TromOti

Donating has never been easier

בניית מערכות ממוחשבות מבוססות אינטרנט (WEB)

חלק ג' – צד שרת

מגישים:

אלון קליימן	319126108
רועי פלדמן	311246227
נוי כשר	314963810
יניב ליפוביצקי	305917569

Github repository: https://github.com/alon222/web-project-g18

Branch: master

תוכן

שימוש בפרויקט הבסיס הנתון באתר

הפרויקט הראשוני שיצרנו הומר לקובץ flask במבנה של blueprints:

- קובץ ה-app.py הראשי מקושר לכל ה-blueprints של הדפים הרלוונטיים למימוש הפרויקט (Homepage, About, Contact, My Account, Main Menu).
- תיקיות static ו-templates מכילות קבצים סטטיים (תמונות, CSS או JS) ועמודי html בהתאמה, המשותפים לכל הפרויקט לדוגמה base.html שמכיל את הבסיס לכל העמודים, קבצי css שמשותפים לכל העמודים וקבצי js שמשותפים לכל העמודים.
- בתקיה components יש blueprint שמטרתו להכיל את כל ה-layout הקבוע של הדף שהם לדוגמה, header, footer והחלון הצף של Sign in.
- תיקיית pages מכילה בתוכה את ה-blueprints עבור כל עמודי האתר שהוזכרו לעיל. כל blueprints מכיל גם הוא את התיקיות static ו-templates וקובץ python עם כל הנתיבים הרלוונטיים לאותו עמוד.

יצירת בסיס נתונים

הרצת בניה של בסיס הנתונים
פרטי בסיס הנתונים שמוגדרים בקובץ env

```
DB_PORT = '3306'
DB_USER = 'root'
DB_PASSWORD = 'root'
DB_NAME = 'webprojectg18'
```

לא ניתן היה לבנות בסיס נתונים בשם web-group-18 קיבלנו שגיאות אז השם הוא webgroup18
בתיקיית utilities בקובץ initialize_db.py כאשר מריצים אותו הוא בונה את בסיס הנתונים ומאכלס כמה דוגמאות אבל קודם נעשה שימוש בספרייה שחייב להתקין
Pip install phonenumbers

בתיקיית utilities בקובץ initialize_db.py יצרנו את בסיס הנתונים של האתר. בסיס הנתונים כולל שלוש טבלאות:

- <u>טבלת users:</u>

```
_CREATE_TABLE_SQL = "CREATE TABLE if not exists users" \
                    " (`id` INT AUTO_INCREMENT PRIMARY KEY," \
                    " `username` varchar(255) NOT NULL," \
                    " `email` varchar(255) UNIQUE KEY NOT NULL," \
                    " `phone_number` varchar(255) UNIQUE KEY NOT NULL," \
                    "`password` varchar(255) NOT NULL)"
```

- <u>טבלת donations:</u>

```
_CREATE_TABLE_SQL = "CREATE TABLE if not exists donations" \
                    " (`id` INT AUTO_INCREMENT PRIMARY KEY," \
                    " `category` varchar(255) NOT NULL," \
                    " `description` varchar(255) NOT NULL," \
                    " `available_until` TIMESTAMP NOT NULL," \
                    " `address` varchar(255) NOT NULL," \
                    "`donating_user` INT NOT NULL," \
                    "`donation_image_path` varchar(255) NOT NULL," \
                    f"`availability_status` ENUM({_statues}) NOT NULL DEFAULT '
                    " FOREIGN KEY (donating_user) REFERENCES users(id) )"
```

- <u>טבלת donation_assignment (הקישור בין משתמש לתרומה):</u>

```
_CREATE_TABLE_SQL = "CREATE TABLE if not exists donation_assignment" \
                    " (`id` INT AUTO_INCREMENT PRIMARY KEY," \
                    " `requested_user` INT NOT NULL," \
                    " `donation` INT NOT NULL," \
                    " FOREIGN KEY (requested_user) REFERENCES users(id), " \
                    " FOREIGN KEY (donation) REFERENCES donations(id)," \
                    " UNIQUE (donation) )"
```

את יצירת בסיס הנתונים עשינו בשלושה שלבים: ראשית, ביצענו בדיקת כפילויות כלומר בדקנו האם קיימות טבלאות ישנות עם אותו השם. אם כן, נמחק טבלאות אלו וניצור טבלאות חדשות. לבסוף, אכלסנו נתונים בטבלאות על ידי פונקציית ()populate_tables_data.

