



Disko Project: API integration and Frontend

שם החברה: OCTOPUS

קבוצה: מס' 1

שם הפרויקט: DISKO



DISKO PROJECT



Our Team

מגישים ות"ז:		חלק בפרויקט
דור קרת	212213201	API and Frontend
בר גוטמן	323871459	API and Frontend
אמיר שלומיוק	212071112	CI/CD Workflow
לינוי אדרי	318943776	UX/UI and Frontend



OCTOPUS

COMPUTER SOLUTIONS

על החברה: OCTOPUS

המוצר:

מוצר החברה עליו עבדנו נקרא DISKO, המוצר נועד לניהול IMAGES ב-REGESTRIES- השונים הקיימים בשוק. המוצר מיעל ההעברה ומיון של IMAGES הקיימים בקלאסטרים המקומיים של הלקוח, המוצר חוסך ללקוח זמן וכסף.

החברה:

OCTOPUS היא חברת טכנולוגיה ומחשוב מלאה שבסיסה בישראל. הם מספקים ללקוחות פתרונות מחשוב ואחסון בענן, ייעוץ IT, מומחיות בקוד פתוח. הם שותפי גוגל, ספקים של Microsoft 365 ויועצי חומרה ותוכנה עבור כל סוגי העסקים הם מספקים תמיכת IT.

הבעיה:

אתגרים שפגשנו בפיתוח המוצר

• הבעיה המרכזית:

המוצר הגיע אלינו בלי ממשק משתמש (GUI) ובלי API שיחבר בין ה-Frontend ל-Backend. זה אילץ אותנו לפתח את החלקים האלו מאפס.

• מה היה לנו מאתגר:

1. ללמוד המון בזמן קצר:

היינו צריכים ללמוד מאפס, תוך שבועיים בלבד, טכנולוגיות כמו JavaScript, React, Node.js, HTML, CSS, ו-Python. כל זה דרש מאיתנו הרבה מאוד מאמץ וזמן ללמוד ולהבין את הבסיס של כל טכנולוגיה, ולהשתמש בהן כדי לפתח את המוצר.

2. לחבר בין כל החלקים:

נדרשנו לעשות אינטגרציה בין ה-API שיצרנו לבין ה-Backend וה-Frontend. החיבור בין כל החלקים האלו היה מורכב ודרש המון בדיקות ותיקונים כדי לוודא שהכל עובד כמו שצריך.

3. להתמודד עם בדיקות CI/CD:

בזמן הקצר שהיה לנו, למדנו גם איך להגדיר ולבצע בדיקות CI/CD עם

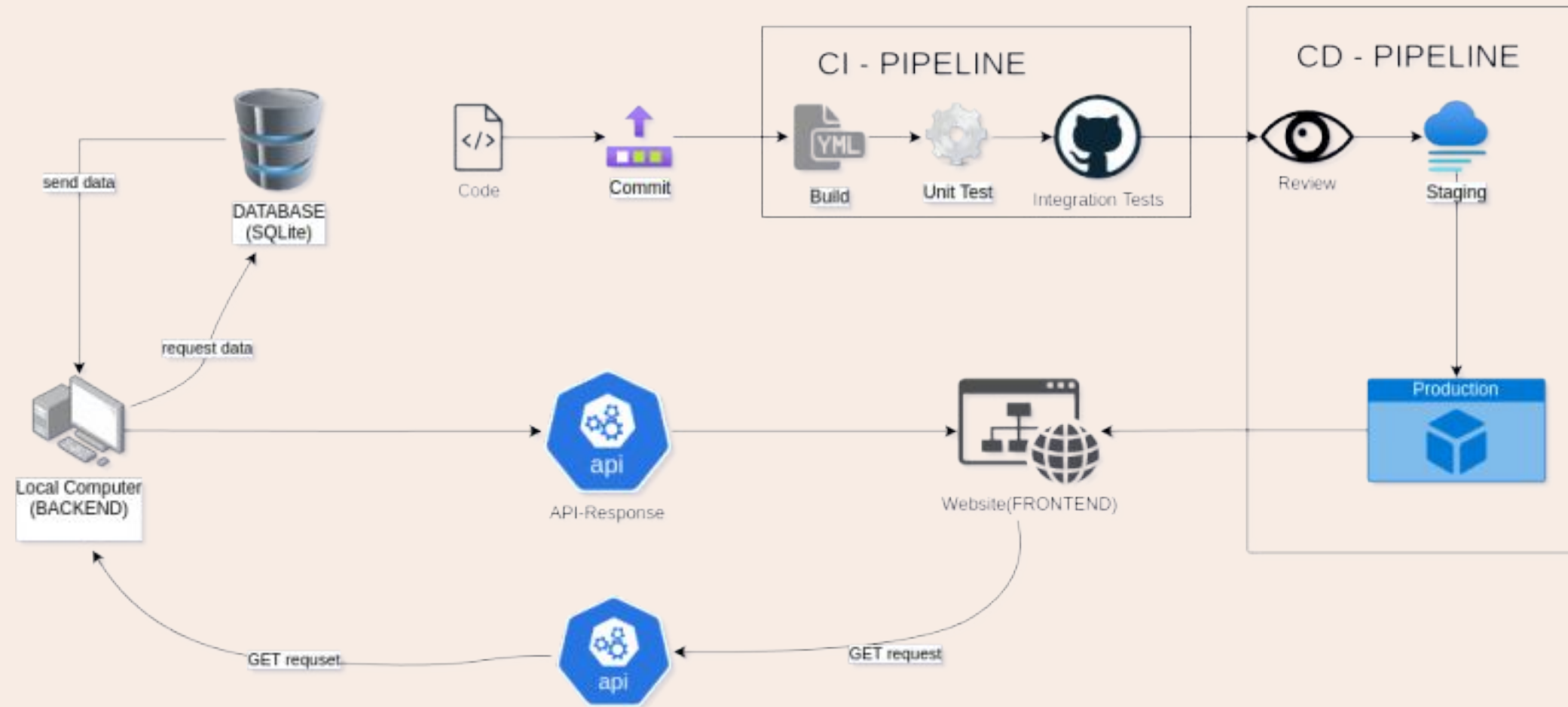
- GitHub Actions כתיבת סקריפטים לבדיקה אוטומטית, אימות של

הקוד, וביצוע Deployment במידת הצורך.



