Peleg Adi Hatuka – 208713594
Eden Hayat - 206921280

נושאים במערכות מידע ושפות תכנות

GoLang

מגישים:

- פלג עדי חתוכה
- עדן חייט

Peleg Adi Hatuka – 208713594
Eden Hayat - 206921280

תוכן עניינים:

כיצד להתקין את המיני פרוייקט ולהריץ אותו:

ראשית, דרישות הקדם שמצריך הפרוייקט הן:

- התקנת MySQL relational database management system (DBMS)
- התקנה של Go

נקודות חשובות לפני הרצת הפרוייקט:

- בעת התקנת MySql בצורה לוקאלית, יש ליצור שם משתמש וסיסמא, אשר אותם תכניסו כמשתנים גלואבאליים בקוד. יש להחליף את שם המשתמש והסיסמא בקובץ dbInit.go בשורות הללו:

```go
func CreateDb() {
    config := mysql.Config{
        User:   "root",
        Passwd: "Pelegedendb", //
        Net:    "tcp",
        Addr:   "127.0.0.1:3306",
    }
```

```go
func connectToDb() (error, *sql.DB) {
    config := mysql.Config{
        User:   "root",
        Passwd: "Pelegedendb",
        Net:    "tcp",
        Addr:   "127.0.0.1:3306",
        DBName: DatabaseName,
    }
```

- בנוסף, אם אתם מעוניינים להריץ את הפרוייקט תחת Port ספציפי, יש לשנות את הפורט לפורט שבו אתם מעוניינים בקובץ ApiInit.go :
- $port \in \{8080,9000,3000\}$

```go
    // Start to listen and server
    log.Fatal(http.ListenAndServe( addr: ":9000", c.Handler(r)))
    fmt.Printf( format: "Server start working on port 9000")
```

כעת יש להוריד את הפרוייקט מדף GitHub שלנו, שבו ניהלנו את הפרוייקט. לאחר מכן יש להכנס לתקיית הפרוייקט אשר הורדנו כעת, ולבסוף יש לבצע בנייה של הפרוייקט והרצת קובץ ה-executable אשר נוצר מבניית הפרוייקט, ושמו לפי שם הפרוייקט.
ניתן לבצע את רצף הפעולות הללו בצורה הבאה:

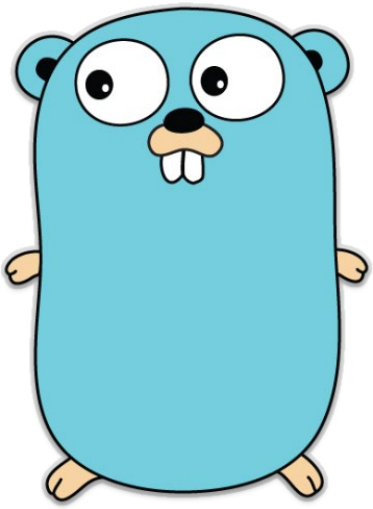
- git clone https://github.com/peleghat/miniProject.git
- cd /miniProject
- go build && ./miniProject

- כששם הפרוייקט הוא miniProject.
- לבסוף יש לפתוח את ה-Swagger בPort אשר נבחר ולשלוח פקודות API לשרת אשר מאזין.

Peleg Adi Hatuka – 208713594
Eden Hayat - 206921280

<u>מידע כללי על – Golang</u>:

