

מטלת גמר קורס תכנות צד שרת – לניהול סטודנטים

תוכן עניינים

כללי

תיאור פעולת היישום

הערות דרישות נוספות והסברים פרטניים לגבי חלק מנושאי העבודה

המלצה לשלבי העבודה על היישום

נספח א' – מבנה מסד הנתונים

כללי

1. את העבודה הסופית יש להעלות כקובץ ZIP (או RAR) של כל תיקיית העבודה הכוללת כל הקבצים הנדרשים וכן קובצי תיעוד. את הקובץ המכוון הזה יש להעלות לתיקייה ייעודית שתיפתח במודל בתיקיית כללי. במידה וקובץ העבודה המכוון יהיה גדול מדי ממה שאפשרי להעלות למודל, ניתן להעלות לתיקיית הדרופבוקס שלי בכתובת:

<https://www.dropbox.com/request/wdoL3ZDEKbDR28cLdPb1>

אין להסתפק בהודעה שהקובץ הועלה, אלא יש לשלוח אלי דוא"ל מפורש המודיע לי שהקובץ הועלה ל-Homework assignments בדרופבוקס ולחכות לאישור קבלה מפורש שלי (בעת העלאה, יש לרשום שם ושם משפחה ברורים כדי שאוכל לדעת של מי הקובץ). אם מעלים במודל, אין צורך לקבל אישור ממני על קבלת העבודה.

2. במסגרת מטלת הגמר, יש לבנות שרת node.js אשר יקבל מלקוח (דפדפן) בקשה לקרוא, להוסיף, לעדכן ולמחוק מסמכי (אובייקטי) סטודנט. שמירת נתוני הסטודנטים תבוצע במסד נתונים MongoDB (שם המסד הספציפי בו ינוהל המידע לגבי הסטודנטים ייקרא academy). השרת הייעודי הזה ידע לפעול בשני מצבים – מצב HTML שבו השרת מחזיר ללקוח את קוד ה-HTML המתאים בהתאם לפעולה הנדרשת. במצב השני, מצב JSON, כל התעבורה בין הלקוח לשרת (בשני הכיוונים) הינה תעבורת JSON. את היישום יש לממש בעזרת express ועם ממשק mongoose.

3. כל סטודנט צריך לבצע את המטלה בעצמו (דהיינו אין להגיש את העבודה כצוות). חוץ מאשר בדיקת קוד העבודה, מספר סטודנטים (הכמות תוחלט על ידי המרצה לאחר קבלת העבודות ואילו הבחירה תהיה בהגרלה) יתבקשו לקיים עם המרצה, כל אחד בנפרד, מפגש זום "אחד על אחד" במהלכו יענו על שאלות לגבי העבודה. בנוסף, במהלך המפגש יתבקשו הסטודנטים שעלו בגורל לבצע שינוי בקוד שלהם כדי לממש פונקציונליות שלא הייתה מוגדרת בעבודה (על הסטודנט יהיה לבצע share לחלונות העבודה שלו הקשורים ליישום). ההתרשמות מתשובות הסטודנטים תהיה **מרכיב משמעותי מאד** בציון העבודה.

4. כחלק מהמטלה, תיקיית העבודה תכלול קובץ וורד בשם ReadMe.docx המתאר באופן כללי ביותר את מרכיבי העבודה (התיאור הזה יכול להיות עד עמוד אחד בלבד!). יש לציין בקובץ זה תוספים מיוחדים שבהם השתמשתם (אם רלוונטי) וכן לציין בעיות מיוחדות שלא נפתרו (אם יש כאלו).

5. כחלק מהחומר הסופי שיש לשלוח עבור המטלה, נדרש לכלול ייצוא (או dump) של האוסף אתו תעבדו (האוסף students אשר מבנהו מוגדר בנספח א'). את מסמכי (אובייקטי הסטודנט) אפשר ליצור באופן

ידני על מנת שיהיה ניתן לבדוק את היישום כבר בשלבים הראשונים של הפיתוח ואפשר כמובן ליצור אובייקטי סטודנטים דרך היישום עצמו (הכול שאלה של איך אתם רוצים להתקדם בבדיקה. בדרך כלל רצוי שיהיו כמה אובייקט סטודנטים מראש כבר במסד). אם יוצרים מסמכי סטודנטים באופן ידני, הכי נוח לבצע זאת בעזרת MongoDB Compass. את קובץ הייצוא (שיהיה בפורמט JSON) תקבלו על ידי בצוע export מתוך ה-Compass. כחלק מהבדיקה שלי, אני מבצע import לקובץ ששלחתם וכך יהיה לי את אותו מסד נתונים שאתם בדקתם אתם את היישום שלכם (ואולם אני גם אבדוק עם תכולת מסד משלי).

6. יש לכלול בקוד הערות תיעוד. אפשר להסתפק בהערות כלליות בראש קטעי קוד מרכזיים (אלא אם כן יש צורך להסביר פקודה ספציפית). יש "להגן" על היישום ממקרי שגיאה אפשריים. **אין צורך להשקיע** במיוחד בעניין ווידוא הקלט אלא רק בדברים ממש קריטיים. בכל מקרה לא יתאפשר למשל להוסיף אובייקט סטודנט אם האובייקט לא יכלול מאפיינים שהוגדרו כחובה בסכמה (Mongoose Schema).