ארכיטקטורה:

DISKO - Diagrams



4. המחשב מבקש ממסד הנתונים את כל התמונות שנמצאות לו בטבלה על פי שם Cluster

5. מסד הנתונים שולח למשתמש את התמונות מתוך הטבלה בעזרת בקשת GET

6. התמונות מוצגות למשתמש בחזרה

1. המשתמש מבקש לראות את כל התמונות ב-Cluster שלו.

1. הבקשה נשלחת ל-API Endpoint

1. המחשב המקומי מקבל את הבקשה ויוצר מסד נתונים וטבלה בשם Cluster

טכנולוגיות בהם השתמשנו:



Kubernetes

פלטפורמת ניהול קונטיינרים לניהול והרחבת היישום שלנו.



Docker

דוקר הוא טכנולוגיה ל-package והרצה של יישומים בקונטיינרים.



GitHub

הוא מערכת ניהול גרסאות שנועדה לנהל ולעקוב אחרי שינויים בקוד תוכנה לאורך זמן.



JavaScript

שפת תכנות לבניית אפליקציות אינטרנט דינמיות ואינטראקטיביות.



TypeScript

סופרסט של JavaScript עם טיפוס סטטי, לשיפור איכות הקוד ויכולת התחזוקה.



Node.js

סביבת ריצה ל-JavaScript לפיתוח צד השרת ובניית APIs.



Next.js

מסגרת עבודה של React לבניית אפליקציות אינטרנט מהירות וידידותיות למנועי חיפוש (SEO)



React

ספריית JavaScript לבניית ממשקי משתמש, הידועה בארכיטקטורת הקומפוננטות שלה.



HTML

שפת סימון (Markup) ליצירת מבנה לתוכן של דפי אינטרנט.



CSS

שפת עיצוב לקביעת המראה הוויזואלי של דפי אינטרנט.



Python

שפת תכנות כללית שנעשית בה שימוש לניתוח נתונים, למידת מכונה ואוטומציה.



Helm

מנהל חבילות ל-Kubernetes, שמפשט את הפריסה והניהול של אפליקציות.



GitHub Actions

פלטפורמת אוטומציה לבניית ופריסת אפליקציות בתוך GitHub.

Cluster Migration

X

Registry:

Enter registry URL

Tag:

Enter image tag

Username:

Enter your username

Password:

Enter your password

Helm Chart Path:

Submit

כפתור לפונקציה : Migration


דף הבית של DISKO

כפתור להפעלת DISKO

HomeAbout

Work with Disko

Disko is an open source tool, designed to manage and facilitate operations in disconnected (air-gapped) environments. It has three main features: Statistics of Images per Registry, Copy Images Between Registries, and Migrate Images in Kubernetes.



Start Disko

Copy Image

X

Images:

registry.k8s.io/coredns/coredns:v1.11.1
registry.k8s.io/etcd:3.5.12-0
docker.io/kindest/kindnetd:v20240202-8f1494ea
registry.k8s.io/kube-apiserver:v1.30.0
registry.k8s.io/kube-controller-manager:v1.30.0

New Registry URL:

Enter new registry URL

Target Username:

Enter target username

Target Password:

Enter target password

Tag:

Enter image tag

Submit

כפתור לפונקציה: Copy images

HomeAbout

kind-kindSubmit

Registry	Number of Images	Percentage
registry.k8s.io	6	75%
Dockerhub	2	25%

Hide Images

Image Name	Date	Registry
registry.k8s.io/coredns/coredns:v1.11.1	2024-09-02 12:19:29	registry.k8s.io
registry.k8s.io/etcd:3.5.12-0	2024-09-02 12:19:29	registry.k8s.io
docker.io/kindest/kindnetd:v20240202-8f1494ea	2024-09-02 12:19:29	Dockerhub
registry.k8s.io/kube-apiserver:v1.30.0	2024-09-02 12:19:29	registry.k8s.io
registry.k8s.io/kube-controller-manager:v1.30.0	2024-09-02 12:19:29	registry.k8s.io
registry.k8s.io/kube-proxy:v1.30.0	2024-09-02 12:19:29	registry.k8s.io
registry.k8s.io/kube-scheduler:v1.30.0	2024-09-02 12:19:29	registry.k8s.io
docker.io/kindest/local-path-provisioner:v20240202-8f1494ea	2024-09-02 12:19:29	Dockerhub

Open Cluster Migration Form

Open Copy Image Form

אמיר שלומיוק

```

- name: Verify Flask Installation
  run: |
    python -c "import importlib.metadata; print(f'Flask version: {importlib.metadata.version(\"flask\")}')"

- name: Check Python Syntax
  run: |
    find backend/ -name "*.py" -print0 | xargs -0 -n1 python -m py_compile

- name: Run API Finder Script
  run: |
    echo "Running Flask server detection..."
    python tests/api_finder.py > flask-server-results.txt
    cat flask-server-results.txt
  shell: /usr/bin/bash -e {0}

- name: Run Tests
  run: |
    echo "Running API validation tests..."
    pytest tests/ --disable-warnings > api_test.py || exit_code=$?
    cat api_test.py
    if [ $exit_code -ne 0 ]; then
      echo "Tests failed! Analyzing failed tests..."
      grep -E "^FAIL|^ERROR" api-validation-results.txt
      exit $exit_code
    fi

- name: Print Test Status
  if: failure()
  run: |
    echo "API validation tests failed! ❌💀. Please check the results above."

- name: Final Message
  if: success()
  run: echo "All tests passed successfully! ✅😊"

```

- הקטע קוד הזה, מגדיר שלבי CD/CI עבור הפרויקט ב-github actions
- אימות התקנה: בודק אם Flask מותקן על מנת להפעיל את שרתי ה-API
- בדיקת תחביר של Python: מאמת את התחביר של כל קבצי ה-Python בפרויקט על ידי שימוש ב-py_compile
- הרצת סקריפט למציאת שרתי אי פי איי והתראה אם נמצאו כאלה, דבר שמצריך בדיקה על מנת לשמור על ריצת קוד תקינה.
- הרצת בדיקות API: מבצע בדיקות אוטומטיות ל-API באמצעות Pytest ובודק את תוצאות הבדיקות.
- הצגת סטטוס הבדיקות: אם הבדיקות נכשלות, מציג הודעה מתאימה ומנתח את התוצאות הכושלות.
- הודעה סופית: אם כל הבדיקות עוברות בהצלחה, מציג הודעה חיובית.

