<u>קורס תכנות מתקדם - תיעוד הפרויקט</u>

שמות חברי הקבוצה: הדס שניידר, דנה פוגץ', רואד ספייה, תום פשינסקי

<u>חלק 1 - Furniture Store Classes</u> (ה-Domain, קובץ Furniture.py)

אלה המחלקות שקשורות לרהיטים עצמם:

1. DiscountStrategy

זוהי מחלקה שמוגדרת כ-ABC לכל סוגי ההנחות האפשריות.

מוגדרת בה מתודה אבסטרקטית בשם get_discount(self) שמגדירה את אחוז ההנחה שכל מחלקה מעניקה.

השתמשנו ב-Strategy Pattern כדי לטפל בכל מקרה של הנחה:

1) ללא הנחה (NoDiscount) - זו ברירת המחדל עבור אובייקטים שהם רהיטים - מחזירה 0 (0 %).
2) הנחת חגים (HolidayDiscount) - מחזירה 15, כלומר תינתן הנחה של 15%.
3) הנחת VIP (VIPDiscount) - מחזירה 20, כלומר תינתן הנחה של 20%.
4) הנחת חיסול מלאי (ClearanceDiscount) - מחזירה 30, כלומר תינתן הנחה של 30%.

2. Furniture

עבור הרהיטים, יצרנו מחלקת אב בשם Furniture שתומכת במחלקה DiscountStrategy. כל רהיט מקבל תכונה של מס' רהיט, שם, תיאור, חומר, צבע, שנות אחריות, מחיר בדולרים, גודל, כמות במלאי, מדינת ייצור, סוג רהיט (ברירת המחדל היא Generic) ואסטרטגיית חישוב הנחה (ברירת המחדל היא NoDiscount).

היא תומכת ב-DiscountStrategy ע"י מתודה שמגדירה את אסטרטגיית ההנחה המתאימה:

set_discount_strategy(self, discount_strategy: DiscountStrategy)

להלן המתודות של המחלקה:

- **calculate_discount()** - מתודה אבסטרקטית שבאמצעותה מחשבים את סך אחוז ההנחה שתינתן.
- **apply_discount()** - מתודה אבסטרקטית שבאמצעותה מחשבים את מחיר הרהיט בהתאם לאחוז ההנחה.

  שתי המתודות האלו יקבלו מימוש עבור כל מחלקה יורשת של רהיטים בהתאם לתכונות הייחודיות שלה, כפי שהגדרנו.
- **price_with_discount(price: float, discount: float)** - מתודה סטטית שמחזירה את המחיר לאחר הנחה.

- **apply_tax(self, tax_percentage: float)** - מתודה המחזירה את המחיר לאחר מיסוי.
- **is_available(self)** - מתודה שבודקת האם הפריט קיים במלאי. אם הפריט לא קיים, מודפסת הודעה שאומרת שהפריט אינו קיים במלאי.

מלבד המחלקה Furniture, **יצרנו 5 מחלקות יורשות עבור כל רהיט: Chair, Table, Sofa, Bed, Wardrobe**, כאשר לכל רהיט יש את המאפיינים הייחודים שלו. כדי לייצר אובייקטים של רהיטים בצורה דינמית, מימשנו Factory Pattern באמצעות הגדרת המחלקה FurnitureFactory שמייצרת את הרהיטים. אלו המאפיינים הייחודיים של כל רהיט:

- כיסא - האם יש לו משענת יד.
- שולחן - הצורה של השולחן (מרובע, עגול), האם הוא שולחן נפתח.
- ספה - מספר מושבים, האם יש לה משענת.
- מיטה - גודל המיטה, האם יש לה מקום לאחסון חפצים.
- ארון בגדים - מספר דלתות, האם יש בו מראות.

חלק 2 - Store Inventory (קובץ Inventory.py)

ניהול מערכת המלאי והרהיטים נעשה באמצעות "מנהל מערכת מלאי" שמתריע על שינויים במלאי (Observer Pattern), וכן באמצעות המחלקה FurnitureFactory (מהקובץ Furniture.py) שמייצרת אובייקטים של רהיטים באופן דינמי. התבנית של Observer מאפשרת למערכת לרשום צופים (Observers) שמתעדכנים כל פעם שהמלאי משתנה.

