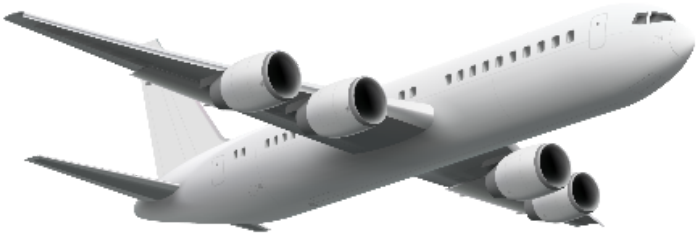
פרויקט מערכת טיסות

קורס פייתון - John Bryce

בניית Web API ב-Node JS



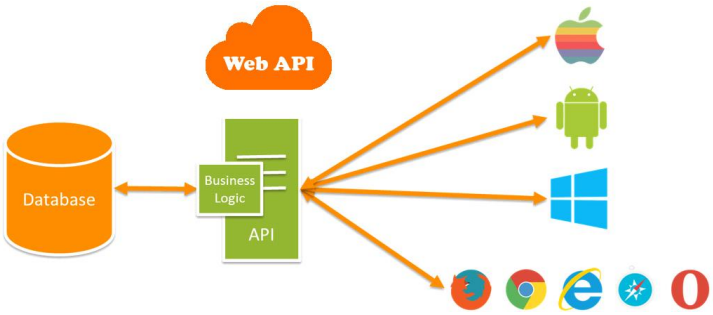
JOHN BRYCE
תלמד ה'-טק. זה עובד!
a matrix company

תיאור:

כפי שאנו כבר יודעים, כדי לחשוף את האפליקציה שלנו לעולם החיצון, עלינו לבנות Web API.

בחלק זה של הפרויקט נבנה אלטרנטיבה (חלופה) ל-Web API שבנינו בחלק ג' באמצעות Flask.
לצורך כך נשתמש ב-Node JS, ב-Express ובידע שצברנו בשיעורים האחרונים.

מטרתו היא לספק את הפעולות הסטנדרטיות של REST, והן: GET, PUT, POST, DELETE ו-PATCH.



עקרונות ביצוע:

1. שרת ה-Node JS יקבל בקשות מן ה-Client ויטפל בהן מול ליבת הפרויקט (שכתבנו בפייתון)
2. ה-Node יטפל גם בבקשות הזדהות (Authentication) בעת ביצוע Login ו-Sign Up.
3. כל ההתממשקות מול ליבת הפרויקט תתבצע באמצעות הודעות Rabbit.
4. (*אתגר) ניהול המשתמשים יתבצע באמצעות replica (העתקה) של טבלת המשתמשים (Users) אל MongoDB.

שלב 1:

ראשית נקים את הפרויקט באמצעות npm, ונתקין בו (עם npm install) את החבילות הדרושות:

- express
- cors
- mocha
- chai
- amqplib
- mongoose (למי שפותר את האתגר)

אפשר (ומומלץ!) להשתמש **בפרויקט smoothie מהבלוג שלנו**, כבסיס.

כל משתמש שיבצע Login יקבל JWT Token.
משתמש שרוצה לבצע פעולה (לקוח, חברה או אדמין) יצטרך להזדהות באמצעות ה-Token שלו.

באמצעות Express נבנה Web API אשר חושף ל-Client את כל הפונקציות של ליבת הפרויקט שלנו:
ניהול טיסות, חברות, לקוחות, משתמשים; הוספה, עדכון, מחיקה וכו'.
נשתמש בתצורה של GET POST PUT DELETE

כמו שעשינו ב-Flask, אנו נייצר ארבעה Controllers, אחד לכל סוג משתמש:

- *Anonymous Controller*
- *Customer Controller*
- *Airline Controller*
- *Admin Controller*

בתוך ה-Controllers יהיו הפונקציות השונות.

לכל פונקציה ניתן Route ייחודי משלה. נקפיד לתת שמות משמעותיים.
בפונקציות מסוג GET ניתן להשתמש ב-query parameters.

הערה:
כך תוכלו לפתוח את חסימת ה-CORS, על מנת להקל על הפיתוח בעתיד (לא חובה):

```
var cors = require('cors')
app.use(cors())
```

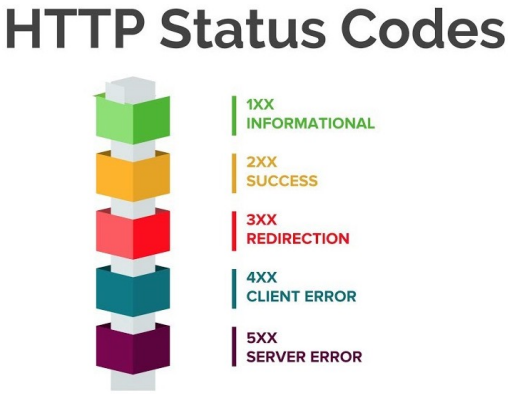
לצורך ה-authentication נשתמש בטכנולוגיית JWT כפי שלמדנו.