אמיר שלומיזק

```

import unittest
from unittest.mock import patch, MagicMock
import sys
import os

# Add the backend directory to the Python path
backend_path = os.path.abspath(os.path.join(os.path.dirname(__file__), '../backend'))
sys.path.insert(0, backend_path)

from flask import Flask
from api import api, get_image_controller

class APITestCase(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        cls.app = Flask(__name__)
        cls.app.register_blueprint(api) # Register the combined API blueprint
        cls.client = cls.app.test_client()

    @patch('api.ImageController.get_kubernetes_clusters')
    def test_get_clusters(self, mock_get_clusters):
        """Test getting a list of Kubernetes clusters."""
        mock_get_clusters.return_value = [
            {'name': 'cluster1', 'status': 'active'},
            {'name': 'cluster2', 'status': 'inactive'}
        ]
        response = self.client.get('/api/clusters')
        self.assertEqual(response.status_code, 200)
        self.assertEqual(response.get_json(), [
            {'name': 'cluster1', 'status': 'active'},
            {'name': 'cluster2', 'status': 'inactive'}
        ])
        mock_get_clusters.assert_called_once()

    @patch('api.ImageCollector.collect_images')
    def test_select_cluster(self, mock_collect_images):
        """Test selecting a cluster and processing it."""
        mock_collect_images.return_value = None
        response = self.client.post('/api/selected-cluster', json={'cluster': 'test-cluster'})
        self.assertEqual(response.status_code, 200)
        self.assertEqual(response.get_json(), {'message': 'Cluster test-cluster selected and processed successfully'})
        mock_collect_images.assert_called_once_with('test-cluster')

```

- קטע הקוד הזה ב-Python משתמש ב-Unitest לבדיקת API שנבנה ב-Flask. הקוד הוא חלק מקובץ פייתון שלמעשה אחראי על הבדיקה, הקוד כולל:
- הוספת נתיב ה-backend ל-Python path: מאפשר לקוד לגשת למודולים שנמצאים בתיקיית ה-backend.
- יצירת אפליקציית Flask: נרשמת Blueprint עבור ה-API המשולב כך שהבדיקות יכולות להתבצע על נקודות הקצה.
- שימוש ב-Mocking: משתמש ב-patch.mock unittest כדי להחליף את הפונקציה clusters_kubernetes_get במתודה מדומה במהלך הבדיקה
- בדיקת נקודת קצה: מבצע קריאה לנקודת הקצה clusters/api/ ומוודא שהתגובה כוללת את הנתונים המדומים שהוגדרו ושמצב התגובה הוא 200.
- הקוד מסייע לוודא שה-API פועל כמצופה ומחזיר את הנתונים הנכונים, גם כאשר חלקים מסוימים של המערכת מדומים לצורך הבדיקה.

אמיר שלומיזק


```
- name: Set up Node.js
  uses: actions/setup-node@v3
  with:
    node-version: '18.17.0'

- name: Install Root Dependencies
  run: npm ci

- name: Install Frontend Dependencies
  run: |
    cd frontend
    npm install

- name: Build the application
  run: |
    cd frontend
    npm run build # Build the Next.js application for production

- name: Start the application
  run: |
    cd frontend
    npm run start & # Start the Next.js application in production mode
    npx wait-on http://localhost:3000

- name: Run Cypress tests
  run: npx cypress run
```

הגדרת גרסא ל-Node.js
התקנת תוספים בפרונטאנד - ניגש לתיקיית
ה-Frontend ומתקין את התוספים הדרושים
להרצת האפליקציה.

בניית האפליקציה - ניגש לתיקיית
ה-Frontend ובונה את האפליקציה
באמצעות `npm run build`, מותאם
ל-`Next.js`.

הרצת האפליקציה - ניגש לתיקיית
ה-Frontend ומתחיל את האפליקציה במצב
`Production`, תוך המתנה לחיבור
הרצת בדיקות Cypress - מבצע בדיקות
אוטומטיות לאפליקציה באמצעות Cypress.

אמיר שלומיוק


```
describe('Navigation Tests', () => {
  beforeEach(() => {
    // Visit the homepage before each test
    cy.visit('http://localhost:3000');
  });

  it('should navigate to the About page when the About button is clicked', () => {
    // Click the 'About' button
    cy.contains('About').click();

    // Verify that the URL is correct
    cy.url().should('eq', 'http://localhost:3000/about');
    cy.contains('About').should('be.visible');
  });

  it('should navigate to the Home page when the Home button is clicked', () => {
    cy.contains('Home').click();
    cy.url().should('eq', 'http://localhost:3000/');

    // Verify that the Home page contains expected content
    cy.contains('Disko').should('be.visible');
  });

  it('should navigate to the Start Disko page when the Start Disko button is clicked', () => {
    // Click the 'Start Disko' button
    cy.contains('Start Disko').click();
    cy.url().should('eq', 'http://localhost:3000/statistics');
    cy.contains('Cluster').should('be.visible');
  });
});
```

סדרת בדיקות ניווט באמצעות Cypress עבור ה-Frontend, נכתבת ב-JavaScript ומבצעת את הפעולות הבאות: לפני כל בדיקה, היא מוודאת שהעמוד הראשי של האפליקציה נטען ב-localhost:3000.

לאחר מכן, לוחצת על כפתור "About", מאמתת שה-URL משתנה ל-

<http://localhost:3000/about> מוודאת שהטקסט ABOUT גלוי בעמוד

בהמשך, לוחצת על כפתור "Home", מאמתת שה-URL משתנה ל-http://localhost:3000/

ומוודאת שהעמוד הראשי מכיל את התוכן הצפוי לבסוף, לוחצת על כפתור Start Disko, מאמתת שה-URL משתנה ל-http://localhost:3000/statistics

ומוודאת שהטקסט "Cluster" גלוי בעמוד. הבדיקות הללו נועדו להבטיח שהכפתורים באתר עובדים כראוי ומובילים את המשתמשים לעמודים הנכונים, וכן שהעמודים מכילים את התוכן הצפוי.

