אפליקציית WEB

להצגת הפרויקט

MusicAI

https://musicai.pythonanywhere.com/

מגישים:

ליאור שרמן 307932277

טל רוזנצוויג 307965806

שני אופיר 204512396

תוכן עניינים

אופן פעולת אתר אינטרנט

על מנת להבין כיצד בונים $web\ applications$ נסביר תחילה מספר מושגים בסיסיים לצורך זה.

דפדפן מוגדר כתוכנת לקוח המציגה דפי אינטרנט ומאפשרת לעבור בין דפים שונים.

האתר יכיל שני חלקים:

$\boldsymbol{FRONT-END}$ - המחשב הביתי מהווה "לקוח" שמקבל את המידע שהגיע מהשרת ומציג אותו למשתמש באמצעות תוכנה ייעודית, במקרה שלנו(ולרוב) הדפדפן.

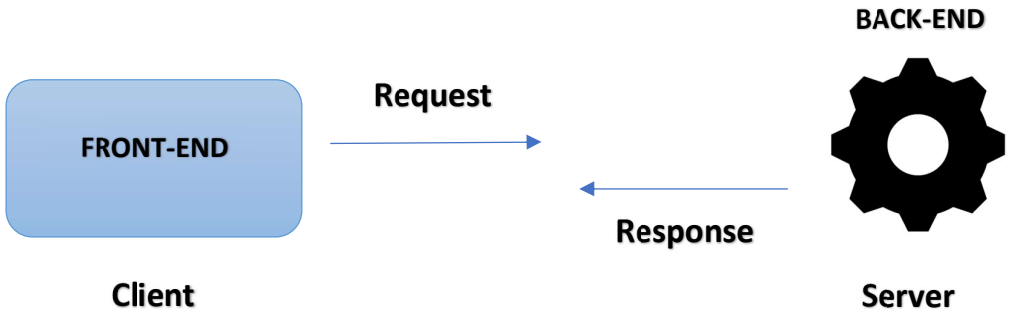
$\boldsymbol{BACK-END}$ - צד השרת מתייחס לפעולות המבוצעות על ידי השרת במערכת שרת-לקוח. לרוב, שרת היא תוכנת מחשב כדוגמת שרת $HTTP$ אשר רצה על גבי מכונת שרת מרוחקת ונגישה מהמחשב או תחנת העבודה המקומית של המשתמש. בנוסף, פעולות צד שרת כוללת גם עיבוד ואחסון נתונים המגיעים מצד הלקוח לשרת.

באתר שלנו בחרנו להשתמש בשרת $HTTPS$ העושה שימוש בפרוטוקול $HTTP$ על שכבת SSL/TLS ובכך מקנה יכולות אבטחה של סטנדרט SSL/TLS לתקשורת $HTTP$ רגילה.

אופן פעולת ארכיטקטורת שרת-לקוח:

דפדפן האינטרנט פועל באמצעות קבלה ושליחה של נתונים. דפדפן האינטרנט של המשתמש מתחבר אל שרת $HTTP$ דרך פרוטוקול התקשורת $HTTP$ המאפשר לדפדפנים ולשרתים לתקר בצורה דו-צדדית, בה הדפדפן גם שולח נתונים לשרת וגם מקבל ממנו נתונים. הדפדפן מוצא את דפי האינטרנט על ידי כתובת מיוחדת המכונה URL, $Uniform\ Resource\ Locator$.

באיור מטה, ניתן לראות כיצד הלקוח משתמש בדפדפן על מנת לשלוח בקשת URL לשרת, כלומר האתר בו הוא רוצה לבקר בו, וכתגובה השרת מעבד את הבקשה ומעביר את התוצאה חזרה ללקוח וכך מספק את מה שביקש.



אם כן, נוכל להסיק כי על מנת להקים אתר אינטרנט נצטרך את שני החלקים, כלומר האפליקציות $FRONT-END$ ו- $BACK-END$.

צד שרת - BACK-END

בתקשורת נתונים, צד שרת הוא מונח המתייחס לפעולות המבוצעות על ידי השרת במערכת שרת-לקוח.
שרת היא תוכנת מחשב, כדוגמת שרת $HTTP$, אשר רצה על גבי מכונת שרת מרוחקת ונגישה מהמחשב או תחנת העבודה המקומית של המשתמש.
קיומן של פעולות המבוצעות רק בצד השרת הינה הכרחית וזאת מאחר שקיימות פעולות בצד השרת הדורשות גישה לנתונים או לפונקציונליות שאינה זמינה בצד הלקוח.
סיבה נוספת להכרחיות היא קיומן של פעולות שונות הדורשות התנהגות מסוימת שאינה בטוחה כאשר היא נעשית בצד הלקוח וכך גם הימצאותם של משאבים שלא קיימים בצד הלקוח, כדוגמת GPU.

