



שבעת השיבולים - תרגיל צה"לי



רקע

עקב גידול האוכלוסייה בבסיס "שבעת השיבולים", נוצר מצב בעייתי מאוד במגורים של הבסיס. יש יותר מבקשי דיור מאשר דירות פנויות. הדבר גורם לבעיה גדולה של ניהול החיילים הישנים במגורים, קושי ביצירת סדר והוגנות, וחוסר שקיפות בהחלטות השיבוץ. חיילים שגרים בקריית שמונה מוצאים את עצמם חוזרים כל יום הביתה עד שימצא להם פתרון.

מפקד הבסיס, אל"ם שיבולי, הורה על הקמת מערכת דיגיטלית שתאפשר ניהול מיטבי של הבעיה.

עליכם לבנות מערכת שתוכל:

- לנהל בקלות את המגורים בבסיס

- לאפשר ליצור צדק באופן שבו משבצים חיילים למגורים
 - לספק תמונת מצב מלאה של תפוסת המגורים ומי נשאר בחוץ
-

הנחיות

1. עליכם לבנות מערכת (שרת API) לניהול מגורי חיילים בבסיס "שבעת השיבולים" בפייתון המערכת תנהל:

- חיילים המבקשים מגורים בבסיס
 - בתי מגורים - כרגע שניים (Dorm A, Dorm B), כל אחד מכיל 10 חדרים
 - בכל חדר יש 8 מקומות לחיילים (סה"כ 80 חיילים לבית מגורים, 160 לכל הבסיס)
 - רשימת המתנה לחיילים שלא נמצאו להם מקומות
2. שימוש בעזרים - מותר להשתמש בכל מה שלא AI (ולא AI-Mode)
3. אופן הבדיקה של התרגיל - הציון מחושב לפי דגש על פונקציונליות עובדת, SOLID, OOP, קונבנציות פייתון **לפי הסדר הזה**

תוצר המערכת יהיה Rest API שבבסיסו הלקוח יוכל לשלוח קובץ CSV של חיילים, והשרת יבצע שיבוץ אוטומטי לפי כללים מוגדרים מראש. פירוט מופיע בהמשך לפי שלבים.

על הרכיבים (כלל הפרטים על כל רכיב צריכים להופיע בקוד)

חייל

חייל הוא ישות בעלת הנתונים הבאים:

- מספר אישי (מתחיל ב8, ומכיל מספרים בלבד)
- שם פרטי
- שם משפחה
- מין (זכר או נקבה)
- עיר מגורים
- מרחק מהבסיס בקילומטרים (מספר)
- סטטוס שיבוץ:

- שובץ למגורים
 - ממתין (לא שובץ)
-

בתי המגורים

בבסיס יש שני בתי מגורים בלבד:

- בית מגורים א'
- בית מגורים ב'

כמובן שכמו בחיים האמיתיים, בעתיד יכולים להיבנות בתי מגורים נוספים, ולכן הקוד צריך להיות פתוח להרחבות

כל בית מגורים מכיל:

- 10 חדרים
 - בכל חדר 8 חיילים
 - סה"כ 80 חיילים לבית מגורים, 160 חיילים בבסיס כולו
-

בסיס שבעת השיבולים

בסיס "שבעת השיבולים" הוא המקום שבו נמצאים בתי המגורים, והחיילים. ניתן להניח שכל חייל שמגיע לבסיס באידיאל היה רוצה לקבל חדר שינה בו.

מערכת השליטה (API)

מערכת השליטה היא הממשק של המשתמש (איש שלישות, מש"קית ת"ש, קצינת קישור וכדומה) מול המערכת.

באמצעות מערכת השליטה המשתמש יוכל:

- לייבא את רשימת החיילים מקובץ CSV (העלאת CSV)
- להריץ שיבוץ מחדש (לפי כללי קדימות)
- להציג דוחות מגורים
- להציג רשימת חיילים שלא שובצו
- ועוד דברים שיתווספו בהמשך

אופן הגשה

הגשת התרגיל תתבצע באמצעות GIT, ל-repository אישי המיועד לתרגיל.

