.js
  > node_modules
  > public
  > routes
  > seeders
  > views
    .env
    .gitignore
    package-lock.json
    package.json
    JS server.js
```

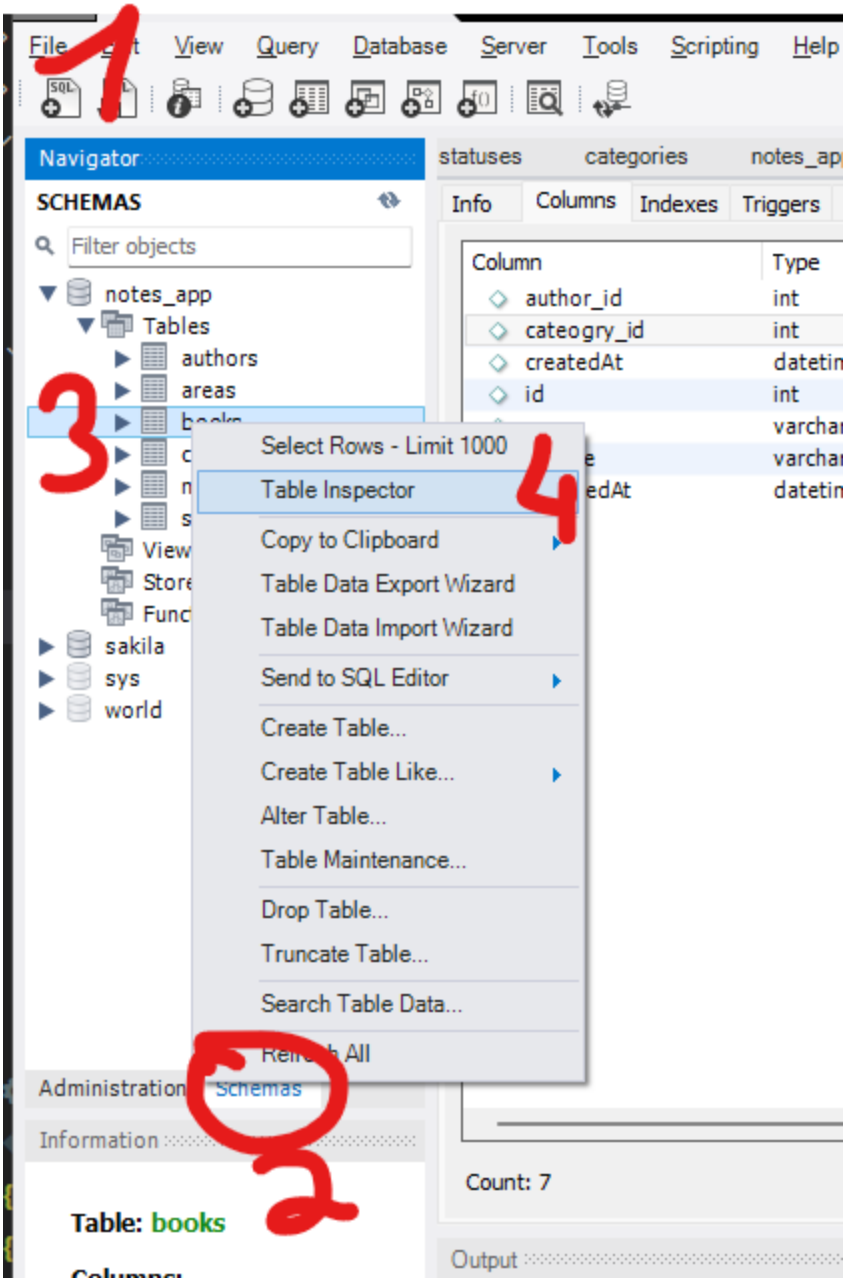
JS index.js × | JS category.js | JS note.js

ex2 > models > JS index.js > ...

