ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ
ΣΧΟΛΗ ΟΙΚΟΝΟΜΙΑΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ
ΠΜΣ ΣΤΗΝ ΕΠΙΣΤΗΜΗ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΡΓΑΣΙΑ ΣΤΟ ΜΑΘΗΜΑ

«ΘΕΜΑΤΑ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ»

Developing RESTful Web Services with Node.js and Express

Καραπιπεράκης Εμμανουήλ

Α.Μ.: 2022202302008

Τρίπολη | Φεβρουάριος 2024

Περιεχόμενα

Ευρετήριο Εικόνων

Ευρετήριο Πινάκων

Περίληψη

Σκοπός της συγκεκριμένης εργασίας αποτελεί η μελέτη των υπηρεσιών δικτύου (web services) καθώς και η πρακτική τους εφαρμογή κάνοντας χρήση του Node.js και της βιβλιοθήκης Express, εμβαθύνοντας στην αρχιτεκτονική REST και στις θεμελιώδεις αρχές του πρωτοκόλλου HTTP. Αρχικά θα πραγματοποιηθεί αναφορά στο πρωτόκολλο HTTP, βασικό πυλώνα της αρχιτεκτονικής REST, που θα αναλυθεί στη συνέχεια, σε συνδυασμό με τις μεθόδους ασφάλειας και τα μοτίβα σχεδίασης που εφαρμόζονται σε αυτό. Το Node.js και η βιβλιοθήκη express θα χρησιμοποιηθούν ως το μέσο μετάβασης από τη θεωρία στην πράξη, κάνοντας αναφορά σε έννοιες όπως Middleware, Routing, Authentication, API Validation & Documentation. Τέλος θα πραγματοποιηθεί συνοπτική αναφορά στην εφαρμογή αρχών μηχανικής μάθησης ως μέρος των υπηρεσιών δικτύου, εστιάζοντας στην αρχιτεκτονική των εικονικών βοηθών.

Λέξεις – κλειδιά: API, Web Services, HTTP, REST, Node.js, Express, DialogFlow

1 Εισαγωγή

Καθημερινά ένας επισκέπτης του διαδικτύου αποκτά πρόσβαση σε μεγάλο όγκο πληροφορίας, είτε ανοίγοντας κάποια εφαρμογή στο κινητό του ή κατά την επίσκεψή του σε κάποια ιστοσελίδα μέσα από ένα πρόγραμμα περιήγησης. Ο χρήστης, χωρίς να αναλαμβάνει ενεργά μέτρα, αποκτά πρόσβαση σε πληροφορίες που προσφέρονται από διάφορες υπηρεσίες, δημιουργώντας ερωτήματα τόσο για την προέλευση των πληροφοριών (πηγή) όσο και για τον τρόπο μετάδοσής τους από τη μία άκρη στην άλλη. Αυτή η επικοινωνία επιτυγχάνεται μέσω Διεπαφών Προγραμματισμού Εφαρμογών (**APIs**), προσφέροντας στις επιχειρήσεις τη δυνατότητα να μοιράζονται πόρους τους με το κοινό και παρέχοντας στον χρήστη, ο οποίος μπορεί να βρίσκεται σε οποιαδήποτε άλλη άκρη του κόσμου, πρόσβαση σε αυτούς. Οι υπηρεσίες που υποστηρίζουν αυτή την αμφίδρομη επικοινωνία ανάμεσα σε δύο άκρα, χρησιμοποιώντας ως μέσο το διαδίκτυο γνωστοποιούνται ως υπηρεσίες δικτύου (**web services**).

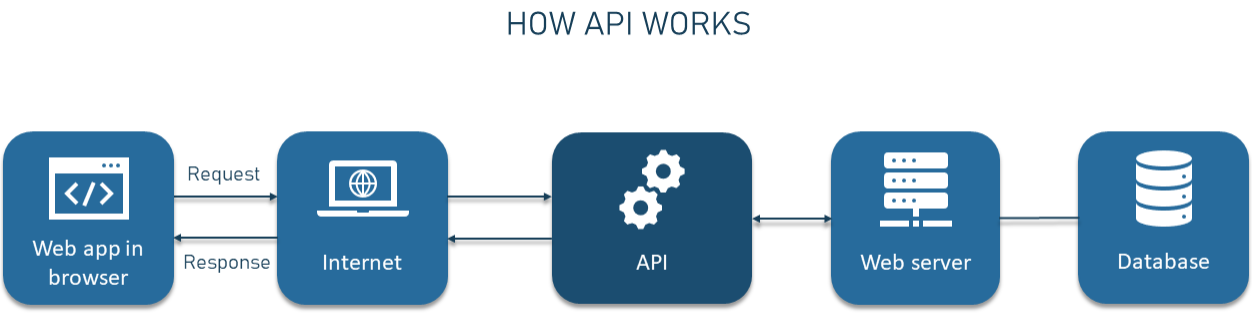
Καθώς χρησιμοποιείται ως μέσο μετάδοσης αυτής της πληροφορίας το διαδίκτυο, σημαντικό ρόλο στην επίτευξη της επικοινωνίας διαδραματίζει το πρωτόκολλο **HTTP**. Αποτελώντας τον βασικό πυλώνα της αρχιτεκτονικής **REST** (Representational State Transfer), η οποία αποτελεί ένα είδος υπηρεσίας διαδικτύου και χρησιμοποιείται στην ανάπτυξη διαδικτυακών εφαρμογών για την ανταλλαγή πληροφοριών μεταξύ πελάτη και διακομιστή. Για την προγραμματιστική υλοποίηση των restful web services προσφέρονται πολλές επιλογές, μερικές εκ των οποίων είναι Django (Python), Spring Boot (Java) και Express JS (JavaScript). Τα τελευταία χρόνια, η άνοδος του **Node.js** έχει καταστήσει τη JavaScript μια πολύ δημοφιλή επιλογή στην προγραμματιστική κοινότητα, προσφέροντας τη δυνατότητα εκτέλεσης κώδικα γραμμένο σε JavaScript έξω από τον περιηγητή ιστού και με τη βοήθεια της βιβλιοθήκης Express την ανάπτυξη εφαρμογών που βασίζονται σε REST APIs.

Η δομή του υπόλοιπου της εργασίας έχει ως ακολούθως: Στο εισαγωγικό κεφάλαιο παρουσιάζονται οι βασικές έννοιες που σχετίζονται με τις υπηρεσίες διαδικτύου. Στη συνέχεια, στο κεφάλαιο 3 θα αναλυθούν διάφορες πτυχές του πρωτοκόλλου HTTP, όπως οι μέθοδοι, τα αιτήματα, οι αποκρίσεις και οι επικεφαλίδες που μπορεί να περιλαμβάνουν. Βασιζόμενο στο πρωτόκολλο HTTP, στο κεφάλαιο 4 θα αναλυθεί ο όρος REST API μαζί με τις βέλτιστες πρακτικές που εφαρμόζονται σε αυτό σε θέματα ασφαλείας και σχεδιασμού. Στο επόμενο κεφάλαιο θα πραγματοποιηθεί μετάβαση από τη θεωρία στην πράξη, εισάγοντας το Node.js και τη βιβλιοθήκη express ως ένα τρόπο ανάπτυξης Restfull APIs, εστιάζοντας σε διάφορες τακτικές όπως middleware, routing, controllers, πιστοποίηση χρήστη, API Documentation & Validation. Κλείνοντας στο κεφάλαιο 6 θα αναλυθούν οι αρχές μηχανικής μάθησης ως μέρος των υπηρεσιών δικτύου, εστιάζοντας στην αρχιτεκτονική των εικονικών βοηθών με χρήση του Dialogflow της Google.

2 Web Services

APIs (Application Programming Interface-Προγραμματιστική Διασύνδεση Εφαρμογών) είναι ένα σύνολο κανόνων και πρωτοκόλλων που επιτρέπουν την επικοινωνία ανάμεσα σε δύο στοιχεία λογισμικού. Πρόκειται για μια διεπαφή λογισμικού προς λογισμικό (software-to-software) που καθορίζει τη σύμβαση ανάμεσα σε εφαρμογές ώστε να επικοινωνούν μεταξύ τους χωρίς να απαιτείται αλληλεπίδραση με το χρήστη. Κάθε συναλλαγή που περιλαμβάνει APIs αποτελείται από τρία μέρη:

1. Αποστολή αιτήματος
2. Εκτέλεση κώδικα
3. Επιστροφή απόκρισης



Εικόνα 1 – API Transaction

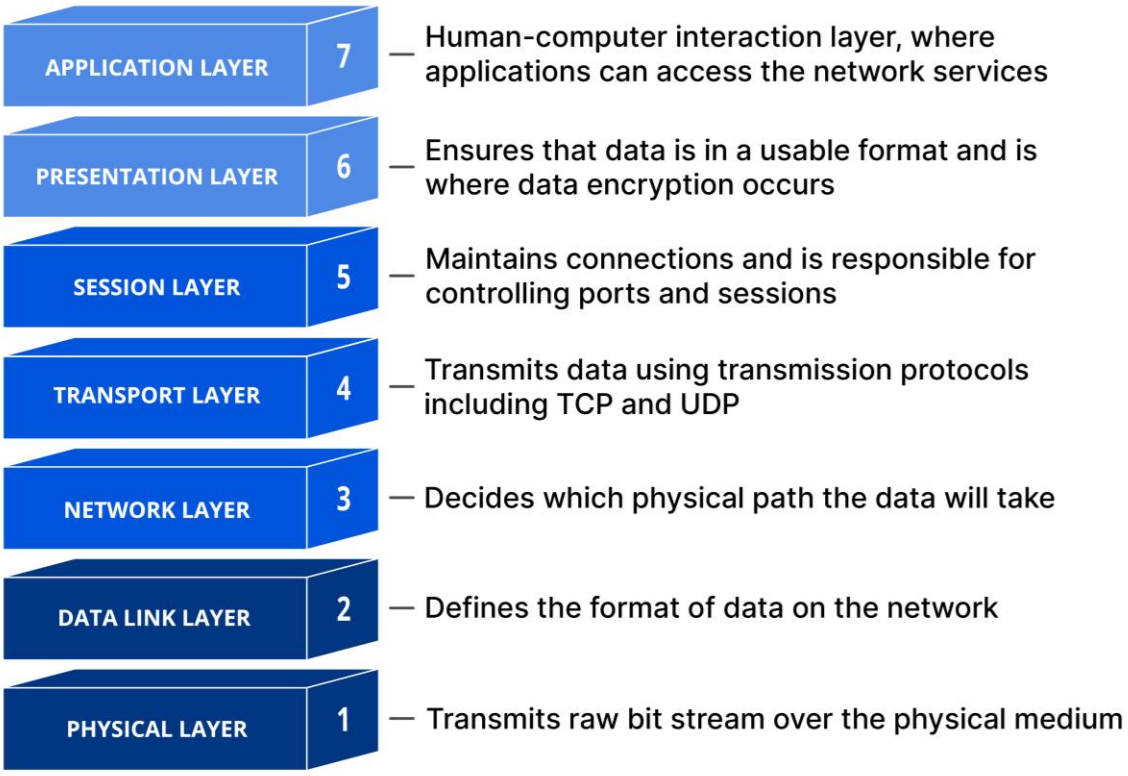
Ως Web Service (υπηρεσία δικτύου) χαρακτηρίζεται ένα σύστημα λογισμικού ή μία συνιστώσα λογισμικού, το οποίο έχει σχεδιαστεί για να υποστηρίζει δια λειτουργική αλληλεπίδραση ανάμεσα σε δύο μηχανήματα μέσω δικτύου. Με απλά λόγια πρόκειται για ένα API που χρησιμοποιεί το διαδίκτυο ως μέσο μεταφοράς δημιουργώντας τον ακόλουθο ισχυρισμό:

- Κάθε web service υλοποιεί ένα API
- Ένα API δεν είναι απαραίτητα web service

Εικόνα 2 – Web Service

3 HTTP

Το HTTP (Hypertext Transfer Protocol) αποτελεί το θεμέλιο του παγκόσμιου ιστού και χρησιμοποιείται για τη φόρτωση ιστοσελίδων εφαρμόζοντας συνδέσμους υπερκειμένου. Το HTTP είναι ένα πρωτόκολλο επιπέδου εφαρμογής (Application Layer) σύμφωνα με το μοντέλο του OSI (Open System Interconnection) που έχει σχεδιαστεί για τη μεταφορά πληροφοριών μεταξύ δικτυωμένων συσκευών.