- הפיתוח ייעשה ב-Branched בשם **dev**
- בתוך Branch ראשי, **master** (או **main**), תהיה תמיד גרסה עדכנית **שמתקמפלת ועובדת**.
- לאחר סיום כל שלב בתרגיל, יש לעשות **Merge Request** מ-**dev** אל **master**.

שימו לב:

- מומלץ לעשות הרבה **Commit**ים ל-**dev**, גם על שינויים קטנים.
- אין חובה שכל **Commit** ב-**dev** יהיה עובד - אבל מה שנכנס ל-**master** **חייב** להיות עובד.
- הקפידו על הודעות **Commit** ברורות שמתארות מה השתנה.
- **רק הcommit האחרון בMaster הוא זה שיבדק**

לקראת ההגשה (עשר דקות אחרונות של התרגיל):

- יש לוודא בפעם האחרונה כי **master** מכיל גרסה עדכנית **שמתקמפלת ועובדת**
- יש לצרף **dump** של המידע **db** שלכם (מדובר בקובץ אחד)
- במידה ויצרתם קבצי **csv** נוספים, צרפו אותם לפרויקט תחת התיקיה "csv"
- יש לוודא כי יש קובץ **README** עם פירוט מינימלי על הפרויקט
- יש למלא את [השאלון הגשה הסופי](#)

הערות

- חשבו על עיצוב המערכת בצורה סולידית (SOLID)
- השתמשו בעקרונות OOP שלמדתם:
 - הפרדה בין לוגיקת שיבוץ לבין ייצוג הנתונים
 - שימוש בממשקים/מחלקות מופשטות במקום **switch**ים ענקיים
 - פתיחות להרחבה וסגירות לשינוי (Open/Closed Principle)
- חשבו איך המערכת תתמודד עם הרחבות עתידיות:

- הוספת בית מגורים שלישי
- שינוי מדיניות עדיפויות
- הוספת תנאים מיוחדים (פרופיל, תפקיד, קבע/סדיר וכו')
- החלפה בין DBים מבלי לפגוע בליבת הקוד
- התייחסות למקרי קצה - יש מקרי קצה, למשל: כמות השורות באקסל נמוך מכמות החדרים בפועל. עליכם לפתור כל מקרה קצה שיש לפי שיקולכם
- יש לבצע ולידציות בהתאם לצורך ולשיקול דעתכם - למשל, בהעלאת אקסל עם מספר אישי שמכיל אות, אפשר לפסול את כל האקסל, או להחליט ששורה אחת באקסל לא תעלה. **אתם נדרשים לפתור סוגיות כאלה בעצמכם, בהתאם למחשבה של "מה הדבר הנכון ביותר לפי העקרונות שלמדתי?"**
- אתם אלו שמחליטים כיצד יראו המחלקות בקוד, ואיך הן מתקשרות אחת עם השנייה, ע"פ עקרונות SOLID ו-OOP
- שימו לב שבשלב א' וב' יקרה מצב שבו ריסוט האפליקציה (שיכולה לקרות אחרי כל שמירה של קובץ בגלל hot-reload) יגרור מחיקה של המידע. אתם יכולים לפתור את זה בכל דרך שתבחרו
- תוכלו לבדוק את התוצר שלכם בעזרת כלים שיודעים לבצע בקשות HTTP (כמו Postman, cURL)
- השמות של routen יהיו בדיוק כפי שהוגדרו בתרגיל. שאר השמות בתרגיל (מחלקות, משתנים וכו') יכולים להיבחר על ידכם וצריכים לעמוד בקונבנציות של פייתון

משימות

בשלים א'-ב' המידע ישמר בזיכרון RAM בלבד (לא בקובץ, ולא במסד נתונים, אלא - על ידי אובייקטים ומשתנים לפי המחלקות שהגדרתם) בשלב ג' נעבור לשמירת המידע בבסיס נתונים.