7. שם הקובץ הראשי של השרת יהיה server.js כאשר הפעולות השונות בקשר עם ניהול הסטודנטים ימומשו על ידי router ייעודי לטיפול בסטודנטים. נתב הסטודנטים "יורכב" על ה-express בנתיב `"/student"`. לגבי מיקום קובץ ה-router ראו בהמשך.

8. כל הפניות לשרת "יתועדו" במסד נתונים ייעודי (academylog) (ראו הסבר בהמשך).

9. רצוי להתייחס להמלצות בפרקים "הערות נוספות" ו-"סדר פיתוח מומלץ" בהמשך.

תיאור פעולת היישום

10. במצב GET בנתיב היחסי `"/"` (דהיינו בתגובה לכתובת <http://localhost:8080/student/>) היישום יציג בחלק התחתון של דף ה-HTML רשימה של הסטודנטים הנמצאים כרגע במסד הנתונים ואילו בחלק העליון יופיע טופס אשר יאפשר לסנן אילו סטודנטים להציג. מסך זה ייראה באופן עקרוני כך (כל מראה המסכים המודגמים במסמך זה הינם אופציונאליים ואתם יכולים לבחור תצוגה כרצונכם):

Toar: -- All --

City:

Min. avg. grade:

id	name		
123456789	רותי	<input type="button" value="delete"/>	<input type="button" value="update"/>
2222222	Danny	<input type="button" value="delete"/>	<input type="button" value="update"/>
8888	נדב	<input type="button" value="delete"/>	<input type="button" value="update"/>

[Add a student](#)

11. המשתמש יכול לבחור לסנן על פי התואר הנלמד ו/או עיר המגורים ו/או הציון הממוצע של הקורסים שלקח הסטודנט עד כה. בתגובה להקלקה על "סנן" יישלחו נתוני הטופס ויטופלו בנתיב היחסי "/" אבל במצב POST.

12. על אף שאפשר באופן עקרוני לייצר את קוד ה-HTML הנדרש לשם קבלת התצוגה לעיל בעזרת קוד JS ייעודי, מומלץ ביותר להשתמש לשם כך בתבנית pug ובמתודה `res.render()`.

13. במצב GET על הנתיב היחסי `"/add/"` (אליו מגיעים דרך הקישור בתחתית הדף), היישום יציג למשתמש את המסך הבא:

Id:

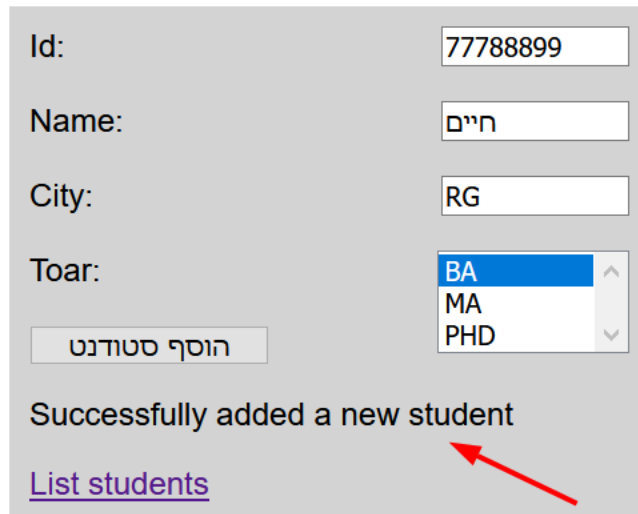
Name:

City:

Toar: BA
MA
PHD

[List students](#)

14. המשתמש ימלא את שדות הטופס ויקליק על "הוסף סטודנט". הטופס יישלח ב-POST ויטופל בנתיב היחסי "/add/" (אבל כאמור במצב POST). הקוד יוסיף את הסטודנט למסד הנתונים ויחזיר הודעה על ההוספה המוצלחת כמתואר להלן:



שימו לב שקוד ה-HTML המוחזר מבצע תמידות (persistence) של השדות שמולאו. המשתמש יוכל להוסיף מספר סטודנטים בזה אחר זה. הקלקה על List students תעביר את היישום לדף הראשי. מסמכי הסטודנט שמתווספים למסד בחלק זה של היישום אינם כוללים כרגע קורסים. הוספת קורסים תבוצע ממסך אחר, מסך ה-update.

חשוב: מה שמוצג למשתמש הם "קודי" התואר BA, MA, PHD אבל מה שמוזן בתוך מסד הנתונים (וזה גם מה שיכול להיות מועבר בבקשת הוספת הסטודנט שמעביר הדפדפן לשרת) אלו המחרוזות "ba", "ma" או "phd" בהתאמה.

15. בעת הקלקה על כפתור ה-DELETE (ראו במסך התצוגה הראשי) ולאחר ווידוא שהמשתמש אכן מעוניין במחיקת הסטודנט¹, היישום יבצע POST בנתיב היחסי "/delete/xxxx" כאשר xxxx הוא ה-id של אובייקט המסמך של הסטודנט האמור כפי שהוא מוגדר במסד הנתונים. פעולת היישום במצב POST על הנתיב היחסי "/delete/:id" תהיה למחוק הסטודנט מהמסד (כולל כמובן הקורסים שלו המהווים תת-מסמך במסמך הראשי), ולאחר מכן הקוד יבצע מעבר תכנותי חזרה אל הנתיב היחסי "/" כדי להביא לתצוגת רשימת סטודנטים מעודכנת.