Εικόνα 3 - OSI model

3.1 HTTP Request

Ως HTTP request (αίτημα HTTP) ορίζεται ο τρόπος με τον οποίο ένας πελάτης (client), για παράδειγμα ένας περιηγητής ιστού ζητάει πρόσβαση σε κάποια πληροφορία από έναν διακομιστή (server). Ένα ορθά διατυπωμένο HTTP request αποτελείται από:

1. Τη γραμμή αιτήματος (Request line)
2. Τις επικεφαλίδες (HTTP Headers)
3. Το σώμα (Body)

```
POST /data/api/login HTTP/1.1
Host: c328-2a02-587-807d-c300-9188-6378-a64d-16ae.ngrok.io
User-Agent: Expo/1017565 CFNetwork/1410.0.3 Darwin/22.6.0
Content-Length: 39
Accept: */*
Accept-Encoding: gzip, deflate, br
Accept-Language: el-GR,el;q=0.9
Content-Type: application/json
{"username":"admin","password":"admin"}
```

Εικόνα 4 - HTTP Request

3.1.1 HTTP Request Line

Το Request line συνήθως αποτελείται από τρία μέρη:

1. HTTP method: Μονολεκτική εντολή που δηλώνει στο διακομιστή την ενέργεια που αναμένεται να πραγματοποιηθεί
2. URL path: Η διαδρομή που προσδιορίζει τον πόρο (resource) του διακομιστή (στο παράδειγμα /data/api/login)
3. HTTP version number: Η έκδοση του πρωτοκόλλου HTTP, την προδιαγραφή της οποίας ακολουθεί το αίτημα

Επιπρόσθετα ένα request line μπορεί να περιλαμβάνει:

- Query string: Επιπλέον κείμενο πληροφορίας διαχειρίσιμο από το διακομιστή. Ακολουθεί μετά το path, χρησιμοποιώντας ως διαχωριστικό το σύμβολο '?'
- Scheme and host components του URL: Γνωστό ως absolute URI, χρησιμοποιείται συνήθως όταν το αίτημα απαιτείται να περάσει από proxy server

```
POST /data/api/login HTTP/1.1
```

Εικόνα 5 - HTTP Request Line

3.2 HTTP Methods

Η μέθοδος HTTP καθορίζει την ενέργεια που αναμένεται να πραγματοποιηθεί από την πλευρά του διακομιστή.

- GET: Χρησιμοποιείται αποκλειστικά για τη λήψη δεδομένων από κάποιον διακομιστή χρησιμοποιώντας ως αναγνωριστικό το URI του αιτήματος. Δύναται να περιλαμβάνει παραμέτρους στο URL (ως τμήμα της διαδρομής ή στο query string) ή Headers αλλά όχι body. Θεωρείται ως μια ασφαλής μέθοδος καθώς λαμβάνει δεδομένα και δεν τροποποιεί τον πόρο του διακομιστή με κανένα τρόπο. Στη περίπτωση που το αίτημα πραγματοποιήθηκε με επιτυχία, συνήθως επιστρέφονται δεδομένα σε μορφή JSON ή XML πίσω στον πελάτη. Στο header *accept* που ορίζεται από τον πελάτη, περιγράφεται η αναμενόμενη μορφή (format) που θα έχει το κείμενο προς επιστροφή. Επιπρόσθετα το εισερχόμενο αίτημα μπορεί να περιλαμβάνει επιπλέον headers που καθορίζουν την πληροφορία που θα επιστραφεί σε ένα

GET αίτημα. *If-Modified-Since, If-Range, If-Match* αποτελούν μερικά παραδείγματα τέτοιων επικεφαλίδων που έχουν ως αποτέλεσμα η μέθοδος GET να χαρακτηρίζεται ως *υπό συνθήκη μέθοδος*. Ο διακομιστής δηλαδή ανταποκρίνεται στο αίτημα, μόνο όταν οι συνθήκες που περιγράφονται στο αίτημα ικανοποιούνται, μειώνοντας έτσι ανεπιθύμητη χρήση του δικτύου. Σε περίπτωση επιτυχίας, η μέθοδος GET επιστρέφει status code 200 (OK) ενώ σε περιπτώσεις σφάλματος μπορεί να επιστραφεί:

  - 400 (Bad request): Το αίτημα δεν είναι ορθό
  - 401 (Unauthorized): Ο χρήστης δεν είναι πιστοποιημένος
  - 404 (Not found): Ο πόρος που στοχεύει το αίτημα δεν υπάρχει, το μονοπάτι το οποίο αναγράφεται στο αίτημα δεν αντιστοιχεί σε υπαρκτό πόρο του διακομιστή

- POST: Η συγκεκριμένη μέθοδος μπορεί να περιλαμβάνει τόσο headers όσο και body. Συνήθως εφαρμόζεται σε δύο περιπτώσεις:
  - Δημιουργία δεδομένων σε μια συλλογή (Βάση Δεδομένων)
  - Εκτέλεση προγραμματιστικής λογικής (functions) που περιγράφονται μέσα σε controllers του διακομιστή. Ως είσοδος σε αυτές τις συναρτήσεις συνήθως είναι δεδομένα που περιλαμβάνονται στο request body

  Σε περίπτωση επιτυχίας, συνήθως ο διακομιστής ανταποκρίνεται με status code 201 (Created), δηλώνοντας πως δημιουργήθηκαν δεδομένα. Σε περίπτωση που δεν πραγματοποιήθηκε εισαγωγή δεδομένων, ενδέχεται να επιστραφεί και 204 (no content). Πολλές φορές όμως η μέθοδος POST δεν ταυτίζεται απαραίτητα με δημιουργία δεδομένων, μπορεί να περιλαμβάνει απλά την εκτέλεση κάποιας λειτουργικότητας επομένως ο διακομιστής δύναται να επιστρέψει και 200 (OK)

- PUT: Ενημέρωση υπάρχουσας εγγραφής. Εάν η εγγραφή προς δημιουργία που περιγράφεται στο αίτημα δεν υπάρχει, τότε δημιουργείται μια νέα. Γνωστό ως Upsert (από τα update/insert) στέλνοντας κάθε φορά όλο το body. Συνήθως το αναγνωριστικό που δηλώνει την εγγραφή προς ενημέρωση, περιλαμβάνεται στο URI. Η απόκριση του διακομιστή εξαρτάται από την ενέργεια που πραγματοποιήθηκε:
  - 201 (Created): Νέος πόρος δημιουργήθηκε
  - 200 (OK) ή 204 (No content): Κάποιος υπάρχων πόρος ενημερώθηκε επιτυχώς
- PATCH: Παρόμοια μέθοδος με την PUT με τη διαφορά πως πραγματοποιεί μερική ενημέρωση, στέλνοντας οδηγίες για το πως να συμπληρωθεί ένα resource και όχι όλο το body όπως προηγουμένως
- DELETE: Χρησιμοποιείται για διαγραφή δεδομένων σύμφωνα με το URI. Σε περίπτωση επιτυχίας επιστρέφεται 200 (OK) ή 204 (No content). Επειδή όμως στην περίπτωση που επιστρέφεται 200, μπορεί να περιλαμβάνεται και η αναπαράσταση του πόρου προς διαγραφή, προτείνεται η χρήση του status code 204 με σκοπό την αποφυγή μεταφοράς περιττού εύρους ζώνης έτσι ώστε να μην παρουσιάζεται συμφόρηση του δικτύου και να βελτιώνεται η απόδοση.
- HEAD: Παρόμοια μέθοδος με τη GET με τη διαφορά πως επιστρέφονται μόνο response lines και headers. Δε μεταφέρονται τα δεδομένα μιας οντότητας αλλά μόνο μεταδεδομένα αυτής, με αποτέλεσμα να μειώνεται το χρησιμοποιούμενο εύρος ζώνης δικτύου. Συχνά χρησιμοποιείται

για σκοπούς ελέγχου έπειτα από τροποποιήσεις αλλά και για εξέταση εγκυρότητας και προσβασιμότητας των υπερσυνδέσμων.

- OPTIONS: Η μέθοδος OPTIONS καθορίζει τις δυνατότητες ενός διακομιστή HTTP, καθώς και τις υποστηριζόμενες κεφαλίδες που επιτρέπουν την αλληλεπίδραση με έναν συγκεκριμένο πόρο. Αν και η μέθοδος αυτή δεν εκτελεί καμία από τις λειτουργίες CRUD (Create, Read, Update, Delete), παρέχει στον πελάτη πληροφορίες σχετικά με τον τρόπο που μπορεί να αλληλεπιδρά με τον συγκεκριμένο πόρο. Ο πελάτης έχει τη δυνατότητα να καθορίσει μια διεύθυνση URL για τη μέθοδο, επιτρέποντάς του να αναφερθεί σε έναν συγκεκριμένο πόρο. Ο αστερίσκος (*) δύναται να χρησιμοποιηθεί στην περίπτωση όπου ο πελάτης επιθυμεί να μάθει πληροφορίες για ολόκληρο το διακομιστή.

Method	Description
GET	Read data
POST	Insert data
PUT	Update entirely or create data
PATCH	Update existing data
DELETE	Delete Data
HEAD	Retrieve only the headers of the response
OPTIONS	Describes the communication options for the target resource

Πίνακας 1 - HTTP methods

3.2.1 Idempotent and Safe Methods

Μερικές μέθοδοι HTTP μπορούν να κληθούν πολλές φορές χωρίς κανένα πρόβλημα επιστρέφοντας πάντα το ίδιο αποτέλεσμα και χωρίς να παράγουν ανεπιθύμητα πλευρικά αποτελέσματα (side effects) ενώ μερικές άλλες όχι. Έτσι δημιουργείται η έννοια χαρακτηρισμού μιας μεθόδου ως **Idempotent** ή/και **Safe**:

- Idempotent: Χαρακτηρίζεται μια μέθοδος, η οποία όσες φορές και να κληθεί, καταλήγει πάντα στο ίδιο αποτέλεσμα και δεν παράγει πλευρικά αποτελέσματα
- Safe: Χαρακτηρίζεται μια μέθοδος η οποία δεν τροποποιεί την κατάσταση ενός resource

Method	Idempotent	Safe
GET	NAI	NAI
POST	OXI	OXI
PATCH	OXI	OXI
PUT	NAI	OXI
DELETE	NAI	OXI
HEAD	NAI	NAI
OPTIONS	NAI	NAI

Πίνακας 2 - Idempotent & Safe methods

3.3 HTTP Responses

Ως HTTP response (απάντηση HTTP) ορίζεται η απόκριση του διακομιστή σε ένα εισερχόμενο αίτημα και αποτελείται από:

- Τη γραμμή κατάστασης (status line)
- Τις επικεφαλίδες (HTTP Headers)
- Το σώμα (body)

```
HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=utf-8
Content-Length: 215
ETag: W/"d7-DtjGyX+0XzL+RtXNxbKkYIr9PsY"
Date: Wed, 01 Nov 2023 16:25:21 GMT
Connection: keep-alive

{"token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImFkbWluIiw
idXNlcklkIjoyNTksImlhdCI6MTY5ODg1NTkyMSwiZXhwIjoxNjk4ODU5NTIxfQ.vQhfM8JLFw
aQUYPBzJcNH0d5ngOWEoU4svet1-cUGwA","username":"admin","userId":259}
```

Εικόνα 6 - HTTP Response

Η γραμμή κατάστασης αποτελείται από τρία μέρη:

1. HTTP Version
2. Status Code
3. Reason Phrase

3.3.1 HTTP Status Code

Το HTTP ορίζει 40 διαφορετικά status codes μέσω των οποίων ο client ενημερώνεται για το αποτέλεσμα του αιτήματός του. Διακρίνονται στις ακόλουθες κατηγορίες:

- 1xx Informational
- 2xx Success
- 3xx Redirection
- 4xx Client Error
- 5xx Server Error

Ακολουθεί ένας πίνακας με τα πιο συχνά χρησιμοποιούμενα status codes, ο οποίος προέρχεται από το σύγγραμμα ***API Management_ An Architect's Guide to Developing and Managing APIs for Your Organization***

Status Code	Reason Phrase	Description
200	OK	Indicates that the request has been processed successfully.
201	Created	Indicates that the request has been processed and a new resource has been created successfully.
202	Accepted	Indicates that the request has been received by the server and is being processed asynchronously.
204	No Content	Indicates that the response body has been purposely left blank.
301	Moved Permanently	Indicates that a new permanent URI has been assigned to the client's requested resource.
303	See Others	Indicates that the response to the request can be found in a different URI.
304	Not Modified	Indicates that the resource has not been modified for the conditional GET request of the client.