```
1   const dbConfig = require('../config/dbConfig');
2   const {Sequelize, DataTypes} = require('sequelize');
3
4   const sequelize = new Sequelize(
5       dbConfig.DB,
6       dbConfig.USER,
7       dbConfig.PASSWORD, {···
19      }
20  )
21
22  sequelize.authenticate().then(() => {
23    console.log('Connection has been established successfully.');
24  }).catch((error) => {
25    console.error('Unable to connect to the database: ', error);
26  });
27
28
29  const db = {}
30
31  db.Sequelize = Sequelize
32  db.sequelize = sequelize
33  //*******************MODELS************************//
34  db.notes  = require('./note')(sequelize, DataTypes)
35  db.categories  = require('./category')(sequelize, DataTypes)
36
37
38  //*******************END MODELS********************//
39
40  db.sequelize.sync({ force: false })
41  .then(() => {
42      console.log('yes re-sync done!')
43  })
44  module.exports = db
```

נריץ עכשיו את השרת

כעת שנבדוק במסד הנתונים תחת SCHEMA נראה שהמערכת יצרה לנו את הטבלה אוטומטית.



אם נסתכל על מבנה הטבלאות נבחין שהמערכת הוסיפה לנו 3 שדות

s notes_app.books notes_app.books notes_app.books notes_app.books notes_ap

Info | Columns | Indexes | Triggers | Foreign keys | Partitions | Grants | DDL

Column	Type	Default Value	Nullable	Character Set	Co
color	varchar(255)	red	YES	utf8mb4	utf
createdAt	datetime		NO		
id	int		NO		
name	varchar(255)		NO	utf8mb4	utf
updatedAt	datetime		NO		

- מפתח id
- תאריך יצירה createdAt
- תאריך עדכון updatedAt

ובנוסף נראה שהמערכת קראה לטבלה בשם categories למרות שאנו הגדרנו את השם category

בברירת מחדל המערכת שומרת את שמות הטבלאות ברבים, במידה ואנו רוצים להישאר עם השם של הטבלה שלנו

הגדרות נוספות

על מנת להגדיר למערכת להשתמש בשם שהגדרנו ולא לשנות לרבים, כאן אנו מגיעים להגדרות נוספות שזה **הפרמטר השלישי של הפונקציה**

freezeTableName - האם לקרוא לטבלה בשם שהוגדר
timestamps - האם להוסיף לטבלה נוצר בתאריך עודכן בתאריך

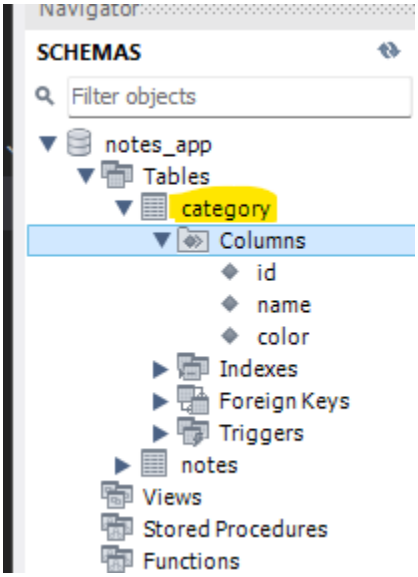
```
module.exports = (sequelize, DataTypes) => {
    const Category = sequelize.define(
        "category",
        {
            name:{
                type:DataTypes.STRING,
                allowNull: false,
            },
            color:{
                type:DataTypes.STRING,
                defaultValue: 'red'
            }
        },
        {
            freezeTableName:true,
            timestamps: false,
```

```
        }

    );

    return Category;
};
```

More Sequelize Options

Option	Meaning
timestamps: false	Don't add the updatedAt and createdAt attributes.
freezeTableName: true	Disable the modification of table names.
tableName: 'my_custom_name'	Define the table's name.
version: true	Sequelize adds a version count attribute to the model and and throws an OptimisticLockingError when stale instances are saved.
paranoid: true	Don't delete database entries but set the attribute deletedAt to when the deletion was done. Only works if timestamps are enabled.

נמחק את הטבלה על ידי הMYSQL ונריץ שוב את המערכת שלנו
ונוספה הטבלה בשם שהוגדר (לא ברבים) ולא קיימים האפשרויות של הוספת עדכון תאריך



הגדרת מפתח ידנית

במידה ורוצים להגדיר מפתח בשם אחר מברירת המחדל (ברירת המחדל היא id)

