

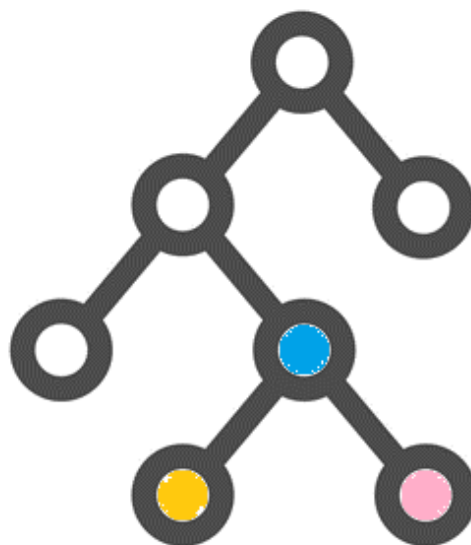
תרגול 9 עץ חיפוש מטומפלט

רקע

בשיעור למדנו על סוג חדש של מבני נתונים – עצים 🌲 🌳 🌴. בנוסף למדנו על templates שמאפשרים לנו להגדיר מחלקות גנריות, כלומר מחלקות שאינן מוגדרות עבור סוג נתונים ספציפי. אחד השימושים הנפוצים ל-templates הוא ליצור מחלקות מבנה נתונים שיוכלו להחזיק סוגי משתנים שונים.

מטרה

בתרגיל נממש את מבנה הנתונים עץ חיפוש גנרי (Binary Search Tree), תחילה שיחזיק מחרוזות (std::string), ובהמשך עם templates, כך שיוכל לשרת סוגי נתונים שונים.

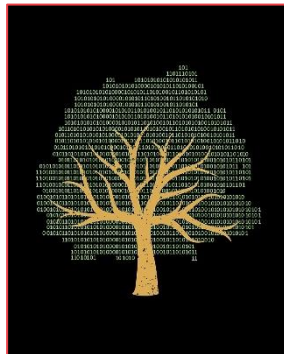


אלה השלבים שנעבור:

מימוש BST עבור מחזורות	תרגול כללי Templates	מימוש BST עם Templates	שלב בונוס AVL
<ul style="list-style-type: none">השלמת קובץ ה-Header ותיקון חתימות.מימוש מתודות ואופרטור =קישור הספרייה printTreeToFile.lib	<ul style="list-style-type: none">מימוש פונקציות נבחרותתיקון תרגיל 2 מתחילת הסמסטר	<ul style="list-style-type: none">מימוש מתודות העץ.	<ul style="list-style-type: none">מימוש עץ מאוזןשימוש ב-rotations

נתרגל מיומנויות חשובות:

- שימוש רקורסיה
- מימוש מבני נתונים מוכרים
- יצירת מחלקות templates
- עבודה עם ספריות
- את התרגיל צריך להגיש ב-GIT: [לינק להוראות שימוש ב-GIT](#). כדאי לקרוא גם [דגשים לתכנות נכון](#).



"PRACTICE MAKES PERFECT"

בהצלחה יא אלופות ואלופים!

שלב 1: עץ חיפוש עבור מחרוזות


שלב בונוס AVL	מימוש BST עם Templates	תרגול כללי Templates	מימוש BST עבור מחרוזות
------------------	---------------------------	-------------------------	---------------------------

סעיף א' – השלמה ותיקון קובץ ה-Header

עברו על קובץ ה-Header שסופק יחד עם הוראות התרגיל (BSNode.h) ובצעו את המשימות הבאות:

- הוסיפו את **סוג הנתונים** של הבן השמאלי והבן הימני (`_left`, `_right`) בשורות 32,33. רשמו את ה-`type` במתודות האחרות היכן שנחוץ. (רמז: היזכרו בהגדרה של רשימה מקושרת).
- הוסיפו `const` היכן שצריך (גם בסוף מתודה וגם על פרמטרים המועברים למתודה).
- העבירו אובייקטים כ-**reference** היכן שניתן.