307	Use Proxy	Indicates that the request should be accessed through a proxy URI specified in the Location field.
400	Bad Request	Indicates that the request had some malformed syntax error due to which it could not be understood by the server. Probable reason is missing mandatory parameters or syntax error.
401	Unauthorized	Indicates that the request could not be authorized, possibly due to missing or incorrect authentication token information.
403	Forbidden	Indicates that the request was understood by the server but it could not be processed due to some policy violation or the client does not have access to the requested resource.
404	Not Found	Indicates that the server did not find anything matching the request URI.
405	Method Not Allowed	Indicates that the method specified in the request line is not allowed for the resource identified by the request URI.
408	Request Timeout	Indicates that the server did not receive a complete request within the time it was prepared to wait.
409	Conflict	Indicates that the request could not be processed due to a conflict with the current state of the resource.
414	Request URI Too Long	Indicates that the request URI length is longer than the allowed limit for the server.
415	Unsupported Media Type	Indicates that the request format is not supported by the server.
429	Too Many Requests	Indicates that the client sent too many requests within the time limit than it is allowed to.
500	Internal Server Error	Indicates that the request could not be processed due to an unexpected error in the server.
501	Not Implemented	Indicates that the server does not support the functionality

		required to fulfill the request.
502	Bad Gateway	Indicates that the server, while acting as a gateway or proxy, received an invalid response from the back-end server.
503	Service Unavailable	Indicates that the server is currently unable to process the request due to temporary overloading or maintenance of the server. Trying the request at a later time might result in success.
504	Gateway Timeout	Indicates that the server, while acting as a gateway or proxy, did not receive a timely response from the back-end server.

Πίνακας 3 - HTTP Status Code

3.4 HTTP Headers

Τα http headers χρησιμοποιούνται για τη μετάδοση πρόσθετων πληροφοριών ανάμεσα σε client και server μέσα από τα headers του αιτήματος και της απόκρισης του διακομιστή αντίστοιχα. Αναλυτικότερα χωρίζονται σε 4 κατηγορίες:

- **Entity headers**: Περιέχουν μεταδεδομένα αναφορικά με τα body των resources όπως Content-length
- **General headers**: Προσφέρουν πληροφορία που μπορεί να εφαρμοστεί τόσο στα αιτήματα όσο και στις αποκρίσεις όπως connection information
- **Client request headers**: Τα συγκεκριμένα headers περιλαμβάνονται μόνο στα αιτήματα που στέλνονται στον server. Πληροφορίες σχετικά με authorization, encoding και γλώσσα που μπορεί να δεχτεί ο client περιλαμβάνονται σε αυτή την κατηγορία.
- **Server response headers**: Τα συγκεκριμένα headers περιλαμβάνονται μόνο στις αποκρίσεις που στέλνονται από το server πίσω στο client. Πληροφορία όπως το age του response, Etag για caching σκοπούς αποτελούν χαρακτηριστικά παραδείγματα.

Header	Type	Description	Example
Accept	Client Request Header	Χρησιμοποιείται για να ορίσει τα media types (MIME) που γίνονται αποδεκτά από τον client σε μια απόκριση του διακομιστή	Accept: text/html
Authorization	Client Request Header	Προσθέτει authentication πληροφορία που χρειάζεται για να αποκτήσει πρόσβαση ο client σε ένα πόρο του διακομιστή. Εάν απαιτείται από το server και λείπει ή δεν είναι το αναμενόμενο, τότε error response επιστρέφεται με status code 401	Bearer kGmxYZk3cz==
Host	Client Request Header	Το συγκεκριμένο header προσδιορίζει τη διεύθυνση και το port του διακομιστή	Host: en.wikipedia.org
Location	Server Response Header	Χρησιμοποιείται από το διακομιστή για να κάνει redirect τον παραλήπτη σε διαφορετικό URI από αυτό που αναφέρεται στο αίτημα ή σε περιπτώσεις που δημιουργείται ένα νέο resource: • Μετά από επιτυχημένο PUT ή CREATE request (201) • Ο πόρος προς αναζήτηση έχει μετακινηθεί αλλού ή το αποτέλεσμα βρίσκεται σε διαφορετική τοποθεσία (3XX)	Location: http://www.w3.org/pub/WWW/People.html
ETag	Server Response Header	Παρέχει ένα μηχανισμό που επιτρέπει στο server να στέλνει πληροφορία για τη τρέχουσα κατάσταση ενός entity. Πρόκειται για ένα αλφαριθμητικό το οποίο αναγνωρίζει μια συγκεκριμένη έκδοση ενός πόρου. Εάν ο πόρος έχει αλλάξει, τότε και αυτή η τιμή θα έχει αλλάξει. Έτσι συγκρίνεται αυτή η τιμή στον client και το server και στην απόκριση του αιτήματος περιλαμβάνεται	ETag: "6966ac86be"

		μόνο η απαραίτητη πληροφορία.	
Content-Type	General Header	Το συγκεκριμένο header προσδιορίζει το media type του payload	Application/json

Πίνακας 4 - HTTP Headers

3.5 HTTP Body

Http Body είναι τα δεδομένα που στέλνονται κατά την αποστολή ενός αιτήματος από τον client ή κατά την απόκριση του server. Μπορεί να είναι ένα αρχείο εικόνας ή κειμένου, ένα βίντεο ή απλά κείμενο. Στην περίπτωση όπου στέλνονται δεδομένα κειμένου, απαιτείται κατάλληλη μορφοποίηση ώστε να μπορέσει να σταλεί και να επεξεργαστεί από το διακομιστή. Τα πιο γνωστά και χρησιμοποιημένα text formats είναι:

- JSON
- XML

3.5.1 JSON

Το JSON (JavaScript Object Notation) ξεχωρίζει ως μια ελαφριά και ευέλικτη μορφή ανταλλαγής δεδομένων. Η ανταλλαγή μπορεί να γίνει στο πλαίσιο των web services, πάνω από το πρωτόκολλο HTTP. Είναι ευανάγνωστο για έναν άνθρωπο και εύκολο για ένα μηχάνημα να το αναλύσει και να το δημιουργήσει. Προέρχεται από ένα υποσύνολο της γλώσσας προγραμματισμού JavaScript αλλά είναι ανεξάρτητο από την ίδια τη γλώσσα, βασιζόμενο όμως σε συμβάσεις που είναι διαδεδομένες στην προγραμματιστική κοινότητα γλωσσών C όπως C#, C++, Java, Python κ.α. Δομικά το JSON περιστρέφεται γύρω από δύο θεμελιώδη στοιχεία:

1. Μια συλλογή ζευγών ονόματος/τιμών, παρόμοια με αντικείμενα, εγγραφές και δομές
2. Μια ταξινομημένη λίστα τιμών, που εμφανίζεται ως πίνακες, διανύσματα, λίστες ή ακολουθίες

Στο JSON, ένα αντικείμενο περιλαμβάνει ένα μη ταξινομημένο σύνολο ζευγών ονόματος/τιμής, ενσωματωμένα σε άγκιστρα ‘{}’, με κάθε ζεύγος να οριοθετείται με άνω-κάτω τελεία ‘:’ και να διαχωρίζεται με κόμματα ‘,’. Σε ένα HTTP αίτημα, όταν χρησιμοποιείται JSON ως text format πρέπει να ενημερωθεί κατάλληλα το header ως **Content-Type: application/json**. Ακολουθεί παράδειγμα αναπαράστασης δεδομένων κειμένου με JSON :

```
{
    "users": [
      {
        "id": 1,
        "name": "Manos",
        "age": 23,
        "hobbies": [
          "coding",
          "movies"
        ]
      }
    ]
}
```

Εικόνα 7 - JSON example

3.5.2 XML

Η XML (eXtensible Markup Language) εξυπηρετεί ως μια ευέλικτη και επεκτάσιμη μορφή σήμανσης για ανταλλαγή δεδομένων. Γνωστή για την ευελιξία και την αναγνωσιμότητα από τον άνθρωπο, η XML διευκολύνει τόσο τη χειροκίνητη επεξεργασία όσο και την αυτοματοποιημένη επεξεργασία. Σε αντίθεση με το JSON, η XML δε συνδέεται με κάποια συγκεκριμένη γλώσσα προγραμματισμού και παρέχει έναν γενικό τρόπο δομής και οργάνωσης δεδομένων. Έχει ξεχωρίσει, καταφέρνοντας να κερδίσει εξέχουσα θέση ως πρότυπο του W3C και χρησιμοποιείται ευρέως σε διάφορους τομείς, συμπεριλαμβανομένων των υπηρεσιών δικτύου, των αρχείων διαμόρφωσης και αναπαράστασης δεδομένων. Η XML δομεί δεδομένα χρησιμοποιώντας ετικέτες που περικλείονται σε αγκύλες '<>' σχηματίζοντας στοιχεία. Αυτά τα στοιχεία μπορούν να έχουν χαρακτηριστικά και να περιέχουν ένθετα στοιχεία, επιτρέποντας την ιεραρχική αναπαράσταση τους δημιουργώντας σχέσεις γονέα – παιδιού. Ενώ το JSON εστιάζει στην απλότητα και την ευκολία χρήσης, η δύναμη της XML έγκειται στην ευελιξία και την ικανότητά της να αναπαριστά περίπλοκες ιεραρχικές δομές με τυποποιημένο και δομημένο τρόπο. Είτε χρησιμοποιείται για αρχεία διαμόρφωσης είτε για ανταλλαγή δεδομένων μεταξύ ετερογενών συστημάτων, η XML παραμένει μια ισχυρή και προσαρμόσιμη επιλογή στον τομέα των γλωσσών σήμανσης. Ακολουθεί παράδειγμα αναπαράστασης δεδομένων κειμένου με XML:

```
<users>
        <id>1</id>
        <name>Manos</name>
        <age>23</age>
        <hobbies>coding</hobbies>
        <hobbies>movies</hobbies>
</users>
```

Εικόνα 8 - XML example

3.5.1 Διαφορές ανάμεσα σε JSON & XML

Στον ακόλουθο πίνακα φαίνονται οι βασικές διαφορές ανάμεσα στο JSON και XML όπως αναγράφονται στο άρθρο που είναι διαθέσιμο στη σελίδα του aws με τίτλο **What's the Difference Between JSON and XML?**

