# מסמך תיאור פרויקט High-Level

## פרויקט goGo

## נושאים במערכות מידע ושפות תכנות

**גיל ידגר**

**אביב אוזלבו**

## חלק 1 – דברים לדעת על Golang כדי להתמצא בקוד:

- מבני נתונים - השימוש במבנה נתונים בשפה מתבצע באמצעות Structים דומי מילונים (בדומה לJS). structה יכלול את השדות השונות של האובייקט, ועל מנת להוסיף מתודות לאובייקט נשתמש בפונקציות בsyntax הבא –
  func (p *<StructName>) <FunctionName> (<Parmaeter> <Parameter Type>) <ReturnType>
  Go מאפשר טעינה קלה של מידע מתוך API/DB ישירות לתוך מבני הנתונים הללו, ופרסור קל שלהם לאובייקט JSON לטובת החזרת תשובה ללקוח (יכולת בה השתמשנו רבות לאורך הפרויקט).
- מתודה/אובייקט/שדה יוגדרו כPrivate במידה ושמם יחל באות קטנה, או public במידה ויחל באות גדולה.
- סטנדרט עבודה מול הרבה מחלקות וחבילות בסיסיות בGolang כולל החזרת אובייקט error מפונקציות ובדיקה בפונקציה הקוראת האם הפונקציה נכשלה עם בדיקת err != nil. נורמת הקוד הזאת מאפשרת מעקב קל וברור אחר תוצאת ההפעלה, והתממשקות קלה עם המתודות השונות.
- IDEs נוחים למעבר ועבודה עם go – VS Code, Goland by Jetbrain

## חלק 2 – איך להריץ את הפרויקט:

1. ניתוב Git לשכפול הקוד - https://github.com/Avoz194/goGo.git

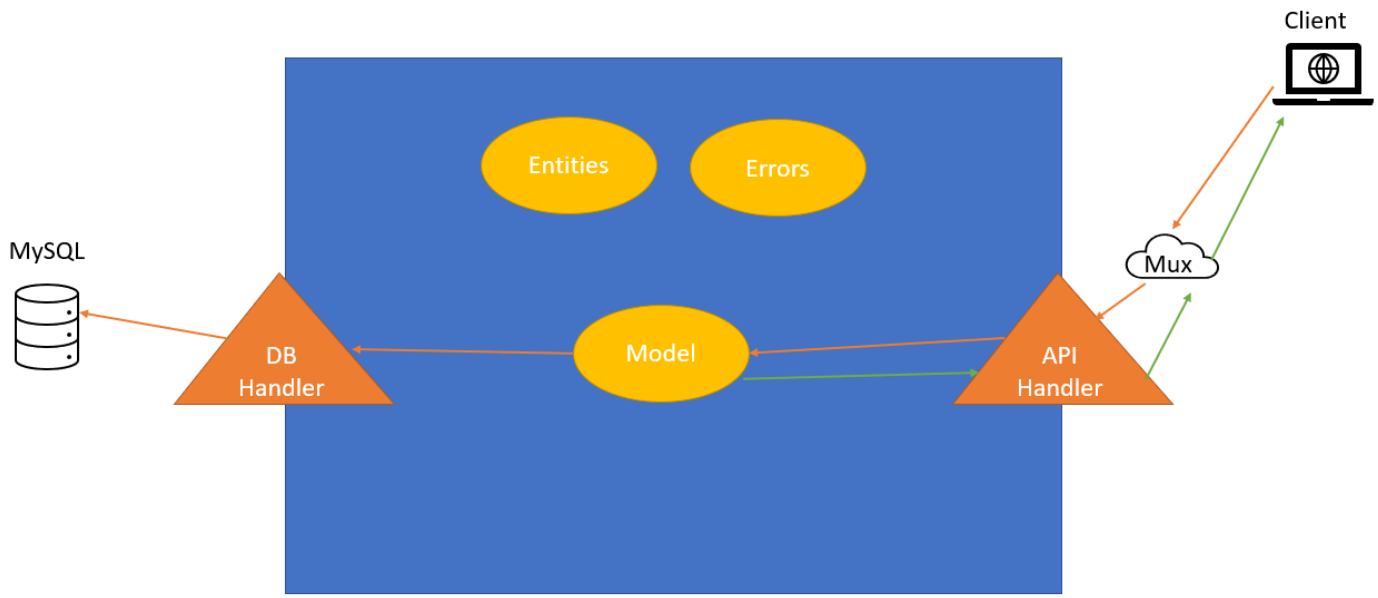
2. MYSQL – לטובת הרצת הפרויקט, חשוב לדאוג להורדת והתקנת שרת MYSQL לוקאלי במחשב, ולדאוג לשם משתמש וסיסמא של Admin User בשרת.