```
        category_id:{
            type: DataTypes.INTEGER,
            autoIncrement: true,
            primaryKey: true
        },
```

נוסיף את השדה למודל ונריץ
לא נוסף שדה המפתח שיצרנו
וזה בגלל שהמערכת יוצרת את הטבלה רק אם היא לא קיימת בברירת מחדל

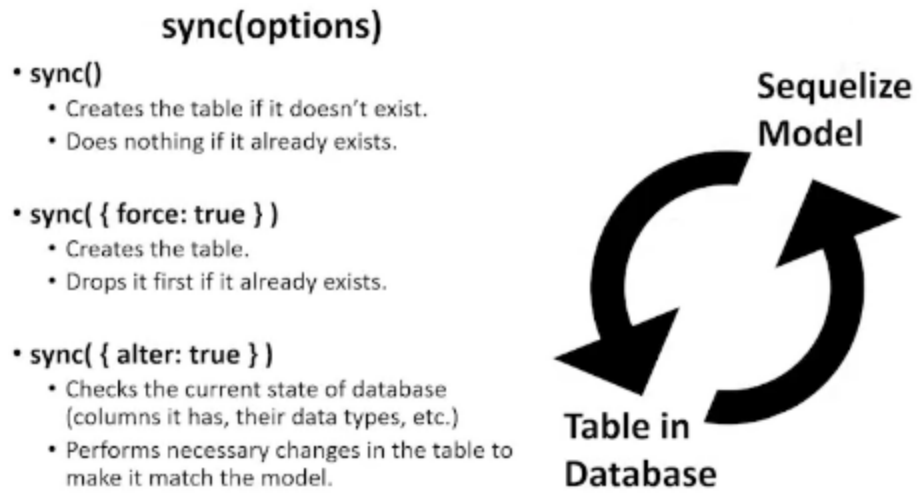
סנכרון מסד הנתונים

וכאן אנו עוברים לנושא של SYNC

ברירת מחדל המערכת יוצרת את הטבלה אם היא לא קיימת.
בקובץ index.js תחת models בו קיימת ההגדרה של החיבור למסד הנתונים

```
//*******************MODELS************************//
db.notes  = require('./note')(sequelize, DataTypes)
db.categories  = require('./category')(sequelize, DataTypes)


//*******************END MODELS********************//

db.sequelize.sync({ force: false })
.then(() => {
    console.log('yes re-sync done!')
})
module.exports = db
```

הגדרות סנכרון



1. יצירת טבלה רק אם היא לא קיימת ברירת מחדל או ההגדרה **force:false**
2. יצירת הטבלה מחדש ואם היא קיימת מוחקים אותה **force:true**
3. בדיקה את השדות ועדכון מסד הנתונים בהתאמה במידה ויש שינויים **alter:true**

נעדכן אצלינו