אלה המחלקות שקשורות לניהול המלאי:

- **InventoryObserver** - מחלקה מסוג ABC שמגדירה ממשק לצופים - כל מחלקה שרוצה לקבל עדכון על שינויים במלאי חייבת לממש את ()update.
- **LowStockIdentifier** - יורשת מ-InventoryObserver ומתריעה במקרה שיש מעט מלאי (מתעדכנת כשהמוצר במלאי נמצא בכמות נמוכה מה-threshold).
- **Inventory** - אחראית לנהל את המלאי של הרהיטים, כולל הוספה, הסרה, עדכון כמות, חיפוש פריטים והצגת רשימת המלאי והתראות. המתודות של המחלקה הן:
  1. **add_observer()** - הוספת צופים לרשימת הצופים שיעודכנו על שינוי במלאי.
  2. **remove_observer()** - הסרת צופים מרשימת הצופים שיעודכנו על שינוי במלאי.
  3. **.notify_observer()**
  4. **get_furniture_type(self, furniture_type: str)** - מחזירה את סוג הרהיט לפי שמו.
  5. **add_item()** - הוספת רהיטים למלאי.
  6. **remove_item()** - הסרת רהיטים מהמלאי.

7. **update_quantity(self, name: str, furniture_type: str, new_quantity: int)** - עדכון כמות של פריט מסוים.
8. **search_by_type(self, furniture_type: str)** - חיפוש פריט ספציפי.
9. **search(self, **filters)** - חיפוש פריט לפי מאפיינים.
10. **get_all_items(self)** - הצגת כל המלאי.
11. **view_inventory()** - הצגת כל המלאי מסודר לפי קטגוריית פריט.
12. **check_low_stock(self, threshold=5)** - בודק סוגי פריטים שיש מעט מהם במלאי.

חלק 3 - User Management (קובץ User.py)

כל משתמש בחנות מיוצג כאובייקט בשם User. לכל משתמש יש שם, מייל, סיסמה, כתובת, היסטוריית הזמנות, wishlist (פריטים שהוא מעוניין להזמין בעתיד), אופן תשלום (ברירת המחדל "Credit Card"). אלה המתודות של User:

1. **add_observer()**.
2. **remove_observer()**.
3. **notify_observer()**.
4. **validate_password(password: str)** - מתודה סטטית שבודקת אם הסיסמה שהמשתמש בחר עומדת בדרישות.
5. **hash_password(self, password: str)** - מחברת את הסיסמה וה-salt והופכת אותן ל-string מוצפן.
6. **check_password(self, password: str)** - עושה "reverse" לסיסמה המוצפנת ובודקת אם היא אכן שקולה לסיסמה הרגילה של המשתמש.
7. **update_profile()** - עדכון פרופיל (המשתמשים יכולים לעדכן את שמם וכתובת המשלוח).
8. **update_payment_method()** – מאפשרת למשתמש לעדכן את אמצעי התשלום.
9. **add_order_to_history()** - מוסיפה הזמנה חדשה להיסטוריית ההזמנות.
10. **view_order_history()** - צפייה בהיסטוריית ההזמנות.
11. **add_to_wishlist(self, item)** - הוספת פריט ל-wishlist.
12. **remove_from_wishlist(self, item)** - הסרת פריט מה-wishlist.
13. **view_wishlist(self, item)** - הצגת ה-wishlist.
14. **שמירה וטעינת משתמש מקובץ CSV -** המידע על המשתמשים שמור בקובץ csv שנקרא users_database.csv. המידע שנשמר הינו שם המשתמש, מייל, סיסמה, salt, hashed_password, כתובת, אמצעי תשלום, היסטוריית הזמנות ו-wishlist. באמצעות המתודה **save_to_csv()**, ניתן לשמור בקובץ

משתמשים חדשים או לעדכן פרטים של משתמשים קיימים. באמצעות המתודה **load_from_csv()**, ניתן לגשת למשתמש שנמצא בקובץ זה.

**דוגמת שימוש בקובץ נתוני יוזרים:

```
user = User("Alice", "alice@example.com", "securepass", "123 Main St")
user.add_to_wishlist("Luxury Chair")
CSV user.save_to_csv()
CSV loaded_user = User.load_from_csv()
print(loaded_user.view_wishlist())
```

חלק 4 - Shopping Cart (קובץ shopping_cart.py)

החלק הזה מתייחס לניהול עגלת הקניות. יצרנו את המחלקה **CartObserver** שמוגדרת כ-ABC, ויש לה מתודה אבסטרקטית **update(self, cart, change_type, item=None)** (שימוש ב- Observer Pattern).