על כן, תכנות צד שרת היא טכניקה המשמשת לפיתוח אינטרנט הכוללת פעולות עיבוד ואחסון נתונים המגיעים מצד הלקוח לשרת כך שהשרת מבצע פעולות בהתאם לבקשות שקיבל מהלקוח ומחזירה תשובה בהתאם לצורך. פעולות אלו יכולות להיות מגוונות כדוגמת גישה לבסיסי נתונים, חישובים מתמטיים, גישה לקבצים ולאחר כל העיבוד השרת מייצר קוד צד לקוח ושולח אותו לדפדפן. הדפדפן, משמש לצרכי תצוגה של התוצר שעבר עיבוד ע"י השרת.

בפרויקט שלנו, בחרנו להשתמש ב- $Django\ framework$ לצורך בניית צד השרת. מסגרת זו מעודדת פיתוח מהיר ועיצוב נקי ופרגמטי ונחשבת לאחת הסביבות הנפוצות המשמשות כיום לצרכי בניית צד שרת.

פונקציות תצוגה

פונקציית $view$ היא פונקציית $Python$ שלוקחת בקשת אינטרנט ומחזירה תגובת אינטרנט.

תגובה זו יכולה להיות תוכן $HTML$ של דף אינטרנט, הפנייה מחדש, שגיאת 404, מסמך XML, תמונה ולמעשה כל דבר. התצוגה עצמה מכילה לוגיקה שרירותית הדרושה על מנת להחזיר תגובת אינטרנט זו.

אם כך, פונקציות התצוגה מהוות את נקודת הכניסה משלב בקשת הלקוח אל מול השרת.

פונקציית התצוגה אחראית על החזרת אובייקט מסוג $HttpResponse$ (ברוב המקרים מסוג זה), המכיל את התגובה שנוצרה.

<u>דוגמא ל- $view\ function$:</u>

```
def rnn_preprocess_view(request):
    return render(request, 'pages/rnn_preprocess_inter.html')
```

כאשר המשתמש מבקש לראות את הפרטים על העיבוד המקדים, הפונקציה מחזירה $response$ עם עמוד ה-$HTML$ המתאים.

מיפוי כתובות URL לפונקציות תצוגה

כדי לעצב כתובות URL עבור אפליקציה, ניצור מודול $Python$ הנקרא $URLconf$ (תצורת כתובת URL).
מודול זה הוא קוד $Python$ הינו מיפוי בין ביטויי נתיב כתובת URL לפונקציות $Python$ (התצוגות שלנו).

מיפוי זה יכול להיות קצר או ארוך לפי הצורך וכך גם להתייחס למיפויים אחרים. בנוסף, מאחר וזהו קוד $Python$ טהור, ניתן לבנות אותו באופן דינמי.

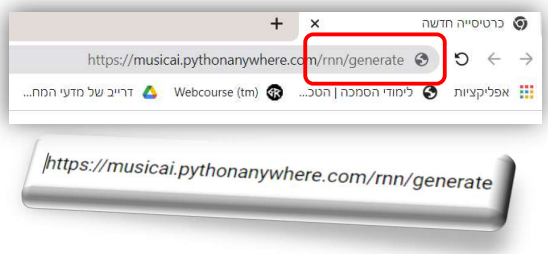
<u>תהליך הבקשה בשלמותו:</u>

כאשר משתמש מבקש דף מהאתר המופעל באמצעות Django, המערכת עובדת לפי האלגוריתם הבא כדי לקבוע איזה קוד $Python$ להפעיל:

1. המשתמש שולח בקשת $Request$ באמצעות הזנת כתובת URL בדפדפן.
2. $Django$ מחפש את דפוסי ה-$URL\ pattern$ המשתנים.
3. $Django$ עובר על כל דפוס כתובת אתר ($URL\ pattern$), לפי הסדר, ועוצר בתבנית הראשונה שתואמת לכתובת ה-URL המבוקשת.
4. ברגע שאחד מדפוסי ה-URL תואם, $Django$ מייבא וקורא לתצוגה הנתונה, שהיא פונקציית $Python$.
5. אם אף $URL\ pattern$ לא תואם או אם הועלתה חריגה בנקודה כלשהי בתהליך, $Django$ מפעיל תצוגה מתאימה לטיפול בשגיאות, כדוגמת שגיאה 404.