```
db.sequelize.sync({ alter: true })
.then(() => {
    console.log('yes re-sync done!')
})
```

ספציפית בהגדרת המפתח זה לא יעבוד בגלל שאין אפשרות להוסיף 2 מפתחות עם מספר רץ - אבל בגדול זה יעדכן כל שדה שנוסיף במהלך הפיתוח

נקל על הפיתוח באמצעות intellisense

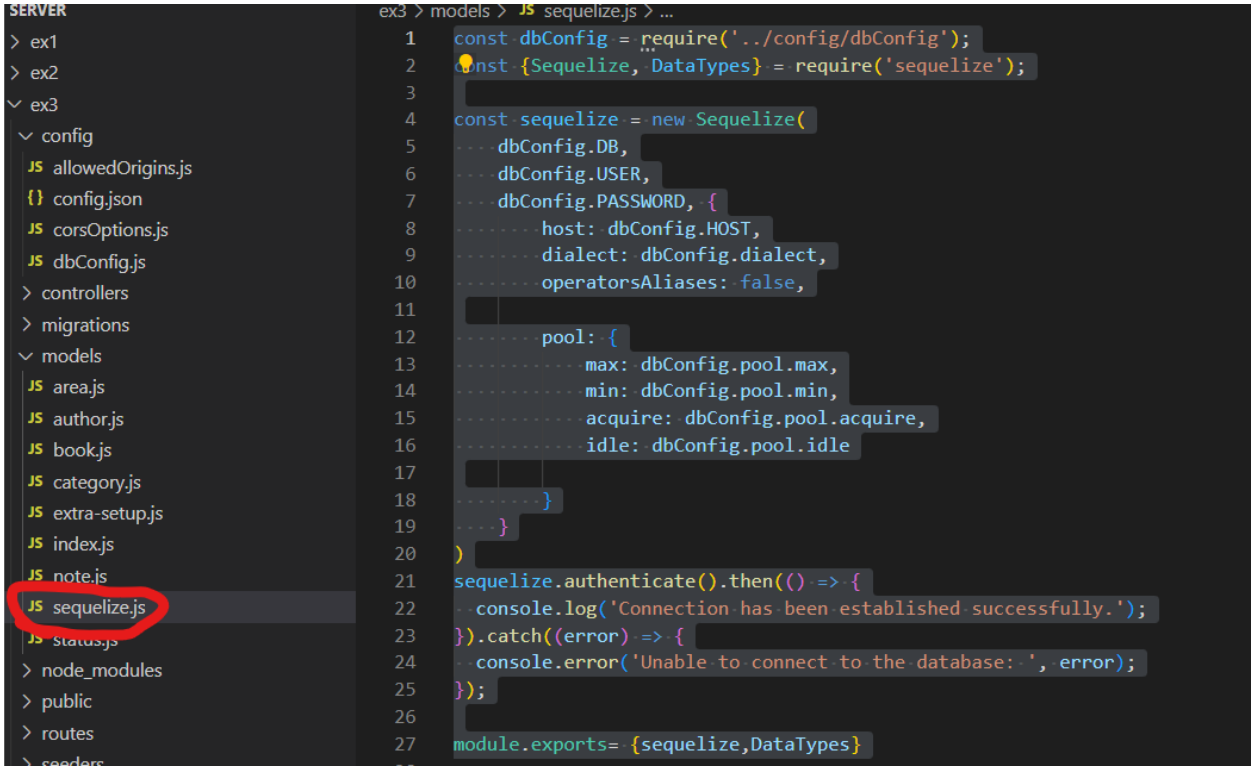
כרגע בפיתוח של המודל אין לנו את העזרה בקוד, המערכת לא נותנת את האפשרויות הרלוונטיות

```
JS index.js    JS category.js 1 ●    JS note.js
ex2 > models > JS category.js > <unknown> > exports > Category > name > type
1
2   module.exports = (sequelize, DataTypes) => {
3       const Category = sequelize.define(
4           "category",
5           {
6               name:{
7                   type:DataTypes.,
8                   allowNull: fals
9               },
10              color:{
11                  type:DataTypes.
12                  defaultValue: '
13              }
14          },
15
16      );
17
18      return Category;
19  };
20
```

```
abc Category
abc DataTypes
abc STRING
abc allowNull
abc color
abc defaultValue
abc define
abc exports
abc module
abc name
abc sequelize
abc type
```

נבצע עדכונים קלים

1. ניצור קובץ שיגדיר לנו את הקישור למסד הנתונים כקובץ נפרד נקרא לו בשם sequelize.js, ובמקום שיהא בקובץ הINDEX נעביר אותו לקובץ נפרד




```
const dbConfig = require('../config/dbConfig');
const {Sequelize, DataTypes} = require('sequelize');

const sequelize = new Sequelize(
    dbConfig.DB,
    dbConfig.USER,
    dbConfig.PASSWORD, {
        host: dbConfig.HOST,
        dialect: dbConfig.dialect,
        operatorsAliases: false,

        pool: {
            max: dbConfig.pool.max,
            min: dbConfig.pool.min,
            acquire: dbConfig.pool.acquire,
            idle: dbConfig.pool.idle