- Go היא שפת תכנות התומכת במקביליות ובתכנות מונחה עצמים באופן חלקי. תחביר השפה דומה לשפת C. השפה נכתבה על ידי מפתחים של גוגל והוכרזה רשמית בשנת 2009. מטרת השפה היא לאפשר כתיבת תוכניות יעילות כמו שפות מקומפלות בעלות טיפוסים סטטיים, ומצד שני לאפשר את נוחות התכנות שנותנות השפות הדינאמיות.
- השפה אמנם מושפעת תחבירית משפת C, אך בצורה יותר קריאה ופשוטה שמזכירה את שפת Python. מפתחי השפה העידו על שפת Golang שהיא "קלה לכתיבה כמו פייתון ומהירה כמו שפת C"
- אחד מהמוצרים בעלי השימוש הרב ביותר בשוק כיום ושכנכתב בשפת Go הוא Kubernetes (k8s).
- אחד מחסרונות השפה (לפחות לטעמנו) היא ש-Go אינה מנהלת היררכייה של עצמים, אך מאפשרת קומפוזיציה של עצמים, מה שאמור לדמות ברמה מסויימת תכנות מונחה עצמים.
- בדומה ל-C, מבנה השפה אינו מכיל Classes, אלא עובד בתצורת Structs.
- אחד מהייתרונות של שימוש בStructs הוא שפרסור הקלט אשר מגיע מקריאת הAPI כ-Json ואף החזרת הפלט מStruct לJson מתבצע בצורה פשוטה.
- ממשק העבודה שבו בחרנו לעבוד הוא Goland by jetBrains. ניתן לעבוד גם כן עם VSCode, אך Goland התגלה כנוח יותר לעבודה מפני שמבצע לך באופן אוטומטי קונפיגורציות שונות, כך שההגעה לתחילת מלאכת הקוד נהייתה מהירה יותר (לדוגמא, בשימוש עם VSCode יש להגדיר באופן אוטומטי את הGoPath, ואילו בשימוש GoLand הדבר מתבצע בתצורה אוטומטית ומגדיר את הGoPath כתיקיית הפרוייקט).



* להלן הקמע של GoLang

מבנה הפרוייקט:

- ברמת המאקרו:

  - הלקוח שולח בקשת API לוקאלית על ידי שליחת Endpoint כ-Url בצירוף ה-Port, פרמטר ו-Request Body במידת הצורך (תלוי שאילתא).
  - הבקשה מגיעה לServer שמאזין על הPort אשר נבחר.
  - Mux (אשר עליו נפרט בהמשך), הינו הAPI Router אשר מנתב את הבקשה למקום הרלוונטי.
  - APIFunctionHandler מקבל בקשות מהMux ומטפל ברמת פרסור הקלט ובדיקת תקינותו. במידה והקלט מתגלה כתקין, ה-APIFunctionHandler את הקלט לרמת הdatabase.
  - dbFunctions – בתפקידו לנהל את הבקשות מול הdatabase המקומי ולבצע בקשות של הכנסת, שליפת ועדכון מידע בהתאם לבקשה המתקבלת מה-APIFunctionHandler מול הdatabase.
  - Database – מסד הנתונים המקומי שלנו אחראי על אחסון המידע.

** במידה ובאחת הרמות התקבלה שגיאה, השגיאה מפועפעת למעלה עד לרמת הApiFunctionHandler ונשלחת כHttp Error ללקוח בצירוף הסבר לסיבת השגיאה.

- ברמת המיקרו:

בשלב זה, נפרט על כל אחד מהשכבות אשר מימשנו, מבחינת שדות ופונקציות רלוונטיות והחשיבה שמאחורי המימוש בצורה הזאת.

ראשית, כפי שנהוג בדוקומנטציה של RestAPI, חילקנו את מבנה הפרוייקט לתיקיות שונות אשר מייצגות שכבות שונות בפרוייקט, כעת נפרט:

1. Entities Folder – תקייה זו מכילה את כל ה-Structs אשר בפרוייקט:
   a. עבור Task – יצרנו אובייקט כללי מסוג Task אשר מכיל את השדות הגנריים שמשותפים גם ל Chore וגם ל-HomeWork כלומר את השדות (type, status , ownerId, id וDescription). בחרנו להתייחס אל Task אל מחלקה שתורחב על ידי Chore ו HomeWork, תוך ההנחה כי כל Task חייב להיות או מסוג Chore או מסוג Homework. לאחר הבנת מבנה הפרוייקט, החלטנו לאחד את השדות details וdescription כשדה משותף בTask. הדבר לא נראה לעין ברמת הAPI כך שללקוח יש תחושה שלChore יש את הdescription שלו ולכל HomeWork יש את הdetails שלו. דבר זה אפשר לנו לחסוך בשדה ובנוסף בעמודה שלמה במסד הנתונים שלנו (עליו נפרט בהמשך). בנוסף לכך, יצרנו עצם אשר נקרא TaskInOut שלמעשה הוא משמש כמעין מבנה ביניים המפריד בין שכבת מסד הנתונים לבין שכבת הAPI, ובנוסף עזר לנו בפרסור הקלט והפלט ושליחתם.
   b. עבור Person – יצרנו אובייקט כללי מסוג Person אשר מכיל את כל השדות הרלוונטיים (כפי שהוצג במסמך דרישות הפרוייקט), והן : (id, name, email, favoriteProgrammingLanguage ו ActiveTaskCount) יצרנו עצם אשר נקרא PersonInOut שלמעשה הוא משמש כמעין מבנה ביניים המפריד בין שכבת מסד הנתונים לבין שכבת הAPI, ובנוסף עזר לנו בפרסור הקלט ובדיקת תקינותו ובנוסף לפרסור הפלט ושליחתו.

c. עבור Status וSize – מימשנו את שני הEnums הללו, כפי בהתאם לדרישות הפרוייקט. הוספנו לאובייקטים אלו פונקציות אשר אחראיות על ההמרה של הEnum לString על מנת שנוכל לפרסר זאת אל הלקוח.

2. dbFolder – תקייה זו אחראית על אתחול מסד הנתונים וביצוע הפעולות השונות בו
   a. הדרך שבה שמרנו את המידע היא באמצעות שתי טבלאות שונות:
      i. Person Table – אשר מכיל את כל השדות הרלוונטיים של Person. הPrimary key של Person הוא הid הייחודי שלו
      ii. Task Table – אשר מכיל את כל השדות הרלוונטיים של Task, הPrimary key של Task הוא הid הייחודי שלו. הForeign key שלו הוא הOwnerID אשר הוא משמש כ Reference לPersonid בטבלאת הPersons. לאחר נסיונות וחשיבה, החלטנו כי הדרך היעילה היא יצירת טבלא אחת משותפת לTask,Chore,HomeWork, ובכך נחסוך שליפה נוספת ממסד הנתונים. מימשנו זאת בצורה הבאה: אם העצם שאותו נרצה להכניס למסד הנתונים הוא מסוג Chore נמלא את השדות הרלוונטיים לו, ואילו את השדות של HomeWork נכניס למסד הנתונים כNull ולהפך.
   b. dbInit, הוא קובץ המכיל את יצירת הטבלאות וההתחברות אל הdatabase.
   c. dbFunctions מכיל בתוכו את כל הפונקציות אשר מבצעות פעולות שונות על מסד הנתונים. בכל פונקצייה ביצענו התחברות מחדש למסד הנתונים, ובכך המימוש שלנו מאפשר ריצה מקבילית של הקוד, מפני שבכל פעם רק משתמש יחיד יוכל להכנס אל מסד הנתונים ולשלוף/ לעדכן שם מידע. במידה ובמהלך שלב כלשהו התקבלה שגיאה, שלחנו שגיאה רלוונטית אשר תפעפע כלפי מעלה (נפרט על נושא השגיאות בהמשך).