16. בעת הקלקה על כפתור ה-UPDATE (ראו במסך התצוגה הראשי) היישום יבצע פעולת GET לנתיב היחסי "/update/xxxx" כאשר xxxx הוא ה-id של מסמך הסטודנט הרלוונטי. תגובת היישום על GET בנתיב היחסי "/update/:id" תהיה לשלוח את נתוני הסטודנט המבוקש ולהחזיר את קוד ה-HTML אשר ירונדר כלהלן:

¹ ווידוא שהמשתמש אכן מעוניין למחוק את רשומת הסטודנט תבוצע על ידי קריאה ל-confirm- (זוהי פקודת JS בדקו תיעוד לגביה בוב).

Id:	123456789
Name:	Yael
City:	RG
Toar:	BA ▼
<input type="button" value="עדכן סטודנט"/>	
course grade	
66666	72
35000	87
<hr/>	
Course Id:	<input type="text"/>
Grade:	<input type="text"/>
<input type="button" value="הוסף קורס"/>	
List students	

המשתמש יראה פרטים נוספים לגבי הסטודנט שלא נראו במסך הראשי (כמו מקום מגורים והתואר אליו הסטודנט רשום) וכן רשימת הקורסים שהסטודנט סיים. לרשות המשתמש יעמדו שני כפתורי שליחת טופס (כל כפתור מקושר לטופס אחר). הקלקה על הכפתור העליון "עדכן סטודנט" תגרום לשיגור טופס עם נתוני הסטודנט (מספר זיהוי, שם, עיר מגורים ותואר) ותגובת השרת תהיה לעדכן לסטודנט את הנתונים הללו (דהיינו את השם, העיר והתואר אבל לא את ה-id שלו שנקבע בעת יצירת הסטודנט ויותר לא ישונה). שימו לב שתיבת הטקסט שמכילה את זיהוי הסטודנט אינה מאפשרת שינוי/הקלדה של ערך.

בעזרת הכפתור שבתחתית המסך "הוסף קורס" יוכל המשתמש להוסיף קורס נוסף לרשימת הקורסים של הסטודנט והקלקה על כפתור זה תשגר טופס עם זיהוי הקורס והציון שקיבל בו הסטודנט ובתגובה השרת יוסיף את הקורס לרשימת הקורסים של הסטודנט.

הערות:

שתי פעולות שיגור הטופס המתוארות לעיל יהיו ב-POST לכתובת שרת מהצורה "/update/xxx" כאשר xxx הוא ה-id של מסמך הסטודנט במסד ה-Mongodb ועל כן יהיה צורך לשלוח בטפסים שדה מיוחד בשם action (מסוג hidden) אשר יחזיק ערך מחרוזתי שיסמן לשרת איזה סוג טופס ואיזו סוג פעולה בעצם נדרש (כי כאמור, פעולות העדכון הללו יגיעו ויטופלו על ידי אותו נתיב בשרת). הערך של action במקרה הראשון יהיה "updatestudent" ובמקרה השני "addcourse".

בסיום בצוע העדכון (בשני סוגי העדכון לעיל) יבצע השרת העברת GET (בעזרת redirect) אל אותה הכתובת (אפשר לכתוב "/") על מנת להציג את הסטודנט עם עדכון הפרטים כולל קורס חדש שהתווסף (המקרה השני) ולאפשר למשתמש להוסיף ולעדכן את פרטי הסטודנט או להוסיף קורס נוסף. הקלקה על List students תעביר למסך הראשי.

בסעיף 33 מובאת רשימה שמתמצתת את כל הפעולות של היישום בכל הנתיבים. כפי שתראו בסעיף זה, הרשימה המובאת שם מתייחסת למצב ריצת היישום ב-mode JSON (על כך בהמשך) ומפרטת מה בכל מצב על היישום להחזיר (מבחינת JSON). ואולם, עצם הרשימה תעזור לכם בבדיקה שממשתם את כל הנתיבים/פעולות הנדרשות (הן במצב "הרגיל", מצב ה-HTML והן במצב "JSON" שיוסבר בהמשך), כולל טיפול בנתיבים לא מוכרים (ראו בסוף הרשימה בסעיף 33).

הערות דרישות נוספות והסברים פרטניים לגבי חלק מנושאי העבודה

17. כאמור הגישה למסד הנתונים תעשה בממשק mongoose. לצורך כך תיצרו קובץ אשר יכיל את הגדרת הסכמה (mongoose schema) של מסמכי סטודנט. בנוסף לסכמה ראשית זו, הקובץ יכיל גם הגדרה לתת הסכמה של לקיחת-קורס שהסכמה הראשית עושה בה שימוש כפי שהודגם בשיעור הסיכום (לצורך הגדרת הסכמות תשתמש בהגדרת מסד הנתונים כפי שמובאת בנספח א'). האובייקט האחרון שיוגדר בקובץ האמור, יהיה מודל (mongoose model) העושה שימוש בסכמת הסטודנט. הקובץ הזה יעשה export לאובייקט מודל הסטודנט. קובץ מודל זה (המכיל כאמור את כל מה שנדרש: סכמות + מודל) ימוקם בתת-תיקייה של הפרויקט בשם models.