        }
    }
)
sequelize.authenticate().then(() => {
  console.log('Connection has been established successfully.');
```

```
}).catch((error) => {
  console.error('Unable to connect to the database: ', error);
});

module.exports= {sequelize,DataTypes}
```

נייבא אותו בקובץ הINDEX - ונוריד את יצירת הקוד הקודמת ונעדכן את קריאת המודלים ללא פונקציה

```
const {Sequelize} = require('sequelize');
const {sequelize} = require('./sequelize');


const db = {}

db.Sequelize = Sequelize
db.sequelize = sequelize
//*******************MODELS*************************//
db.notes  = require('./note')
db.categories  = require('./category')



//*******************END MODELS*********************//


db.sequelize.sync({ alter: true })
.then(() => {
    console.log('yes re-sync done!')
})
module.exports = db
```

```
const {Sequelize} = require('sequelize');
const {sequelize} = require('./sequelize');


const db = {}
```

```
db.Sequelize = Sequelize
db.sequelize = sequelize
//*******************MODELS************************//
db.notes  = require('./note')
db.categories  = require('./category')
//*******************END MODELS********************//

db.sequelize.sync({ alter: true })
.then(() => {
    console.log('yes re-sync done!')
})
module.exports = db
```

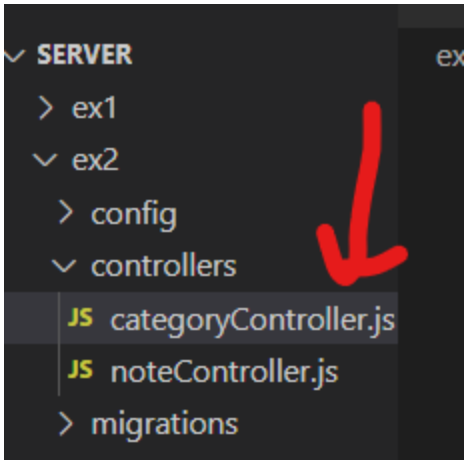
גם בקובץ המודל עצמו נייבא את אובייקט במקום כפמרטר לפונקציה ונייצא את המודל במקום את הפונצקיה

```
const { sequelize, DataTypes } = require("./sequelize");
const Category = sequelize.define("category", {
  name: {
    type: DataTypes.STRING,
    allowNull: false,
    unique: true,
  },
  color: {
    type: DataTypes.STRING,
    defaultValue: "red",
  },
});
module.exports = Category;
```

3. הוספת קונטרולר

categoryController.js

נוסיף קונטרולר של פעולות (קבלת כל הרשומות, הוספה, עדכון, מחיקה, קבלה באמצעות ID)



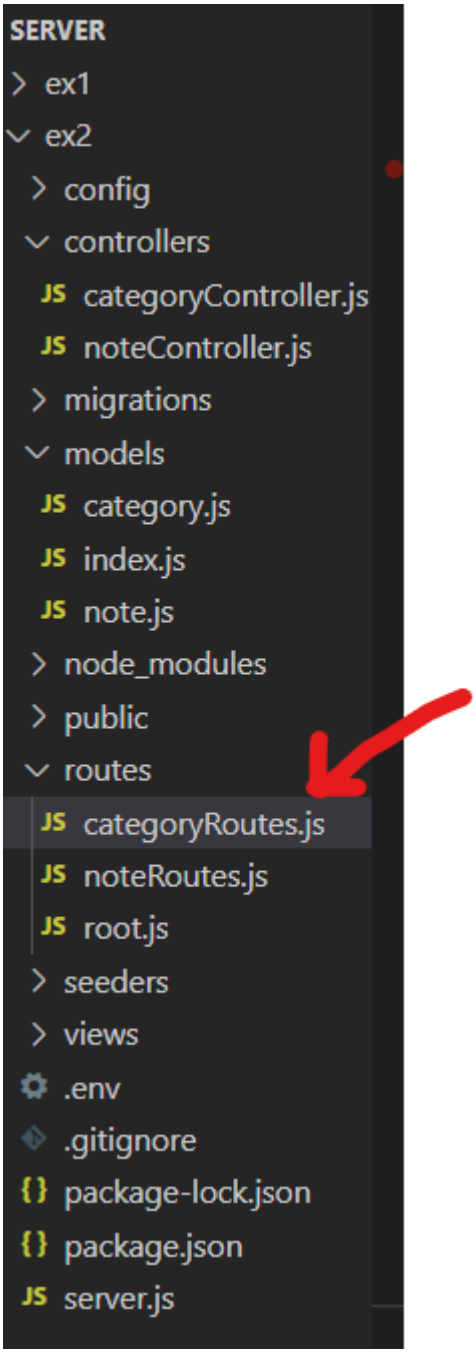