סעיף ב' – מימוש מתודות המחלקה BSNode

 BSNode	
תיאור	מתודה/פעולה
בנאי המקבל מחרוזת	<code>BSNode(string text)</code>
בנאי העתקה (חישבו למה לא ניתן להשתמש בבנאי ה-default)	<code>BSNode(const BSNode& other)</code>
מפרק	<code>~BSNode()</code>
מכניסה ערך לעץ בהתאם לחוקיות של BST. אם מכניסים ערך שכבר קיים הפונקציה מעלה את ערכו של השדה <code>count</code> ב-1. (אין כפילויות בעץ).	<code>void insert(string value)</code>
אופרטור העתקה (חישבו למה לא ניתן להשתמש באופרטור ה-default)	<code>BSNode& operator=(const BSNode& other)</code>

ממשו את המתודות הבאות בקובץ BSNode .cpp

BSNode (המשך...)	
מתודה/פעולה	תיאור
<code>bool isLeaf()</code>	מחזירה האם הצומת היא עלה או לא
<code>string getData()</code>	מחזירה את המחרוזת שהצומת מחזיקה
<code>/**/ getLeft()</code>	מחזיר את תת העץ השמאלי (הבן השמאלי) (חישבו מה הסוג שהמתודה צריכה להחזיר)
<code>/**/ getRight()</code>	מחזיר את תת העץ הימני (הבן הימני) (חישבו מה הסוג שהמתודה צריכה להחזיר)
<code>bool getCount()</code>	מחזירה כמה פעמים מופיע הנתון שבשורש העץ (השדה count עולה ב-1 בכל פעם שמכניסים ערך שקיים בשורש, ויורד ב-1 שמסירים אותו)
<code>bool search(string val)</code>	מחזירה האם הערך שהתקבל קיים בעץ או לא
<code>int getHeight()</code>	מחזיר את הגובה של העץ (עבור הצומת שעליה הפעילו את הפונקציה)
<code>int getDepth (const BSNode& root)</code>	מחזירה את העומק של צומת כלשהי ביחס לצומת ששולחים כפרמטר לפונקציה. לכן אם שולחים את השורש מקבלים את העומק האמתי. אם הצומת הנוכחי הוא לא צאצא של הצומת שנשלח כפרמטר לפונקציה, זוהי שגיאה ויש להחזיר -1.

שימו 

אל התרגיל מצורף קובץ main.cpp שבו ניתן להשתמש כדי לבדוק את התקינות של המתודות.

הנחיות והערות חשובות:

- בהוראות לא רשומות החתימות המדויקות, אבל האחריות שלכם/ן היא להוסיף `const` ולקבל פרמטרים כ-reference היכן שצריך.
- אסור לשנות את החתימות של המתודות בקובץ ה-header (מלבד הוספת הסוג של הבנים)
- את המתודות יש לממש באופן **רקורסיבי**, אין צורך בלולאות.
- הפונקציה `getCurrNodeDistFromInputNode` היא פונקציית עזר לחישוב העומק

סעיף ג' – קישור ספרייה לתרגיל

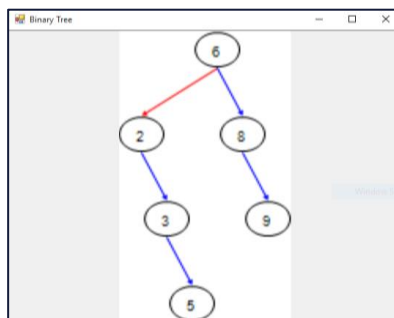
על מנת שנוכל לראות את העץ **בצורה גרפית**, מצורף קובץ EXE. ה-EXE מריץ תוכנית שיוצרת לקרוא נתונים מקובץ ולהציג אותם כעץ.

בקובץ ה-**main.cpp** שסופק ביחד עם הוראות התרגיל נעשה שימוש בקובץ ה-EXE ובספרייה **printTreeToFile** על מנת להציג את העץ.

כדי שהקוד יוכל לרוץ ראשית וודאו שיש ברשותכם/ן את כל **הקבצים שניתנו לשאלה 1**, כולל:

- BSTData.txt
- printTreeToFile.h
- printTreeToFile.lib
- printTreeToFile.pdb

וודאו שהקובץ **BSTData.txt** ו-**BinaryTree.exe** באותה תיקייה ו**הריצו את ה-EXE**. נסו להבין מה הקשר בין תוכן הקובץ **BSTData.txt** לבין מבנה העץ שהתוכנית מציגה



קישור הספרייה **printTreeToFile**

כדי שנוכל להציג בצורה גרפית עץ אשר מוגדר בתוך התכנית שלנו, נצטרך לקשר את הספרייה **printTreeToFile.lib**.