18. עיקר הקוד של היישום ירוכז בקובץ ייעודי שיממש student router (ישנם מספר דוגמאות בשעורים האחרונים לשימוש ב-router). קובץ "נתב" הסטודנטים יכיל הגדרות לפעולות השונות בנתיבים השונים (פעולות get ו-post) כאשר הנתיבים יוגדרו באופן יחסי (קובץ הנתב "לא יודע" היכן הוא "יותקן" (mount) בידי קוד ה-express הראשי (אנחנו יודעים שהוא יותקן על "/student"). לצורך בצוע הפעולות השונות, קוד ה-router יעשה שימוש באובייקט ה-model של הסטודנט ולפיכך יבצע require שלו. קובץ נתב הסטודנטים ימוקם בתת-תיקייה של הפרויקט בשם routes.

19. קובצי ה-pug ימוקמו בתת-תיקייה של הפרויקט בשם views (זו תיקיית ברירת המחדל עבור pug). אם נדרש, אתם יכולים לכלול בתוך דף ה-pug קטע style עם הוראות CSS, אבל את רוב (או את כול) הוראות ה-CSS יש לרשום בקובץ CSS חיצוני ולדאוג בתבנית ה-pug לכלול הוראת link כדי לייבא את הקובץ הזה. במילים אחרות, **בכל מקרה עליכם להשתמש בקובץ CSS חיצוני** אשר ימוקם בתיקייה בשם public תוך שיש לוודא ששרת ה-express יודע "לשרת" את הקובץ הסטטי הזה כאשר הוא נדרש.

20. להלן מבנה הקבצים ותיקיות היישום:

application root folder:

- **json** (test files for the JSON running mode. See in the last section of this document)
- **models** (student model and log model files will be in this folder)

- **node_modules** (all Node.js installed packages will be located here)
- **public** (style sheet(s) or any other static file)
- **routes** (student router file)
- **views** (PUG files)

server.js – the main file and the starting point of the application

21. כדי לסנן את רשימת הסטודנטים על פי ציון ממוצע שיהיה גדול או שווה מהערך שהמשתמש נתן, ישנן שתי אופציות כיצד לבצע זאת. דרך אחת היא לקבל מערך של כל הסטודנטים אשר עונים על יתר התנאים (תואר ומקום מגורים) ואז לעבור על המערך הזה בלולאה ולהפנות כל אובייקט סטודנט לפונקציית שירות שלכם שתחשב את הממוצע. אם הערך גדול מערך הסף הנדרש, תוסיפו את הסטודנט הזה למערך מיוחד שיכיל את כל הסטודנטים שעונים לדרישות (כולל כאמור עניין ממוצע הציונים). דרך אחרת היא לבצע את הסינון ישירות מול MongoDB. הדרך הזו תדרוש מבנה שאילתת חיפוש מורכב יותר ממה שהודגם בקורס תוך שימוש במבנה סינון אגרגטיבי מסוג expression (האופרטור \$expr) יחד עם פונקציית ה-\$and וה-\$avg. ישנה הדגמה של שימוש בפילטר כזה בסוף חלק ההקלטה הראשון (ותחילת השני) של מפגש הסיום אבל תצטרכו לחפש מידע נוסף ברשת (עבור השימוש בפונקציה \$avg).

22. הפעלת היישום תתבצע על ידי הרצת הקובץ server.js כלהלן (הסוגריים המרובעים בסוף השורה מציינים שהתוספת --json היא אופציונלית):

```
node server.js [--json]
```

אם ההרצה תהיה ללא התוספת --json (דהיינו node server.js) היישום יפעל במצב של HTML ואם היישום יופעל עם --json הוא יפעל במצב JSON.

לצורך מימוש דרישה זו, יש לכלול בקוד קובץ השרת (server.js) בדיקה של הקלט לתוכנית ולרשום במשתנה כלשהו האם מדובר ב-html mode או ב-json mode (למדנו בתחילת הסמסטר כיצד לבדוק קלט להרצת היישום). המקום שבאמת יהיה צורך לדעת מה לעשות בכל אחד משני המצבים הללו, הוא הקוד שמבצע את פעולת ניהול הסטודנטים בפועל, שזה הקוד שבתוך הקובץ המטפל ב-student router. ואולם, הקוד הזה נמצא בקובץ פנימי שייבא אל תוך ה-server.js. לפיכך, יש צורך שה-mode של ריצת השרת יירשם בתוך משתנה גלובלי שיהיה נגיש בכל מקום ביישום (כולל כמובן ב-router). דרך פשוטה לעשות זאת היא לרשום את המשתנה כמאפיין של האובייקט הגלובלי הנקרא בשפת node.js - global. הפקודה תיראה אפוא כך²:

```
global.runmode = ...
```

ואילו בכל מקום שנדרשים לדעת את הערך של המשתנה הזה כותבים למשל:

```
if (global.runmode == ...)
```

² בפועל גם אם סתם מכניסים ערך למשתנה ללא ציון המילה var הוא משויך אוטומטית ל-global. זו פשוט צורת קריאה יותר ברורה בקוד.