```
const db = require('../models/index')
const Category = db.categories


const getAll = async (req, res) => {
    const categories = await Category.findAll({})
    if (!categories?.length) {
        return res.status(400).json({ message: 'No categories found' })
    }
    res.json(categories)
}
const getOne = async (req, res) => {
    const id = req.params.id
    const category = await Category.findOne({where:{id:id}})
    res.json(category)
}

const create = async (req, res) => {
    const { name, color } = req.body
    // Confirm data
    if (!name) {
        return res.status(400).json({ message: 'All fields are required'
})
    }
    const category = await Category.create({ name, color })

    if (category) { // Created
        return res.status(201).json({ message: 'New category created' })
    } else {
```

```
        return res.status(400).json({ message: 'Invalid category data
received' })
    }

}
const update = async (req, res) => {
    const {id, name, color } = req.body
    // Confirm data
    if (!id ||  !name) {
        return res.status(400).json({ message: 'All fields are required'
})
    }
    const category = await Category.update({ name, color},{where:{id:id}})

    if (!category) {
        return res.status(400).json({ message: 'category not found' })
    }
    res.json(category)
}
const deleteOne = async (req, res) => {
    const { id } = req.body

    // Confirm data
    if (!id) {
        return res.status(400).json({ message: 'category ID required' })
    }
    await Category.destroy({
        where: {
          id: id
        }
    });
    res.json( `Category  with ID ${id} deleted`)
}

module.exports = {
    getAll,
    getOne,
    create,
    update,
    deleteOne
```

```
}
```

4. הוספת RUTES

נוסיף categoryRoutes



\

```
const express = require('express')
```

```
const router = express.Router()
const categoryController = require("../controllers/categoryController")

router.route('/')
    .get(categoryController.getAll)
    .post(categoryController.create)
    .patch(categoryController.update)
    .delete(categoryController.deleteOne)
router.get('/:id', categoryController.getOne)
module.exports = router
```

5. עדכון ROUTES במערכת

ובקובץ הserver הראשי נוסיף הפנייה לROTES שיצרנו

```
app.use("/api/categories", require("./routes/categoryRoutes"));
```

6. בדיקה POSTMAN

נבדוק את הROUTES עבור קטגוריות באמצעות POSTMAN

POST http://localhost:3600/api/categories

Params Authorization Headers (10) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{
    "name":"נוער ספרי",
    "color":"ירוק"
}
```

Body Cookies (1) Headers (9) Test Results

Pretty Raw Preview Visualize JSON

```
{
    "message": "New category created"
}
```

