# תרגיל 5

הרפו ריק, יש להוסיף קובץ README.md ואת הקבצים לפי הנחיות המצגת להכנת פרויקט Express. אחר כך יש להוסיף את הספריות הבאות: sequelize sqlite3 connect-session-sequelize express-session cookie

- **נביאים**:
- https://classroom.github.com/a/KnqVbps7

  תאריך הגשה: 29/02/2024 23:59
- **שטראוס נשים**:
- https://classroom.github.com/a/Pl2hbNhO

  תאריך הגשה: 02/03/2024 23:59
- **שטראוס גברים**:
- https://classroom.github.com/a/bnmpyCmG

  תאריך הגשה: 04/03/2024 23:59

## Goal

Build a simple Yad2-like website. Anyone can post new ads and see them. An admin user can edit and approve ads.

### Models

An ad is a Model of our app, it includes:

1. a title
2. a long description
3. a price
4. a phone number
5. an email

Sequelize automatically adds:

6. an id (primary key)
7. a date of insertion
8. a date of modification

Title, price and email are mandatory. Title length limited to 20 characters, price is a number >= 0, phone number in format XXX-XXXXXXX (for example 02-1231212 or 055-1231212), long description limited to 200 characters, email as commonly defined with 2 parts and a "@" character in between (see https://en.wikipedia.org/wiki/Email_address ).

You will also define a User model including login and password fields, and initialize the app in your code with 2 sample users: admin (password: admin), admin2 (password: admin2).

You may of course add other models, add more fields if needed in the models, as long as you don't duplicate data inefficiently.

## Landing page:

1. Landing page means default page for the website http://localhost:3000 (route “/”)
2. Displays all approved ads, most recent first.
3. Contains an “admin area” link/button that leads to a “login” page.
4. Contains an “new ad” link/button that leads to a “Post new ad” page.
5. *Bonus (+4): contains a search form to filter ads and show only ads that contain a specific string in anywhere in the title*.
6. This page can be implemented either as a SPA page or a regular EJS page.

## Post new ad page

1. Displays a form where anyone can post a new ad.
2. Displays a link to go to the landing page.
3. After successful post, the user sees a confirmation message (such as “your ad was successfully posted and is waiting for approval”). The message can be a new page with a link to go to landing page, or you could directly display the landing page with the message.
4. The ad will not be visible in the landing page until admin approves it.
5. We will use cookies to remember the user after they post a new ad. Next time the same user posts another ad, we will display a “welcome back <email>, your previous ad was posted on <date>” where we display the email of the user and the date of last ad sending.
6. This page must be implemented as a EJS page (regular form sending)

## Login Page:

1. The form submission checks if credentials are valid and if so, it redirects to the admin page.
2. If credentials are wrong, it shows the login page with a relevant error message.
3. This page must be implemented as an EJS page

## Admin page:

1. This page is available only for the admin user.
2. Displays a logout button
3. It displays all ads in the database, allows the user to approve or delete an ad.
4. Must be implemented as a SPA page (based on a REST API to handle ads in the database)

# Implementation details

Landing page: if you implement a SPA page (as opposed to a server side EJS page), you will need to develop REST API endpoints to get the list of ads. This endpoint should not reveal unapproved ads.

Admin page: you will develop a REST API to handle ads in the admin page. The endpoint reading ads should return all ads (including not approved). Note that this API requires

permission: it is available only for an admin (logged in) user. Otherwise, it should return an error.

Validation must be done at the server side and optionally at the client side. Client-side validation saves the hassle of handling the display of specific error messages (for example “please enter a valid email” just next to the email input). Nevertheless, server side cannot rely on client and must always check for valid data, upon error you must return appropriate error responses to the client (you don’t always need to produce full error report inside the html, in some cases it is enough to display a generic error. If you have doubts, ask us). A simple way to check the server side is to disable client-side validation. The least expected is that the website shows some error message. There should be a different server message for validation errors or network/system errors.

REST API: should follow the guidelines (consistency, naming, endpoints, type of responses). As usual, client code should display a spinner while loading the response.

Express: should follow the middleware and MVC architecture (separating models, controllers and views).

**הצעת תכנית עבודה:**

1. לפתח דך הכנסת הודעה כדי שיהיה תוכן – אפשר לטפל בנושא הקוקיז בסוף.
2. לפתח את דף ה-login (שימוש ב-session) ואחריlogin מוצלח להציג דף סמלי (בהמשך את דף ה-admin)
3. לפתח REST API לניהול הודעות
4. לפתח את מסך ה-admin מבוסס על REST API (אפשר לטפל בנושא הרשאות בהמשך). חלק זה דומה מאוד לתרגיל 4 (בעיקר קוד צד לקוח)

**הערות כלליות:**

יש להגיש את כל הפרויקט ללא תיקייהnode_modules . שימו לב לא לשכוח להכניס את הקובץ package.json או תמונות, css. אין צורך להכניס את הקובץ sqlite

יש גישות שונות לאתר מסוג זה, ואין ספק שזה אתר חלקי ואפשר להתמקד בדברים אחרים, לכן מותר להציע שינוים בדרישות, פשוט תכתבו לי ואם תקבלו אישור ממני, תודיעו לבודק.

התרגיל משלב את כל החומר הנלמד ובכל זאת, לא מדובר בתרגיל גדול אבל כן מורכב בארכיטקטורה. רצוי קודם להבין לעומק את הקוד לדוגמא באתר. אנחנו נחזור על כל החומר בשבועיים אחרונים לסמסטר ונדריך אתכם על מנת לסיים את התרגיל בזמן. מאוד מומלץ להיות נוכח בשיעורים. הניסיון מוכיח שעבודה עצמית במקביל לקורס יכולה להאריך את הזמן ולסבך אתכם בצורה משמעותית.

בהצלחה!

# Short Guide: How to Initialize your Repo

1. Open the GitHub repo in Webstorm, you have an empty project, add a README.md file.
2. Open the terminal and type
   ```
   npm install -g express-generator
   express -e
   ```
   At this point you have an express project that can run.
   If you have trouble with this step, create another express/ejs project from WebStorm menu as shown in class, and drag all the files into your empty project.
3. Optionally add a run configuration as shown in class, or simply type in terminal:
   ```
   npm start
   ```
   Open your browser on http://localhost:3000 and make sure you see a welcome page
4. Now you want to add the required libraries (you might need more libraries later)
   ```
   npm install  cookie express-session connect-session-sequelize
   npm install sqlite3 sequelize
   ```
   If you have errors (ERR), look for solution in the sequelize slides.
5. Prepare the models and config folder:
   Create a models folder
   Copy into that folder the file models/index.js from the 12-express-lite code example
   Copy the config folder from 12-express-lite code example
6. In your app.js you will add the following code to initialize the database (it may vary upon your model definition)

```
////////////////////////////////////////////////////////////

const db = require('./models/index');
// create the tables if don't exist
db.sequelize.sync()
    .then(() => {
        console.log('Database Synced');
        return Promise.all([
            db.User.findOrCreate({
                where: {login: 'admin'},
                defaults: {login: 'admin', password: 'admin'}
            }),
            db.User.findOrCreate({
                where: {login: 'admin2'},
                defaults: {login: 'admin2', password: 'admin2'}
            })
        ]);
}).then(() => {
    console.log('Admin user created');
}).catch((err) => {
    console.log('Error syncing database or creating admin users');
    console.log(err);
});
```

7. You're all set!
   you are of course expected to write your own routes and views, therefore you should delete all routes and views from the generated express template. Please do not leave irrelevant code in your project!