יצירת מחלקות

מחלקת donation

```
def __init__(self, category: DonationCategory, descrip
    self._category = category
    self._description = description
    self._available_until = available_until
    self._address = address
    self._donation_id = donation_id
    self._donating_user_id = donating_user_id
    self._availability_status = availability_status
    self._donation_image_path = donation_image_path
```

את השדה category הגדרנו על ידי אינומרטור:

```
class DonationCategory(enum.Enum):
    FRUITS = enum.auto()
    VEGETABLES = enum.auto()
    GRAINS = enum.auto()
    DAIRY = enum.auto()
    CLOTHS = enum.auto()
    FURNITURE = enum.auto()
    OTHER = enum.auto()
```

גם את השדה availability_status הגדרנו על ידי אינומרטור. כאשר המשתמש הנתרם מזמין מוצר המוצע לתרומה, הסטאטוס שלו הוא reserved. המשתמש התורם יכול לשנות את הסטאטוס ל-delivered או ל-available במידה והמוצר נשלח או נעשה זמין שוב, בהתאמה.

```
class DonationAvailabilityStatus(enum.Enum):
    AVAILABLE = enum.auto()
    RESERVED_FOR_USER = enum.auto()
    DELIVERED = enum.auto()
```

בנוסף, המחלקה כוללת מספר שיטות בניהן שיטות של קבלת נתונים ובדיקות ולידציה. לדוגמה, נבדוק שזמן איסוף התרומה לא חרג מזמן האיסוף המקסימלי המוגדר ל-24 שעות.

מחלקת donation management

מחלקה שמנהלת את התרומות באתר ואת התקשורת עם בסיס הנתונים.

שם השיטה	הסבר
create_table	יצירת הטבלה donations על ידי שימוש ב-db manager
delete_table	מחיקת הטבלה donations על ידי שימוש בפונקציית execute שניתנה לנו
get_all_available_donations	הצגת כל התרומות (לשימוש בדף הבית)
get_user_donations	הצגת כל התרומות של משתמש מסוים
get_donation	הצגת תרומה ספציפית
add_donation	יצירת תרומה חדשה על ידי שימוש בשיטה _insert_donation_to_db
_insert_donation_to_db	יצירת תרומה חדשה על ידי שימוש בפונקציית commit שניתנה לנו
_parse_or_raise	שלוש שיטות עבור השדות available_until, category ו-availability_status הממירה אותם לפורמט של date או של אינומרטור.
delete_donation	השיטה מקבלת מזהה של תרומה ומוחקת אותה
update_donation	השיטה מקבלת מזהה של תרומה, משתמש ואת השדות אותם מעוניינים לעדכן

מחלקת user

```
def __init__(self, username: str, email: str, phone_number: str,  user_id: Optional[int] = None, password_plaintext: Optional[str] = None, password_hashed: Optional[str] =
    self._user_id = user_id
    self._username = username
    self._email = email
    self._phone_number = phone_number
    assert bool(password_hashed) ^ bool(password_plaintext), 'Must provide either plain text password or already hashed password'
    self._password_hashed = self.hash_user_password(password_plaintext) if password_plaintext else password_hashed
```

הסיסמה של המשתמש עוברת הצפנה, כלומר הסיסמה שהמשתמש הזין אינה הסיסמה שנשמרת בבסיס הנתונים וכאשר המשתמש מתחבר לאתר נשווה בין ההצפנות.

```
@staticmethod
def hash_user_password(password_plaintext: str) -> str:
    return hashlib.sha1(password_plaintext.encode('utf-8')).hexdigest()
```

בנוסף, המחלקה כוללת מספר שיטות בניהן שיטות של קבלת נתונים ובדיקות ולידציה על השדות טלפון, אימייל (על ידי regex שהוגדר מראש) ושם משתמש.

מחלקת user management

מחלקה שמנהלת את המשתמשים באתר ואת התקשורת עם בסיס הנתונים.