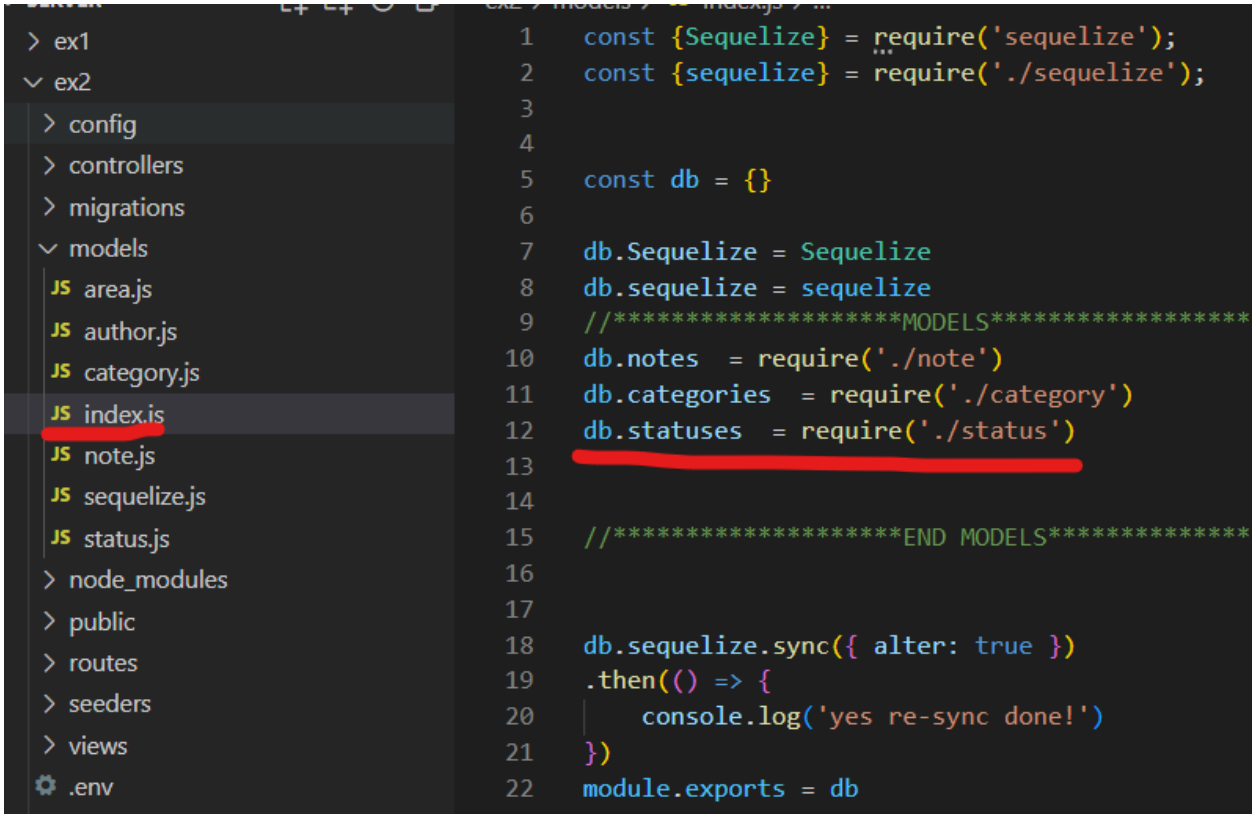
נחזור על שלבים 1-6 עבור מחברים סטסוס ואיזורים

1. יצירת המודל

```
ex1
ex2
  config
  controllers
  migrations
  models
    area.js
    author.js
    category.js
    index.js
    note.js
    sequelize.js
    status.js
```

```
const { sequelize, DataTypes } = require("./sequelize");
const Status = sequelize.define("status", {
  name: {
    type: DataTypes.STRING,
    allowNull: false,
    unique: true,
  }
});
module.exports = Status;
```

2. עדכון אובייקט מסד הנתונים



3. הוספת קונטרולר

SERVER
- ex1
- ex2
  - config
  - controllers
    - categoryController.js
    - noteController.js
    - statusController.js
  - migrations
  - models
    - category.js
    - index.js
    - note.js
    - status.js
  - node_modules
  - public
  - routes
    - categoryRoutes.js
    - noteRoutes.js
    - root.js
  - seeders
  - views
  - .env
  - .gitignore
  - package-lock.json
  - package.json
  - server.js