עד שנעשה זאת, הקוד שקשור לספרייה **לא** יעבוד ונקבל שגיאות לינקג'. אחרי שנקשר את הספרייה בצורה נכונה ניתן יהיה להשתמש בקוד שלה מתוך הקוד של התוכנית שלנו.

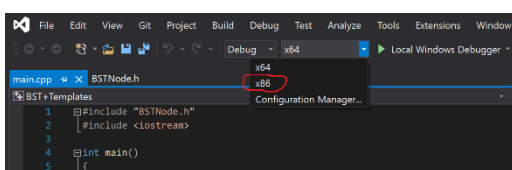
כדי לעשות זאת יש לשים את הקבצים בתיקייה של הפרויקט (אותה תיקייה בה נמצא קובץ מסוג **vcxproj**) ולעשות **#include** לקובץ **printTreeToFile.h** (בצורה דומה למה שכתוב בקובץ **main.cpp** הניתן עם התרגיל).

איך מוסיפים קובץ lib לפרויקט ה-VS שלנו? יש כמה דרכים...

עשיתם/ן את זה בקורס מבוא בשנה שעברה, אבל תוכלו גם להיעזר ב-[StackOverFlow](https://stackoverflow.com)

וודאו שהספרייה מקושרת כראוי ושהעץ המוגדר בקובץ ה-**main.cpp** מוצג בצורה נכונה.

משהו נוסף, הקפידו לקמפל את הספרייה ב-32 bits אחרת יהיו שגיאות לינקג'. אפשר לעשות זאת ע"י בחירת **x86** בהרצה של התכנית ב-Visual Studio.



סעיף ד' – הדפסת עץ

ממשו את המתודה `printNodes`

הפונקציה `printNodes` צריכה להדפיס למסך (באמצעות `cout`) את תוכן העץ מילה אחר מילה על פי סדר אלפביתי, וגם את מספר המופעים של כל מילה. כל מחרוזת צריכה להיות בשורה נפרדת בהדפסה.

דוגמה:

friends 1

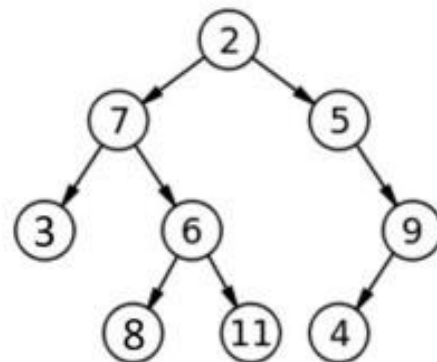
hello 2

קראו על שלוש השיטות למעבר על איברי עץ חיפוש בינארי: `postorder`, `preorder`, `inorder`. בחרו בשיטה המתאימה למשימה הזו (רמז: אנחנו רוצים להדפיס את האיברים בסדר אלפביתי – מהקטן לגדול).

דוגמה להדפסות השונות:

שימו 🖤, בדוגמה ישנו עץ בינארי רגיל, הוא לא BST.

- Inorder : 3 7 8 6 11 2 5 4 9
- Preorder : 2 7 3 6 8 11 5 9 4
- Postorder : 3 8 11 6 7 4 9 5 2



שלב 2: Function Templates

שלב בונוס AVL	מימוש BST עם Templates	תרגול כללי Templates	מימוש BST עבור מחרוזות
------------------	---------------------------	-------------------------	---------------------------

סעיף א' – Function Templates

ממשו את הסעיפים הבאים בקובץ **Functions.h**

סעיף א' – compare

כתבו template לפונקציה בשם **compare**, אשר מקבל שני פרמטרים מאותו טיפוס. הפונקציה מחזירה **1** אם הראשון גדול יותר מהשני, **0** אם הם שווים, **1** אם הראשון קטן יותר מהשני.

שימו  במידה והטיפוס הוא אובייקט, **חובה** עליו לתמוך באופרטורים: $<$ ו $=$ (החובה חלה רק על האופרטורים האלו)

סעיף ב' – bubbleSort

כתבו template לפונקציה בשם **bubbleSort**, אשר מקבלת **מערך של אובייקטים** מטיפוס כלשהו ואת **גודל המערך**. הפונקציה תמיינ את המערך מהקטן לגדול.