שם השיטה	הסבר
create_table	יצירת הטבלה users על ידי שימוש ב-db manager
delete_table	מחיקת הטבלה users על ידי שימוש בפונקציית execute שניתנה לנו
register_user	יצירת משתמש חדש על ידי שימוש בפונקציית commit שניתנה לנו
delete_user	השיטה מקבלת מזהה של משתמש ומוחקת אותו
get_all_users	החזרת את כל המשתמשים באתר
get_users_from_email	שיטה שמקבלת מזהים של משתמשים ומחזירה רשימה שלהם
get_user_from_email	שיטה שמקבלת מזהה של משתמש ומחזירה אותו
_get_users_from_raw_details	הפיכת משתמש מבסיס הנתונים לאובייקט של משתמש
authenticate_user	אימות משתמש כאשר הוא מתחבר לאתר על ידי המזהה שלו (אימייל) ועל ידי סיסמה שעוברת הצפנה
update_user_info	השיטה מקבלת מזהה של משתמש, ואת השדות אותם מעוניינים לעדכן

מחלקת user donation assignment

מחלקה המקשרת בין משתמש לתרומה. על מנת לעשות זאת, נשתמש בשיטה assign_donation_to_user אשר מקבלת את המזהים של המשתמש ושל התרומה אשר משנה את הסטאטוס של המוצר ל"שמור". את המזהה של המשתמש שמבקש ניקח דרך ה-session.

בנוסף יש שיטה בשם delete_assign כדי להוריד הצמדה של מוצר בין שני משתמשים כאשר הבעלים של המוצר יכול לבצע בעמוד ניהול המשתמש שלו כאשר מוצר נתפס.

שאילתות SQL

השאילתות הן במטרה לממש את השיטות של המחלקות שהצגנו.
השתמשנו בהרבה שאילתות כי נדרשנו לנהל הרבה מרכיבים דינמיים שהם תרומות עם כל הסטאטוסים שלהן, ההצמדה של התרומות ללקוחות, ולנהל רישום לקוחות.

מחלקת donation management

```
_DELETE_TABLE_SQL = "DROP TABLE donations;"

_INSERT_DONATION_SQL = "INSERT INTO donations (category, description, available_until, address, donating_user, donation_image_path) VALUES (%s, %s, %s, %s, %s, %s)"
_GET_ALL_AVAILABLE_DONATIONS_SQL = f"SELECT * FROM donations WHERE availability_status = %s AND available_until > now()"
_GET_ALL_USER_DONATIONS_SQL = "SELECT * FROM donations WHERE donating_user = %s;"
_GET_DONATION_SQL = "SELECT * FROM donations WHERE id = %s;"
_DELETE_DONATION_SQL = "DELETE FROM donations WHERE id = %s AND donating_user = %s;"
```

מחלקת user management

```
_DELETE_TABLE_SQL = "DROP TABLE users;"
_INSERT_USER_SQL = "INSERT IGNORE INTO users (username, email, phone_number, password) VALUES (%s, %s, %s, %s)"
_GET_ALL_USERS_SQL = "SELECT * FROM users"
_GET_USERS_BY_EMAIL_SQL = """SELECT * FROM users WHERE email IN (%s);"""
_GET_USERS_BY_EMAIL_AND_PASSWORD_SQL = """SELECT * FROM users WHERE email = %s AND password = %s ;"""
_DELETE_USER_SQL = "DELETE FROM users WHERE id = %s; "
_UPDATE_USER_SQL = """UPDATE users SET username = %s , email = %s, phone_number = %s , password = %s WHERE id = %s; """
_UPDATE_USER_SQL_without = """UPDATE users SET username = %s , email = %s, phone_number = %s WHERE id = %s; """
```

מחלקת user donation assignment

```
_DELETE_TABLE_SQL = "DROP TABLE donation_assignment"

_INSERT_ASSIGNMENT_SQL = "INSERT IGNORE INTO donation_assignment (requested_user, donation) VALUES (%s, %s)"

_DELETE_ASSIGNMENT_SQL = "DELETE FROM donation_assignment WHERE donation = %s"
```

תוכן דינמי – מימוש עמודי התבנית

עמוד My Account

עמוד החשבון של משתמש הוא עמוד דינמי המשתנה ממשתמש למשתמש. לדוגמה:

- עבור ה-route: `@myaccount.route('/myaccount')` נציג את כלל התרומות שהמשתמש תרם באתר. עם ניהול מצבים כאשר מוצר לא הוזמן אז אני כבעלים של המוצר יכול למחוק אותו. כאשר המוצר הוזמן אני יכול לסמן כ- "סופק" או להוריד את ההצמדה ככה שהמוצר יחזור להופיע בקטלוג המוצרים.
- עבור ה-route: `@myaccount.route('/my-account/edit', methods=['GET', 'POST'])` נעדכן את פרטי המשתמש. כלל הפרטים של המשתמש יוזרקו לשדות הטופס למעט סיסמה כי כאשר עושים hash לא ניתן לשחזר את הסיסמה מטעמי הבטחת מידע. בשני המקרים, את המזהה של המשתמש ניקח דמתוך ה-session.

בקשת מוצר בעמוד Home

כאשר משתמש מבקש לשריין תרומה הוא עובר לעמוד סיכום של המוצר. עמוד זה הינו עמוד דינמי אשר מוזרק לתוכו תוכן הנוגע לתרומה הספציפית שהמשתמש ביקש. נעשה זאת ב-route: `@homepage.route('/request-donation', methods=['POST','get'])`
בנוסף כאשר לקוח לא מחובר הוא לא יכול לבקש מוצר או להוסיף מוצר לכן קופץ לו popup הרשמה או כניסה למערכת לפני.

Header/ footer

שני החלקים האלו גם דינמיים כדי לאפשר תפריט שונה בין אנשים שמחוברים לבין אלו שלא מחוברים לאתר.

סליידר מוצרים בעמוד Home

בעמוד הבית יש מספר Slider המכיל את כלל המוצרים המוצעים לתרומה מקטגוריה מסוימת (אלו שהסטאטוס שלהם הוא "זמין"). כל Slider הינו רספונסיבי ודינמי משום שתוכנו תלוי ב-DB. נעשה זאת ב-route: `@homepage.route('/')`

מימוש הטפסים

- טופס login בעמוד הבית: `@homepage.route('/login', methods=['POST'])`
- טופס signup בעמוד הבית: `@homepage.route('/register', methods=['POST'])`
- טופס edit בעמוד my account: `@myaccount.route('/my-account/edit', methods=['GET', 'POST'])`
- טופס contact us: `@contact.route('/contact-us', methods=['POST','GET'])`
- טופס הוספת מוצר בעמוד הבית (לחצן הפלוס): `@homepage.route('/donation', methods=['POST'])`
- הכפתורים בעמוד החשבון שלי גם הם טפסים `@myaccount.route('/my-account/donation', methods=['POST'])`

בנוסף, בטופס הוספת מוצר הוספנו אפשרות להעלאת תמונה של המוצר:

```
if request.files:
    image=request.files['donation_image']
    image.save(os.path.join(settings.UPLOAD_FOLDER, image.filename))
    donation_image=image.filename
```

בעצם מה שקורה כאשר לקוח מוסיף מוצר הוא מעלה תמונה שנכנסת לתיקייה static/uploaded_donation_images ובנוסף בבסיס הנתונים נשמר השם של התמונה כדי להציג את התמונה בעמוד הבית. בנוסף ביצענו ולידציה האם מדובר בקובץ תמונה. שלב זה היווה אתגר רציני.

מודולריות

utilities/api_utils.py – טפסים

פונקציה אשר מוציאה את הפרמטרים מתוך טופס עם method=POST:

```
def extract_from_form(request, field):
    field_value = request.form.get(field)
    if not field_value:
        raise app_errors.InvalidAPIUsage(f"{field} not provided")
    return field_value
```

פונקציה אשר מוציאה את ה-Query parameters מתוך טופס עם method=GET:

```
def extract_from_args(request, field):
    arg_value = request.args.get(field)
    if not arg_value:
        raise app_errors.InvalidAPIUsage(f"{field} not provided")
    return arg_value
```

המרה לפורמט של תאריך - utilities/datetime_utils.py

```
def from_datetime_str_to_datetime(datetime_str: str) -> datetime.datetime:
    return datetime.datetime.fromisoformat(datetime_str)


def convert_datetime_to_timestamp(ts: datetime.datetime) -> str:
    return ts.strftime('%Y-%m-%d %H:%M:%S')
```

פונקציות עזר לניהול Session

הקמנו מחלקה בשם sessionHelper אשר עוזרת לניהול ה-sessions. ה-session key הוא המשתמש עצמו. למחלקה ארבע שיטות: login_user, logout_user (מחיקת ה-session והחזרה למצב "אורח"), is_user_logged ו- get_user_from_session (שיטה שמחזירה את ה-session).