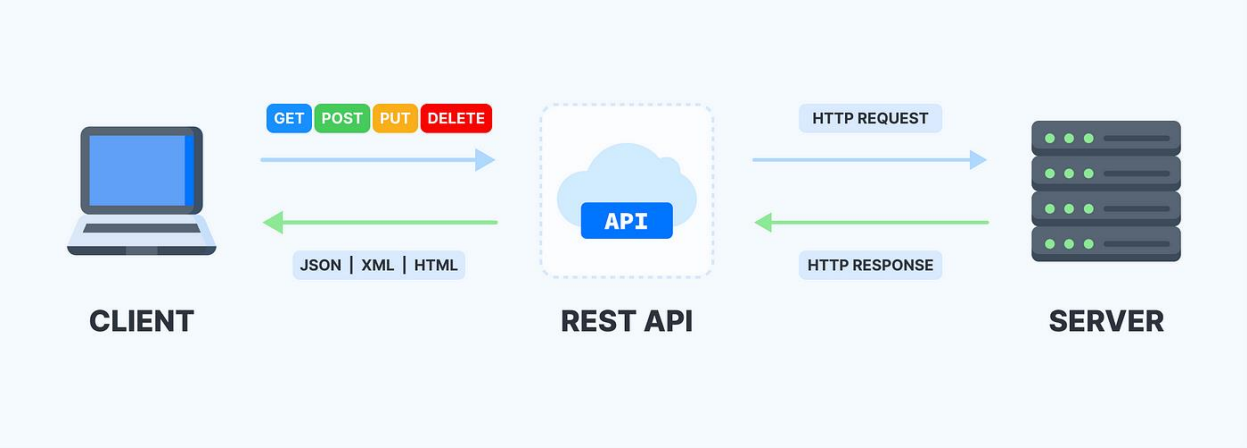
	JSON	XML
Format	Το JSON χρησιμοποιεί ζεύγη κλειδιών-τιμών	Η XML αποθηκεύει δεδομένα σε μια δεντρική δομή με namespaces για διαφορετικές κατηγορίες δεδομένων
Syntax	Η σύνταξη του JSON είναι πιο συμπαγής και ευκολότερη στην ανάγνωση και εγγραφή	Η XML είναι πιο περιεκτική και αντικαθιστά ορισμένους χαρακτήρες για αναφορές οντοτήτων. Για παράδειγμα, αντί για τον χαρακτήρα <, η XML χρησιμοποιεί την αναφορά οντότητας ⁢. Η XML χρησιμοποιεί επίσης ετικέτες τέλους, κάτι που το κάνει μεγαλύτερο από το JSON
Parsing	Μπορεί να προσπελαστεί και να αναλυθεί εύκολα χρησιμοποιώντας μια απλή συνάρτηση γραμμένη σε JavaScript	Απαιτείται ειδικός XML parser για να αναλυθεί
Schema documentation	Είναι απλό και ευέλικτο	Η XML είναι πολύπλοκη και λιγότερο ευέλικτη
Data types	Υποστηρίζει αριθμούς, αντικείμενα, συμβολοσειρές και πίνακες Boolean.	ποστηρίζει όλους τους τύπους δεδομένων JSON και πρόσθετους τύπους όπως Boolean, ημερομηνίες, εικόνες και namespaces
Ease of use	Λόγω του μικρότερου μεγέθους του, έχει ταχύτερο χρόνο μετάδοσης σε σχέση με το XML.	Ως γλώσσα σήμανσης, η XML είναι πιο περίπλοκη και απαιτεί

	Δεν χρησιμοποιεί ετικέτες, γεγονός που το καθιστά πιο συμπαγές και πιο ευανάγνωστο για τον άνθρωπο	μια δομή ετικέτας
Security	Η ανάλυση JSON είναι ασφαλέστερη από την XML	Η δομή της XML είναι ευάλωτη σε μη εξουσιοδοτημένες τροποποιήσεις, γεγονός που δημιουργεί έναν κίνδυνο ασφάλειας που είναι γνωστός ως έγχυση εξωτερικής οντότητας XML (XXE). Είναι επίσης ευάλωτο σε μη δομημένη δήλωση τύπου εξωτερικού εγγράφου (DTD). Και τα δύο ζητήματα μπορούν να αποτραπούν επιτυχώς απενεργοποιώντας τη δυνατότητα DTD κατά τη μετάδοση.

Πίνακας 5 - Διαφορές ανάμεσα σε JSON & XML

4 REST API

Ο όρος REST (Representational State Transfer) και οι αρχές που σχετίζονται με αυτόν εισήχθησαν από τον Roy Fielding στη διδακτορική του διατριβή με τίτλο "Architectural Styles and the Design of Network-based Software Architectures“, που παρουσιάστηκε το 2000. Υπήρξε ανάμεσα στους συγγραφείς της προδιαγραφής HTTP/1.1 και έπαιξε καταλυτικό ρόλο στη διαμόρφωση του Παγκόσμιου Ιστού όπως είναι σήμερα. Το REST είναι ένα διαδεδομένο αρχιτεκτονικό στυλ σχεδιασμού διαδικτυακών εφαρμογών. Είναι χτισμένο πάνω στο πρωτόκολλο HTTP, εφαρμόζοντας τα headers που έχουν οριστεί. Χρησιμοποιεί τέσσερα βασικά functions **C**reate, **R**ead, **U**pdate, **D**elete (CRUD), τα οποία αντιστοιχούν στις HTTP μεθόδους και ταυτίζονται με ενέργειες που μπορούν να πραγματοποιηθούν σε κάποιο σύστημα βάσης δεδομένων.



Εικόνα 9 - REST API

4.1.1 REST API Design Practices

Ορισμένες βέλτιστες πρακτικές οφείλονται να ακολουθούνται κατά το σχεδιασμό ενός REST API, μερικές εκ των οποίων αναγράφονται παρακάτω, οι οποίες αντλήθηκαν από ένα άρθρο στο Stack Overflow με τίτλο **Best practices for REST API design:**

- Αποδοχή αιτήματος και απόκριση σε αυτό χρησιμοποιώντας JSON: Το JSON format έχει καθιερωθεί ως πρότυπο για τη μεταφορά δεδομένων και σχεδόν κάθε τεχνολογία στο διαδίκτυο μπορεί να το αναγνωρίσει και να το επεξεργαστεί. Στη μεριά του πελάτη, η JavaScript περιλαμβάνει ενσωματωμένες μεθόδους για κωδικοποίηση και αποκωδικοποίηση JSON όπως το fetch API. Αντίστοιχα οι τεχνολογίες από την πλευρά του διακομιστή διαθέτουν βιβλιοθήκες που μπορούν να αποκωδικοποιήσουν το JSON χωρίς να κάνουν πολλή δουλειά. Υπάρχουν διάφοροι τρόποι για τη μεταφορά δεδομένων, καθένας με τα δικά του πλεονεκτήματα. Η XML για παράδειγμα παρότι μπορεί να αντικαταστήσει το JSON, δε λαμβάνει ευρεία υποστήριξη από frameworks χωρίς την ανάγκη μετατροπής των δεδομένων σε μορφή που μπορεί να χρησιμοποιηθεί ευκολότερα, η οποία συνήθως καταλήγει να είναι JSON.

Αντίθετα, το JSON έχει γίνει στάνταρ επιλογή, καθώς προσφέρει ευελιξία και ευαναγνωστία. Η δυσκολία στο χειρισμό των XML δεδομένων από την πλευρά του πελάτη, ιδίως στα προγράμματα περιήγησης, καθιστά την κανονική μεταφορά δεδομένων μια πρόκληση. Επιπλέον το μειωμένο μέγεθος που έχει το JSON σε σύγκριση με την XML επιφέρει ταχύτερους ρυθμούς μετάδοσης δεδομένων βελτιώνοντας τη γενική απόδοση. Τέλος προκειμένου να εξασφαλιστεί ότι η απάντηση της εφαρμογής REST API ερμηνεύεται ως JSON από τους πελάτες, απαιτείται ο σωστός καθορισμός του Content-Type στην κεφαλίδα απόκρισης. Μετά την υποβολή του αιτήματος, θα πρέπει να οριστεί το Content-Type σε "application/json". Αυτή η παράμετρος δείχνει στους πελάτες ότι η απόκριση περιλαμβάνει δεδομένα σε μορφή JSON, επιτρέποντάς τους να ερμηνεύσουν και να διαχειριστούν σωστά τις πληροφορίες.

- Χρήση HTTP μεθόδων για λειτουργίες CRUD: Είναι πολύ σημαντικό να χρησιμοποιούνται οι γνωστές μέθοδοι του πρωτοκόλλου HTTP, δηλαδή οι GET, POST, PUT, DELETE και όχι οι λειτουργίες που αντιστοιχούν στο CRUD (Create, Read, Update, Delete).
- Εφαρμογή εμφωλευμένων endpoints: Για να επιτευχθεί καλύτερη οργάνωση, προτείνεται η ομαδοποίηση των αιτημάτων που περιέχουν σχετιζόμενες πληροφορίες. Δηλαδή στην περίπτωση όπου ένα αντικείμενο μπορεί να περιλαμβάνει με τη σειρά του ένα επιπλέον αντικείμενο, πρέπει να σχεδιαστεί ένα αντίστοιχο endpoint για αυτό το σενάριο.
- Ορθή διαχείριση Σφαλμάτων: Το REST API θα πρέπει να είναι σχεδιασμένο για να προλαμβάνει και να διαχειρίζεται με σωστό τρόπο σφάλματα. Στην περίπτωση εντοπισμού σφάλματος πρέπει να επιστρέφεται κατάλληλο μήνυμα, φέροντας το αντίστοιχο status code ακολουθώντας τους κανόνες που έχουν οριστεί στο πρωτόκολλο HTTP (400, 404 ,500 κ.α)
- Έμφαση στις πρακτικές ασφάλειας: Η πληροφορία που ανταλλάσσεται ανάμεσα σε πελάτη και διακομιστή οφείλει να είναι εμπιστευτική καθώς συνήθως στέλνονται και λαμβάνονται ιδιωτικά δεδομένα. Έτσι κρίνεται αναγκαία η χρήση SSL/TLS. Επιπλέον, πρέπει να λαμβάνονται μέτρα ώστε ο κάθε χρήστης να λαμβάνει όση πληροφορία χρειάζεται. Αυτό επιτυγχάνεται θεσπίζοντας ρόλους ανάμεσα στους χρήστες ή προσθέτοντας authentication στα αιτήματα.
- Προσωρινή αποθήκευση (cache) δεδομένων για βελτίωση της απόδοσης: Υπάρχει η δυνατότητα ανάκτησης δεδομένων από την τοπική προσωρινή μνήμη αντί να πραγματοποιούνται συνεχώς αιτήματα στη βάση δεδομένων κάθε φορά που πρέπει να ληφθούν δεδομένα για λογαριασμό κάποιου χρήστη. Χρησιμοποιώντας την τοπική μνήμη, οι χρήστες λαμβάνουν τα δεδομένα τους πιο γρήγορα αλλά απαιτείται προσοχή καθώς τα δεδομένα μπορεί να μην είναι ενημερωμένα σωστά (outdated).
- Συμμόρφωση με τις βέλτιστες πρακτικές ονοματοδοσίας: Σύμφωνα με το άρθρο από το forum restfulapi.net με τίτλο **REST API URI Naming Conventions and Best Practices**, πρέπει να ακολουθούνται ορισμένοι άτυποι κανόνες κατά την ονοματοδοσία των Rest APIs, μερικοί εκ των οποίων αναγράφονται παρακάτω:
    - Αποφυγή χρήσης κεφαλαίων γραμμάτων στα URIs
    - Χρήση forward-slash (/) στο μονοπάτι του URI ώστε να υποδείξει μια ιεραρχική σχέση ανάμεσα στους πόρους, για παράδειγμα εάν θέλουμε να εκθέσουμε τις δραστηριότητες ενδιαφέροντος που μπορεί να έχει ένας συγκεκριμένος χρήστης, δύναται να χρησιμοποιηθεί ένα URI της μορφής: /users/{id}/hobbies