שלב א' (60%) - ממשו את מערכת השליטה

צרו API המאפשר קריאת CSV ושיבוץ ראשוני בהינתן CSV של חיילים [\(להלן דוגמה\)](#). אתם צריכים לשבץ אותם למגורים. שימו לב שיש חיילים שישארו בחוץ. סדר העדיפות של השיבוץ (הנקרא גם "אסטרטגיית השיבוץ") צריך להיות לפי קילומטר מהבסיס, לדוגמה:

1. חייל בעל מרחק של 100 קילומטרים מהבסיס ישובץ לפני חייל בעל מרחק של 99 קילומטרים

2. במידה וישנו אותו מרחק, אפשר לבחור באופן שרירותי

מימוש מפורט:

1. מימוש שרת

- צרו שרת בPython המסוגל לקבל בקשות HTTP לפי הכלים שלמדתם (למשל בFlask)

2. הוספת Route מסוג POST - /assignWithCsv

- הוסיפו לשרת שיצרתם Route מסוג POST שידע לקבל קובץ CSV
- הערה: במידה ואתם מסתבכים בשליחת הקובץ בבקשה עצמה, תוכלו להשתמש

בקובץ ישירות בשרת

3. קריאת קובץ CSV

- ממשו לוגיקה בשרת שקוראת את הCSV וממירה אותו לאובייקטים הרלוונטיים
- צרו עבור כל שורה את האובייקטים המתאימים במערכת.

4. שיבוץ החיילים במגורים

- התחילו לשבץ חיילים לפי סדר העדיפות לבתי המגורים ולחדרים עד שהבסיס מלא (160 מקומות).
- אם אין מקום פנוי - החייל נכנס לרשימת המתנה.
- אתם מחליטים איך נראה השיבוץ בקוד

5. תשובה למשתמש - החזירו תשובה בפורמט JSON לשימוש המשתמש

- החזירו בבקשת HTTP את הנתונים לפי הדרישות הבאות:
■ הדפיסו סיכום:

1. כמה חיילים שובצו בפועל

2. כמה נשארו ברשימת המתנה

■ הציגו עבור כל חייל:

1. שובץ? כן/לא

2. אם כן - בית מגורים + מספר חדר

3. אם לא - ציון שהוא ברשימת המתנה

■ חשבו על המבנה JSON המתאים ביותר להחזרת הנתונים המופיעים פה

6. בדקו את מה שעשיתם עם cURL או Postman:

- דוגמה לבקשת cURL שיכולה לעבוד:

```
curl -X POST http://localhost:5000/assignWithCsv \ -F "file=@soldiers.csv"
```

הערה - במידה וכבר בוצע שיבוץ בעבר ע"י המערכת, פנייה לroute שוב תדרוס את מה שהיה בעבר, ותפעל רק לפי המידע החדש

נקודה למחשבה:

מדיניות השיבוץ לבסיס יכולה להשתנות (למשל, בעתיד יתווספו תנאי קדימות אחרים כמו תפקיד, ותק, פרופיל, אוכלוסייה וכו').
נסו לבנות את המערכת כך שהלוגיקה של השיבוץ אינה "תפורה" ישירות למחלקת החייל, אלא ניתנת להחלפה/שינוי בקלות.

שלב ב' (20%) - אל"ם שיבולי דורש נתונים

בשלב זה תוסיפו למערכת יכולות דו"ח ובקרה.

צרו Route ים נוספים בAPI:

1. דוח תפוסה לפי בית מגורים - /space

○ מימוש Route מסוג GET המחזיר עבור כל בית:

■ מספר החדרים המלאים

■ מספר החדרים החלקיים

■ מספר החדרים הריקים

2. דוח רשימת המתנה - /waitingList

○ מימוש Route מסוג GET המחזיר את כל החיילים שלא שובצו, לפי סדר עדיפות השיבוץ.

3. חיפוש חייל לפי מספר אישי /search

○ מימוש Route מסוג GET המחזיר פרטים על החייל לפי מספר אישי:

■ שובץ?

■ אם כן - איפה (בית+חדר)

■ אם לא - ברשימת המתנה.

על התשובות לחזור כJSON (שאתם מחליטים איך נראה). כמו כן, אתם מחליטים כיצד לקבל את הפרמטרים בבקשת HTTP ועל הסטטוס חזרה

שלב ג' (20%) - התממשקות עם בסיס נתונים (DB)

רקע:

מה הבעיה במה שעשיתם עד עכשיו? כשסוגרים את התוכנית (או שהמחשב נכבה), הלך כל המידע במערכת!