אמיר שלומיזק

DISKO PROJECT

בר גוטמן

הקטע קוד הזה הוא חלק מיישום Flask, שמאפשר עבודה עם API לצורך העתקת תמונות למאגר רישום חדש תחילה מוגדר נתיב (route) בשם /copyimage, שניתן לגשת אליו באמצעות בקשת GET. הפונקציה copy_images מתבצעת כאשר מתקבלת בקשה לנתיב זה, ומטרתה להעתיק תמונות למאגר רישום חדש בהתבסס על הפרמטרים שסופקו בבקשה.

הפרמטרים מתקבלים מהבקשה באמצעות request.args.get() ו-request.args.getlist(), וכוללים את שם המאגר החדש (new_registry), התג (tag), שם משתמש (target_username), סיסמה (target_password), ורשימה של תמונות (images). בתוך הפונקציה, קריאה ל-controller.copy_images מבצעת את ההעתקה של התמונות על בסיס הפרמטרים שסופקו. במידה והפעולה מצליחה, מוחזרת תגובה בפורמט JSON עם הודעה על הצלחה וקוד סטטוס 200. אם מתרחשת שגיאה כלשהי, תופס ה-except את החרוג ומשיב תגובה בפורמט JSON עם הודעה המתארת את השגיאה וקוד סטטוס 500 בר גוטמן

```
@api.route('/copyimage', methods=['GET'])
def copy_images():
    # Copies images to a new registry based on provided parameters
    registry = request.args.get('new_registry')
    tag = request.args.get('tag')
    username = request.args.get('target_username')
    password = request.args.get('target_password')
    images = request.args.getlist('images')

    try:
        # Call the copy_images method
        controller.copy_images(images, registry, tag, username, password)

        # Return a success message in JSON format if the operation was successful (200)
        return jsonify({'message': 'Images copied successfully!'}), 200
    except Exception as e:
        # Return an error message in JSON format if the operation failed (500)
        return jsonify({'message': str(e)}), 500

if __name__ == '__main__':
    # Configure the Flask app and enable CORS for the API
    app = Flask(__name__)
    CORS(app, resources={r"/api/*": {"origins": "*"}})
    app.register_blueprint(api)
    app.run(debug=True)
```

הקוד הזה יוצר נתיב API שמטפל בבקשות GET לכתובת `clusters/`. הנה מה שקורה בו:
הגדרת הנתיב: `clusters/` הנתב נוצר עבור בקשות GET. כשמישהו שולח בקשה לכתובת הזו, הפונקציה `get_clusters` תופעל.
שליפת רשימת קלאסטרים: הפונקציה פונה ל-`controller` (שכבר מוגדר במקום אחר בקוד) ומבקשת ממנו את רשימת קלאסטרי ה-Kubernetes באמצעות קריאה לפונקציה `get_kubernetes_clusters()`.
החזרת התשובה בפורמט JSON: התוצאה שמתקבלת היא רשימת הקלאסטרים, והפונקציה מחזירה אותה כתשובת JSON למי ששלח את הבקשה.

בר גוטמן

```
controller.get_image_controller(),

@api.route('/clusters', methods=['GET'])
def get_clusters():
    # Fetches a list of Kubernetes clusters and returns it as a JSON response
    clusters = controller.get_kubernetes_clusters()
    return jsonify(clusters)
```


הפונקציה migration:

הקוד הזה מגדיר נתיב API שמטפל בבקשות GET לכתובת [clustermigration/](#). הנה הסבר על מה שקורה בו:

הגדרת הנתיב: הנתיב [clustermigration/](#) מיועד לטיפול בבקשות GET. כשמישהו שולח בקשה לכתובת הזו, הפונקציה [migration](#) מופעלת.

קבלת פרמטרים מהבקשה: הפונקציה שולפת את הפרמטרים שנשלחו בבקשה דרך ה-URL, כולל [password](#), [username](#), [tag](#), [registry](#), ו-[helm_chart_path](#). אלה הם פרמטרים שמגיעים כחלק מהשאלתה ב-GET request.

ביצוע פעולת ההעברה: הפונקציה מעבירה את התמונות של הקלאסטר לרשומת רישום חדשה על בסיס הפרמטרים שהתקבלו, באמצעות קריאה לפונקציה [cluster_migration](#) של ה-[controller](#).

החזרת תשובה מוצלחת: אם הפעולה מצליחה, הפונקציה מחזירה תשובת JSON שמכילה הודעה על הצלחת החלפת שם התמונה בקובץ ה-Helm Chart, יחד עם סטטוס HTTP 200 (הצלחה).

טיפול בשגיאות: אם מתרחשת שגיאה כלשהי במהלך ההעברה, הפונקציה תופסת את החריגה (Exception) ומחזירה הודעת שגיאה בפורמט JSON עם סטטוס HTTP 500 (שגיאת שרת).

בר גוטמן

```
@api.route('/clustermigration',methods=['GET'])
def migration():
    # Migrates a cluster's images to a new registry based on provided parameters
    registry = request.args.get('registry')
    tag = request.args.get('tag')
    username = request.args.get('username')
    password = request.args.get('password')
    helm_chart_path = request.args.get('helm_chart_path')

    try:
        controller.cluster_migration(registry, tag, username, password, helm_chart_path)
        return jsonify({'message': 'Image name replaced successfully in ' + helm_chart_path}), 200
    except Exception as e:
        return jsonify({'message': str(e)}), 500
```

קביעת מיקום מסד הנתונים: הפונקציה מוצאת את הנתוב לתיקיה שבה נמצא הקובץ הנוכחי ומגדירה את הנתוב המלא למסד הנתונים בשם `image_data.db`, שנמצא במיקום יחסית לתיקיה הזו.

חיבור למסד הנתונים: הפונקציה יוצרת אובייקט שמאפשר לבצע פעולות CRUD (יצירה, קריאה, עדכון, מחיקה) על מסד הנתונים.

שליפת כל התמונות לפי שם קלאסטר: הפונקציה שולפת את כל התמונות מתוך מסד הנתונים לפי השם של הקלאסטר שנשלח לה כפרמטר.

החזרת התמונות: לבסוף, הפונקציה מחזירה את רשימת התמונות שנשלפה מתוך מסד הנתונים.

בקיצור, הפונקציה הזו מקבלת שם של קלאסטר ומחזירה את כל התמונות השייכות אליו מתוך מסד הנתונים.