האיור בעמוד הבא ממחיש את התהליך.

FRONT-END



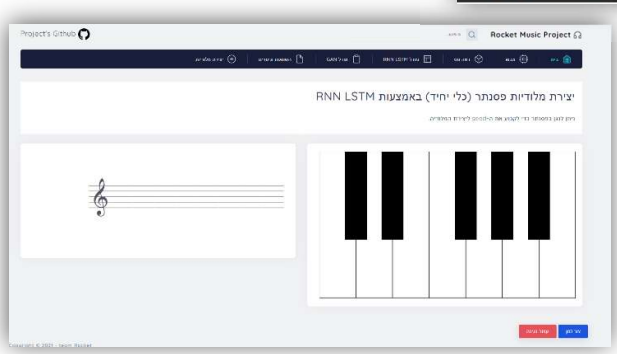
Request

URL configuration

```python
app_name = 'pages'
urlpatterns = [
    # home
    path('', TemplateView.as_view(template_name='pages/index.html'), name='home'),
    path('team', TemplateView.as_view(template_name='pages/the_team.html'), name='team'),

    # intro
    path('intro', TemplateView.as_view(template_name='pages/intro_to_midi.html'), name='intro'),

    # datasets
    path('dataset', TemplateView.as_view(template_name='pages/dataset.html'), name='dataset'),
    path('dataset/esac', TemplateView.as_view(template_name='pages/esac_dataset.html'), name='dataset-esac'),
    path('dataset/snes', TemplateView.as_view(template_name='pages/snes_dataset.html'), name='dataset-snes'),

    # RNN LSTM model
    path('rnn/preprocess', TemplateView.as_view(template_name='pages/rnn_preprocess.html'),
        name='rnn-preprocess'),
    path('rnn/preprocess/interactive', TemplateView.as_view(template_name='pages/rnn_preprocess_inter.html'),
        name='rnn-preprocess-inter'),
    path('rnn/generate', rnn_lstm_generate, name='rnn_lstm_generate'),
    path('rnn/intro', TemplateView.as_view(template_name='pages/rnn_intro.html'), name='rnn-intro'),
    path('rnn/training', TemplateView.as_view(template_name='pages/rnn_training.html'), name='rnn-training'),
    path('rnn/generate/multi', rnn_lstm_generate_multi, name='rnn_lstm_generate_multi'),
    path('rnn/generate/process', TemplateView.as_view(template_name='pages/rnn_generate_process.html'),
        name='rnn-generate-process'),
    path('rnn/experiments', TemplateView.as_view(template_name='pages/rnn_experiments.html'), name='rnn-experiments'),
```

View function

```python
def rnn_lstm_generate(request):
    if request.method == "POST":
        pressed = request.POST.get('pressed').split(',')
        if pressed != ['']:
            notes_to_midi = {'C': 60, 'D': 62, 'E': 64, 'F': 65, 'G': 67, 'A': 69, 'B': 71, 'Db': 64, 'Eb': 63, 'Gb': 66,
                             'Ab': 68, 'Bb': 70}
            seed = (' _ '.join([str(notes_to_midi[note]) for note in pressed]))
        else:
            seed = None
        trained_model = os.path.join(BASE_DIR, 'rnn_lstm', 'static', 'trained_models', 'trained_erk')
        gen = MelodyGenerator(trained_model)
        song = gen.generate_melody(500, 64, 0.85, seed=seed)
        if os.getenv('DEBUG'):
            mel_path = os.path.join(BASE_DIR, 'rnn_lstm', 'static', 'midi', 'generated', 'mel.mid')
        else:
            mel_path = os.path.join(STATIC_ROOT, 'midi', 'generated', 'mel.mid')
        gen.save_melody(song, file_name=mel_path)
        return render(request, 'pages/rnn_generate.html', {'generated': True})
    return render(request, 'pages/rnn_generate.html', {'generated': False})
```



Response

שימוש בתבניות $HTML$

בתחילת דרכה של רשת האינטרנט, עלה הצורך להציג באינטרנט מידע גם בצורה גרפית, ולשם כך פותחה שפת התגיות $HTML$. התגיות, המהוות הוראות מוסכמות ליצירת מצד גרפי סטטי, נשמרות כקובץ טקסט בעל סיומת $html$ או htm אשר מאוחסן על השרת. כך, כאשר המחשב המרוחק מבקש מהשרת גישה אל הקובץ הנ"ל, נשלח אליו תוכנו ולפי הדפדפן בונה את הדף בצד הלקוח.