למחלקה הזו יש מחלקה יורשת בשם CartNotifier שמתריעה כאשר מוסיפים או מסירים פריט מהעגלה באמצעות המתודות add_item ו-remove_item (מתבצעת בדיקת מלאי אוטומטית לפני הוספת פריט כדי לוודא שלא מוסיפים יותר ממה שקיים במלאי, ובדיקה כי כמות המלאי ושם הפריט תקינים).

יש מחלקה נוספת בשם ShoppingCart, שבה ליוזר יש attribute של מילון שמייצג את העגלה שלו, וכן attribute של אסטרטגיית ההנחה.

עבור המימוש של ה-Observer Pattern, מוגדרות 2 מתודות add_observer() ו- notify_observers().

- המתודה **set_cart_discount_strategy()** מגדירה את סוג ההנחה שצריך ליישם על ההזמנה.
- המתודה **view_cart()** נועדה כדי לצפות בתוכן העגלה.
- המתודה **calculate_total()** נועדה כדי לחשב את סכום העגלה הסופי לתשלום, תוך יישום הנחות ומס.
- כללנו 3 מתודות האחריות לניהול קבצי csv של עגלות קניות:

1) מתודת **save_cart_to_csv()** ששומרת את פרטי העגלה של כל משתמש לקובץ csv לפי כתובת המייל שלו.

2) מתודת **load_cart_from_csv()** שמשחזרת את העגלה האחרונה של המשתמש - עבור מקרה שבו המשתמש לא יאבד את הפריטים אם יצא והתחבר מחדש, או למשל אם המשתמש התחבר לאתר ומילא עגלת קניות בעבר אבל עדיין לא הזמין אותה - האתר יראה למשתמש את העגלה האחרונה שמילא.

3) מתודת **clear_cart_from_csv()** שמוחקת את העגלה מקובץ ה-csv, לאחר שהמשתמש ביצע תשלום (כלומר לאחר שהמשתמש ביצע checkout).

חלק 5 - Checkout Process (קובץ shopping_cart.py)

תהליך התשלום - המתודה checkout()

- מתחילים בהדפסת תהליך התשלום - הדפסה של כתובת המשלוח ואת שיטת התשלום שבחר היוזר.
- מתבצע אימות זמינות המלאי לפני הרכישה - התממשקות עם Inventory.
- לאחר מכן מחשבים את המחיר הכולל של העגלה בעזרת המתודות הרלוונטיות.
- כללנו תהליך תשלום מדומה עם Mock ע"י המתודה הסטטית **process_payment**.
- אחר כך ממשיך הממשק עם Inventory, כאשר המלאי מתעדכן עם יצירת וביצוע ההזמנה.
- בשלב ביצוע ההזמנה והשלמתה נוצר הממשק עם Order, כאשר אנחנו מבצעים את ההזמנה ושומרים אותה לקובץ ה-csv של ההזמנות ע"י קריאה למתודה **save_order_to_csv()** אבל מסירים את העגלה מקובץ ה-csv של העגלות ע"י המתודה **clear_cart_from_csv()**.
- לבסוף אנחנו מוסיפים את ההזמנה להיסטוריית ההזמנות של היוזר. לאחר מכן העגלה של היוזר מתרוקנת ואנחנו מחזירים את ההזמנה שבוצעה.

חלק 6 - Order Management (קובץ order.py)

ניהול הזמנות

- כדי להטמיע את מחלקת Order שתאחסן את פרטי הרכישה של כל יוזר, שמרנו את מס' ההזמנה, היוזר, הפריטים, המחיר הכולל של ההזמנה, סטטוס ההזמנה (ברירת המחדל היא "Pending"), כתובת המשלוח ואת אמצעי התשלום. כל הזמנה תשמור כעת את כתובת המשתמש שנמצאת בפרופיל שלו ואמצעי התשלום שבחר עבור ההזמנה.
- המתודה **complete_order()** - כאשר ההזמנה מסתיימת בהצלחה, סטטוס ההזמנה הופך ל-"Completed".
- המתודה **mark_as_delivered()** - אם ורק אם ההזמנה נשלחה בהצלחה, סטטוס ההזמנה הופך ל-"Delivered".
- המתודה **__str__()** – הדפסה של ההזמנה הכוללת את כל הפריטים שלה, כולל את כתובת המשלוח ואמצעי התשלום.

