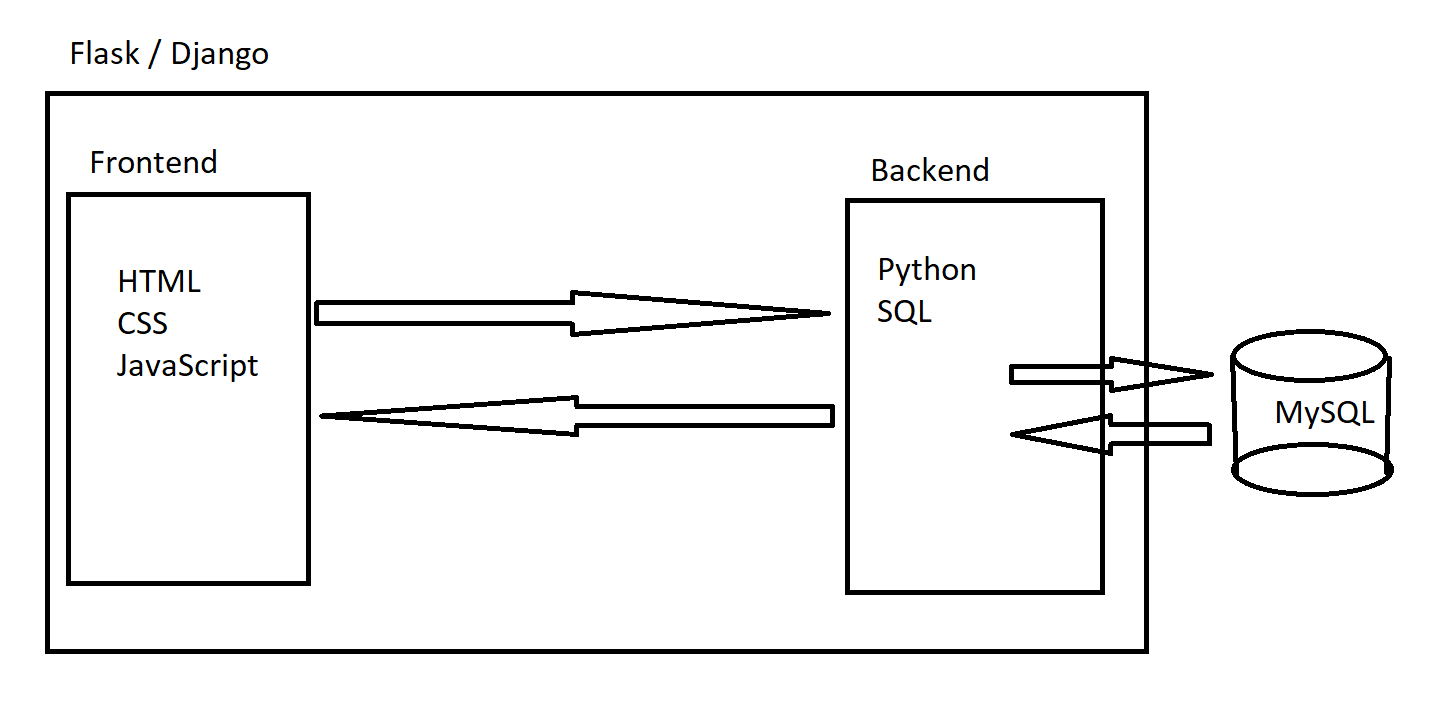
# **Full Stack**

**Framework**

מערכת מוכנה שאנו עובדים בתוכה ובמסגרת החוקים שלה והיא מאפשרת לבנות עבורנו חלק מהאתר או את כולו.

בפייתון יש שני Frameworks נפוצים:

1. Flask: מאפשרת לבנות בפייתון מערכת Full Stack, הינה מאוד Light, מיועדת לאתרים לא גדולים מידי, לא יותר מידי קובעת בשביל המתכנת איך לעבוד – יותר גמישה.
2. Django: מאפשרת לבנות בפייתון מערכת Full Stack, מיועדת לאתרים יותר גדולים, קובעת בשביל המתכנת איך לעבוד – פחות גמישה.