$HTML$, שפת סימון $Hyper\ Text$, היא אבן הבניין הבסיסית ביותר של האינטרנט. הוא מגדיר את המשמעות והמבנה של תוכן אינטרנט ומשתמש ב-"$markup$"(סימון) כדי להוסיף הערות לטקסט, תמונות ותוכן אחר לתצוגה בדפדפן האינטרנט.

$Django$ מהווה מסגרת אינטרנט($web\ framework$) ולכן זקוקה לדרך נוחה ליצור דפי $HTML$ באופן דינמי. הגישה הנפוצה ביותר מתבססת על תבניות, שימוש ב- $templates$ המכילות את החלקים הסטטיים של פלט ה- $HTML$ הרצוי וכך גם תחביר מיוחד המתאר כיצד יוכנס תוכן דינמי.

באיור מטה ניתן לראות את תהליך החזרת התגובה תוך שימוש ב- *HTML template*.

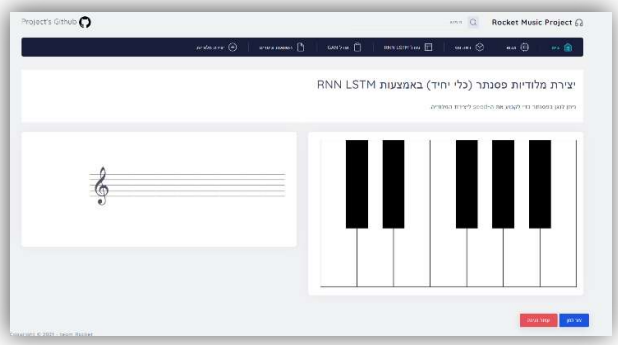
View function

```python
def rnn_lstm_generate(request):
    if request.method == "POST":
        pressed = request.POST.get('pressed').split(',')
        if pressed != ['']:
            notes_to_midi = {'C': 60, 'D': 62, 'E': 64, 'F': 65, 'G': 67, 'A': 69, 'B': 71, 'Db': 64, 'Eb': 63, 'Gb': 66,
                             'Ab': 68, 'Bb': 70}
            seed = (' _ '.join([str(notes_to_midi[note]) for note in pressed]))
        else:
            seed = None
        trained_model = os.path.join(BASE_DIR, 'rnn_lstm', 'static', 'trained_models', 'trained_erk')
        gen = MelodyGenerator(trained_model)
        song = gen.generate_melody(500, 64, 0.85, seed=seed)
        if os.getenv('DEBUG'):
            mel_path = os.path.join(BASE_DIR, 'rnn_lstm', 'static', 'midi', 'generated', 'mel.mid')
        else:
            mel_path = os.path.join(STATIC_ROOT, 'midi', 'generated', 'mel.mid')
        gen.save_melody(song, file_name=mel_path)
        return render(request, 'pages/rnn_generate.html', {'generated': True})
    return render(request, 'pages/rnn_generate.html', {'generated': False})
```

HTML template

```html
<audio id="C" src="{% static 'notes/C.mp3' %}"></audio>
<audio id="Db" src="{% static 'notes/Db.mp3' %}"></audio>
<audio id="D" src="{% static 'notes/D.mp3' %}"></audio>
<audio id="Eb" src="{% static 'notes/Eb.mp3' %}"></audio>
<audio id="E" src="{% static 'notes/E.mp3' %}"></audio>
<audio id="F" src="{% static 'notes/F.mp3' %}"></audio>
<audio id="Gb" src="{% static 'notes/Gb.mp3' %}"></audio>
<audio id="G" src="{% static 'notes/G.mp3' %}"></audio>
<audio id="Ab" src="{% static 'notes/Ab.mp3' %}"></audio>
<audio id="A" src="{% static 'notes/A.mp3' %}"></audio>
<audio id="Bb" src="{% static 'notes/Bb.mp3' %}"></audio>
<audio id="B" src="{% static 'notes/B.mp3' %}"></audio>


<div class="col-xl-6 col-lg-6 col-md-12 col-sm-12 col-12 layout-spacing">
    <div class="widget widget-one">
        <div class="piano">
            <div data-note="B" class="key white"></div>
            <div data-note="Bb" class="key black"></div>
            <div data-note="A" class="key white"></div>
            <div data-note="Ab" class="key black"></div>
            <div data-note="G" class="key white"></div>
            <div data-note="Gb" class="key black"></div>
            <div data-note="F" class="key white"></div>
            <div data-note="E" class="key white"></div>
            <div data-note="Eb" class="key black"></div>
            <div data-note="D" class="key white"></div>
            <div data-note="Db" class="key black"></div>
            <div data-note="C" class="key white"></div>
        </div>
    </div>
</div>
```