1. כאשר המשתמש יבצע login אנו נבצע פעולת decode ונחזיר אליו jwt token.
2. כאשר המשתמש יבצע signup המערכת תייצר עבורו public id ותשלח לו.

את ה-secret key של ה-jwt יש לאחסן בקובץ config.

על השרת להחזיר את הסטטוסים הבאים:

- 200 - כאשר הפעולה הצליחה (נשתמש בזה בפעולות GET או POST)
- 204 - כאשר הפעולה הצליחה ואין תוכן שצריך להחזיר (נשתמש בזה ב-PUT ו-DELETE)
- 400 - כאשר משהו אינו תקין בבקשה שהגיעה
- 401 - כאשר למשתמש אין גישה למערכת (לא סיפק token, למשל)
- 403 - כאשר המשתמש מנסה לעשות פעולה לא מורשית (למשל customer שמנסה לבצע פעולה של airline)
- 500 - כאשר התרחשה שגיאה כלשהי בצד השרת

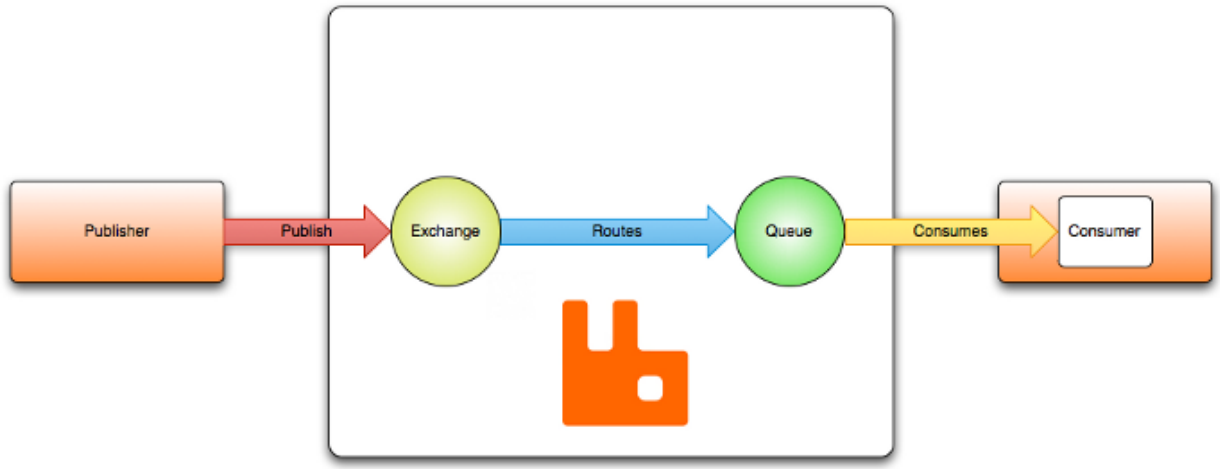


שלב 2:

נחבר בין ה-Node JS לליבת הפרויקט באמצעות RabbitMQ.

בכל פעם שה-Node יקבל בקשה מן ה-Client, הוא ישגר הודעה באמצעות ה-Rabbit אל צד ה-backend, שיקשיב כל העת. לאחר הטיפול בבקשה, תוחזר תשובה אל ה-Node והוא בתורו ישיב תשובה ל-Client.

כל הפעולות במערכת (כולל אותנטיקציה) יבוצעו בשיטה זו.



יש להעביר דרך ה- rabbit את הבקשות של הפרוייקט (שהגיעו בתצורת GET POST PUT DELETE) עבור כל אחד מהמשתמשים: anonymous, admin, airline, customer. ולאחר מכן להמתין לתשובה מה- backend. ולהשיב http status מתאים ללקוח

שלב 3:

עלינו לכתוב טסטים ל-Web API שלנו באמצעות mocha ו-chai כפי שלמדנו בשיעור, על מנת לבדוק את אמינותו ויציבותו.

לפני כתיבת הטסטים, מומלץ לבדוק את הפונקציות שלנו ב-Postman.

נכתוב לפחות 4 טסטים אשר בודקים ארבעה פונקציות שונות ב-Web API.

*אתגר:

בכל פעם שהשרת יידרש לבצע אימות משתמש (Login) או הרשמת משתמש חדש, הוא יפנה להעתק של טבלת המשתמשים שניצור ב-MongoDB (וזאת במקום לעבוד מול צד ה- backend)

1. ראשית נייצר טבלה בשם Users ב-MongoDB (ניתן באופן מקומי על המחשב, או בענן Atlas) ובה יהיו כל נתוני המשתמשים (כולל הסיסמאות שלהם).

2. נחבר את ה-Node ל-Mongo כפי שלמדנו, באמצעות mongoose.

3. כל נושא האותנטיקציה (לוגין, הרשמה) יתבצע מול ה-Mongo.

הנחיות נוספות:

- השתמשו ב-Middlewares כפי שלמדנו בשיעור.
- שמרו על קוד נקי וגנרי כמה שיותר.
- החזיקו קובץ קונפיגורציה לצורך שמירת connection string וכדומה.
- מומלץ לבדוק את כל הפונקציות באמצעות Postman.

בהצלחה !!!