**Route**

זהו הניתוב הפנימי באתר שמוצג בדפדפן. לדוגמה: בכתובת הבאה: [http://mysite.com**/abc/def**](http://mysite.com/abc/def) ה-/abc/def זהו ה-Route

**View**

פונקציה בפייתון הנמצאת בצד השרת אליה הלקוח גולש ע"י תיאור Route בכתובת הדפדפן.

Blueprint זהו אובייקט המסוגל לייצר Views

**Model**

מחלקת מידע טהור

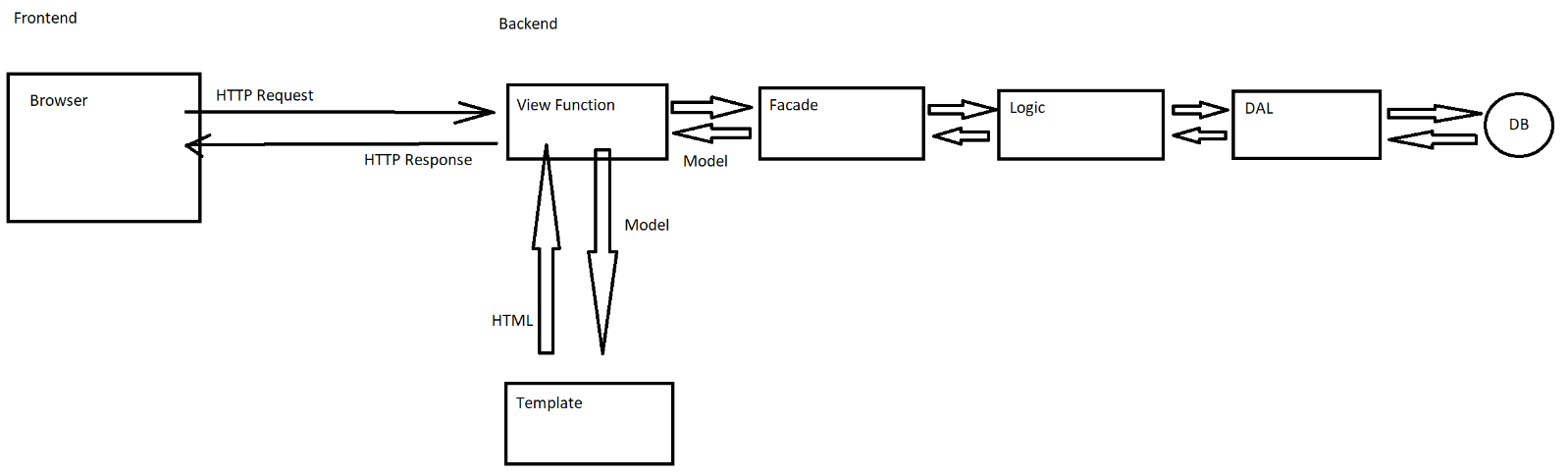
**Template**

זה דף HTML שפונקציית ה-View מחזירה ללקוח.

כלל השיטה הזו נקראת MVT (Model-View-Template)

**ה-Flow של Flask:**

1. משתמש גולש ע"י הדפדפן ל-Route ספציפי.
2. ה-Route מנתב לפונקציה ב-View.
3. פונקציית ה-View ניגשת ל-DB ומביאה מידע (Model).
4. הפונקציה שולחת את המידע ל-Template (קובץ HTML) לשילוב המידע
5. ה-Template מחזיר HTML ל-View
6. ה-View מחזיר את ה-HTML בחזרה ללקוח.



יצירת פרויקט Flask

1. יצירת Environment:

py -m venv env

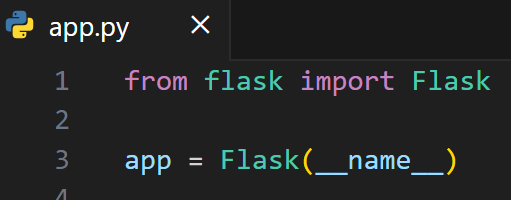
1. הרצת ה-Environment:

env/scripts/activate

1. התקנת Flask פנימית בפרויקט:

pip install flask

1. יצירת תיקיית src ו-app.py בתוכה.
2. הגדרת אובייקט app:



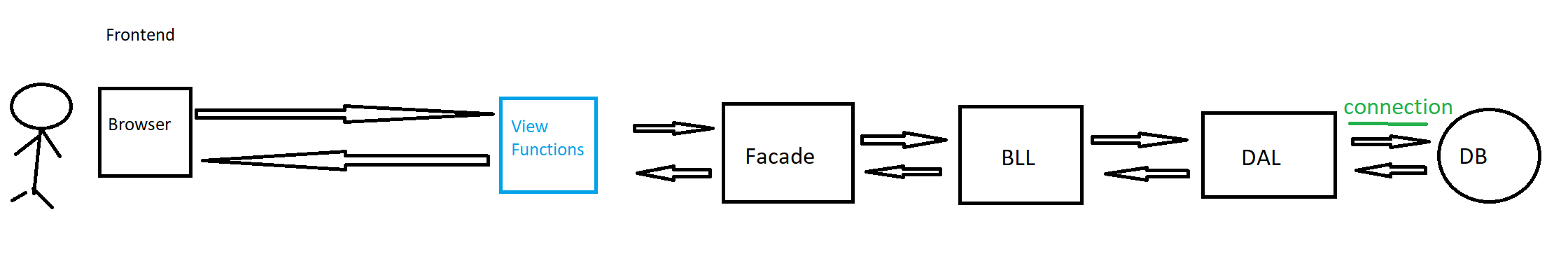
1. הרצת האפליקציה ע"י פקודת הטרמינל:

flask --app src/app.py run --debug

Route Parameter

משתנה שניתן לשלוח ל-Route ופונקציית ה-View יכולה לקבל אותו לפי שמו.

ארכיטקטורת שכבות:



טרמינולוגיית תמונות:

Client Image Name: שם קובץ התמונה המקורי שקיים במחשב הלקוח. לדוגמה: apple.jpg

Server Image Name: שם קובץ התמונה שהשרת ממציא לתמונה המתקבלת מהלקוח.

לדוגמה: 01a5a6d5-6cc4-4e72-8f5d-e44efd3bc3d7.jpg

שם זה נכנס ל-DB.

המחרוזת הארוכה הזו נקראת UUID (גרסה 4): Universal Unique Identifier. זו מחרוזת ייחודית שניתן לייצר ולא תחזור על עצמה Ever.

Image Path: הנתיב המלא במחשב השרת לתמונה כולל שם וסיומת.

לדוגמה: C:\MySite\src\static\images\products\01a5a6d5-6cc4-4e72-8f5d-e44efd3bc3d7.jpg

Image File: קובץ התמונה (ביטים של פיקסלים) שהלקוח שולח ואנו שומרים בתיקייה.

Image URL: הכתובת ה-Web-ית לתמונה. לדוגמה:

<http://mysite.com/products/images/01a5a6d5-6cc4-4e72-8f5d-e44efd3bc3d7.jpg>

בגלישה לנתיב זה השרת מחזיר את קובץ התמונה

הוראה לטפסים לשלוח גם קבצים שנבחרו בתיבות file:

enctype="multipart/form-data"

**טיפול בשגיאות**

חובה לטפל במערכת במספר קטגוריות של שגיאות. כל קטגוריה הטיפול הוא אחר.

1. שגיאת 404: דף לא קיים שהמשתמש מנסה לראות.

יש לבנות 404.html עבור דף כזה.

יש לבנות פונקציה שמפנה לדף במקרה וגולשים לדף לא קיים.

@app.errorhandler(404)

def page\_not\_found(error):

    return render\_template("404.html")

1. שגיאת 404 עבור Resource Not Found. כאשר Resource זה אובייקט במסד הנתונים שמנסים לראות אך הוא לא קיים. לדוגמה, מוצר לא קיים.

יש לבנות מחלקת חריגה המתארת שגיאה כזו ולזרוק אותה מה-Facade אם זה קורה.

יש לבדוק ב-Facade האם id לא קיים, ואם לא – להעלות את החריגה.

יש לתפוס את החריגה ב-View (עטיפת הקריאה ל-Façade ב-try-except) והצגת דף 404 הכולל את השגיאה.

1. שגיאת 400: שגיאת וולידציה – פרטים שגויים שמשתמש מכניס.

חובה לבצע בדיקות וולידציה גם ב-Front וגם ב-Back.