- Αποφυγή χρήσης forward-slash στο τέλος του URI, προκειμένου να μην μπερδευτεί ο χρήστης:

```
/users/{id}/hobbies/          //bad practice
/users/{id}/hobbies           //good practice
```

- Προτίμηση χρήσης απλής παύλας (-) αντί για κάτω παύλα (_) όταν πρέπει να ενωθούν λέξεις στο URI. Αυτή η τακτική βελτιώνει την αναγνωσιμότητα των URIs:

```
application_users/{id}/favorite_colors      //bad practice
application-users/{id}/favorite-colors      //good practice
```

- Αποφυγή χρήσης επεκτάσεων αρχείων, καθώς επηρεάζουν την εικόνα των αιτημάτων, αυξάνουν το μήκος του URI και γενικότερα δεν προσφέρουν κάποιο πλεονέκτημα:

```
application-users/{id}/favorite-colors/list.json      //bad practice
application-users/{id}/favorite-colors/list           //good practice
```

- Αποφυγή χρήσης λειτουργιών CRUD στα ονόματα των URIs. Σκοπός κάθε URI είναι να αναγνωρίσει τους προσφερόμενους πόρους και όχι να περιγράψει την αναμενόμενη ενέργεια. Αυτόν το σκοπό εξάλλου καλούνται να εξυπηρετήσουν οι μέθοδοι HTTP:

```
HTTP GET data/api/users/{id}              //get a specific user
HTTP PUT data/api/users/{id}              //update a specific user
HTTP DELETE data/api/users/{id}           //delete a specific user
```

- Χρήση query component για φιλτράρισμα – έλεγχο συλλογής. Συχνά απαιτείται μόνο συγκεκριμένη πληροφορία από ένα πόρο. Έτσι θεμιτό είναι να πραγματοποιούνται ενέργειες ταξινόμησης, σελιδοποίησης και φιλτραρίσματος αντί να συντάσσονται νέα αιτήματα:

```
data/api/users                    //get all users
data/api/users?country=GREECE     //get only users from Greece
```

4.2 API Security

Οι πόροι που προσφέρουν οι διακομιστές μέσα από τα APIs μπορεί να είναι διαθέσιμοι στο ευρύ κοινό ή η πρόσβαση να περιορίζεται σε όσους διαθέτουν κατάλληλη εξουσιοδότηση. Για να περιοριστεί κατάλληλα η πρόσβαση, τα APIs υλοποιούν μηχανισμούς διακρίβωσης ταυτότητας (authentication) και εξουσιοδότησης (authorization).

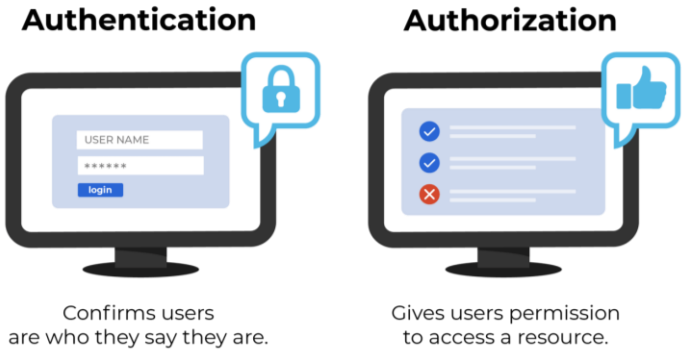
Οι διαφορετικές απειλές ασφαλείας API μπορούν να ταξινομηθούν στις ακόλουθες κατηγορίες:

- Authentication
- Authorization
- Message or content-level attacks
- Man-in-the-middle attack
- DDoS attacks (distributed denial-of-service)

4.2.1 Authentication & Authorization

Οι έννοιες authentication και authorization αν και ταυτίζονται μεταξύ τους, συχνά χρησιμοποιούνται λανθασμένα. Ο όρος **Authentication** προσδιορίζει την ταυτότητα του χρήστη που προσπαθεί να αποκτήσει πρόσβαση σε κάποιον πόρο του διακομιστή. Από την άλλη μεριά ο όρος **Authorization** προσδιορίζει τα δικαιώματα και το επίπεδο πρόσβασης του χρήστη που πραγματοποιεί κάποιο αίτημα. Για να γίνει πιο αντιληπτό, έστω ένα παράδειγμα όπου ένα άτομο θέλει να πάρει μέρος σε μια συναυλία. Εάν δεν είναι πιστοποιημένος, τότε η είσοδος δεν θα του επιτραπεί. Έτσι πρέπει να αποκτήσει ένα εισιτήριο, πηγαίνοντας στο ταμείο και δίνοντας την ταυτότητα του ώστε να εξακριβωθούν τα στοιχεία του καθώς, μπορεί να υπάρχει για παράδειγμα όριο ηλικίας. Αυτή η διαδικασία ονομάζεται authentication, όπου ο χρήστης πρέπει να προσδιορίσει την ταυτότητα του προκειμένου να αποκτήσει πρόσβαση σε κάποιον πόρο (είσοδος στο χώρο της συναυλίας). Ως αποτέλεσμα της επιτυχής ταυτοποίησης του, αποκτάει ένα εισιτήριο το οποίο θα χρησιμοποιήσει ο κάτοχος για να μπορέσει να εισέλθει στη συναυλία. Αυτή η διαδικασία ονομάζεται authorization. Αξίζει να σημειωθεί πως το εισιτήριο δεν αναγράφει πάνω προσωπικά στοιχεία του κατόχου, όπως ονοματεπώνυμο, ηλικία κ.α. Χρησιμοποιείται απλά για να προσδιορίσει πως ο κάτοχος του έχει πλέον πρόσβαση στον πόρο.

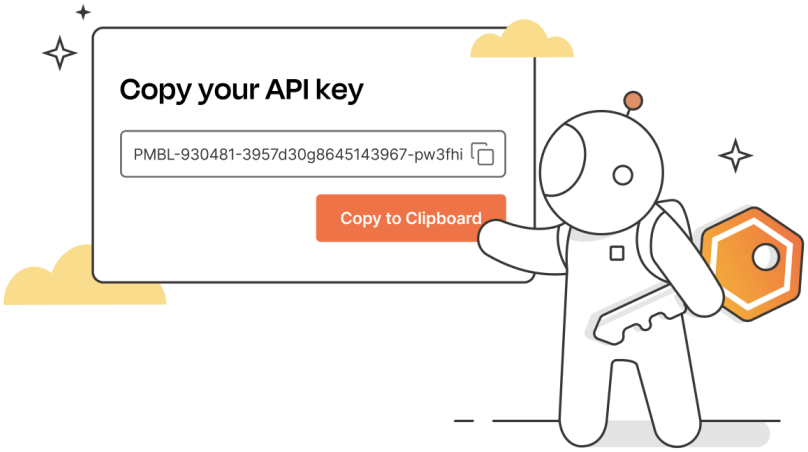
Δύο γνωστές και συχνά χρησιμοποιούμενες κατηγορίες authentication & authorization είναι τα **API Keys** και το **Username and Password**



Εικόνα 10 - Authentication & Authorization

4.2.1.1 API Keys

Ένα κλειδί API χρησιμοποιείται για να αναγνωριστεί η εφαρμογή που χρησιμοποιεί ένα API. Τα API keys παρέχουν έναν απλό μηχανισμό για έλεγχο ταυτότητας των εφαρμογών, επιτρέποντας σε ένα API να προσδιορίζει ποιες εφαρμογές το χρησιμοποιούν. Πρόκειται για μια σειρά ψευδοτυχαίων χαρακτήρων και αριθμών, που προσφέρεται από τον πάροχο του API και είναι μοναδικό για κάθε χρήστη (εφαρμογή). Επιλέγοντας να ενσωματώσει ένα API στην εφαρμογή του, ο προγραμματιστής θα λάβει το κλειδί που δημιουργήθηκε από τον πάροχο και θα το προσαρμόσει στον κώδικα του, περιλαμβάνοντάς το στις επικεφαλίδες (authorization headers) κάθε αιτήματος που στέλνει στην υπηρεσία που το προσφέρει. Με αυτό τον τρόπο ο πάροχος γνωρίζει εάν το αίτημα που έλαβε, για την παροχή πόρων του είναι έγκυρο και ανταποκρίνεται αναλόγως.



Εικόνα 11 - API Key

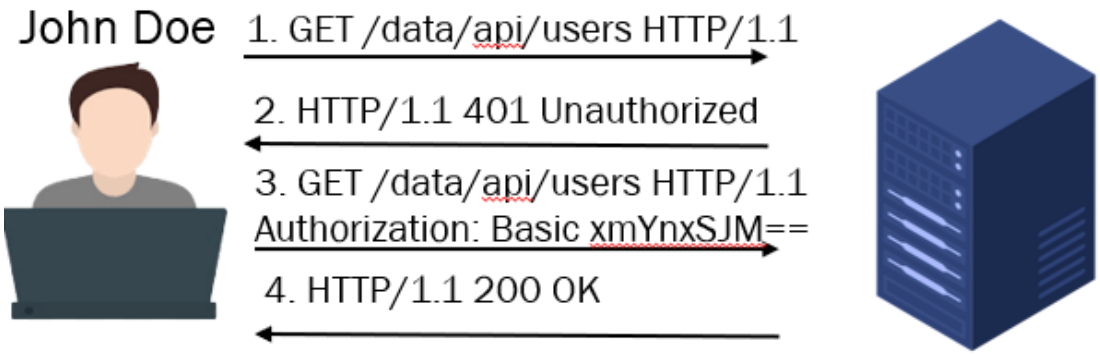
4.2.1.2 Username and Password

Το όνομα χρήστη και ο κωδικός πρόσβασης είναι η πιο κοινή μορφή ελέγχου ταυτότητας. Σε αυτή τη μορφή ελέγχου ταυτότητας, ο πελάτης παρουσιάζει στον διακομιστή ένα μοναδικό όνομα και έναν μυστικό κωδικό. Ο διακομιστής συγκρίνει τα στοιχεία που έλαβε, με αυτά που είναι αποθηκευμένα στη βάση δεδομένων του και προσφέρει πρόσβαση στους πόρους μόνο μετά από επιτυχή ταυτοποίηση και αντίστοιχο έλεγχο ύπαρξης δικαιωμάτων.