בר גוטמן

```
# Function for Kubernetes cluster migration
def cluster_migration(self, registry, tag, username, password, helm_chart_path):
    values_file_path = os.path.join(helm_chart_path, "values.yaml")
    chart_file_path = os.path.join(helm_chart_path, "Chart.yaml")
    release_name = self.get_release_name(chart_file_path)
    current_image = self.get_current_image(values_file_path)
    self.copy_images(current_image, registry, tag, username, password)
    self.replace_image(values_file_path, registry)
    ##self.helm_upgrade_release(release_name, helm_chart_path)

def present_images_per_cluster(self, cluster):
    base_dir = os.path.dirname(os.path.abspath(__file__))
    db_name = os.path.join(base_dir, '../../../image_data.db')
    db = SQLiteCRUD(db_name)
    images = db.select_all(cluster)
    return [images]
```


מניעת התנהגות ברירת מחדל: הפונקציה מתחילה במניעת התנהגות ברירת המחדל של הטופס, כך שהדף לא יתרענן כששולחים את הטופס.

בניית השאילתה: לאחר מכן, היא יוצרת מחרוזת שאילתה (`query`) באמצעות הפרמטרים שהוזנו בטופס (`registry`, `tag`, `username`, `password`, ו-`helm_chart_path`). המחרוזת הזו מתווספת לכתובת ה-URL כדי לשלוח את הנתונים כפרמטרים ב-GET request.

שליחת הבקשה לשרת: הפונקציה מנסה לשלוח בקשת GET לשרת בכתובת

`http://localhost:5000/api/clustermigration` עם הפרמטרים שהוזנו.

בדיקת תגובת השרת: אם השרת מחזיר תשובה מוצלחת (סטטוס 200), הפונקציה מעדכנת את המצב (`state`) של ההודעה ל"הצלחה", ומציינת שהתמונה עברה וה-`Helm chart` עודכן. אם התשובה מהשרת לא מוצלחת (סטטוס שגיאה), הפונקציה מעדכנת את המצב עם הודעת שגיאה לפי הטקסט של סטטוס התגובה מהשרת.

טיפול בשגיאות רשת: אם מתרחשת שגיאה במהלך שליחת הבקשה (למשל, בעיית חיבור), הפונקציה תופסת את השגיאה ומציגה הודעת שגיאה שמציינת שהייתה שגיאת רשת.

בר גוטמן

```
const handleSubmit = async (e: React.FormEvent<HTMLFormElement>) => {
  e.preventDefault();

  const query = new URLSearchParams({
    registry: formData.registry,
    tag: formData.tag,
    username: formData.username,
    password: formData.password,
    helm_chart_path: formData.helm_chart_path,
  }).toString();

  try {
    const response = await fetch(`http://localhost:5000/api/clustermigration?${query}`, {
      method: 'GET',
      headers: {
        'Content-Type': 'application/json',
      },
    });

    if (response.ok) {
      setStatusMessage('Success: Image migrated and Helm chart updated.');
```

דור קרת

הגדרת לוגים: הקוד מגדיר מערכת לוגים ברמת פירוט גבוהה כדי לעקוב אחר פעולות המערכת ולנפות שגיאות בעת הצורך.

יצירת אפליקציית Flask: הקוד יוצר אפליקציה מבוססת Flask, שהיא מסגרת לפיתוח אפליקציות רשת ב-Python.

הגדרת תיקיית בסיס: הקוד מוצא ושומר את הנתיב לתיקיה שבה נמצא הקובץ הנוכחי, כדי לשמש אותו בעבודה עם קבצים יחסית למיקום הזה.

הגדרת Blueprint: הקוד יוצר קבוצת מסלולים (routes) מאורגנים תחת קידומת `/api`, כדי לנהל את ה-API של האפליקציה בצורה מסודרת.

חיבור למסד נתונים: הקוד מכין אובייקט שמנהל תמונות, שמתחבר למסד נתונים הנמצא בתיקיית הבסיס של האפליקציה.

אתחול האובייקט המנהל: לבסוף, הקוד קורא לפונקציה שיוצרת את אובייקט ניהול התמונות ומאחסן אותו לשימוש בהמשך.

בקיצור, הקוד מגדיר ומכין את האפליקציה לעבוד עם מסד נתונים של תמונות ולנהל בקשות דרך API.

דור קרת

```
11 logging.basicConfig(level=logging.DEBUG)
12
13 app = Flask(__name__)
14
15 base_dir = os.path.dirname(os.path.abspath(__file__))
16
17 api = Blueprint('api', __name__, url_prefix='/api')
18
19
20 def get_image_controller():
21     # Initializes and returns an ImageController instance connected to the database
22     db_name = os.path.join(base_dir, 'image_data.db')
23     print(db_name)
24     return ImageController(db_name)
25
26 controller = get_image_controller()
27
```

הקוד הזה מגדיר נתיב API שמטפל בבקשות POST לכתובת `/selected-cluster`. הנה תיאור כללי של מה שקורה בו:

הגדרת הנתיב: `/selected-cluster` נוצר לטיפול בבקשות POST. כשמישהו שולח בקשה לכתובת הזו, הפונקציה `select_cluster` מופעלת.

קבלת הנתונים מהבקשה: הפונקציה מקבלת את הנתונים שהתקבלו בגוף הבקשה בפורמט JSON, ושולפת את שם הקלאסטר שנבחר באמצעות המפתח `'cluster'`.

בדיקת קלט: אם לא נשלח שם קלאסטר בבקשה, הפונקציה מחזירה הודעת שגיאה עם סטטוס HTTP 400 (בקשה לא תקינה), מה שמעיד על כך שלא נבחר קלאסטר.

עיבוד הקלאסטר: אם נבחר קלאסטר, הפונקציה יוצרת מופע של `ImageCollector`, ואז קוראת לפונקציה `collect_images` של המופע הזה כדי לאסוף את התמונות עבור הקלאסטר שנבחר.

החזרת תשובה מוצלחת: אם הפעולה מצליחה, הפונקציה מחזירה תשובת JSON עם הודעה על הצלחת בחירת הקלאסטר והטיפול בו, יחד עם סטטוס HTTP 200 (הצלחה).