23. עבור כל פנייה לשרת יוכנס למסד נתונים בשם academylog (באוסף log) מסמך (אובייקט) המתעד את הפנייה לשרת אשר יכלול את 4 השדות הבאים:

- **method** - שיטת הפניה (get או post)
- **when** - זמן הפנייה (תאריך/זמן) (רצוי להגדיר שדה זה בסכמה עם ברירת מחדל של Date.now).
- **path** - נתיב הפנייה (למשל /student/delete/xxxx)
- **runmode** - מצב עבודת השרת (JSON או HTML)

לצורך מימוש דרישה זו, יש להגדיר בקובץ נפרד מודל (mongoose model) עבור הטיפול במסמכי ה-Log בדומה לקובץ מודל הסטודנט, גם קובץ זה יגדיר את הסכמה הנדרשת וייצא את אובייקט המודל הנבנה על בסיס סמכה זו. גם קובץ זה ימוקם בתת התיקייה models. ואולם לצורך בצוע פעולת ה-Log (שהיא למעשה פעולת שמירת נתונים בלבד ללא קריאה או עדכון או מחיקה) לא תגדירו router ייעודי אלא תבצעו את הפעולה בפונקציית ביניים (intermediate function) שתגדירו בקובץ השרת הראשי, דהיינו בקובץ router.js. אתם יכולים לקרוא לפונקציה זו למשל my_log() (יש דוגמאות והסברים מפורטים בנושא פונקציות ביניים במפגשים בנושא express). בתוך פונקציית ביניים זו, תצרו אובייקט המתאים לסכמה כאשר את הנתונים עבור אובייקט ה-log תוכלו לקבל באופן הבא:

- את ערך ה-method ניתן לקבל על ידי המאפיין req.method (req הוא אובייקט ה-request שכל פונקציית ביניים מקבלת כאחד מהקלטת שלה).
- את ערך ה-path ניתן לקבל על ידי המאפיין req.path
- את ערך מצב הריצה (runmode) אפשר לקבל על ידי הערך ששמרתם במשתנה הגלובאלי למטרה זו.
- את ערך הזמן הנוכחי אין צורך לספק בעת בניית אובייקט ה-log, הוא יתקבל אוטומטית (אם דאגתם להגדיר זאת כך בסכמה, אם לא יש לספקו כעת – Date.now).

את פונקציית הביניים תשלבו בבצוע הכללי של השרת על הנתיב הראשי של היישום וחשוב לא לשכוח בסיום מוצלח של שמירת אובייקט הלוג לקרוא ל-next(). אגב, אם אתם מחליטים לממש את הקריאה לשמירת ה-log בעזרת promise, הגדירו את פונקציית הביניים כאסינכרונית (המילה async לפני function).

24. את השרת יש להריץ בפורט (port) 8080. אם אצל משהו שער זה תפוס והוא נאלץ לעבוד עם port בערך אחר, יש לרשום זאת בתיעוד.

25. יש להשתמש בשמות השדות במסד הנתונים (וכמובן בסכמה) כמוגדר בנספח א'. כמו כן בשדות הטפסים השונים של היישום יש להשתמש בשמות הבאים (ערך המאפיין name):

- id עבור ה-id של הסטודנט
- name עבור השם של הסטודנט
- toar עבור התואר של הסטודנט (בטופס הוספת סטודנט ובטופס חיפוש סטודנטים)
- city עבור יישוב המגורים של הסטודנט (הן בטופס ההוספה והן בטופס החיפוש)
- avg עבור ממוצע מינימום בטופס חיפוש סטודנטים.
- cid עבור ה-id של קורס
- grade עבור ציון הקורס