סעיף ג' – printArray


כתבו template לפונקציה בשם **printArray**, המקבלת **מערך של אובייקטים** מטיפוס כלשהו ואת **גודל המערך**. הפונקציה תדפיס את המערך, כל איבר במערך בשורה נפרדת.

סעיף ד' – בדיקות

בקובץ ה-main של שאלה 2 בדרייב ישנה דוגמא לבדיקות בשביל הפונקציות שכתבתם עם **float**.

a. הוסיפו בדיקות גם עבור **char**

b. הוסיפו בדיקות עבור **אובייקט של מחלקה** משלכם/ – מחלקה פשוטה שמכילה member אחד. למען הפשטות אפשר שהוא יהיה פומבי.

שימו  , כדי שהפונקציות יעבדו על מחלקה כלשהי, עליה לממש אופרטור $<$, $=$ ואופרטור $<<$ כי פונקציות ההשוואה וההדפסה מפעילות את האופרטורים הללו על הסוג הטמפלייטי **T**.
כדי לממש את $<$ ו $=$ אפשר לנסות לבד, ואם נתקעים אפשר למשל לחפש בגוגל `implement operator`, להיכנס לקישור השני ולגלול לאזור המתאים.

כדי לממש את $<<$ אפשר להיעזר בתרגילים 3 (הבונוס), 4 או לחפש בגוגל `implement operator` $<<$

סעיף ב' – Class Templates

אחרי שהתנסינו בבתיבת פונקציות בסיסיות עם **templates**, וגם תרגלנו שימוש בהן, נוכל לתקן בעיה שצצה בתחילת הסמסטר.

היזכרו בתרגיל 2, הנה קישור להצעת פתרון:

<https://drive.google.com/drive/u/0/folders/1v2tQwjOmHoR6WfROcc8N7YjmWF0svQ53>

בתרגיל כתבנו מחלקות שונות שביחד יצרו מערכת של רשת חברתית. יחד עם התרגיל, ניתנו לנו 3 קבצים שבהם היו מוגדרות מחלקות של רשימות מקושרות:

- **DeviceList** - רשימה שמיועדת להחזיק אובייקטים של **Device**
- **UserList** - רשימה שמיועדת להחזיק אובייקטים של **User**
- **ProfileList** - רשימה שמיועדת להחזיק אובייקטים של **Profile**


כבר בתחילת הסמסטר היינו צריכים להרגיש שמהו פה לא כל כך יעיל, כי אנחנו חוזרים על אותו קוד 3 פעמים, וכל מה ששונה זה רק ה-**type** שהרשימות מחזיקות. אז בשביל לתקן את זה יצרנו מחלקות חדשות שנקראות **GenericList** ו-**GenericNode**. גם יצרנו קובץ בשם **testList.cpp** שבו נעשה שימוש ברשימה החדשה.

ההגדרה של המחלקה נמצאת בקובץ **GenericList.hpp** *שניתן עם קבצי התרגיל, אבל **היא לא שלמה**. לאורך הקובץ תוכלו למצוא חלקים חסרים בקוד שבהם כתוב **/* complete code here */**, שצריך להשלים כדי שהקובץ יתקמפל והטסטר שבו מוגדרת פונקציית ה-**main** יוכל לרוץ.

השלימו את ההגדרה וודאו שהקובץ יוכל לרוץ.

אנא צרו **פרויקט VS חדש** בשביל החלק הזה, ודאגו להוסיף אליו את כל הקבצים שבתיקייה '2' (שב-**drive**):

```
Device.cpp
Device.h
GenericList.hpp
Page.cpp
Page.h
Profile.cpp
Profile.h
testList.cpp
User.cpp
User.h
```

שימו , אם אתם/ן לא בטוחים/ות איך להשלים את החלקים השונים, תוכלו להסתכל על המימוש המקורי של מחלקות ה-**List** בפתרון של תרגיל 2. מעבר לכך, המימוש של המחלקה מופיע באופן מלא באותו קובץ, ומכיל חלקים מסוימים שבהם תוכלו להיעזר כדי להשלים את הקוד שבהגדרת המחלקה.

* קובץ **hpp**

קובץ עם סיומת **hpp**. הוא קובץ שמכיל הגדרות (בדומה לקבצי **h**) וגם עשוי להכיל מימוש (כמו קבצי **cpp**). כשמגדירים מחלקות **templates**, חייב לכתוב גם את ההגדרות וגם את המימוש באותו הקובץ, ולכן זה מאוד נפוץ להגדיר מחלקות כאלו בקבצי **hpp**.