3. APIFolder – זוהי השכבה העליונה, אשר מולה הלקוח מתקשר, ומעביר את בקשותיו הלאה, לאחר בדיקת הבקשה
   a. APIInit – קובץ אשר מאתחל את הRouter (mux), מתחבר אל הפורט הרלוונטי ומבצע האזנה לבקשות חדשות עד לקבלת interrupt.
   b. APIFunctionHandler – אמון על ניתוב הEndpoint אשר התקבל הלקוח בשילוב של סוג הבקשה שקיבל (Post, get וכדומה), הפרמטרים הרלוונטיים, והRequest body של הקריאה (אם קיים). אחראי לקיחת כל המידע של הבקשה המתקבלת מהלקוח והעברתו אל APIFunctions
   c. APIFunctions – קובץ אשר מרכז פונקציה לכל Endpoint אשר יכול להתקבל במהלך הפרוייקט. כל פונקצייה מפרסרת את המידע אשר הלקוח שלח, מבצעת בדיקת תקינות קלט ברמתה ובנוסף קוראת לפונקציה הרלוונטית במסד הנתונים. במקרה ובו הבקשה לא הצליחה הפונקצייה מנתבת את סוג השגיאה אשר התקבלה ושולחת Http Error בליווי הסיבה לשגיאה. במידה והפונקצייה מצליחה לבצע את הבקשה אשר קיבלה, היא מחזירה Http Created/OK, והודעה אשר מכריזה על הצלחת הבקשה, ושולחת Headers רלוונטים בעת הצורך.

4. ErrorFolder – קובץ אשר מכיל את סוגי השגיאות הרלוונטיות במערכת. השגיאות הללו מתקבלות ברמת הdatabase ומפעפעות עד לרמת הלקוח. עברנו על כל התרחישים אשר יכולים לקרות במהלך ביצוע שאילתת SQL (לכל פונקצייה) ויצרנו אובייקט של אותה שגיאה וגרמנו לפונקציות של הdatabase להחזיר את אותן האובייקטים ולהעלותן הלאה, במקום לבצע Panic ולעצור את התוכנית.

5. Main – בעת בניית והרצת הקובץ, קובץ הMain מורץ ומבצע סך הכל 2 פקודות:
   a. יצירת מסד הנתונים
   b. אתחול הServer

אז איך היה לנו ומה למדנו:

1. ראשית, בחרנו את השפה Go, מפני שזו שפה יחסית חדשה, ובנוסף לכך, לא יצא לנו לכתוב בה במהלך התואר.

ייתרונות וחסרונות של השפה (מבחינתנו):

ייתרונות:

- Go מאפשר טעינה קלה של מידע מפורסר מהJson אל הStructs ללא חשיבות לסדר הפרמטרים אשר מתקבלים בבקשת הAPI. דבר זה מתבצע בצורה קלה, מפני שליד כל שדה בStruct של ה Input וOutput הוספנו את המיפוי הרלוונטי שלו ל-Json:

```
ID              string `json:"id"`
Name            string `json:"name"`
```

- פונקציות יכולות להחזיר ערכי החזרה מרובים ומסוגים שונים (להבדיל מJava שהיא קצת יותר אדוקה במקרים אלו).
- כאשר השתמשנו בפונקצייה מחבילה חדשה, השפה ידעה לייבא בעצמה את אותה החבילה.
- ב Go, יש הרבה חבילות אשר ניתן להרחיב, מה שמקל על הכתיבה.
- יש הרבה מידע באינטרנט על השפה, בפרט בקהילות שונות כגון StackOverFlow ו Geeks4Geeks. במקרים בהם נתקלנו בשגיאות מסויימות/ חיפשנו דרכי מימוש שונות/ חיפשנו פונקצייה מסוימת, קיבלנו את התוצאות הרלוונטיות במהירות ולאחר חיפוש פשוט.
- קל להפוך מתודות ושדות מPrivate לPublic וההפך (רק באמצעות שינוי האות הראשונה של השדה/המתודה לאות גדולה באנגלית – בשביל להפוך אותה לפומבית, ואות קטנה בשביל להפוך אותה לפרטית). דבר זה הקל עלינו בעת כתיבת פונקציות וStructs שונים.
- כשפונקצייה מחזירה פלט מרובה, ניתן להתעלם מחלק מההחזרות של הפונקציה בסינטקס די פשוט, לדוגמא:

```
a, _ = func()
```