טיפול בשגיאות: אם מתרחשת שגיאה במהלך עיבוד הקלאסטר, הפונקציה תופסת את החריגה (Exception), רושמת את השגיאה בלוגים, ומחזירה הודעת שגיאה בפורמט JSON עם סטטוס HTTP 500 (שגיאת שרת).

דור קרת

```
33
34 @api.route('/selected-cluster', methods=['POST'])
35 def select_cluster():
36     # Processes the selected cluster by collecting images for it
37     data = request.json
38     cluster = data.get('cluster')
39
40     if not cluster:
41         return jsonify({'error': 'No cluster selected'}), 400
42
43     try:
44         collector = ImageCollector()
45         collector.collect_images(cluster)
46         return jsonify({'message': f'Cluster {cluster} selected and processed successfully'}), 200
47     except Exception as e:
48         logging.error("Error in select_cluster: %s", traceback.format_exc())
49         return jsonify({'error': str(e)}), 500
50
```

הקוד הזה מגדיר נתיב API שמטפל בבקשות GET לכתובת `/statistics`. הנה תיאור כללי של מה שקורה בו:

הגדרת הנתיב: `/statistics` נוצר לטיפול בבקשות GET. כשמישהו שולח בקשה לכתובת הזו, הפונקציה `get_statistics` מופעלת.

קבלת פרמטר מהבקשה: הפונקציה מקבלת את שם הקלאסטר שהתקבל בפרמטר `cluster` מתוך ה-URL של הבקשה.

בדיקת קלט: אם לא סופק שם קלאסטר בבקשה, הפונקציה מחזירה הודעת שגיאה עם סטטוס HTTP 400 (בקשה לא תקינה), מה שמעיד על כך שלא סופק קלאסטר.

שליפת נתונים סטטיסטיים: אם סופק שם קלאסטר, הפונקציה פונה ל-`controller` ומבקשת ממנו לחשב את האחוזים עבור הקלאסטר הזה באמצעות הפונקציה `calculate_percentages`.

עיבוד הנתונים: הפונקציה יוצרת רשימה של תוצאות (`results`), כאשר כל תוצאה כוללת את שם הרג'יסטרי, כמות התמונות, והאחוז שלהן מתוך כלל התמונות בקלאסטר.

החזרת תשובה מוצלחת: אם הכל מתבצע כמתוכנן, הפונקציה מחזירה את הנתונים הסטטיסטיים כ-JSON עם סטטוס HTTP 200 (הצלחה).

טיפול בשגיאות: אם מתרחשת שגיאה במהלך שליפת הנתונים או עיבודם, הפונקציה תופסת את החריגה, רושמת את השגיאה בלוגים, ומחזירה הודעת שגיאה כללית של "שגיאת שרת פנימית" עם סטטוס HTTP 500.

דור קרת

```
@api.route('/statistics', methods=['GET'])
def get_statistics():
    # Retrieves statistical data for a specified cluster and returns it as a JSON response
    cluster = request.args.get('cluster')

    if not cluster:
        return jsonify({'error': 'No cluster provided'}), 400

    try:
        percentages = controller.calculate_percentages(cluster)
        results = []
        for item in percentages:
            results.append({
                "registry": item[0],
                "amount": item[1],
                "percentage": item[2]
            })

        return jsonify({'results': results}), 200
    except Exception as e:
        logging.error("Error in get_statistics: %s", traceback.format_exc())
        return jsonify({'error': 'Internal Server Error'}), 500
```


הקוד הזה מגדיר פונקציה בשם `fetchStatistics`, אשר מבצעת בקשת GET לשרת ומחזירה נתונים סטטיסטיים עבור קלאסטר מסוים. הנה מה שקורה בקוד: **התחלת הבקשה**: הפונקציה מתחילה עם הגדרת מצב טעינה (`setLoading(true)`) ומאפסת כל שגיאה קודמת שהייתה (`setError(null)`). **ביצוע בקשת GET**: הפונקציה משתמשת ב-`axios` כדי לשלוח בקשת GET לכתובת ה-API המקומית, עם שם הקלאסטר כפרמטר ב-URL. הבקשה נעשית לנתיב `api/statistics/`, והפרמטר `cluster` מועבר בכתובת.

עדכון סטטיסטיקות: אם הבקשה מצליחה, התשובה שמתקבלת מכילה נתונים סטטיסטיים, והפונקציה מעדכנת את המצב (`setStatistics`) עם התוצאות שהתקבלו מהשרת.

טיפול בשגיאות: אם ישנה שגיאה במהלך הבקשה, הפונקציה תופסת את השגיאה, מעדכנת את המצב עם הודעת שגיאה (`setError('Failed to load statistics')`), ומדפיסה את השגיאה לקונסול לצורך ניפוי שגיאות.

סיום מצב הטעינה: לבסוף, ללא קשר אם הבקשה הצליחה או נכשלה, הפונקציה מעדכנת את מצב הטעינה `False` כדי לסמן שהבקשה הסתיימה.

דור קרת

```
// setLoading(false);  
const fetchStatistics = async (cluster: string) => {  
  try {  
    setLoading(true);  
    setError(null);  
    const response = await axios.get(`http://localhost:5000/api/statistics?cluster=${cluster}`);  
    setStatistics(response.data.results);  
  } catch (err) {  
    setError('Failed to load statistics');  
    console.error("Error fetching statistics:", err);  
  } finally {  
    setLoading(false);  
  }  
};
```


הקטע הזה של הקוד מייצג את החלק של הקומפוננטה ב-React שאחראי על הצגת התוכן בהתאם למצב של הנתונים שנטענים. הנה מה שקורה:
מצב טעינה: אם הקומפוננטה נמצאת במצב טעינה (**loading** הוא **true**), יוצג הטקסט "Loading..." כדי ליידע את המשתמש שהתוכן עדיין נטען.
טיפול בשגיאות: אם יש שגיאה (**error** לא שווה ל-**null**), יוצג הטקסט של השגיאה על המסך, המידע נמצא במשתנה **error**.
הצגת טבלה עם נתונים: אם לא מדובר במצב טעינה ולא קיימת שגיאה, יוצגת טבלה עם הנתונים הסטטיסטיים. הטבלה כוללת:
כותרות עמודות: "Registry", "Number of Images", ו-"Percentage".
שורות טבלה: עבור כל פריט בנתונים הסטטיסטיים (**statistics**), נוצרות שורות בטבלה שמציגות את המידע עבור כל פריט:

שם הרגיסטרי (**stat.registry**)
מספר התמונות (**stat.amount**)
אחוז התמונות (**stat.percentage**)

דור קרת

```
return (  
  <div>  
    {loading ? (  
      <p>Loading...</p>  
    ) : error ? (  
      <p>{error}</p>  
    ) : (  
      <table>  
        <thead>  
          <tr>  
            <th>Registry</th>  
            <th>Number of Images</th>  
            <th>Percentage</th>  
          </tr>  
        </thead>  
        <tbody>  
          {statistics.map((stat, index) => (  
            <tr key={index}>  
              <td>{stat.registry}</td>  
              <td>{stat.amount}</td>  
              <td>{stat.percentage}%</td>  
            </tr>  
          ))}  
        </tbody>  
      </table>  
    )}  
  </div>  

```

לינוי אדרי

- השתמשתי במודאל גנרי (GENERIC MODAL) כדי להכיל את ה-FORMS תוך שמירה על שפה עיצובית אחידה

לינוי אדרי

```
205
206 export default function ClusterMigration() {
207   const [isModalOpen, setIsModalOpen] = useState(false);
208
209   const openModal = () => setIsModalOpen(true);
210   const closeModal = () => setIsModalOpen(false);
211
212   return (
213     <div>
214       {/* Button to open the modal */}
215       <button
216         className="button-small"
217         onClick={openModal}
218       >
219         Open Cluster Migration Form
220       </button>
221
```

Frontend - יצירת קומפוננטות - NEXTJS

- יצרתי שתי קומפוננטות עבור הגירת קלאסטרים (בשם CLUSTER MIGRATION) וקומפוננטה נוספת בשם COPY IMAGE שתפקידה הוא העתקת אימג'ים לרג'יסטרי חיצוני.
- הקומפוננטות בנויות כך שהן מרנדרות HTML FORM בהינתן המידע מעלות שאילתה ב- REST API את תוכן המוזן אל הבקאנד להמשך טיפול.

```
return (  
  <div style={styles.overlay}>  
    <div style={styles.modal}>  
      <button onClick={onClose} style={styles.closeButton}>X</button>  
      <h1>Cluster Migration</h1>  
      <form onSubmit={handleSubmit} className="form-container">  
        <div className="form-body">  
          <div>  
            <label htmlFor="registry">Registry:</label>  
            <input  
              type="text"  
              id="registry"  
              name="registry"  
              value={formData.registry}  
              onChange={handleChange}  
              placeholder="Enter registry URL"  
              style={styles.input}  
            />  
          </div>  
          <div>
```


עיצוב ממשק - Frontend

- כפי שצוין, חפשתי להשתמש בשפה עיצובית אחידה עבור הממשק כולו - החל מטבלאות, דפים, כפתורים וכיוצ"ב. במקטע הנ"ל ניתן לראות את עיצוב הטבלאות ב-CSS שחל על כל הקומפוננטות בפרויקט.

לינוי אדרי.

```
/* Table Styling */
table {
  width: auto; /* Make table width auto to be compact */
  margin: 20px auto; /* Center the table */
  border-collapse: separate;
  border-spacing: 0; /* Remove extra space in tables */
  background-color: #ffffff; /* Distinct background color for the table */
  border-radius: 5px;
  box-shadow: 0 0 3px rgba(0, 0, 0, 0.1);
  border-radius: 5px;
}

th, td {
  padding: 10px 15px; /* Trim unneeded space */
  text-align: left; /* Align table headers to the left */
  border: 1px solid #ccc; /* Clearer table borders */
}
```

לוגו חדש למוצר ושימוש - Frontend

בחרתי ליצור לוגו חדש למוצר שיהיה מופשט וקליל. הלוגו משלב את אלמנט הקוברנטיס (הגה) וכדור הדיסקו (לפי שם המוצר).

הלוגו שולב בפרויקט הן בעמוד הראשי, הן ב- NAV BAR והן בתור FAVICO לכל עמוד. לינוי אדרי.

```
export function Navbar() {
  return(
    <nav className=" bg-body-tertiary justify-between px-[20px] py[16py] lg:container lg:mx-4" >
      <div className="container-fluid flex items-center justify-between px-4 py-2">
        <div className="flex items-center space-x-3">
          <Image src={Logo} alt="Logo" width="45" height="40"/>
          <div className="flex gap-x-6">
            {navLinks.map((link, index) => (
              <a className="nav-link" href={link.href} key={index}>
                {link.name}
              </a>
            ))}
          </div>
        </div>
      </div>
    </div>
  )
}
```



בנוסף למשימה הקבוצתית שניתנה לי
(עיצוב הfrontend) של מערכת Disko שכללה בעיקר
שימוש ב HTML, CSS וNEXTJS
ניתנה לי המשימה לעטוף את הBackend וFrontend בקונטיינרים של Docker

הכנת files לdocker לbackend

- במסמך הוגדר שרת python בהתאם דרישות הסביבה
- התקנות חשובות לסביבה כגון apt וpip.
- הותקן git לשם משיכת הפרויקט עצמו.
- בנוסף הוגדר הפורט עליו נמצא הסביבה

```
python:3.10-slim
# /app
apt-get -y update && apt-get -y install git
# hello
git clone https://github.com/gatmbarz123/Disko-back.git
# /app/Disko-back
git checkout API
# install kubernetes flask Flask-Cors docker ruamel.yaml
5000

# docker client
apt-get update && apt-get install -y \
  apt-transport-https \
  ca-certificates \
  curl \
  gnupg \
  lsb-release \
  && curl -fsSL https://download.docker.com/linux/debian/gpg | apt-key add - \
  && echo "deb [arch=amd64] https://download.docker.com/linux/debian $(lsb_release -cs) stable" \
  && tee /etc/apt/sources.list.d/docker.list > /dev/null \
  && apt-get update && apt-get install -y docker-ce-cli

["bash"]
["python3", "backend/api.py"]
```