- מתודה של שמירת הנתונים של ההזמנות לקובץ csv ושחזור שלהם ע"י **save_to_csv()** ו-**load_orders_from_csv()**.

חלק 7 - RESTtful API

בחלק הזה מימשנו API לניהול חנות הרהיטים באמצעות flask. המערכת מאפשרת:

- רישום משתמשים
- חיפוש מוצרים (ניהול קטלוג)
- הוספה והסרה של פריטים לעגלה
- ביצוע רכישה
- ניהול מלאי

לשם חלוקת הפונקציונליות חילקנו את הגישה באתר ל-2 סוגי משתמשים:

1) בעל העסק (Admin) = גישה לניהול מלאי, משתמשים והזמנות.
2) לקוחות (Client) = גישה לחיפוש רהיטים, ניהול עגלה ורכישות.

- גישה נאיבית שיישמנו: יצרנו מילון admins שמזהה את סוג המשתמש לפי כתובת המייל שלו- אם היא מכילה כתובת עם הצירוף admin.

איך החלטנו ליישם את ההפרדה בפועל?

1) יצרנו **prefix נפרד** לנתיבים עבור הלקוחות ובעלי החנות:
   a) **/admin/...** – פעולות לבעלי החנות
   b) **/cart/, .../furniture/...** – פעולות ללקוחות
2) **בדיקת הרשאות ליוזרים** - השתמשנו בכמה מנגנונים לבדיקת הרשאות בהתאם לרמת הגישה הנדרשת לכל פעולה.

מנגנון האימות הראשי (JWT Token Authentication)

השתמשנו ב-JWT כדי לוודא שמשתמשים מחזיקים בהרשאות המתאימות. לשם כך ייבאנו את הספרייה **HTTPTokenAuth** של Flask. כדי לתמוך בפעולות ה-HTTP וה-decorator, יצרנו אובייקט שלו בשם auth, כדי להשתמש בו כ-decorator שמסומן כ- @auth עבור המתודות שנראה בהמשך שדורשות מנגנון התחברות - הדקורטור הוא **@auth.verify_token** .

זיהוי משתמשים

מימשנו את המתודה **generate_token()** שאחראית לחולל עבור משתמש שנכנס למערכת Token ייחודי עם מידע על סוג המשתמש: האם הוא Admin או Client, כתובת המייל שלו ותוקף.

לצורך שחזור ה-Token השתמשנו במתודה **get_jwt_token()** שמקבלת כפרמטר משתמש ומחזירה את ה-Token שלו.

דאגנו גם לבדוק את תקינות / תוקף ה-Token לפני כל פעולה מאובטחת ע"י המתודה **verify_token()**. כלומר, בדקנו בכל פעולה מאובטחת אם יש Token חוקי באמצעות המתודה הזו. מכאן, במידה וה-Token לא תקף, מוחזרת שגיאת 401 Unauthorized והבקשה נדחית.

זיהוי משתמש שהוא Admin

בדקנו האם משתמש הוא Admin באמצעות מתודת עזר **is_admin()** שמקבלת כפרמטר כתובת מייל. אם היא מכילה את הצירוף "admin", היא מחזירה True, אחרת היא מחזירה False. היא מקבלת עטיפה של **@auth.verify_token** כדי לשמש אותנו בפעולות המוגדרות עבור בעל העסק.

הפרדת הרשאות של Admin לעומת Client

כל ההרשאות מאוחדות ונשלטות במקום אחד. נבדיל בין סוגי הפעולות:

1. **פעולות שנגישות ללא צורך בהרשמה / כניסה לאתר** - את הפעולות האלו הגדרנו ע"י המתודות הבאות:

1. <u>הרשמה</u> - ע"י המתודה **register_user()** שבה המשתמש יוצר חשבון עם אימייל וסיסמה. היא דורשת מתודת **POST** (כדי לרשום יוזרים חדשים). נשים לב כי הגדרנו שיוזרים מזוהים בהזמנות שלהם לפי ה-email שלהם לכן קודם כל כללנו בדיקה האם המייל של הלקוח כבר קיים במאגר. אם כן, נוציא הודעת שגיאה (400) שהמשתמש כבר קיים במערכת. אחרת, ניצור instance חדש של User עם הפרטים הרלוונטיים , שגם כן יזוהה במאגר ע"י המייל שלו. לבסוף נוציא הודעת אישור למסך כי היוזר אכן נרשם בהצלחה.
2. <u>התחברות/כניסה</u> - ע"י המתודה **login_user()** שבה המשתמש מזין את כתובת המייל והסיסמה שלו. היא דורשת מתודת **POST** (כדי לשלוח לשרת את הנתונים איתם צריך להיכנס לאתר כדי לוודא שהם אלו הנתונים ששמורים

