

תרגיל 5

נביאים: <https://classroom.github.com/a/4Rg9eMR5>

שטראוס: <https://classroom.github.com/a/qkErr9Qk>

תאריך הגשה: 29/01/2026

כחלק מבדיקת התרגיל, לוי"ז יפורסם לפגישה במכללה – למרות שלא סיימתם לעבוד על התרגיל. חובה לכל הצוות להגיע בשבוע האחרון באחד הימים (שני/שלישי 26-27/01). מטרת הפגישה הינה לוודא שאתם/שולטים(ות) בחומר שהוגש או יוגש.

מסמך זה כולל הרבה הנחיות ותשובות לשאלות רבות (מניסיון של שנים קודמות), מומלץ לעבור עליו תוך כדי התקדמות בעבודה.

מטרת התרגיל היא לבנות אתר מסוג chatroom הכולל גם צד שרת וגם צד לקוח בעזרת הטכנולוגיות שנלמדו: javascript, html, css, express-ejs (בשילוב עם מוד נתונים mariadb) (mysql). ראו באתר הנחיות להקמת המסד נתונים ב-docker. דאגו שלא לשנות את ה-credentials ושם המסד נתונים.

קודם נתמקד בחלק א': **רישום משתמשים**. (דף SPA שניתן כבר לממש עם החומר הנלמד) הכינו את הפרויקט לפני ההנחיות בסוף המסמך. עד שנלמד על מסדי נתונים אפשר לבנות דף SPA (html) של הרישום כמו שעשיתם עד עכשיו. דף זה יכול לשבת בתיקייה public או ב-views (ואז כותבים route שעושה render לאותו view). לגבי ה-backend תסתכלו על הדוגמא של ה-rest api. בהמשך תוכלו לשפר את מבנה הפרויקט עם המסד נתונים והפרדה מלאה routes/controllers.

יש להגיש ב-15/01 את חלק א' (אנחנו פשוט נוריד את כל ה-repos בתאריך זה בדיוק בשעה 23:59 – אחר כך תמשיכו לעשות push עד ההגשה סופית). הגשת חלק א' לא חייבת להיות bug-free. המטרה היא לראות התקדמות כלומר מימוש הרישום עם טיפול בקודיז. הגשה זו תחשב כ-20% מציון התרגיל. כלומר מי שלא השלים לרוב את חלק א', הציון של התרגיל לא יעלה על 80.

חלק א': רישום משתמשים

דף נחיתה ("/) של האתר יציג טופס login וקישור לדף רישום. דף הרישום לאתר יכלול קישור לדף login וטופס הרשמה המאפשר הזנה של:

- דואר אלקטרוני
- שם פרטי
- שם משפחה

לאחר הגשת הטופס (submit), צד השרת בודק אם הדואר האלקטרוני כבר בשימוש (הוא מנהל את רשימת המשתמשים הרשומים באתר). אם הדואר לא כבר בשימוש (קיים משתמש שכבר נרשם עם כתובת זו), יש להציג הודעת שגיאה (כגון "this email is already in use, please choose another one") בתוך הדף עצמו.

ולידציה צד לקוח :

- ניתן לנצל את html כדי לממש וולידציה צד לקוח (required, input type etc)
- קלט שאינו ריק, אותיות a-z בלבד בשם ושם משפחה, דוא"ל תקף (ניתן להשתמש באימות מובנה של HTML5), אורך מינימלי 3 תווים לשם ואורך מקסימלי של 32 אותיות בכל השדות (כולל סיסמא)
- אין חשיבות לאותיות גדולות/קטנות (case insensitive) עבור כל קלט. כלומר a@b.c שקול ל-A@B.C
- חלק זה כולו בנוי כ-rest api (כלומר כל חלק זה מומש כ-express views אחד בלבד)

אחרי הזנת השם והדואר האלקטרוני, עוברים בעזרת כפתור לדף שלב הבא להמשך רישום. דף זה מכיל טופס בעזרתו המשתמש יוכל לבחור סיסמא. המשתמש מזין סיסמא פעמיים ("input type="password 2") ולוחץ שוב על כפתור submit כדי לשלוח את הסיסמא לשרת ולסיים את ההרשמה. בסיום ההרשמה חוזרים לדף login עם הודעה מתאימה ("you are now registered"). בדף הזנת הסיסמא קיים גם כפתור back שמחזיר לדף הזנת השם. קלט המשתמש ישמר יוצג בטופס לטווח זמן (ראו הסברים בהמשך).

מטבע הדברים משתמשים רבים יכולים להירשם בו זמנית. נדרש לבצע את בדיקת קיום הדוא"ל לפני כל הוספה ממשית למסד/מבנה נתונים של השרת (כלומר לא רק בבדיקה ראשונית) ולכן יש לבדוק שוב את קיום הכתובת מייל בסיום הרישום של המשתמש בצד שרת. אם הכתובת קיימת יש להחזיר את המשתמש להתחלת הרישום והודעה מתאימה ("this email is already in use, please choose another one"), כאשר קלט המשתמש הוזן ב-input כל עוד הקוקיז קיים (ראה פסקה הבאה).