שלב 3: BST עם Templates

שלב בונוס AVL	מימוש BST עם Templates	תרגול כללי Templates	מימוש BST עבור מחרוזות
------------------	---------------------------	-------------------------	---------------------------

בחלק האחרון של התרגיל ניצור עץ חיפוש בינארי גנרי, שיכול להחזיק סוגי משתנים שונים. מטעמי נוחות ופשטות, יש ליצור **פרויקט חדש** עבור הסעיף הזה (ולא לשנות את הקבצים של סעיף א') את ההגדרות והקוד בבקשה כתבו בקובץ חדש **BSNode.hpp**.

סעיף א' – יצירת עץ חיפוש מטומפלט

הוסיפו לפרויקט החדש קובץ חדש בשם `BSNode.hpp`, הגדירו בו מחלקת `template` בשם `BSNode` והוסיפו אליה את ההגדרות והמימושים שכתבתם בתרגיל 1. אחרי שסיימתם/, נסו ליצור עצי חיפוש בינארי מסוגים שונים כמו `float`, `int`, או **אובייקטים מורכבים** (כמובן, בכל עץ רק `type` אחד).

סעיף ב' – שימוש ב-BSNode הגנרי למיון מערך

השתמשו ב-`BSNode` החדש כדי **למיין מערך**!

- כתבו קובץ `main`, וצרו בו שני מערכים לא ממוינים בגודל 15 – אחד של `int`-ים, ואחד של `string`-ים.
- הדפיסו את תוכן המערכים הלא ממוינים עם רווחים בין האיברים, וירידת שורה בין המערכים.
- הכניסו את שני המערכים לשני עצים נפרדים:
`BSNode<int>`, `BSNode<std::string>`
- הדפיסו את שני המערכים לאחר המיון מתוך העץ בעזרת הפונקציה `printNodes` שכתבתם בשאלה 1.

הנחיות והערות חשובות:

- חשוב שגם ההגדרות וגם המימוש יהיו באותו הקובץ (קובץ ה-`h`), אחרת יהיו שגיאות לינקג'.
- ניתן להשתמש במילה השמורה `typename` או `class`, לנו זה לא משנה.

נקודה למחשבה – מה **הסיבוכיות** של שיטת מיון חדשה זו?

בממוצע כל הכנסת איבר לעץ עולה $O(\log n)$.

כדי למיין עלינו להכניס n איברים לעץ, ולכן סיבוכיות מיון הכללית היא $O(n \log n)$ עם זאת, אם מערך הקלט ממין מראש, נקבל רשימה מקושרת ארוכה, ויעלה $O(n)$ להכניס איבר לעץ. לכן בחישוב ה- O notation נקבל שסיבוכיות המיון הכללית היא $O(n^2)$.

שלב בונוס: AVL

שלב בונוס AVL	מימוש BST עם Templates	תרגול כללי Templates	מימוש BST עבור מחרוזות
------------------	---------------------------	-------------------------	---------------------------



עצים מאוזנים

ישנם סוגים שונים של עצים, BST הוא רק אחד מהם, ובמקרה הממוצע הוא מספק ביצועים טובים מאוד. מימושים שונים של עצים הופכים להיות מאוד חשובים כאשר העצים מכילים מידע רב, כלומר עבור מ-ים גדולים מאוד. AVL הוא אחד מסוגי העצים שמאפשר לנו לקבל ביצועים טובים (אפילו במקרה שהכניסו קלט ממזין), ע"י כך שהוא תמיד שומר על עצמו מאוזן. כדי לממש אותו נצטרך להוסיף לעץ החיפוש הבינארי תכונות נוספות.

השבוע בבונוס תממשו עץ AVL עם templates

מוזמנים לאתגר את עצמכם! 💪


<https://drive.google.com/file/d/17GYAfgjYPBVfucuxrOUnbQQUzgWwCyiK/view?usp=sharing>

בהצלחה!