Response



צד משתמש - *FRONT-END*

צד הלקוח מהווה את ממשק המשתמש, כלומר הוא מופעל על ידי המשתמש ופונה לשרת כאשר הוא זקוק למידע או שירותים ממנו. למעשה, צד הלקוח משתמש בדפדפן על מנת להציג את התוכן בצורה מעוצבת ללא כל חישוב, גישה לקבצים, קריאה לבסיסי נתונים וכדומה.

אם כן, צד הלקוח הוא הדפדפן והקוד שאנו כותבים בצד זה רץ על מחשב הגולש. טכנולוגיית צד הלקוח בפרויקט שלנו היא $JavaScript$, CSS ו $HTML$.

עיצוב באמצעות CSS ו-$Bootstrap$

עיצוב באמצעות CSS:

טכנולוגיית CSS מגדירה כיצד יראה העמוד שכתבנו כדוגמת צבעים, גדלים, תמונות רקע ושאר מאפיינים ייחודיים של האתר. טכנולוגיה זו נועדה לאפשר הפרדה בין תוכן ומבנה דפי האינטרנט לבין עיצובם, הכולל פריסה, צבעים וגופנים.

כלומר, עד כה תוכן האתרים ועיצובם נעשו באותו דף $HTML$ וכך הקוד הפך למסובך ובלתי קריא. כעת, באמצעות שימוש ב- CSS ניתן למקם הגדרות עיצוב בקוד יחיד ששינוי בו ישתקף בבת אחת בכל הדפים העושים בו שימוש. הפרדה זו יכולה לשפר את נגישות התוכן וכך:

- לספק יותר גמישות ושליטה במפרט מאפייני התצוגה,
- לספק לדפי אינטרנט מרובים לשתף עיצוב על ידי ציון ה-CSS בקובץ מסוג css בנפרד וכך להפחית את המורכבות והחזרות בתוכן המבני,
- לאפשר לקובץ מסוג css להישמר במטמון על מנת לשפר את מהירות טעינת הדפים בין הדפים החולקים את הקובץ והעיצוב שלו.

לכן, בחרנו להשתמש בטכנולוגיה זו לצרכי עיצוב האתר.

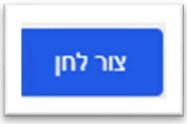
עיצוב באמצעות $Bootstrap$:

טכנולוגיית $Bootstrap$ היא סביבת עבודה בקוד פתוח לצד לקוח, המכילה אוסף של כלים ליצירת אפליקציות רשת ואתרים. טכנולוגיה זו מורכבת משלושה חלקים:

- אוסף מחלקות CSS שמתאימות לסיטואציות נפוצות באתרי אינטרנט.
- רכיבי ממשק משתמש - רכיבים שכתובים ב-CSS וב-$JavaScript$ שנועדו להקל על המפתח לייצר ממשק משתמש איכותי, כדוגמת כפתורים, תמונות, מד-התקדמות והודעות.
- $JavaScript$ - קיומם של סקריפטים בצורת תוספים ל-$JQuery$(ספריית $JavaScript$ הנתמכת ע"י דפדפנים רבים שמטרתה להקל על כתיבת סקריפטים לצד הלקוח), שמוסיפים אפשרויות מתקדמות כמו חלונות קופצים או גרפים. בנוסף, קיימת אפשרות כך שהתוספים ירחיבו פונקציונליות של רכיבים קיימים.

בפרויקט ה-web השתמשנו בטכנולוגיה זו לצרכי הצגת הפסנתר ומימוש הפונקציונליות שלו.