קיימת דרישה להשתמש בקוקיז על מנת לשמור את פרטי המשתמש לטווח זמן של 30 שניות (מהרגע שהמשתמש עבר למסך השני של הרישום). הגדירו מספר זה כ- `const REGISTER_TIMEOUT` בתחילת התכנית כדי שנוכל לשנות אותו בקלות. כל עוד המשתמש נמצא בטווח זמן זה, אנחנו תמיד נציג את הטופס הראשון עם הקלט שהוא כבר הזין ונאפשר להשלים את רישום. בדף הזנת הסיסמאות, אם הוא חוזר למסך הראשון, או אפילו סגר את הדפדפן, וחוזר לכתובת URL של שלב ראשון של הרישום, הטופס יכלול את הקלט שלו כל עוד לא עברו `REGISTER_TIMEOUT` שניות. אם עבר הזמן, שליחה לשרת להשלמת הרישום של דף הזנת הסיסמא, יעביר אותו למסך הראשון של הרישום עם טופס מאופס, במקום להשלים את הרישום.

לאחר סיום הרישום שדות הטפסים מאופסים וכך נאפשר רישום משתמש חדש. מומלץ לבדוק את האתר עם `REGISTER_TIMEOUT` ארוך על מנת לבדוק את כל התסריטים האפשריים.

הכתובת לדף רישום הינו `register/`. יש לממש את `"/` - כדף "נחיתה" - כך שיביא את דף ה-`login` (אך כפי שהוסבר בשיעור דף ה-`login` לא יהיה זמין למשתמש כבר מחובר ויעביר אותו ל-`chatroom`).

סיום הרישום מעביר לדף `login` (זהו URL אחר `"/` כלומר `route` אחר). לאחר הצלחה ברישום, נציג דף חדש `login` עם הודעה בהתאם "you are registered".

חלק ב': chatroom

דף chatroom

זמין רק למשתמש מחובר, אחרת אין להציג אותו בכלל ועוברים למסך `login`. כאשר משתמש מתחבר נציג לו את ההודעות של ה-`chat` בסדר יורד (תאריך עדכני ביותר קודם), כל הודעה תוצג יחד עם שם הפרטי של המשתמש.

דף זה יכלול:

- טקסט: `Welcome <username>` כאשר `<first name>` מכיל את שם של המשתמש המחובר
- `text input` וכפתור לשליחת הודעה לציאט
- כפתור ליציאה מה-`chat` (logout) ומעבר למסך `login`
- ציאט: רשימת ההודעות שנשלחו עם שם השולח, תאריך וזמן בפורמט קריא, ואת ההודעה

הציאט מתעדכן כל 10 שניות: הגדירו מספר זה כ- `const POLLING` בתחילת כדי שנוכל לשנות אותו בקלות. יחד עם מנגנון `polling`, יש להשתמש ב-`rest api` על מנת לחלץ הודעות עדכניות. מומלץ לבדוק את האתר עם ערכים שונים (כגון 5 דק). על מנת לבדוק את כל התסריטים האפשריים. יש לדאוג לכך שכל הגולשים רואים את אותו תוכן של הודעות באותו סדר. תוכן הציאט מתעדכן מיד אחרי הוספה מחיקה עריכת הודעה (ולא רק בסיבוב הבא של ה-`polling`).

יש לאפשר למשתמש לערוך או למחוק כל הודעה שהוא שלח.

- כדי לאפשר למשתמש עריכת הודעה (פעולה זו תמומש ב-`fetch`) תוך כדי זה שיש עדכונים בדף, מומלץ ליצור אלמנט של ה-DOM שלא מתערבב עם תוכן ההודעות המתעדכן בו זמנית.
- כדי לטפל בעניין העדכונים מומלץ להשתמש ב-`paranoid mode` של `sequelize` יוסבר בסדנה.
- פעולות הוספה ומחיקה ימומשו ב-`fetch` (rest api) אלה כטופס רגיל עם route של `express/ejs`. לגבי מחיקה נדאג נציג למשתמש הודעה כגון "Are you sure? Deleting a message cannot be undone". יחד עם תוכן ההודעה. לאחר הוספה או מחיקה (או ביטול), נחזיר את דף ה-`chat`.

ניהול מבנה הנתונים של ההודעות:

חשוב לעדכן את תוכן הציאט רק אם יש עדכון בצד שרת. כלומר יש צורך להחזיק תאריך עדכון לוקאלי בצד לקוח על מנת

להשוות בין שרת-לקוח ולבקש מהשרת עדכונים. נדגיש גם שמבנה הנתונים נמצא בצד שרת ומומלץ לצ להחזיק מבנה דומה בצד לקוח אלא לסמוך על פעולות של rest api על מנת לקבל תוכן תמיד עדכני. נוסיף כי לצורך פשטות אפשר לרענן את כל תוכן ההודעות על כל עדכון במקום לבצע עבודה כירוגית ב-dom. בחלק זה אין להשתמש ב-web socket.