OUTLINE

TIMELINE

ex2 > controllers > statusController.js > <unknown> > deleteOne

```javascript
const db = require('../models/index')
const Status = db.statuses
const getAll = async (req, res) => {
    const statuses = await Status.findAll({})
    if (!statuses?.length) {
        return res.status(400).json({ message: 'No statuses found' })
    }
    res.json(statuses)
}
const getOne = async (req, res) => {
    const id = req.params.id
    const status = await Status.findOne({where:{id:id}})
    res.json(status)
}
const create = async (req, res) => {
    const { name } = req.body
    if (!name) {// Confirm data
        return res.status(400).json({ message: 'All fields are required' })
    }
    const status = await Status.create({ name })
    if (status) { // Created
        return res.status(201).json({ message: 'New status created' })
    } else {
        return res.status(400).json({ message: 'Invalid status data received' })
    }
}
const update = async (req, res) => {
    const {id, name } = req.body
    if (!id ||  !name) { // Confirm data
        return res.status(400).json({ message: 'All fields are required' })
    }
    const status = await Status.update({ name},{where:{id:id}})
    if (!status) {
        return res.status(400).json({ message: 'status not found' })
    }
    res.json(status)
}
const deleteOne = async (req, res) => {
    const { id } = req.body
    if (!id) {// Confirm data
        return res.status(400).json({ message: 'status ID required' })
    }
    await Status.destroy({ where: {id: id}});
    res.json( `Status  with ID ${id} deleted`)
}

module.exports = {getAll, getOne,create,update, deleteOne}
```

4. הוספת ROUTES

```
SERVER
> ex1
v ex2
  > config
  > controllers
  > migrations
  > models
  > node_modules
  > public
  v routes
    JS categoryRoutes.js
    JS noteRoutes.js
    JS root.js
    JS statusRoutes.js
  > seeders
```

ex2 > routes > JS statusRoutes.js > ...

```
1  const express = require('express')
2  const router = express.Router()
3  const statusController = require("../controllers/statusController")
4
5  router.route('/')
6      .get(statusController.getAll)
7      .post(statusController.create)
8      .patch(statusController.update)
9      .delete(statusController.deleteOne)
10 router.get('/:id', statusController.getOne)
11 module.exports = router
12
```

5. עדכון ROUTES במערכת

```
SERVER
> ex1
v ex2
  > config
  > controllers
  > migrations
  > models
  > node_modules
  > public
  v routes
    JS categoryRoutes.js
    JS noteRoutes.js
    JS root.js
    JS statusRoutes.js
  > seeders
  > views
    .env
    .gitignore
    package-lock.json
    package.json
    JS server.js
```

ex2 > JS server.js > ...

```
1  require('dotenv').config()
2  const express = require('express')
3  const app = express()
4  const path = require('path')
5  const cookieParser = require('cookie-parser')
6  const cors = require('cors')
7  const corsOptions = require('./config/corsOptions')
8  const PORT = process.env.PORT || 3600
9
10 //middleware
11 app.use(cors(corsOptions))
12 app.use(express.json())
13 app.use(cookieParser())
14 //routes
15 app.use('/', express.static(path.join(__dirname,'public')))
16 app.use('/', require('./routes/root'))
17
18 app.use("/api/notes", require("./routes/noteRoutes"));
19 app.use("/api/categories", require("./routes/categoryRoutes"));
20 app.use("/api/statuses", require("./routes/statusRoutes"));
21
22
23
24 app.all('*', (req, res) => {
25     res.status(404)
26     if (req.accepts('html')) {
27         res.sendFile(path.join(__dirname, 'views', '404.html'))
28     } else if (req.accepts('json')) {
29         res.json({ message: '404 Not Found' })
30     } else {
31         res.type('txt').send('404 Not Found')
32     }
33 })
34 app.listen(PORT, ()=> console.log(`Server running on port ${PORT}`))
```

6. בדיקה POSTMAN

geshtop-exercise

node app

- POST add note
- POST add Category
- POST add Status
- DEL delete note
- DEL delete category
- PATCH update note
- PATCH update category
- GET get all notes
- GET get all categories
- GET get one note

node app / **add Status**

POST http://localhost:3600/api/statuses

Params | Authorization | Headers (10) | Body | Pre-request Script | Tests | Settin

none | form-data | x-www-form-urlencoded | raw | binary | GraphQL | JS

```
1 {
2     "name" :"במדף"
3 }
```

Body | Cookies (1) | Headers (9) | Test Results

Pretty | Raw | Preview | Visualize | JSON

```
1 {
2     "message": "New status created"
3 }
```