- במסמך הוגדר שרת nodejs בהתאם דרישות הסביבה
- התקנות חשובות לסביבה כגון npm
- הותקן git לשם משיכת הפרויקט עצמו.
- בנוסף הוגדר הפורט עליו נמצא הסביבה

```
1 FROM node:18-alpine
2 WORKDIR /app
3 RUN apk add git
4 RUN echo hello
5 RUN git clone https://github.com/gatmbarz123/disko-web.git
6 WORKDIR /app/disko-web/frontend
7 RUN git checkout Frontend-Functions
8 RUN npm install
9 RUN npm run build
10 EXPOSE 80
11 ENV HOST=0.0.0.0
12 ENV PORT=80
13 #CMD ["npm", "run", "dev"]
14 CMD ["npm", "run", "dev", "--", "-p", "3000", "-H", "0.0.0.0"]
```

docker compose לשם בנייה והרצה של שני הסביבות בו זמנית-

- יצרתי מסמך המקיף ומריץ את שני הdockerfiles
- במסמך זה מצויינים את שמות services והפורטים עליהם יושבים ומקים אותם באופן אוטומטי לפי הדרישות הניתנות

```
1 version: '3'
2
3 services:
4   disko-frontend:
5     image: disko-frontend
6     container_name: disko-frontend
7     build: ./frontend
8     ports:
9       - "3000:3000"
10    environment:
11      - NEXT_PUBLIC_API_URL=http://disko-backend:5000
12    networks:
13      - disko-network
14    depends_on:
15      - disko-backend
16
17   disko-backend:
18     image: disko-backend
19     container_name: disko-backend
20     build: ./backend
21     ports:
22       - "5000:5000"
23     networks:
24       - disko-network
25     volumes:
26       - /var/run/docker.sock:/var/run/docker.sock # Share Docker socket
27       - ~/.kube/config:/root/.kube/config:ro
28       - /usr/local/bin/kubectl:/usr/local/bin/kubectl:ro
29     stdin_open: true # Keep stdin open (optional, useful for interactive shells)
30     tty: true # Allocate a pseudo-TTY (optional)
31
32 networks:
33   disko-network:
34     driver: bridge
```

ערכתי ב-disko-frontend מסמך השולח את בקשת ה-API לבקאנד. המשתנה שערכתי הינו משתנה סביבה שמחזיק את כתובת IP של הבקאנד במידה אם הוא קיים ובמידה אם הוא לא קיים הפרונט יגש ל-localhost

```
import axios from 'axios';  
import React, { useState, useEffect } from 'react';  
  
const apiUrl = process.env.NEXT_PUBLIC_API_URL || 'http://localhost:5000';
```

```
setLoading(false);  
setError(null);  
const response = await axios.get(`${apiUrl}/api/statistics?cluster=${cluster}`);  
setStatistics(response.data.results);
```

מה למדנו:

אמיר שלומיוק: למדתי כיצד עובד חיבור API ל-Frontend של מוצר, תהליך שדורש הבנה כיצד לשלב את ה-API עם הקוד הקיים כדי לאפשר אינטראקציה חלקה בין frontend ל backend

למדתי בנוסף את תהליך יצירת הבדיקה לקוד, תוכן הבדיקה, הטמעה של בדיקות בחלקים שונים של קוד ואיך ליצור בדיקות רלוונטיות שיעזרו למנוע תקלות בעתיד.

החווייה שלי הייתה מעולה מהפרויקט , חלוקת העבודה הייתה טובה ומאתגרת כל אחד נתן עבודה טובה בחלק שלו בעבודה וגם קיבלתי עזרה מכל חבר בקבוצה. העבודה לא הייתה קלה והיינו תחת מגבלת זמן אבל הצלחנו להתגבר על הכול ולסיים את העבודה עם פרוייקט מצטיין .

בר גוטמן: למדתי איך לקחת כלים כמו API ו-NEXT.JS ולדעת לחבר אותם ביחד ככה שב backend ו ה- frontend יכולו לדבר וליצור אינטרקציה אחד עם השני גם אם הם לא יושבים על אותו מכונה.

השפות החדשות שלמדתי יוכלו לעזור לי בהמשך הדרך כדי לעזור לאנשים אחרים בתחום ולתת את דעתי בנושא שאולי לא הייתי יודע אם לא השתמשתי בכלים הללו .

החווייה שלי מהפרוייקט כקבוצה הייתה מעוד טובה , עבדנו בעבודת צוות, שעות על שעות של דיבורים וניסיון לפתור בעיות ביחד , עזרה בנושאים שלא קשורים לתחום חלוקה בפרוייקט .

דור קרת: במהלך הפרויקט, חוויתי את הקסם שבבניית אפליקציה מ-0. הרגשתי כמו מהנדס בניין, שבונה מבנה ענק לבנה אחר לבנה. האתגר הגדול ביותר היה לחבר את ה-backend וה- frontend בצורה חלקה. הרגשתי כמו פאזל, שבו כל חתיכה הייתה קריטית לתמונה הסופית.

למדתי המון על API ו- Next.js, אבל מעבר לכך, רכשתי מיומנויות חשובות בעבודה בצוות. שיתוף הפעולה והתמיכה ההדדית היו הכרחיים להצלחת הפרויקט.

חלוקת העבודה בקבוצה הייתה יעילה מאוד. כל אחד מאיתנו תרם את חלקו, והייתה מוכנות הדדית לעזור וללמוד זה מזה.

למדתי להתמודד עם בעיות מאפס, למדתי להשתמש בכלים חדשים שלא הכרתי כמו ,helm, next.js, typescript. החוויה הזו חיזקה את הביטחון שלי ביכולת שלי לתרום לפרויקטים מורכבים בעתיד.

לינוי אדרי :

- **פרונטאנד** - למדתי באופן יותר נרחב על עיצוב אתר, שימוש בnextjs, איך ממשקים מתקשרים עם בקאנד , עבודה עם css לשם עיצוב נוח,אחיד וברור, היכרות עם npm כבסיס להרצת nextjs, עבודה בצוות בgit
- **דוקר** - עטיפת פרויקט קיים בדוקר, מה שדרש "לצאת" מהמשבצת שלי ולהכיר את הפרויקט כולו. שימוש ב-docker compose , עדכון הקוד הקיים כדי שיתאים לריצה בקונטיינרים



תודות:

רצינו למסור את התודה שלנו לכל המשתתפים בדרך לפרוייקט הזה :

מכללת אורט סינגלובצקי

חברת OCTOPUS

למרצה דניאל שעזר לנו בדרך