המלצה לשלבי העבודה על היישום

26. כל אחד יכול לבחור את הדרך שלו לפיתוח היישום. הדרך המוצעת להלן הינה דרך של שלבים כאשר בכל פעם מתווספת יכולת נוספת. זו אמנם דרך מעט "ארוכה" יותר (מאשר פתוח מלא מראש) ושדורשת בנקודות מסוימות "תיקון" של הקוד שהתקבל בשלב הקודם, ואולם היא מבטיחה התקדמות בטוחה יותר של צעד אחר צעד.
27. שלב 1 – פיתוח העבודה מול מסד הנתונים. בשלב זה לא יורץ שרת (על אף שהקוד יופעל בעיקר מתוך הקובץ server.js הוא לא יכלול בשלב זה יצירת express). בשלב זה גם לא נבצע כתיבת לוג ועל כן מדובר יהיה במסד נתונים אחד (ולכן קישור סטנדרטי אחד של mongoose). השלב כולל את יצירת קובץ המודל לסטודנטים (הכולל כמובן את הגדרות הסכמה) ולאחר מכן (תוך בצוע require מתאים בקובץ server.js) כתיבת קוד לפעולות השונות (בצוע save, find, delete ו- update). לצורך הבדיקה, רצוי להכין מסד נתונים עם 2-3 סטודנטים.
28. שלב 2 – יצירת שרת express. יהיה צורך ליצור את אובייקט ה-router של הסטודנט ולהגדיר בו את כל הפעולות הנדרשות. אובייקט ה-router ייבוא (required) בקובץ server.js ויותקן (mounted) בנתיב '/student'. בשלב זה כל מה שתבצעו בנתב הסטודנט הוא הדפסות console.log שידגימו לכם שהשרת קולט את הנדרש בכל הנתיבים ובכל המקרים (get או post) ובסיום תסיימו עם res.end() המודיע OK או משהו דומה (בכל מקרה זה הרי קוד זמני). במקרי ה-post תדפיסו גם את req.body כדי לוודא קבלה נכונה של המידע. אתם יכולים להפעיל את השרת ולבדוק אותו באמצעות דפדפן אבל גם ובעיקר בעזרת curl. אתם יכולים להכניס את כל הקוד העוסק במסד הנתונים להערה, כי בשלב זה אתם רק בודקים את ריצת השרת. זכרו שבכמעט כל מקרה ישנן שתי פעולות נפרדות: אחד get ואחת post על אותו הנתיב.
29. שלב 3 – טיפול ב-HTML. בשלב זה תגדירו קובצי PUG עבור המקרים השונים ובנתב הסטודנטים, תחליפו את ה-res.end() ב-res.render() כדי להחזיר קוד HTML. מכיוון שעדיין אתם לא עובדים עם מסד הנתונים, הכינו מערך סטודנטים (hard coded), בהתחלה בתוך קובץ ה-PUG עצמו ולאחר מכן בקובץ נתב הסטודנטים, וראו שאתם יודעים ליצור את "דפי" ה-HTML הנדרשים. לאחר שמקבלים את המבנה ה-HTML-י הנכון, יש לדאוג ל-persistence, דהיינו להחזרת ערכי שדות הטופס כפי שנקלטו בעת ה-post (ראו שיעור מסכם).
30. שלב 4 – שילוב העבודה מול מסד הנתונים. זה למעשה השלב המשמעותי בפיתוח שבו היישום אמור לבצע כבר את הפעולה העיקרית שלו. עבור שלב זה, תעבירו את הקוד שכתבתם קודם ב-server.js (הקוד שבצעתם לבדיקת העבודה מול mongoDB) אל המקומות המתאימים בקובץ נתב הסטודנטים (קובץ נתב הסטודנטים יצטרך לבצע כמובן require למודל הסטודנט אבל בנתיב שונה מאשר ה-require בשרת בגלל המיקום היחסי השונה של הקבצים).
- ברוב המקרים, לאחר פעולה מוצלחת מול מסד הנתונים, ייבדק אובייקט השגיאה ואם הוא null (דהיינו אין שגיאה - !err), תבצעו res.render() עם קובץ התבנית המתאים (תוך העברת הפרמטרים הנדרשים). אם קרתה שגיאה, אפשר להסתפק בהחזרת הודעה של מספר מילים על ידי res.end() (למשל "Update student failed"). במקרים אחרים, למשל במצב delete, אם הפעולה מול המסד הצליחה הקוד יבצע res.redirect() ל-"דף" הראשי שמציג את כל הסטודנטים ואם נוצרה שגיאה, אפשר פשוט לדווח עליה עם res.end(). עבור מצב ה-GET לקבלת רשימת כל הסטודנטים ועבור מצב ה-POST לקבלת רשימה חלקית של סטודנטים (לאחר סינון), אין צורך לבדוק האם היה error.

זה יהיה הזמן בו תבצעו בדיקות מקיפות של היישום ובסיום שלב זה אפשר לומר שהיישום התקרב למצבו הסופי.

31. שלב 5 – הוספת ביצוע logging. זה יחסית חלק פשוט שהוסבר בפירוט בחלקים קודמים של המסמך (השימוש בפונקציית ביניים). הדבר היחידי שהיווה הצדקה לדחיית עניין זה לשלב נפרד הוא העובדה שמדובר בשני מסדי נתונים שונים ואין אפשרות להסתמך על האובייקט `mongoose.connetion`. לפיכך יש כעת צורך להחליף בשרת `server.js` את הקוד שיוצר קישור למסד הנתונים היחיד `academy` לקוד שיוצר שני (!) אובייקטי קישור למסד `academy` ולמסד `academylog` (ראו בשיעור הלפני אחרון כיצד לעשות זאת). גם בקובצי המודל - קובץ המודל של סטודנט וקובץ המודל של ה-`log` (דהיינו בשני הקבצים בתת-התיקייה `models`) - יש צורך לשנות את הקוד ובמקום ליצור את המודל בעזרת `mongoose.connection.model()` ליצור את המודל בעזרת `conn1.model()` ו-`conn2.model()` כאשר `conn1` ו-`conn2` הם שני אובייקטי הקשר לשני המסדים בהתאמה. על כן, כאשר אתם יוצרים בקובץ `server.js` את שני אובייקטי הקישור למסדי הנתונים, השתמשו באובייקט `global` ושמרו אותם באופן גלובלי (במאפיינים `conn1` ו-`conn2`) כך שיהיו ניגשים גם בקובצי המודל.