Για ένα αίτημα σε REST API, ο πελάτης μπορεί να στέλνει αυτά τα στοιχεία μέσα από τα headers του αιτήματος χρησιμοποιώντας το **Basic Authentication Scheme:**

- Το όνομα χρήστη και ο κωδικός πρόσβασης συνδυάζονται με μία άνω τελεία ανάμεσα τους: username:password
- Η σύνθετη συμβολοσειρά κωδικοποιείται με τη μέθοδο Base64

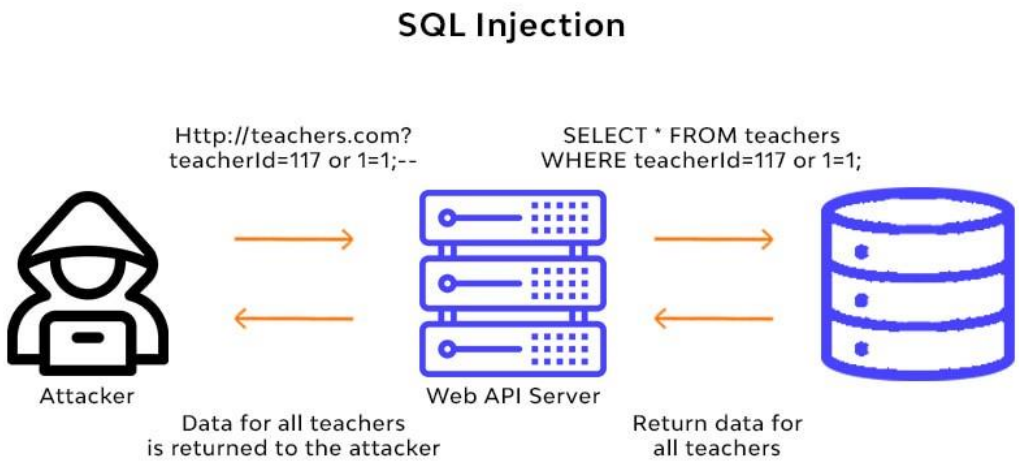
- Το authorization method και ένα κενό (Basic) συμπληρώνονται στο header «Authorization» πριν το encoded string



Εικόνα 12 - Basic Authentication Scheme

4.2.2 Message or content-level attacks

Ένας επιτιθέμενος δύναται να τοποθετήσει κακόβουλο περιεχόμενο, όπως λογισμικό, σε αιτήματα API ώστε να επιτεθεί στο σύστημα. Μπορούν επίσης να εισάγουν scripts στα αιτήματά τους, τα οποία θα εκτελεσθούν στο back end. **Code injections** αποτελούν παράδειγμα τέτοιων επιθέσεων όπου ένας κακόβουλος χρήστης με προγραμματιστικές ικανότητες μπορεί να προσθέσει κώδικα μέσα από τα πεδία εισαγωγής κειμένου, ο οποίος θα εκτελεστεί στο διακομιστή, προκαλώντας σημαντικά προβλήματα. **XML and JSON Bombs** είναι ακόμα μια απειλή, όπου σκοπός είναι η υπερφόρτωση των parsers που χρησιμοποιούνται για να επεξεργαστούν το εισερχόμενο αίτημα. Ως αποτέλεσμα οι πόροι του διακομιστή (RAM, CPU) δεν επαρκούν για να ανταποκριθούν στο αίτημα, καθιστώντας το διακομιστή ανίκανο να ανταποκριθεί σε άλλα αιτήματα. Αυτή η συγκεκριμένη επίθεση παραπέμπει σε DDoS attack που θα αναλυθούν στη συνέχεια. Ένα ακόμα χαρακτηριστικό παράδειγμα επιθέσεων που στοχεύουν στην τροποποίηση του κειμένου που στέλνεται κατά το αίτημα είναι τα **SQL Injection Attacks**. Σε μια τέτοια επίθεση, ένας εισβολέας τροποποιεί το αίτημα για να χειριστεί εντολές SQL, αλληλοεπιδρώντας έτσι με τη βάση δεδομένων μέσω ακούσιων ερωτημάτων, οδηγώντας σε κλοπή, διαγραφή ή χειραγώγηση αποθηκευμένων δεδομένων.

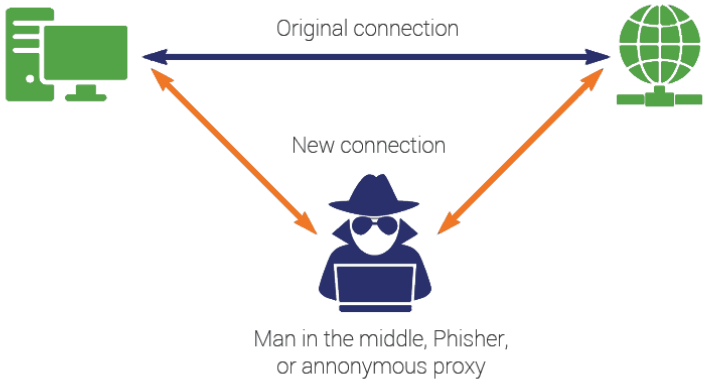


Εικόνα 13 - SQL Injection Attack

4.2.3 Man in the middle attack

Η επίθεση Man In the Middle αναφέρεται σε μια επικίνδυνη μορφή επίθεσης, κατά την οποία ένα ανεπιθύμητος επισκέπτης εισβάλλει σε ένα δίκτυο επικοινωνίας μεταξύ ενός πελάτη και ενός διακομιστή, ενός διακομιστή και άλλου διακομιστή, ή ακόμα και μεταξύ δύο πελατών. Η επίθεση αυτή έχει πολλαπλούς στόχους, συμπεριλαμβανομένης της υποκλοπής ευαίσθητων δεδομένων όπως κωδικοί πρόσβασης, τραπεζικές πληροφορίες, προσωπικά δεδομένα, καθώς και εξαπόλυσης κακόβουλου λογισμικού.

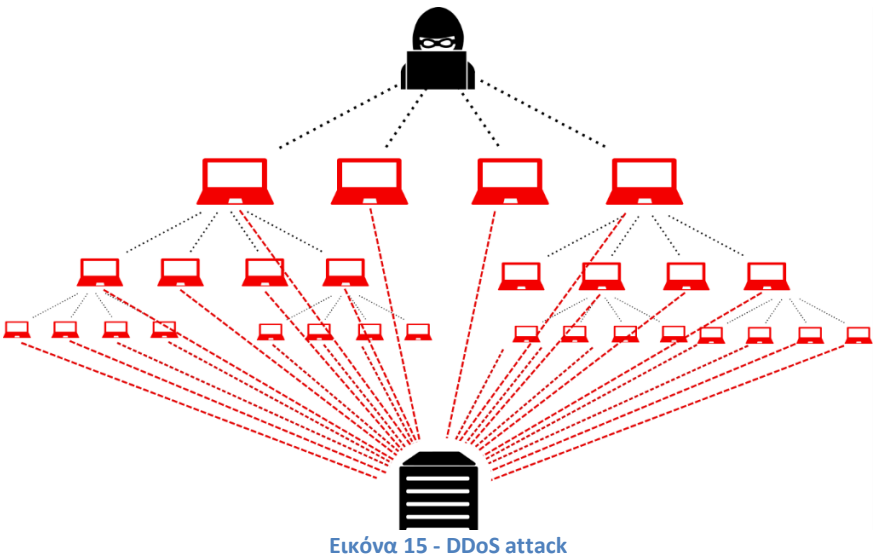
Είναι σημαντικό να σημειωθεί ότι η επιτυχία αυτού του είδους επίθεσης εξαρτάται από το αν οι επικοινωνίες είναι κρυπτογραφημένες. Συνεπώς, είναι ζωτικής σημασίας η διασφάλιση και χρήση κρυπτογραφίας (SSL, TLS) στις επικοινωνίες, καθιστώντας την επίθεση Man In the Middle περισσότερο δύσκολη και αντιμετωπίζοντας αποτελεσματικά αυτόν τον κίνδυνο.



Εικόνα 14 - Man in the middle attack

4.2.4 DDoS attacks

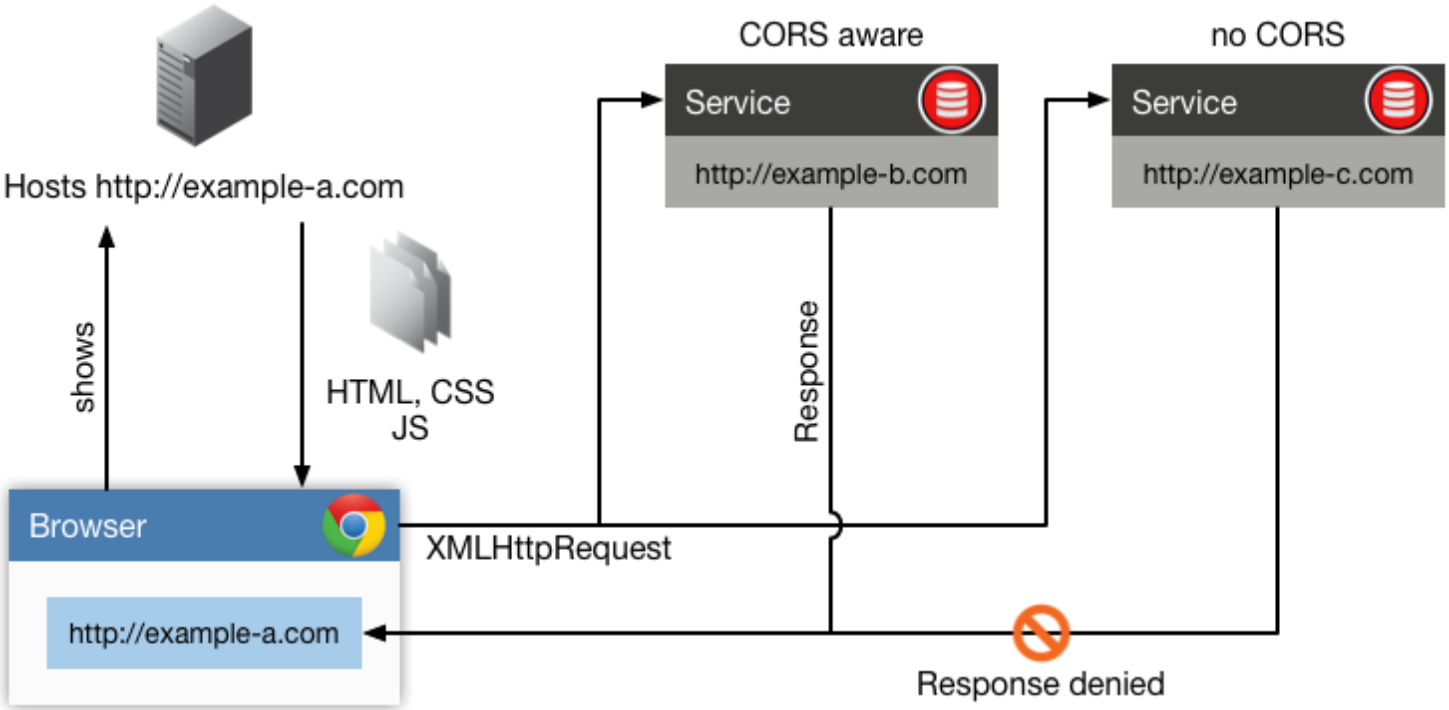
Η άρνηση υπηρεσίας (DoS) είναι μια επίθεση που πραγματοποιείται με σκοπό να καταστήσει τις υπηρεσίες ενός διακομιστή μη διαθέσιμες. Πλημμυρίζοντας ένα API με ψευδή αιτήματα, οι πόροι του αποκλείονται στο να ανταποκρίνονται σε αυτά τα αιτήματα και όχι σε άλλα. Ο στόχος των επιθέσεων DoS δεν είναι η αλλαγή, η διαγραφή ή η κλοπή δεδομένων. Ο στόχος είναι απλώς να βλάψει τη λειτουργία μιας διαδικτυακής υπηρεσίας ή τη φήμη μιας εταιρείας που προσφέρει τέτοιες υπηρεσίες.