סיום ששן:

נניח והמשתמש התנתק: מצב כזה יכול לקרות כאשר המשתמש פתח שני דפים והוא מתנתק באחד מהם, או שהוא מחק קוקיז, או שה session- שלו פג תוקף. במקרה זה אנחנו רוצים להעביר אותו לדף login. מספיק לגלות זאת בתקשורת עם השרת: כלומר במעבר לדף, בשליחת הודעה, חילוץ הודעות, חיפוש או בכל פעולת רשת (כגון מנגנון ה-polling שיכול לטפל בזה תוך כדי). אז נדאג להעביר את המשתמש לדף login (בצד שרת קיימת פעולה redirect, שימו לב בצד לקוח עושים זאת ע"י window.location ובצד שרת response.redirect())

Rest api:

חשוב לאמת את המשתמש בכל פעולות של rest api וכך למנוע פעולות סדוניות כגון מחיקת הודעות של משתמשים אחרים.

חיפוש

זמין רק למשתמש מחובר, כמו כל דף הקשור ל-chat. דף זה יאפשר חיפוש טקסט בתוך מסד הנתונים לפי תוכן הודעות. ניתן לממש את החיפוש באותו דף של ה chatroom או בנפרד.

באופן כללי חייבים לאפשר למשתמש גלישה בין דפים על ידי כפתורי ניווט בדפים עצמם, או לינקים של האתר עצמו ולא להתבסס על הכפתורים של הדפדפן (לדוגמה דף תוצאות חיפוש חייב כפתור להמשך גלישה/חזרה לציאט).

יש לממש:

- כניסה לציאט (login)
- יציאה מהציאט (logout)
- הוספת הודעה
- מחיקה ועריכת הודעה ע"י המשתמש בעל ההודעה
- חיפוש הודעות לפי טקסט בהודעות
- קבלת הציאט על ידי הלקוח ועדכון הדף (rest api)
- טיפול שגיאות שונות (input validation)
- טיפול בסיום ששן (עבור כל דפי הציאט: מעבר אוטומטי ל-login) ולהפך מעבר אוטומטי לציאט במידה והמשתמש כבר מחובר
- טיפול בשגיאות תקשורת (הודעה רלוונטית למשתמש)

הצעת תכנית עבודה לחלק ב':

1. לפתח את דף ה-login (שימוש ב-session) ואחרי login מוצלח להציג דף סמלי
2. לפתח ולבדוק (בעזרת REST API postman) לניהול הודעות (חילוץ, הוספה, חיפוש, מחיקה, עדכון – אפשר לעבוד עם משתמש קבוע סמלי לצורך בדיקה)
3. להוסיף את כל נושא אימות/הרשאות (שימוש בסשן)
4. לפתח את הדפים וקוד צד לקוח לתצוגה ופעולות של הציאט מול ה-rest api

הערות כלליות:

חשוב לשמור על מסד נתונים עקבי וללא תוכן מיותר (לדוגמה לדאוג שלא לשמור מידע חלקי או זמני שלא יהיה רלוונטי להמשך). נצלו את השאילתות: ניתן למיין/לסנן/לשלב בקשות בבקשה אחת, אל לבצעו פעולות בקוד javascript אם ניתן לעשות זאת כבר ב-sql (כגון מיון).

בדקו את האתר שלכם עם 2 דפדפנים ומשתמשים שונים! בדקו את עניין סגירת הסשן בכל מצב / פעולה אפשרית באתר.

ראו את המסמך שגיאות נפוצות שבדרייב עם התרגילים.

יש להגיש את כל הפרויקט ללא תיקייה node_modules. שימו לב לא לשכוח להכניס את הקובץ package.json או תמונות, css.

לא לשכוח trim לכל קלט! קלט שכולו רווחים נחשב ריק.

יש גישות שונות לאתר מסוג זה, ואין ספק שזה אתר חלקי ואפשר להתמקד בדברים אחרים, לכן מותר להציע שינויים 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 and write your names and email (the README.md format is explained [here](https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax) :
<https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax>)
2. Open the terminal and type
`npm install -g express-generator`
`express -e`
At this point you have an express project that can run.
The code includes 2 routes (index.js and users.js), we recommend cleaning up and remove these files and write your own routes (irrelevant code such as code examples from class in your submission is penalized).
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 mysql2 mariadb sequelize`
If you have errors (ERR), look for solution in the sequelize slides.
5. create the *models* and *controllers* folders:
Until we learn about databases you can write the models similarly to 11-MVC-shop example (note the separation between routes and controllers).
Copy from this repo the models/index.js into your folder and add the DB initialization code in your app.js. Regarding the REST api backend, look at the rest api example.
6. Set up your database as instructed in the website