ב-sessions נשתמש על מנת לשלוף נתונים ללקוח; בעמוד החשבון שלי, בבקשת תרומה ולמעשה כל מה שדורש התחברות לאתר.

הקמת שרת smtp

בעמוד Contact Us נרצה לשלוח למשתמש מייל עם תשובה לפנייתו. לשם כך, יש צורך בהקמת שרת. שמשתמש בפונקציה הנמצאת utilities/mail_utils.py

המשתמשת ב-gmail כשרת smtp לשליחת מייל.

```
import settings
import smtplib


def send_mail(email: str, subject: str, message: str):
    server = smtplib.SMTP(settings.EMAIL_HOST, settings.EMAIL_PORT)
    server.starttls()
    server.login(settings.CONTACT_EMAIL, settings.CONTACT_EMAIL_PASSWORD)
    server.sendmail(settings.CONTACT_EMAIL, email, message)
```

כך הלקוח מקבל למייל את ההודעה ששלח כהעתק.

תפיסת שגיאות

כל השאילתות לבסיס הנתונים נבנו עם ולידציה וזריקת שגיאות בעזרת raise לדוגמה

```python
@classmethod
def register_user(cls, username: str, email: str, password: str, phone_number: str) -> bool:
    user = User(username=username, email=email, password_plaintext=password, phone_number=phone_number)
    user.validate_user_details()
    num_rows_inserted = dbManager.commit(
        cls._INSERT_USER_SQL,
        (user.username, user.email, user.phone_number, user.password_hashed)
    )
    if num_rows_inserted == dbManager.ERROR_CODE:
        raise app_errors.AppError('Failed registering user', payload={'email': email})

    if num_rows_inserted == 0:
        raise app_errors.InvalidAPIUsage('User with this info already exists')

    return True
```

כך אנחנו יכולים תפוס אותם להציג ולעשות פעולות נוספות כדי לשפר את חווית המשתמש כאשר דברים לא עובדים כמתוכנן.

בקובץ app_errors.py

אנחנו מגדירים שני מחלקות שתופסות את השיגוע שלנו.

```python
class AppError(Exception):

    def __init__(self, message, status_code=http.HTTPStatus.INTERNAL_SERVER_ERROR, payload=None):
        super().__init__()
        self.message = message
        self.payload = payload
        self.status_code = status_code

    def to_dict(self):
        rv = dict(self.payload or ())
        rv['message'] = self.message
        return rv


class InvalidAPIUsage(AppError):
    _DEFAULT_STATUS_CODE = http.HTTPStatus.BAD_REQUEST

    def __init__(self, message, status_code=None, payload=None):
        status_code = status_code or self._DEFAULT_STATUS_CODE
        super().__init__(message, status_code=status_code, payload=payload)
```

ובקובץ app.py אנחנו מגדירים routes שייצגו ב-frontend את השגיאה ואת התוכן שלה.

```python
@app.errorhandler(AppError)
def app_error(e: AppError):
    return jsonify(e.to_dict()), e.status_code


@app.errorhandler(Exception)
def unhandled_error(e: Exception):
    return jsonify(dict(error=e)), http.HTTPStatus.INTERNAL_SERVER_ERROR
```

במידה והיינו ממשיכים את האתר הזה כאתר אמיתי היינו רוצים לשפר את כדי לתת חווית לקוח יותר טובה.

הנחות:

1) לא ניתן לעדכן מוצר תרומה מתוך מחשבה שאפשר למחוק וליצור חדש.
2) לקוח מוגדר על-ידי 3 מזהים חד ערכיים אימייל, טלפון וסיסמה.
3) קיים בסיס נתונים בשם webgroup18 לפני שמריצים את אתחול בסיס הנתונים
4) התקנת הספרייה Pip install phonenumbers לפני שמריצים את אתחול בסיס הנתונים