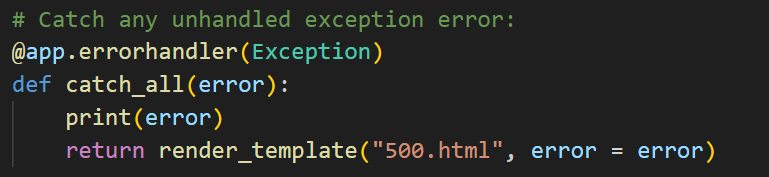
ב-Front – כי אנו לא רוצים לגלוש ל-Back אם המשתמש הכניס פרטים שגויים.

ב-Back – כי תמיד אפשר לעקוף בדיקות ב-Front.

1. שגיאת 500: קריסה כלשהי בשרת.

יש לבנות דף 500.html המציג מידע לגבי השגיאה הכללית.

יש לבנות פונקציה ב-app.py שתופסת כל חריגה כללית כזו. קונספט כזה נקרא Catch All.



Response Status Codes:

1xx – Continue: השרת דורש מידע נוסף. הלקוח שולח מידע שעדיין לא הסתיים.

2xx – OK: הצלחות.

3xx – Redirect: שרת אומר ללקוח לפנות למקום אחר.

4xx – Client Errors: שגיאות לקוח.

5xx – Server Errors: שגיאות שרת, קריסות שרת וכו'.

**Auth**

Authentication & Authorization

Authentication – אימות: זיהוי המשתמש.

Authorization – הרשאות: מה המשתמש יכול לבצע.

פעולות הקשורות ל-Auth:

1. הרשמה – Register
2. כניסה – Login
3. הרשאות – מה כל משתמש יכול או לא יכול לבצע. חסימה אם צריך.
4. יציאה – Logout

Credentials

אלו הפרטים הדרושים לצורך Login בלבד.

**Server-Side Session**

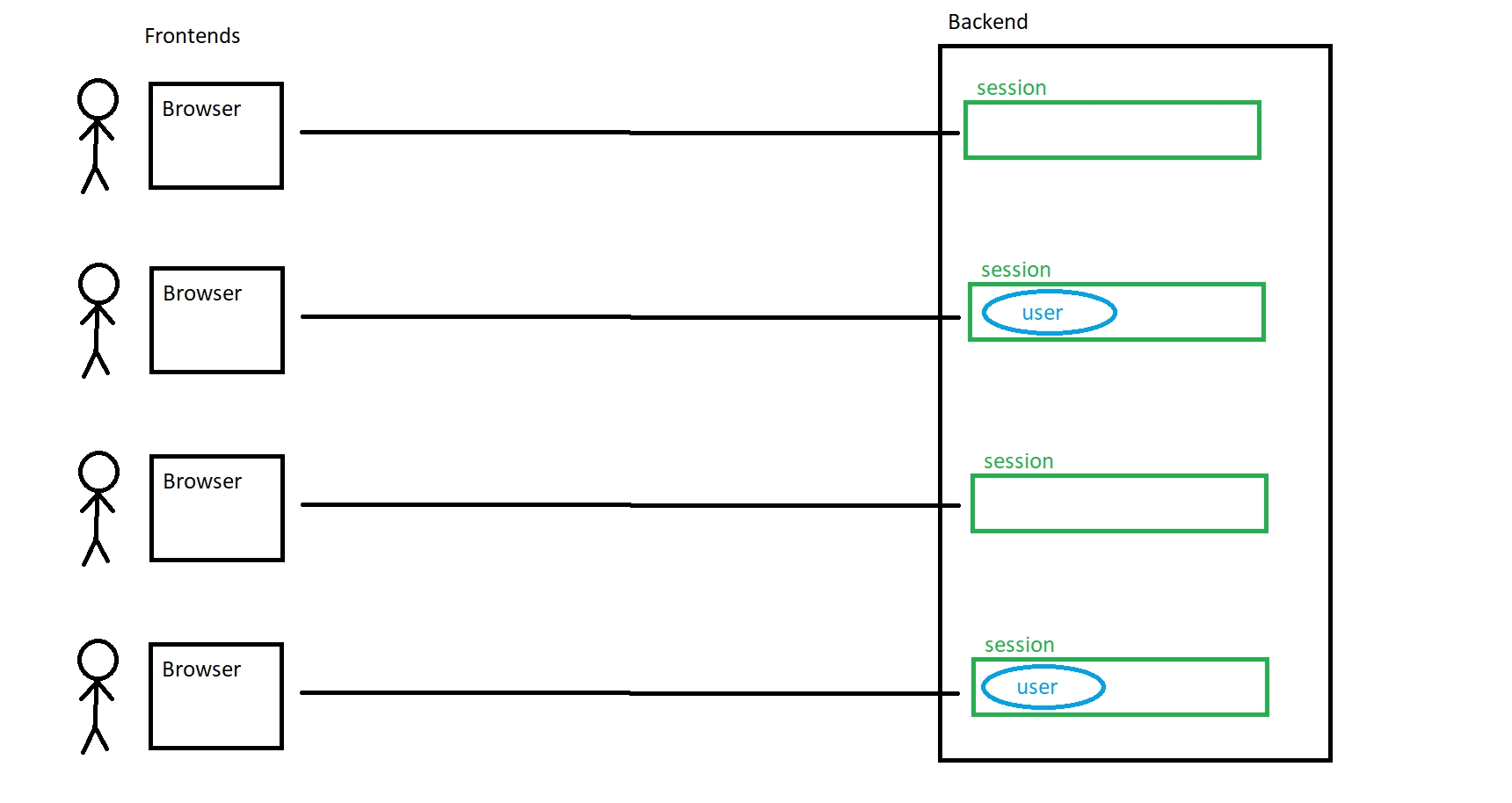
זהו אובייקט Collection הפועל לפי Key-Value, שקיים בנפרד לכל משתמש בצד השרת.