עבורו במערכת). אם יש שגיאות בתיאום הפרטים (למשל אם אחד מהפרטים אינו נמסר / המייל אינו נמצא / הסיסמה/מייל/Token שגויים) נוציא הודעות שגיאה מתאימות. אחרת, נקבל את הפרטים של היוזר.

3. צפייה בכל הרהיטים הקיימים באתר - ע"י המתודה **get_furniture()** . היא דורשת מתודת **GET** כי אנחנו שולפים את הנתונים על המלאי שקיים במאגר של האתר, ע"י שימוש בפונקציה ב-get_all_items() של inventory.py.
4. חיפוש רהיט מסוים - ע"י המתודה **search_furniture()** . היא דורשת מתודת **GET** כי אנחנו שולפים את הנתונים על המלאי שקיים במאגר של האתר, ע"י שימוש בפונקציה ב-search() של inventory.py. אם כל הפרטים המתאימים (שם +סוג רהיט) נמסרו בהתאם, נחזיר את הפרטים הבאים של הפריט: מק"ט, שם, סוג, מחיר וכמות זמינה של הפריט.

2. פעולות שנגישות רק ללקוחות רשומים (*Client Endpoints*) - בעצם כל מתודה שמתאימה תקבל עטיפה של הדקורטור @auth.verify_token , כלומר נאשר שה-Token של המשתמש תקין ולכן הוא יוכל לעשות את הפעולות המוגדרות לו. את הפעולות האלו הגדרנו ע"י המתודות הבאות:

1. צפייה בתוכן העגלה - כללנו מתודת **view_cart()** שדורשת מתודת **GET** (כדי לצפות בפריטי העגלה הנוכחיים, לכן מתבצעת פה שליפת נתונים מהשרת). אם למשתמש אין עגלה ששמורה ב-DB, ניצור לו עגלה חדשה עם הפרטים שלו. אם כבר יש לו עגלה, נחזיר אותה (השתמשנו פה במתודת **view_cart()** שמימשנו ב-Shopping_cart.py).
2. הוספת פריט לעגלה - כללנו מתודת **add_to_cart()** שדורשת מתודת **PUT** (כי אנחנו משנים כמות של פריט בעגלה). כללנו בדיקה שהמלאי הקיים אכן מספיק לעדכון שהיוזר מעוניין בו (אם המלאי אינו מספיק, מוחזרת שגיאה 400), שהמייל של היוזר אכן כלול במאגר ההזמנות (כדי שיוכל לעדכן אותה), אם הפריט אינו נמצא במלאי (מוחזרת שגיאה 404) . השתמשנו פה במתודת **add_item()** שמימשנו ב-Shopping_cart.py.
3. הסרת פריטים מהעגלה - כללנו מתודת **remove_item_from_cart()** שדורשת מתודת **DELETE** (כדי למחוק פריט שהיה קיים בעגלה), וכן השתמשנו במתודת **remove_item()** שמימשנו ב-Shopping_cart.py. התייחסנו למקרים בהם:
   - חסרים פרטים בבקשת המשתמש (מוחזרת שגיאה 400).
   - הפריט הרצוי אינו נמצא במלאי / לא נמצאה העגלה של המשתמש / אם הפריט הרצוי אינו נמצא בעגלה עצמה (מוחזרת שגיאה 404).
   - אם לא אירעה אף אחת מהשגיאות המופיעות לעיל, נדפיס הודעה על איזה פריט הוסר מהעגלה.

4. ביצוע רכישה - כללנו מתודת **checkout()** שדורשת מתודת **POST** (כדי לשלוח לשרת את ההזמנה שהלקוח ירצה לבצע עם הפרטים הרלוונטיים). אם תהליך הרכישה נכשל (המשתמש לא נמצא / העגלה ריקה / בעיות מלאי /

תהליך התשלום נכשל) אז מוחזרת שגיאה 400 ותצא הודעה שתהליך הרכישה נכשל. אחרת, תהליך התשלום הושלם ותודפס הודעה על פרטי ההזמנה של היוזר ופרטי הרכישה (סטטוס, אמצעי תשלום).