- סינטקס יחסית קריא ופשוט. אמנם פחות דומה לשפות התכנות אשר התעסקנו בהן עד עכשיו, אך היה ניתן להתרגל אליו בקלות
- עבודה מול ממשקים כגון MySQL וMux, הייתה מהירה, קלה והתבצעה באמצעות שורות קוד ספורות ואינדיקטיביות.
- השפה מבצעת בצורה יפה ואוטומטית הסקת טיפוסים- לדוגמא:

```
a := 5
```

חסרונות:

- כמעט בכל פונקציה אשר עבדנו איתה, אחד מהערכי ההחזרה הוא שגיאה, ויש צורך לבצע בכל שגיאה טיפול ייחודי.
- מתן מענה חלקי לכל נושא הירושה, מחלקות ופולימורפיזם. היה לנו מעט קשה להבין מה המבנה המקובל בGo אשר אמור לייצג בצורה המיטבית את נושא הירושה.
- לעיתים הרגשנו כי השפה "חסרת סבלנות", לדוגמא : על כל משתנה שלא היה בשימוש, קיבלנו שגיאת קומפילציה.
- השפה משתמשת גם בפויינטרים ורפרנסים לפוינטרים, בדומה ל-C. דבר שגרם לנו לקושי מסויים.

2. Mux – הינו המימוש של RestApi שבו בחרנו להשתמש.

<u>ייתרונות וחסרונות של mux (מבחינתנו):</u>

<u>ייתרונות</u>:

- היות ו Go שפה חדשה, היא נותנת מענה לRestApi דרך חבילה זו. מכיוון שחבילה זו היא Build In בGo, נחסך מאיתנו הגדרות וקונפיגרציות מסוימות שיש לבצע לפני תחילת השימוש בחבילה

<u>חסרונות</u>:

- העבודה מול mux הייתה מורכבת, לקח לנו לא מעט זמן בשביל להבין את אופן הביצוע של הדברים, ולבסוף הבנו כי יש לבצע ממש כל שלב בעת קבלה בקשת API – כלומר, יש לפרסר בעצמנו את הבקשה, את הפרמטרים, את הRequestBody, את הURL, ובנוסף, בזמן התגובה, יש לפרסר בנפרד את ה-Headers. לאור ניסיוננו האישי עם FastAPI של פייתון, בו העבודה היא יותר אינטואיטיבית, ברורה וקלה.

3. הגנה מפני מידע מושחט:

- CORS – בעזרתו הגדרנו מה הן הפונקציות המותרות ללקוח והחזרנו שגיאות במידה והוא משתמש בפונקציות שלא נתנו עליהן מענה בסקופ של הפרוייקט.
- עבור כל רשומה במסד הנתונים, סיפקנו Id ייחודי אשר נשאר ברמת הServer ואין ללקוח את היכולת לשנות אותו. הלקוח יכול רק להשתמש באותו Id לצורך ביצוע פעולות שונות על אותה רשומה במסד הנתונים. בנוסף לכך, שמרנו את ActiveTaskCount כשדה בלעדי של מסד הנתונים, כך שמסד הנתונים הוא היחיד שיכול לעדכנו.
- כאשר הוכנס מידע לא תקין או שהתקבלה שגיאה אצלנו במערכת, דאגנו להחזיר שגיאה רלוונטית (לכל מקרה אשר עלול להתקבל) ולטפל בה כנדרש.
- רק פונקציות אשר נמצאות תחת dbFunctions יכולות להתחבר אל הdatabase, ובנוסף, כל אחת מהן מתחילה את ביצוע המתודה שלה בהתחברות אל מסד הנתונים, על מנת למנוע בעיית סנכרון אשר יכולות לגרום להשחטת המידע.