3. הרצת הפרויקט עם קובץ ה exe.:
   a. נתבו את הShell שלכם לתיקייה בה נמצאת קובץ הexe. של הפרויקט (... cd)
   b. הגדר את המשתנים הבאים לכלול את שם המשתמש והסיסמא לשרת MYSQL
      i. set GOGODBPASS=[AdminPass]
      ii. set GOGODBUSER=[AdminUser]
   c. הרץ את קובץ הexe. (goGo.exe)

4. בנייה והרצת הפרויקט מקבצי הפרויקט (לאחר ביצוע clone):
   a. נתבו את הShell שלכם לתיקייה הראשית של הפרויקט
   b. הרץ את הפקודות הבאות לפי הסדר:
      i. go build .
   c. הגדר את המשתנים הבאים לכלול את שם המשתמש והסיסמא לשרת MYSQL
      i. set GOGODBPASS=[AdminPass]
      ii. set GOGODBUSER=[AdminUser]
   d. הרץ את קובץ הexe. (goGo.exe)

**חלק 3 - הפרויקט בHigh-Level:**

**תיעוד LOW LEVEL של הקוד מופיע בהערות לאורכו.*




1. **תבנית עיצוב כללית:**

השרת מומש תוך דגש על רמה מסוימת של אבסטרקציה בין – API – Model – Data, באופן המשלב תבניות עיצוב כמו MVC המשמש לפיתוח UI (בswift לדוגמא), וה-ORM למימוש שמירה וגישה לנתונים של אובייקטים שונים מול מאגר מידע.

בחרנו לבנות את השרת שלנו בחלוקה לרכיבים שונים המבצעים משימות מתחומים שונים, בצורה המפרידה "התמחות שונה" של כל רכיב ורכיב.

2. **עבודה עם חבילות וFrameworks:**

לאור המענה המקיף שנותנת go כשפה חדשה, הן לתמיכה בrestAPI והן לעבודה מול DBs, בחרנו לעבוד ללא שימוש בFramework נוסף, ולהסתפק בחבילות שgo מאפשרת לשני הדברים.

3. **מקביליות:**

שרת מטבעו נדרש לשרת מספר רב של לקוחות, ולקבל, לעבד ולהגיב למספר של פניות.על כן, השרת נדרש לבצע עבודות בצורה מקבילית, תוך הגנה על פעולות שדורשות סנכרון, ונדרש לקחת זאת בחשבון במימושו. בפרויקט בחרנו להסתמך על ניהול מקביליות המתבצעת בשני הרכיבים איתם עובדים:

- תמיכה בפניות API מלקוחות שונים וביצוע פעולות במקביל – מנוהל ומתבצע ע"י הראוטר שנוצר בחבילה MUX בה משתמשים.
- ניהול הסנכרון והעבודה המקבילית מול מאגר המידע – מנוהל ע"י שרת הMySQL איתו עובדים.

4. **שגיאות ממודלות:**

לאור מגוון הפעולות הלוגיות המתבצעות בשרת, קיימים מקרים רבים בהם נאלץ להחזיר שגיאה ללקוח שהעלה פנייה כלשהי. הסיבה לשגיאות מגוונת, ונרצה להבחין ביניהן מאוחר יותר בהחזרת השגיאה ללקוח. על מנת לעשות זאת, מימשנו את סטנדרט החזרת השגיאות המתואר בחלק 1 במסמך זה, יחד עם מידול ועטיפת הerror הסטנדרטי בgo עם wrapper שנועד להחזיר מידע נוסף על השגיאה.

5. **הגנה מData Corruption:**

לאורך המערכת אנחנו דואגים להגן מפני שימוש לא נאות בדרכים הבאות:

- הגדרת פונקציות מותרות בעבודה אל מול הלקוח, שימוש בפרוטוקול CORS על מנת למנוע שימוש אסור והחזרת שגיאות במידה ומתבצעת שגיאה שכזאת.
- הגנה על מידע שמג'ונרט על ידי המערכת עבור האובייקט הספציפי – כמו id או כמות המשימות האקטיביות של person במערכת.
- הגרלה של ids עבור האובייקטים השונים המנוהלים במערכת על מנת למנוע ניחושים/ניסיונות לשיבוש ומחיקת מידע.
- הגדרת PKs FKs בDB שלא מאפשרים הגעה למצבים אסורים – משימות שרשומות על משתמש שנמחק, עבודה עם ids לא קיימים, הגדרת משתמש עם מייל קיים וכדומה.
-

6. **הרכיבים:**

**Model:**

הרכיב המרכזי בשרת שלנו - יהווה המוח של המערכת. המודל ינתב ויפעיל פעולות לוגיות בודדות לפי לוגיקה מתאימה, ישלח בקשות לגישה למידע הנאגר בשרת (אל מול הDB handler), ויקבל בקשות ויחזיר מידע חזרה אל הלקוחות הפונים למערכת (באמצעות הAPI handler).