השתמשנו פה במתודת **checkout()** שמימשנו ב-Shopping_cart.py.

5. <u>שמירת עגלת הקניות של המשתמש</u> - כללנו מתודת **save_cart_to_csv()** שדורשת מתודת **POST** (כדי לשלוח לשרת את פרטי העגלה של הלקוח) כדי לשמור אותה ב-DB של האתר. אם כתובת המייל של המשתמש אינה שייכת להזמנות האחרונות (שכוללות את זו שרכש כעת), תצא הודעה שלא נמצאה עגלה למשתמש (מוחזרת שגיאה 404). אחרת, נשמור אותה ל-CSV ונוציא הודעה שהעגלה נשמרה בהצלחה.
6. <u>שחזור עגלת הקניות של המשתמש</u> - כללנו מתודת **load_cart_from_csv()** שדורשת מתודת **GET** (כדי לצפות בעגלה האחרונה של המשתמש, לכן מתבצעת פה שליפת נתונים מהשרת). אם כתובת המייל לא מופיעה בהזמנות, ניצור הזמנה חדשה. אחרת, נחלץ את העגלה ששמורה ב-shopping_carts עם כתובת המייל של המשתמש ונטען אותה.

3. פעולות שנגישות רק למנהלים (*Admin Endpoints*) - כאן אנחנו מתייחסים לכל הפונקציות שמשנות נתונים.לכן הן מוגנות בעטיפה של הדקורטור auth.verify_token@ וגם של הדקורטור admin_required@.

הדקורטור מיועד למנהלים בלבד ובודק אם המשתמש מחזיר Token תקף. אם הוא לא Admin, מוחזרת שגיאה 403 Forbidden. כלומר, בדיקת ההרשאות הזו מבטיח שרק מנהלים מבצעים פעולות רגישות לנתונים, כמו צפייה במלאי וניהולו, צפייה בהזמנות לקוחות וניהול משתמשים.

כדי לעדכן את ה-Observers על מצב בו משתמש מתעדכן / נמחק, השתמשנו במתודה **notify_observers()** שהגדרנו ב-User.py. לאחר מחיקת משתמש, מוחקים אותו מהמילון שהגדרנו עבור כל סוגי היוזרים במערכת = users_roles.

דרך התמודדות עם הנתונים:

1. הגדרנו 2 פונקציות עזר לטעינת נתונים ושמירתם: עבור טעינה נעזרים ב-**load_data()**, ועבור שמירה נעזרים ב-**save_data()** ע"י תמיכה של **JSON**.
2. כדי להגיע לקבצי הנתונים, הגדרנו נתיבים עבור הקבצים של ניהול משתמשים, ניהול הזמנות (קיימות) , ניהול עגלות (כאלו שנשמרו עבור משתמשים שעוד לא רכשו אותן).

***עבור ניהול משתמשים, עגלת קניות והזמנות -** כללנו פונקציות של שמירה וטעינה של נתוני משתמשים מקבצי JSON :

- *load_users_json() / load_carts_json()/ load_orders_json()*
- *save_users_json() /save_carts_json()/ save_orders_json()*

***עבור ניהול המלאי** - יצרנו instance של Inventory כדי להתמודד עם <u>הרהיטים</u>.

3. עבור שמירת הנתונים שמתבצעים באתר, הגדרנו מתודה **save_all_data()** ששומרת את כל הנתונים החדשים לקבצי ה-**JSON**. ביציאה מה-API יש שימוש ב- **atexit.register(save_all_data)**. אנחנו נקרא לפונקציה הזו בכל פעם שנממש פונקציה שמשנה נתונים כמו **add_to_cart,()register_user,()checkout()** וכדומה.

****הבדיקות שהחלטנו לא לכלול בפרויקט הסופי****

יש כמה טסטים שהחלטנו לא לכלול לאחר נסיונות רבים לתיקון הקוד בקובץ app.py, זאת כדי שמכלול הקבצים יכיל כמה שיותר טסטים תקינים. אלו הטסטים:

test_remove_item_not_in_cart

test_add_item_not_in_inventory

test_load_cart_from_csv_via_api

test_save_cart_to_csv_via_api

test_admin_can_view_users