Εικόνα 15 - DDoS attack

4.2.5 CORS

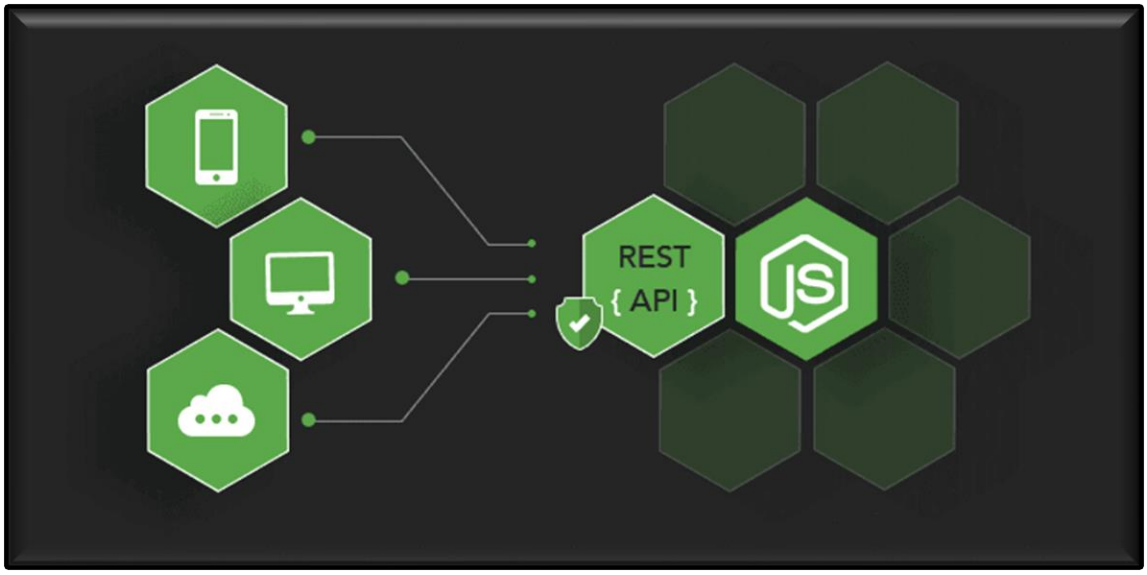
Το Cross-Origin Resource Sharing (CORS) αποτελεί έναν μηχανισμό ασφάλειας που βασίζεται στα HTTP Headers, επιτρέποντας σε έναν διακομιστή να δηλώνει οποιαδήποτε προέλευση, εκτός από την δική του, από την οποία ένα πρόγραμμα περιήγησης θα πρέπει να επιτρέπει τη φόρτωση πόρων. Ένα παράδειγμα αυτού είναι τα **XMLHttpRequest** και το **Fetch API**, τα οποία ακολουθούν την πολιτική ίδιας προέλευσης (same origin). Για λόγους ασφαλείας, τα προγράμματα περιήγησης απαγορεύουν τα cross-origin HTTP αιτήματα που στέλνονται από scripts. Αυτό σημαίνει ότι μια ιστοσελίδα που χρησιμοποιεί αυτά τα API μπορεί να ζητήσει πόρους μόνο από την ίδια προέλευση που φορτώθηκε η ιστοσελίδα, εκτός εάν η απάντηση από άλλες πηγές περιλαμβάνει τις απαραίτητες κεφαλίδες CORS.



Εικόνα 16 - CORS

5 Node.js

Το Node.js είναι ένα ανοικτού κώδικα, περιβάλλον εκτέλεσης πλευράς διακομιστή που επιτρέπει στους προγραμματιστές να εκτελούν κώδικα JavaScript έξω από τον περιηγητή ιστού. Είναι χτισμένο πάνω στη μηχανή JavaScript V8 που αναπτύχθηκε από τη Google και παρέχει μια αρχιτεκτονική μη ανασταλτικού (non-blocking) χειρισμού συμβάντων (events), γεγονός που το καθιστά ιδιαίτερα αποδοτικό για το χειρισμό λειτουργιών εισόδου/εξόδου. Το Node.js είναι ιδιαίτερα χρήσιμο για την ανάπτυξη εφαρμογών ιστού και RESTful APIs.



Εικόνα 17 - Node.js

5.1.1 Express

Το Express αποτελεί ένα ευέλικτο framework για το Node.js, το οποίο κυκλοφορεί ως ελεύθερο λογισμικό ανοιχτού κώδικα υπό την άδεια MIT. Έχει σχεδιαστεί για τη δημιουργία διαδικτυακών εφαρμογών και API. Το Express έχει γίνει μια από τις πλέον δημοφιλέστερες επιλογές για τη δημιουργία διακομιστών ιστού και API στο οικοσύστημα Node.js. Η ελαφριά φύση του επιτρέπει στους προγραμματιστές να δημιουργούν γρήγορα RESTful API ή διαδικτυακές εφαρμογές με ευκολία, καθιστώντας το ένα εξαιρετικό εργαλείο τόσο για αρχάριους όσο και για έμπειρους προγραμματιστές. Ένας χρήστης μπορεί να κατεβάσει τη βιβλιοθήκη δίνοντας την ακόλουθη εντολή σε ένα τερματικό.

```
$ npm install express
```

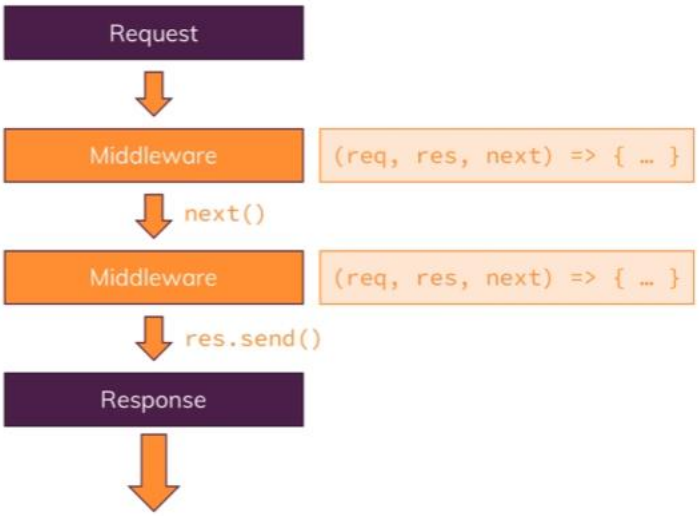
5.1.2 Middleware

Middlewares (ενδιάμεσο λογισμικό) είναι συνιστώσες λογισμικού που παρεμβάλλονται μεταξύ των πελατών (clients) και των δεδομένων, υλοποιώντας κανόνες της επιχειρηματικής λογικής. Σε επίπεδο web services, τα middleware υλοποιούν και δημοσιοποιούν μεθόδους API οι οποίες χρησιμοποιούνται

στα αιτήματα http, προσφέροντας πρόσβαση στο ίδιο το request (req) και στο response (res) του διακομιστή. Τα middlewares μπορούν να εκτελέσουν τις ακόλουθες ενέργειες:

- Εκτέλεση κώδικα
- Αλλαγές στα αιτήματα – αποκρίσεις
- Τερματισμός του κύκλου αιτήματος
- Κλήση του επόμενου middleware

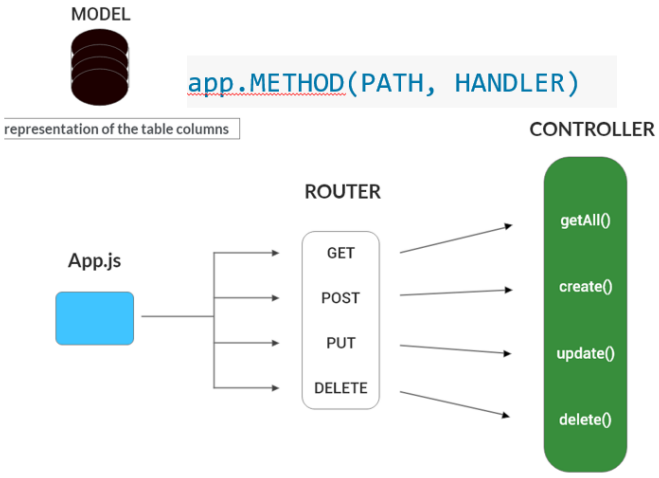
Για να κληθεί το επόμενο middleware, απαιτείται να δοθεί η εντολή next()



Εικόνα 18 - Middleware

5.1.3 Routing

Ο όρος routing αναφέρεται στον τρόπο με τον οποίο αποφασίζει μια εφαρμογή πως να ανταποκριθεί σε κάποιο εισερχόμενο αίτημα, ορίζοντας ένα endpoint (PATH) και μια HTTP μέθοδο. Κάθε Route περιλαμβάνει επίσης ένα handler, ο οποίος διαχειρίζεται το εισερχόμενα αίτημα:



Εικόνα 19 - Routing

5.1.4 Express Example

Η διαδικασία ξεκινά με την εισαγωγή της απαραίτητης βιβλιοθήκης στην εφαρμογή υπό την προϋπόθεση ότι έχει εγκατασταθεί στο τοπικό μηχάνημα. Στη συνέχεια, δημιουργείται ένα instance του Express, το οποίο θα λειτουργήσει ως ο διακομιστής. Ένα route ορίζεται στο μονοπάτι '/' χρησιμοποιώντας τη μέθοδο GET. Όταν ένα αίτημα φτάσει σε αυτό το μονοπάτι, ένα middleware αναλαμβάνει τη διαχείριση του αιτήματος. Το middleware λαμβάνει πληροφορίες για το εισερχόμενο αίτημα όπως headers και body μέσω του αντικειμένου req, ενώ η απόκριση του διακομιστή αντιστοιχεί στο αντικείμενο res. Με τη χρήση της μεθόδου res.send(), η απόκριση του διακομιστή στέλνεται πίσω στον πελάτη. Τέλος, η εφαρμογή ακούει στη θύρα 3000, ώστε να αποδεχτεί εισερχόμενα αιτήματα.

```
const express = require('express')
const app = express()
app.get('/', (req, res) => {
    res.send('hello world')
})
app.listen(3000)
```

5.1.5 Advanced Routing

Για καλύτερη διαχείριση και οργάνωση του κώδικα, τα routes δύναται να χωριστούν σε δύο κατηγορίες. Αρχικά, τα routes με το πρόθεμα "dr" (data-read) αναλαμβάνουν την επιστροφή πληροφορίας, ενώ τα routes με το πρόθεμα "dw" (data-write) χειρίζονται την εισαγωγή πληροφορίας στη βάση δεδομένων ή την εκτέλεση λογικής. Η δυνατότητα να δοθεί ένα μονοπάτι καθολικά για όλα τα αιτήματα προσφέρει επιπλέον ευελιξία στο σύστημα (/data/api).

Σε κάθε route, ένας controller αναλαμβάνει τον ρόλο της διαχείρισης του αιτήματος και της επιστροφής της απόκρισης. Πρόκειται για ένα middleware που έχει οριστεί σε διαφορετικό αρχείο, προσφέροντας έτσι οργανωτική δομή και διαχείριση κώδικα. Επιπλέον, μπορούν να εγκατασταθούν και να χρησιμοποιηθούν συμπληρωματικές βιβλιοθήκες, όπως CORS, προσφέροντας επιπλέον λειτουργικότητα και ασφάλεια στην εφαρμογή.

App.js

```
const express = require('express')
const drRouter = require('./routes/data-read')
const dwRouter = require('./routes/data-write')
const cors = require('cors')
const app = express()
app.use("/data/api", [drRouter, dwRouter])
app.use(cors())
app.listen(3000)
```

routes/data-read.js

```
const express = require('express')
const userController = require('../controllers/user')
const router = express.Router()
router.get("/users",userController.getUsers)
```

routes/data-write.js

```
const express = require('express')
const userController = require('../controllers/user')
const router = express.Router()
router.post("/users",userController.createUser)
```

5.2 Controller

Ο Controller είναι ένα middleware, το οποίο εκτελεί την απαραίτητη λογική κάνοντας χρήση του req όπου απαιτείται και αξιοποιώντας το res επιστρέφει το αποτέλεσμα.

routes/data-read.js

```
const express = require('express')
const userController = require('../controllers/user')
const router = express.Router()
router.get("/users",userController.getUsers)
```

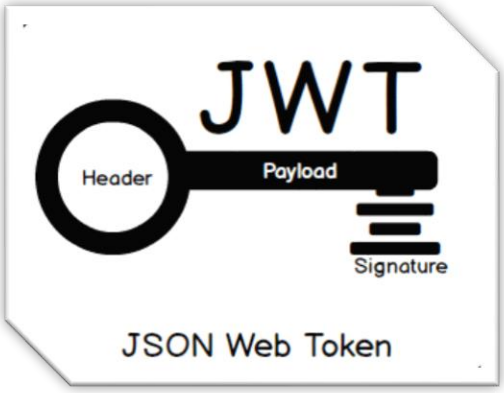
controllers/user.js

```
exports.getUsers = (req, res) => {
  client.query(`SELECT * FROM app.users`, (err, queryResult) => {
    if (!err) {
      res.status(200).json({
        users: queryResult.rows,
      });
    } else {
      res.status(500).json({
        error: "An error occurred while fetching users.",
      });
    }
  });
};
```