קיים בצד השרת כל עוד המשתמש גולש באתר. אם המשתמש סוגר דפדפן – אובייקט ה-Server-Side Session שלו, לא יהיה יותר זמין עבורו ויעלם אחרי זמן מסוים.

ה-session מגיע מ-Flask.

לצורך שימוש בו עלינו להגדיר מחרוזת ייחודית ברמת ה-app.py בה המערכת משתמשת בכדי לקודד את המידע:

app.secret\_key = “some unique key…”



**Secure Coding**

התחום של המתכנתים לגבי אבטחת הקוד שהם כותבים.

קיימים מספר עקרונות שמתכנת חייב לבצע מבחינת Security.

1. TMI

Too Much Information

אין להראות אינפורמציה מיותרת למשתמש. לדוגמה, בכניסה שגויה לדף Login, הצגת ההודעה: "סיסמה נכונה אך לא האימייל". או "אימייל שגוי"... הודעה נכונה: "אימייל או סיסמה לא נכונים".

1. env.

Environment Variables אלו משתנים שמוגדרים ברמת המכונה (המחשב), שניתן להשתמש בהם באפליקציה שאנו בונים. לפעמים יש בהם מידע רגיש. הדבר הנכון לבצע הוא להגדיר ערכים אלו בקובץ מיוחד עבור Environment Variables. קובץ זה נקרא env.

קובץ זה לא עולה ל-GitHub! כי הוא מכיל מידע רגיש.

1. אין להציג שגיאות מערכת בסביבת ה-production, אלא רק בסביבת ה-development. לדוגמה קריסות.

יש לבדוק באיזו סביבה אנו ולפי זה להציג למשתמש שגיאות.

1. אין להחזיר סיסמאות לצד הלקוח (לדוגמה שמירת סיסמה ב-localStorage)
2. אין להשאיר סיסמאות באובייקטים ב-Server-Side Session.
3. אסור לשמור סיסמאות Plain Text במסד הנתונים!

סיסמאות "מגבבים" ("גיבוב" – Hash)

Hash זה ערבול חד כיווני של טקסט. ניתן לקחת מחרוזת, לערבל אותה כך שתתקבל מחרוזת אחרת לגמרי, אך אם יש לנו את המחרוזת המעורבלת, לא ניתן לחזור למחרוזת המקורית.

כל מחרוזת תמיד תייצר את אותו ה-Hash עבור אותו האלגוריתם.

1. יש לבצע פעולה הנקראת Salt על הסיסמה שכן ה-Hash שלה מופיע ב-Rainbow Tables.

לא לשכוח

לאחר שהורדנו פרויקט (אין לנו את ה env ) אבל יש לנו את requirements.txt

pip install -r requirements.txt