הערה חשובה: כאשר אתם עושים `require` למודל המטפל ב-`logging` וכאשר אתם מבצעים `require` ל-`router` של הסטודנט (קובץ שבעצמו משתמש במודל שהוגדר לסטודנט), יש לוודא שאתם מבצעים את פקודות ה-`require` הללו לאחר (!) שיצרתם את אובייקטי ה-`connection` (`conn1` ו-`conn2`) ולאחר ששמרתם אותם ב-`global`. אם לא תקפידו על סדר נכון של הדברים, בעת בצוע ה-`require` יהיה ניסיון ליצור את המודל בעזרת `conn1` או `conn2` מבלי שמשנתים אלו יהיו מוגדרים ותקפים.

32. שלב 6 – טיפול בשני מצבי הריצה: `HTML` ו-`JSON`. ראשית יש לקלוט ב-`server.js` את הקלט הנוסף בעת הרצת היישום (אם היה כזה) ולפי ערכו לרשום במשתנה `global.runmode` ערך מתאים: `"html"` או `"json"`. מכיוון שהשמירה היא גלובלית, היא נגישה מכל מקום ביישום, ובפרט בנתב הסטודנטים. לאחר שקובץ היישום הראשי (ה-`server.js`) יודע איזה מצב ריצה הוא מופעל, הוא ירכיב בהתאם את ה-`body parser` המתאים: את ה-`urlencoded parser` למצב "הרגיל" (מצב ה-`HTML`) ואת ה-`json parser` למצב ריצת `JSON` (זכרו שתנועת ה-`JSON` היא בשני הכיוונים, גם מהלקוח לשרת ולכן יש צורך לטפל גם ב-`body parser`).

בנתב עצמו, בכל מקום שבו בצעתם `res.render()` יש לבדוק כעת (על ידי משפט `if`) מהו מצב הריצה. אם המצב הוא `html`, הרי שהקוד כבר קיים ונבדק ויש לבצע את ה-`res.render()` כמקודם (במקרים מסוימים במקום `res.render()` יש צורך לבצע `res.redirect()`). אם, לעומת זאת, מצב הריצה הוא `json`, יש להודיע על ה-`content-type` המתאים ולאחר מכן לשלוח בעזרת `res.send()` את תוצאת `JSON.stringify()` על הנתונים הנדרשים (בפתרון התרגול של המפגש בו הודגם שימוש ב-`JSON` תוכלו לראות דוגמה כיצד לבצע זאת).

33. השליחה חזרה תהיה כלהלן (בכל המקרים מדובר בביצוע `JSON.stringify()` על הנתונים):

- עבור פעולת ה-`GET` שמחזירה את רשימת כל הסטודנטים – יש להחזיר את הרשימה האמורה (אין צורך לטפל במצב שגיאה).

- עבור פעולת ה-POST שמחזירה רשימת סטודנטים על פי סינון – יש להחזיר הרשימה האמורה (אין צורך לטפל במצב שגיאה).
- עבור פעולת ה-POST של הוספת סטודנט – יש להחזיר את אובייקט הסטודנט שנוצר (אותו אובייקט שמחזר על ידי מתודות ה-save). יש להחזיר "FAIL" במקרה כישלון.
- עבור פעולת ה-POST של עדכון סטודנט – יש להחזיר את אובייקט הסטודנט שמוחזר על ידי הקריאה ל-findOneAndUpdate. יש להחזיר "FAIL" במקרה כישלון.
- עבור פעולת ה-POST של מחיקת סטודנט – יש להחזיר את הערך 1 המציין שהמחיקה הצליחה ואובייקט אחד נמחק מהמסד. יש להחזיר 0 במקרה כשלון המחיקה.
- עבור פעולת ה-HTTP שאו שהיא מסוג שאינו נתמך ביישום זה (למשל מסוג DELETE או UPDATE) או שהנתיב שניתן אינו מוגדר ביישום, על היישום להחזיר דף מיוחד עם הכותרת Page Not Found (יש להחזיר אז את קוד ה-HTTP 404. בדף זה יהיה קישור המאפשר להקליק ולחזור לדף הראשי שמציג את כל הסטודנטים (דף זה יוחזר הן במקרה של HTML והן במקרה של JSON). כדי להבין כיצד לממש תגובה ל-path לא קיים ב-express ראו למשל כאן - <https://stackoverflow.com/questions/42461774/nodejs-handle-unsupported-urls-and-request-types>). גם במקרה זה ניתן להחזיר את קוד ה-HTML הדרוש בעזרת תבנית pug.
- חשוב:** גם כניסה לדף זה (אשר כאמור תנבע מנתיב/סוג פעולה שאינם מוגדרים) תתועד ב-log שהיישום מנהל.

יש מספר נקודות שיש לשים אליהן לב:

דבר ראשון, כאשר עבדתם במצב HTML, ומכיוון שעבדתם עם נתוני POST שמגיעים מדפדפן בטופס (ששדותיו היו required), היה מובטח לכם שכל המידע שהיה צריך להגיע הגיע (גם אם השדה הגיע ריק). כך למשל, בקוד המטפל בפילטר של סטודנטים על פי קריטריונים מסוימים, ייתכן שהיה לכם קוד מהצורה הבאה (שבדוק האם יש צורך לסנן על פי יישוב מגורים של הסטודנט):

```
var city = req.body.city;
if (city.trim() != "") ...
```

הקוד הזה עבד טוב כי הטופס תמיד שלח ערך עבור שדה זה (גם אם היה ריק). ואולם כעת כאשר השרת אמור לקבל את הנתונים ב-JSON, הם יגיעו אליו על ידי תוכנית curl על פי הנתונים שנחליט לספק (אין לנו כעת דרך לבדוק את השרת באמצעות דפדפן!) ואם למשל לא יישלח ערך עבור city, הערך שייכנס ל-city על פי הקוד לעיל יהיה undefined מה שיגרום לשגיאה בניסיון לבצע trim() על ערך זה.