דוגמא לכפור שנעשה ע"י שימוש - $Bootstrap$:



$JavaScript$

שפת תכנות דינמית מונחית-עצמים המותאמת לשילוב באתרי אינטרנט ורצה ע"י הדפדפן בצד הלקוח. השפה מרחיבה את יכולות שפת התגיות הבסיסית $HTML$ ומאפשר בכך ליצור יישומי אינטרנט מתוחכמים יותר.

השפה הייתה הדרך הראשונה להריץ קוד בצד הלקוח ועד היום נשארה הדרך המרכזית והנפוצה ביותר למטרה זו, בנוסף לאחרות שהתווספו היום כדוגמת $Flash$. ל-$JavaScript$ יש את היתרון שהיא אינה דורשת התקנה של רכיב חיצוני נוסף על הדפדפן והחיבור שלה עם שאר מאפייני העמוד היו החלק ביותר.

באתר שלנו, ה- $JavaScript$ בא לידי ביטוי בחלק של הפסנתר האינטראקטיבי.

שרת מארח

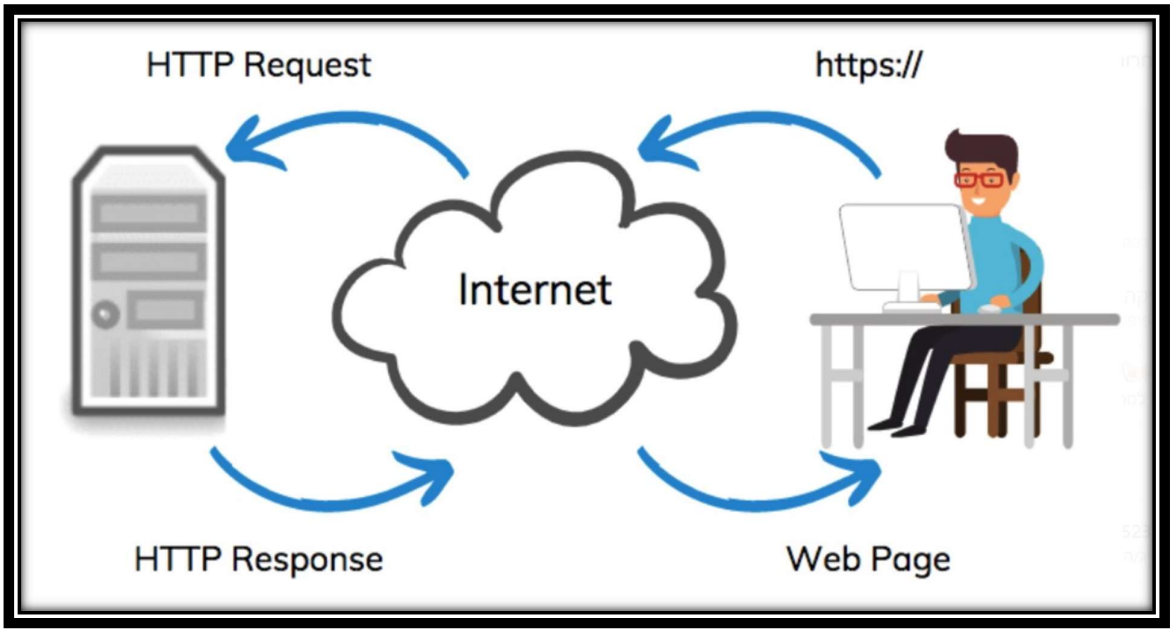
מהו שרת מארח?

שרת מארח הוא מחשב או התקן אחר המחובר לרשת מחשבים, והוא עשוי לעבוד כשרת המציע משאבי מידע, שירותים ויישומים למשתמשים או למארחים אחרים ברשת. מארחים ברשת כוללים לקוחות ושרתים אשר שולחים או מקבלים נתונים, שירותים או יישומים שונים.

מחשב המשתתף ברשתות המשתמשות בחבילת פרוטוקולי האינטרנט נקרא גם מארח IP. במובן זה, מחשבים המשתתפים באינטרנט נקראים מארחי אינטרנט. למארחי אינטרנט ולמארחי IP אחרים יש כתובת IP אחת או יותר שהוקצו לממשקי הרשת שלהם כאשר כתובות אלו מוגדרות באופן ידני על ידי מנהל מערכת או אוטומטית בעת האתחול באמצעות פרוטוקול ה- $DHCP$ או ע"י שיטות קביעת תצורה אוטומטית של כתובות חסרות מצב.