נספחים

הגשה ב-GIT

- את הפרויקט יש לנהל ב-Git, לפתוח repository חדש בתוך קבוצת ה-gitlab שלנו ושל המדריך/ה, ולהגיש לינק לפרויקט ב-NEO (אפשר לעשות comment עם הלינק או להגיש מסמך txt עם הלינק בפנים).
- יש להעלות ל-repository את כל הקבצים הרלבנטיים לתרגיל (קבצי txt, מסמכים, ומשאבים אחרים שבהם השתמשנו). חשוב להעלות את פרויקט ה-Visual Studio השלם ולהתעלם מקבצים לא נחוצים ([הנחיות במסמך הבא](#)), במידה ולא הועלה הפרויקט השלם, אין להעלות את שאר הקבצים שיוצר Visual Studio – הם רבים מאוד, הם לא מכילים מידע נחוץ להרצת הפרויקט אצל המדריך, ורק יוצרים בלגן.
- הבחירה אילו קבצים להעלות ל-repository נעשית באמצעות הפקודות add ו-rm. אופציה נוספת (מומלצת) היא להוסיף קובץ gitignore. אשר יתעלם מהקבצים הלא נחוצים. במידה ותרצו תוכלו להיעזר ב[סרטוני עזר בנושא GIT](#).
- כסיימתם/ן, בדקו שניתן להריץ את הפרויקט בקלות – בצעו Clone אל תיקייה במחשב אשר שונה מזו שעבדתם/ן, ותראו שהפרויקט נפתח ע"י לחיצה על קובץ ה-sln ויכול לרוץ בלי בעיה

 Ex9.sln

דגשים:

- את הפרויקט יש לפתוח בקבוצת ה-gitlab שאליה משותף/ת המדריך/ה כ-Maintainer.
- יש לוודא שכל הקבצים הרלבנטיים נוספו ל-repository (באמצעות הפקודה add), במידת הצורך ניתן להוריד קבצים מיותרים (באמצעות הפקודה rm).
- יש לבצע commit עבור כל סעיף, ובנקודות שבהן הוספנו שינויים חשובים (לפי הדגשים שהועברו בכיתה).
- עבור כל commit, זכרו לכתוב הודעה קצרה ואינפורמטיבית, שאפשר יהיה להבין מה היה השינוי בקוד.
- יש לדחוף את הקוד (באמצעות הפקודה push) ל-repository בסיום העבודה שלנו, חשוב שבסיום העבודה שלנו, ובמידה ונפנה למדריך/ה, ב-repository יהיה הקוד המעודכן ביותר.
- במידה ושכחנו או שאנחנו לא בטוחים איך מעלים קובץ, או מתעלמים מקבצים, כדאי לצפות בסרטוני ההדרכה בנושא GIT. ניתן לגשת לסרטונים בלשונית ה-resources שבכיתה ה-NEO.
- בסיום העבודה יש להגיש לכיתה ה-NEO קישור ל-repository.

כללי

1. יש לבדוק שכל המטלות מתקמפלות ורצות ב-VS2022. מטלה שלא תעבור קומפילציה אצל הבודק לא תיבדק **והניקוד שלה יהיה 0** 😞
2. יש לבדוק שהקוד שכתבתם עובד. יש להריץ בדיקות שלכם ולוודא שהקוד ברמה טובה.
3. כאשר אתם מתבקשים לממש פונקציה, ממשו בדיוק את הנדרש. אין להוסיף הדפסות וכדו'. אם הוספתם תוך כדי הבדיקות שלכם הדפסות, אנא דאגו להוריד אותם לפני ההגשה.
4. להזכירכם! העבודה היא עצמית, ואין לעשות אותה ביחד.
5. על כל שאלה או בעיה יש לפנות למדריך, לפחות 36 שעות לפני מועד ההגשה.

דגשים לתכנות נכון

- כדאי לקמפל כל מספר שורות קוד ולא לחכות לסוף! הרבה יותר קל לתקן כאשר אין הרבה שגיאות קומפילציה. בנוסף קל יותר להבין מאיפה השגיאות נובעות.
- כדאי לכתוב פונקציה ולבדוק אותה לפני שאתם ממשיכים לפונקציה הבאה. כלומר, כתבו תכנית ראשית שמשתמשת בפונקציה ובודקת האם היא עובדת כראוי. חישבו על מקרי קצה ונסו לראות מה קורה.
- בכל פעם שאתם מתקנים משהו, זכרו שיכול להיות שפגעתם במשהו אחר. לכן עליכם לבדוק שוב מהתחלה.
- חשפו החוצה רק את הממשק המינימלי הדרוש (minimal API), הגדירו את שדות המחלקה כפרטיים, וכמה שפחות מתודות כציבוריות.