לכן, יש לכתוב קוד שיבדוק אם בכלל הגיע ערך ב-body עבור משתנה מסוים, ואם לא, להכניס לו ערך ברירת מחדל (מחרוזת ריקה). הקוד יכול להיכתב כך:

```
If (req.body.city)
```

```

var city = req.body.city;

else

var city = "";

```

או בכתיבה מקוצרת כלהלן:

```
var city = (req.body.city) ? req.body.city; : "";
```

תוכלו להניח שבמקרי הוספת הסטודנט ועדכון הסטודנט, כל השדות המצופים אכן מגיעים (גם במצב HTML וגם במצב JSON).

הדבר השני שיש לשים לב אליו הוא שבמצב JSON לא כל מצבי נתב הסטודנטים פעילים. זכרו שבמצב HTML יש במקרים של הוספת סטודנט או עדכון שני שלבים: אחד (get) מגיעים למסך הטופס הראשוני, ובשלב שני, מצבעים את הפעולה. כאשר אנחנו פועלים ב-JSON, השלב הראשון לא קיים, מדובר בממשק CRUD שממש קריאת סטודנטים (או קריאה מסוננת במצב Post), מחיקת סטודנט, הוספת סטודנט ועדכון סטודנט. מדובר ב-5 פעולות שבהן צריך לטפל JSON-ית (הרשימה של 5 הפעולות פורטה בסעיף 33).

זכרו : בדיקות השרת במצב JSON יבוצעו כולם בעזרת CURL (המקרה היחיד שניתן לבדוק גם בדפדפן הוא מקרה ה-GET הבסיסי ללא הסינון). מומלץ לבחון קודם את מסד הנתונים, ולהכין מספר קובצי קלט מסוג json. שתוכנם (מחרוזת JSON של הפרמטרים הנדרשים ליישום) יישלחו בעזרת CURL על הנתיבים המתאימים כך שתדעו מראש מה התוצאה המצופה. תריצו את קריאות ה-CURL ווודאו שקבלתם הנדרש (לפעמים לאחר בצוע קריאת ה-CURL יידרש לבדוק את התוצאה במסד הנתונים ב-Compass). בקובץ פתרון התרגול של מפגש ה-JSON מוסבר כיצד משתמשים עם curl לצורך שליחת JSON.

אם מה שמודפס לכם בחלונות הקונסול בעת הפעלת ה-curl לא ברור (למשל בגלל תווי עברית שלא נראים טוב במסך), אתם יכולים להפנות פלט תוצאת הריצה אל תוך קובץ ואז לפתוח את הקובץ ב-VSC או בדפדפן (עדיף דפדפן פיירפוקס). דוגמה לפקודה כזו הינה:

```
curl http://localhost:8080/student/ > tmp.json
```

ולאחר מכן אפשר לפתוח הקובץ tmp.json בדפדפן פיירפוקס

נספח א' – מבנה מסד הנתונים

שם מסד הנתונים למסמכי סטודנט יהיה: **academy**

המסד יכלול אוסף יחיד בשם **students**

האוסף **students**

מסמך סטודנט יוגדר על ידי סכמה ויהיה מהצורה הבאה (ראו דוגמה בהמשך ושימו לב להערות/דרישות נוספות מכל שדה, אם יש):

id: of type String. It is the student's id, 9 digits or less, not theMongoDB's _id field). This field is required.

name: of type String. This field is required.

city: of type String.

toar: of type String. This field is required, and its value should be only one of the following: "ba","ma","phd")

courses: of type Array. Each of the array's items is of type Object defined by a special take_course schema (see in recording part 1 in the course's final meeting how to define a schema which relies on another schema). This sub schema defines a course the student had taken and includes the following fields:

cid: of type String. This field is required (e.g. '35619'). This cid is not MongoDB's _id property and is not the student id.

grade: of type Number. This field is required. The value of this field must be between 0-100

להלן דוגמה למסמך סטודנט:

```
_id: ObjectId("60cf55e188f0fb2fd4cf1dd2")
name: "Yael"
__v: 0
city: "RG"
toar: "ba"
id: "123456789"
courses: Array
  0: Object
    _id: ObjectId("60d09f7970a30a620c03ed29")
    cid: "66666"
    grade: 72
  1: Object
    _id: ObjectId("60d0a23137e55066a4f5057e")
    cid: "35000"
    grade: 87
```

שם מסד הנתונים למסמכי log יהיה: **academylog**

המסד יכלול אוסף יחיד בשם **log**

האוסף log מוגדר על ידי הסכמה שתוארה בסעיפים קודמים במסמך.

להלן דוגמה למסמך Log:

```
_id: ObjectId("5efec4615680d874404c8eeb")
method: "POST"
path: "/student/add"
runmode: "HTML"
when: 2020-07-03T05:38:41.446+00:00
__v: 0
```