מטרה:

להעביר את הנתונים במערכת (חיילים, בתי מגורים, חדרים, שיבוצים) לאחסון מתמשך בבסיס נתונים SQLite - לפי מה שנלמד בקורס.

משימות:

1. הקמת DB

- הקימו DB בעזרת בו תשתמשו בשביל לאחסן את המידע של המערכת

2. אתחול סכמה בinitializeScheme - DB

- הוסיפו לשרת שיצרתם Route מסוג Post שיאתחל את הסכמה בDB. אין צורך בשלב זה לשים מידע בDB.

- שימו לב! אתם מחליטים כיצד הסכמה תיראה, כמובן שאפשר לבחור ליצור כמה טבלאות

3. עדכון ה-API לעבוד עם DB

- כל Route משלב א' ו-ב' צריך לעבוד מול בסיס הנתונים:

■ קריאת CSV בשלב א' תכניס חיילים לטבלאות הרלוונטיות.

■ לוגיקת השיבוץ תעדכן את טבלאות השיבוץ/סטטוס.

■ דוחות (space, waitingList, חיפוש חייל) יופקו על בסיס שאילתות DB.

טיפ: חשבו על שימור ה-OOP וההפרדה הלוגית:

- ודאו שהמחלקות שכתבתם עדיין קיימות כאובייקטים לוגיים.

- הגישה לבסיס הנתונים לא "נזלגת" לכל מקום בקוד, אלא מרוכזת בשכבה מתאימה.

שימו לב שבסוף שלב זה סגירת התוכנית ופתיחתה מחדש לא אמורה להשפיע על זמינות הנתונים, מעכשיו המערכת שרידה!

שלב ד' (20%) - בונוס (מי שסיים את א'-ג')

💡 שימו ♥ - עדיף לסיים עד שלב ג' באופן מלא ובנוי טוב לפי העקרונות שנלמדו, מאשר לרוץ הלאה. במידה ושלב הבונוס פוגע בתוצר של שלבים א'-ג' אין להגיש אותו!

הרחיבו את המערכת כך שתתמודד טוב יותר עם שינויים לאורך זמן:

1. עזיבת חיילים - /release

- הוסיפו Route מסוג POST ל"שחרור" חייל מחדר לפי מספר אישי

- לאחר שחרור מקום:

- המערכת צריכה לשבץ **אוטומטית** את החייל הבא בתור מרשימת ההמתנה. ע"פ האסטרטגית שיבוץ שבה האקסל עלה

- עדכנו גם את בסיס הנתונים בהתאם (מחיקת שיבוץ / עדכון סטטוס).

2. הוספת אסטרטגיית שיבוץ חדשה - PazamStrategy

- הוסיפו לכל חייל שדה "דרגה". הדרגה תהיה 1-5 כאשר 5 זה הרמטכ"ל ו-1 זה טוראי
- בנו אסטרטגיה חדשה לפיה במידה ויש חיילים עם אותו מרחק מהבסיס, העדיפות תקבע לפי הדרגה.
- בבקשת `POST /assignWithCsv` - אותה מימשתם בעבר, הוסיפו אפשרות לבחור באסטרטגיה הבסיסית (מיון לפי מרחק בלבד) או החדשה (מיון לפי מרחק, ואם המרחק זהה אז דרגה).
אין לפגוע בבקשות POST שכבר עובדות. חשבו איך לעשות את זה בצורה OCP

3. הוסיפו Route מסוג Post חדש של הוספה במקום דריסה - `/appendWithCsv`

- הוסיפו לשרת שיצרתם Route חדש מסוג POST שידע לקבל קובץ CSV
- כל חייל חדש יתווסף למערכת. במידה וקיים לו חדר פנוי הוא ישובץ, במידה ולא, יצטרף לרשימת ההמתנה. במידה ויש ניסיון להעלות מספר אישי שכבר קיים, טפלו בזה לפי שיקולכם