**API Handler:**

מהווה השער של המערכת שלנו ללקוחות והאחראי על התקשורת מול הלקוחות של השרת. התקשורת תתבצע ותנוהל באמצעות ראוטר שיווצר בחבילת MUX. הראוטר יצור go routine (thread) נפרד לכל לקוח, יקבל את הבקשות ממנו ויחזיר את התשובה המתאימה לכל בקשה.

חלקי הרכיב:

- API Handler – פתיחת השרת, הגדרת נהלי CORS המתאימים (Allowed Origins, Methods, Headers), הבקשות המאושרות והפונקציות המתאימות בAPI Handler להתמודדות עם כל בקשה.
- API Function – התאמה בין כל בקשה חוקית בשרת לפונקציה המטפלת בה, מחלצת את הנתונים המתאימים מהבקשה, הפעלת הפונקציה במודל והחזרת המידע המתאים/השגיאה ל Client.
- API Entity Holders – הגדרת מבני נותנים מתאימים לקליטת המידע הנשלח מהלקוח או להמרת מבני הנתונים המגיע מהמודל לפורמט הנדרש לשליחה ללקוח.

**DB Handler:**

הרכיב האחראי על שמירת המידע בשרת. תוך שימוש בחבילות מתאימות, הרכיב יקים DB של MySQL (במידה ולא קיים) וטבלאות מתאימות לאגירת המידע, י

- DB Generator – אחראי על הקמת הDB והתחברות אליו. *(Repository במבנה הORM).*
- DB Handler - פונקציות שונות המפעילות שאילתות לגישה לנתונים שונים בDB *(הDAO במבנה הORM).*
- MYSQL DB – goGODB, שמאפשר שמירת המידע של השרת.
  - Persons
    - ID – PK
    - Email מוגדר כערך חד-חד ערכי
  - Tasks
    - ID – PK
    - ownerID – FK מול טבלת Persons, StatusID מול טבלאת TaskStatus
  - TaskStatus
    - ID – PK
    - טבלה סטטית, מצמידה תיאור לכל StatusID

**GoGoErrors:**

מבנה נתונים שנועד לעטוף את ממשק הerror הדיפולטי של go. המבנה מתאים סוג לכל שגיאה, שומר נתונים מתאימים על השגיאה ומחזיר טקסט מתאים.
מתודות עיקריות:

- GetError() – מחזירה אובייקט מסוג error שמתאר את השגיאה.
- Error() – כמרחיבה את ממשק error של go ובהתאם לסטנדרט שלה, הפונקציה Error() תחזיר מחרוזת שתתאר את השגיאה שהתרחשה.

**Entities:**

מבנה נתונים מתאים לכל אובייקט במערכת – Person, Task, Status. *(הDTO במבנה הORM).*
מבני הנתונים הללו נועדו לקלוט מידע מהסוג הרלוונטי, להפעיל עליו לוגיקה/פרסור מסויים ולהיקלט בהמשך ע"י רכיבי המערכת השונים (בעיקרם ה API Handler וה DB handler).

- **Status** – מבנה נתונים מסוג Enum הממפה את הסטטאוסים החוקיים לTasks בשרת. במידה והתקבל Status לא חוקי, המערכת תתריע על כך ללקוח.
- **Person –** ביצירת אובייקט מסוג Person המערכת תג'נרט לו אוטומטית Unique ID ותעדכן בהמשך את כמות המשימות הפתוחות שלו.
- **Tasks –** ביצירת אוביקיט מסוג Task המערכת תג'נרט גם כן id אוטומטי עבורו ותמיר את הdueDate שהתקבל לאובייקט מסוג Time. נשים לב שלאור אי הבהירות מסביב לפורמט התאריך המדויק בחרנו לתמוך כרגע בפורמט המוצג בswagger כתאריך הכולל יום בלבד, אך ניתן בקלות להמירו לפרוטוקול RFC3339 במידת הצורך.