בנוסף, מארחי רשת המשתתפים ביישומים המשתמשים במודל לקוח-שרת של מחשוב, מסווגים כמערכות שרת או לקוח.

ברשת TCP/IP, לכל מארח יש מספר מארח שיחד עם זהות רשת, יוצר כתובת IP ייחודית משלו. במודל $Open\ Systems\ Interconnection\ (OSI)$, פרוטוקולים בשכבת התעבורה, המכונה גם $Layer\ 4$, אחראים על התקשורת בין המארחים. מארחים משתמשים בפרוטוקולים שונים כדי לתקשר, כולל פרוטוקול בקרת שידור (TCP) ופרוטוקול $User\ Datagram\ (UDP)$.



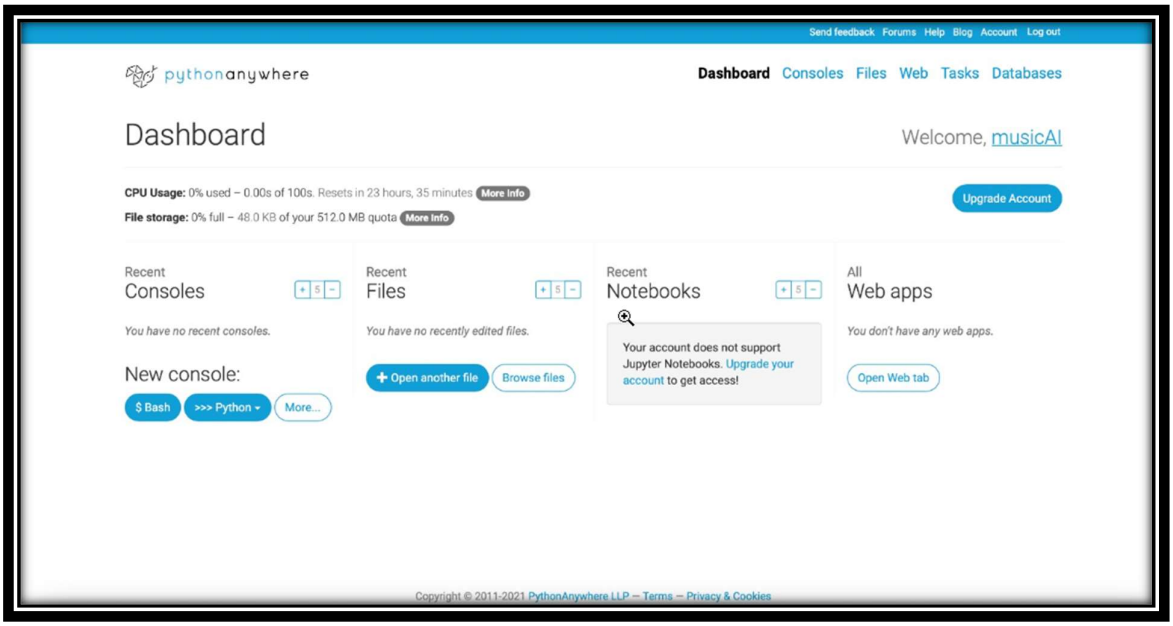
Pythonanywhere

$Pythonanywhere$ היא סביבת פיתוח משולבת מקוונת(IDE) המספקת גם שירות ואירוח אתרים

($Platform\ as\ a\ service$) המבוססת על שפת התכנות $Python$. סביבה זו מספקת גישה בדפדפן לממשקי $Python$ ו- $Bash$ מבוססי שרת, יחד עם עורך קוד עם הדגשת תחביר ($syntax\ highlighting$).

בנוסף, ניתן להעביר קבצי תוכניות אל השירות וממנו באמצעות הדפדפן של המשתמש וכך גם לכתוב יישומי אינטרנט המתארחים בשירות באמצעות כל מסגרת יישום מבוססת $WSGI$ ($Web\ Server\ Gateway\ Interface$) - ממשק שער האינטרנט של שרת האינטרנט היא קונבנציית קריאה פשוטה לשרתי אינטרנט המשתמש להעברת בקשות ליישומי אינטרנט או מסגרות שנכתבו בשפת התכנות $Python$).

בחרנו להשתמש בשרת זה מאחר והוא מספק שירות בסיסי חינמי ובנוסף גם ידידותי למשתמש.



References

- *Django* - https://docs.djangoproject.com/en/3.2/
- *Pythonanywhere* - https://www.pythonanywhere.com/