5.3 JWT

Το JWT ή JSON Web Token, είναι μια αυτοτελής μέθοδος για την ασφαλή μεταφορά πληροφοριών μεταξύ διαφόρων πλευρών σε μορφή JSON αντικειμένου. Χρησιμοποιείται συχνά για τον έλεγχο ταυτότητας και την εξουσιοδότηση σε ιστοσελίδες και υπηρεσίες API. Τα JWT αποτελούνται από τρία μέρη:

- Header: καθορίζει τον αλγόριθμο που χρησιμοποιείται για την υπογραφή (συνήθως χρησιμοποιείται HS256 ή RS256)
- Payload: Περιέχει δεδομένα
- Digital Signature: Ψηφιακή υπογραφή, εξασφαλίζει την ακεραιότητα και την αυθεντικότητα του token



Εικόνα 20 - JWT

Τα JSON Web Tokens αποτελούν μία ευρέως χρησιμοποιούμενη μέθοδος αυθεντικοποίησης. Κατά τη σύνδεση του χρήστη, εάν τα παρεχόμενα στοιχεία είναι έγκυρα, δημιουργείται ένα JWT στον διακομιστή και επιστρέφεται στον πελάτη. Στη συνέχεια, ο πελάτης παρουσιάζει το JWT σε αιτήσεις προς τον διακομιστή προκειμένου να αποδείξει την προηγούμενη αυθεντικοποίησή του. Η παρουσίαση συνήθως πραγματοποιείται μέσω μιας επικεφαλίδας με τη μορφή **Authorization: 'Bearer <JWT-token>'**. Όπως φαίνεται στο ακόλουθο απόσπασμα κώδικα, τα δεδομένα περιλαμβάνουν το όνομα χρήστη και το αναγνωριστικό, ενώ η ψηφιακή υπογραφή είναι μια κωδικοποιημένη σε base64 τιμή. Η διάρκεια του token είναι μία ώρα, και ο προεπιλεγμένος αλγόριθμος είναι HS256 εφόσον δεν έχει οριστεί.

```
const token = jwt.sign(
      {
            username: username,
            userId: queryResult.rows[0].id
      },
      process.env.JWT_PRIVATE_KEY,
      { expiresIn: "1h" }
);
```

Ως αποτέλεσμα του παραπάνω κώδικα, δημιουργείται ένα token της μορφής:

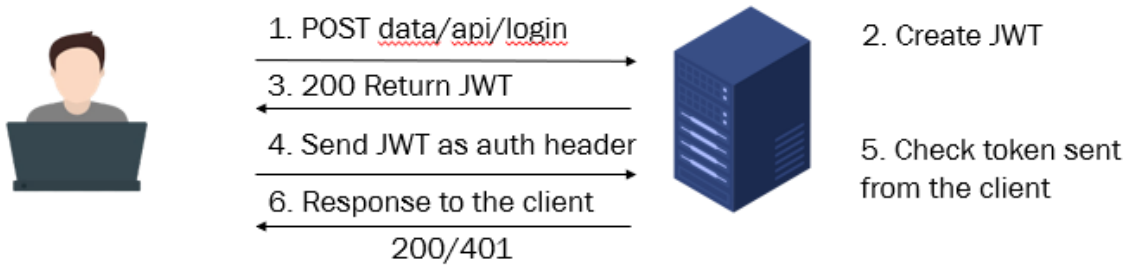
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImFkbWluIiwidXNlcklkIjoyNTksImlhdCI6MTY5NzQ1MzMwMSwiZXhwIjoxNjk3NDU2OTAxfQ.INFr4GDoa5BpzDTFE8KBlfHp8MB1kbGSiL2wkd21E3o

Σε περίπτωση επιτυχημένης αποκωδικοποίησης του token, λαμβάνονται οι ακόλουθες πληροφορίες:

- username: το όνομα χρήστη που χρησιμοποιήθηκε κατά την υπογραφή του token
- userID: ID που χρησιμοποιήθηκε κατά την υπογραφή του token
- iat (issued at): η χρονική στιγμή που υπογράφηκε το token, είναι σε μορφή Epoch Unix Timestamp, δηλαδή ένας αριθμός που εκφράζει τα δευτερόλεπτα που έχουν περάσει από την ημερομηνία 1/1/1970 μέχρι τη στιγμή που υπογράφηκε το token
- exp (expires): η χρονική στιγμή που θα λήξει το token, εφαρμόζοντας την ίδια λογική με το iat

Εικόνα 21 - Decoded JWT

Αφού συνδεθεί ο χρήστης, θα λάβει ένα token, το οποίο θα χρησιμοποιηθεί σε επόμενα αιτήματα που απαιτούν εξουσιοδότηση, προσθέτοντάς το στα headers του αιτήματος.



Εικόνα 22 - JWT Use

5.4 Authentication Middleware

Προκειμένου να υλοποιηθεί αυτός ο έλεγχος προγραμματιστικά με τη βοήθεια της βιβλιοθήκης Express, απαιτείται ένα επιπλέον middleware, το οποίο καλείται πριν τον τελικό controller, στο οποίο

πραγματοποιείται έλεγχος εάν το JWT που έλαβε ο διακομιστής στο αίτημα, αντιστοιχεί σε αυτό που δημιουργήθηκε κατά τη σύνδεση του χρήστη.

```
const jwt = require("jsonwebtoken");

module.exports = (req, res, next) => {
  let token = "";
  if (!!req.get("Authorization")) {
    token = req.get("Authorization").split(" ")[1];
  }

  let decodedToken;
  try {
    decodedToken = jwt.verify(token, process.env.JWT_PRIVATE_KEY);
  } catch (err) {
    err.statusCode = 500;
    res.status(500).json({
      error: "Not authenticated.",
    });
  }

  if (!decodedToken) {
    const error = new Error("Not authenticated");
    error.statusCode = 401;
    throw error;
  }
  next();
};
```

5.5 API Documentation

Ως API Documentation ορίζεται μια συλλογή, στην οποία περιλαμβάνονται όλες οι απαραίτητες πληροφορίες για τα APIs που είναι διαθέσιμα στο διακομιστή, προσφέροντας δυνατότητες όπως:

- Λίστα με διαθέσιμα αιτήματα
- Περιγραφή αιτημάτων
- Παραδείγματα εκτέλεσης (example body, example responses)
- Δυνατότητα εκτέλεσης αιτήματος
- Υποστήριξη πολλαπλών διακομιστών (Development, UAT, Production)
- Προσθήκη authorization όπου χρειάζεται

Παράδειγμα ενός εργαλείου API Documentation είναι το Swagger, μια συλλογή εργαλείων που χρησιμοποιείται για την ανάπτυξη και περιγραφή RESTful APIs.

5.6 YAML

Η YAML δεν αποτελεί γλώσσα προγραμματισμού ή γλώσσα σήμανσης, παρόλο που το ακρωνύμιο προκύπτει από τις λέξεις Yet Another Markup Language. Πρόκειται για έναν απλό τρόπο αναπαράστασης δομημένων δεδομένων που χρησιμοποιείται συχνά σε configuration αρχεία:

- openapi: τρέχουσα έκδοση
- info: Πληροφορίες για το documentation όπως τίτλος και έκδοση
- servers: Οι διαθέσιμοι servers
- tags: Θεματικές ενότητες με σκοπό τη διαχώριση των αιτημάτων για καλύτερη οργάνωση
- paths: Τα διαθέσιμα αιτήματα

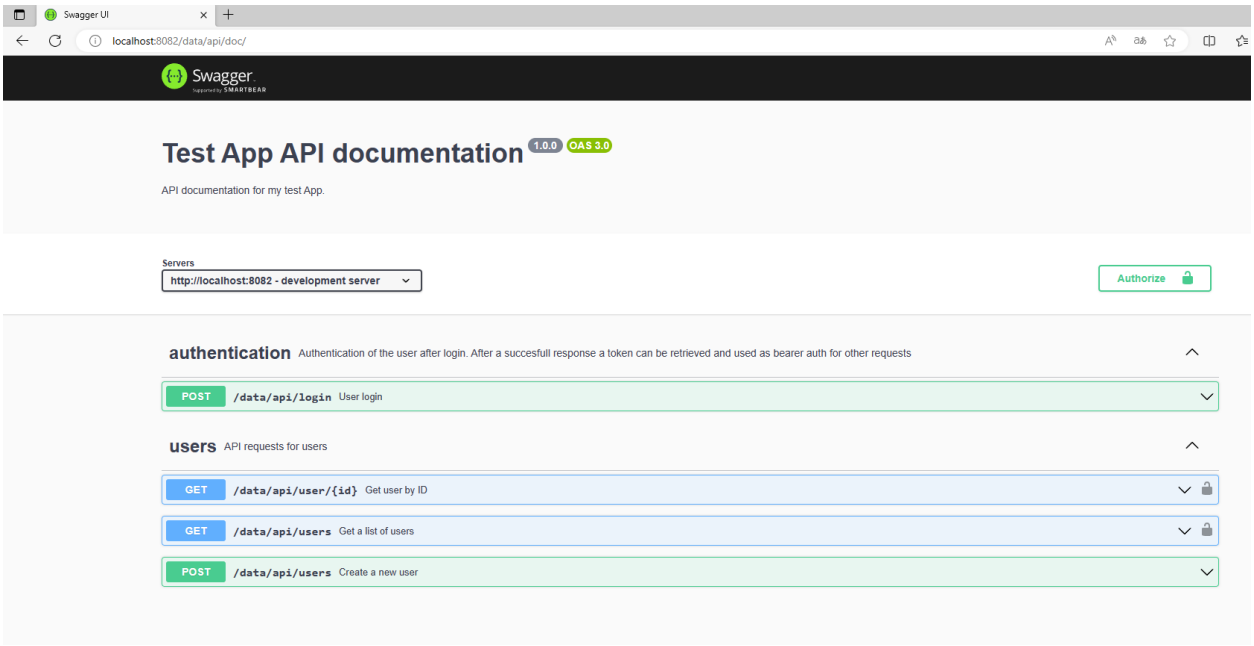
```yaml
openapi: 3.0.1
info:
  title: Test App API documentation
  description: API documentation for my test App.
  version: 1.0.0
servers:
  - url: http://localhost:8082
    description: development server
  - url: https://villa-agapi-344fd44fcd28.herokuapp.com/
    description: production server
tags:
  - name: authentication
    description: Authentication of the user after login.
  - name: users
    description: API requests for user
paths:
  /data/api/user/{id}:
    get:
      tags:
        - users
      summary: Get user by ID
      description: Retrieve user information by their ID.
      parameters:
        - name: id
          in: path
          description: ID of the user to retrieve.
          required: true
          schema:
            type: integer
            example: 1
      responses:
        "200":
          description: Successful response
        "401":
          description: Authorization header required
        "404":
          description: User not found
        "500":
          description: Internal server error
      security:
        - BearerAuth: []
```

5.7 Swagger-ui-express

Το swagger-ui-express επιτρέπει την προβολή εγγράφων API, με βάση ένα αρχείο swagger.json ή swagger.yaml. Το αποτέλεσμα είναι η ζωντανή τεκμηρίωση για τα APIs που φιλοξενούνται από τον διακομιστή μέσω μιας διαδρομής:

```
const swaggerUi = require("swagger-ui-express");
const yaml = require("yamljs");
const swaggerDocument = yaml.load("./swagger.yaml");

const app = express().use("/data/api/doc", swaggerUi.serve,
swaggerUi.setup(swaggerDocument));
```

Εάν ο χρήστης επισκεφτεί το μονοπάτι που έχει οριστεί θα δει το αποτέλεσμα του αρχείου μέσα από ένα γραφικό περιβάλλον:



Εικόνα 23 - Swagger UI

5.8 express-openapi-validator

Το express-openapi-validator αποτελεί ένα middleware για το Express.js, το οποίο βοηθάει στην επικύρωση αιτημάτων και αποκρίσεων API με βάση ορισμένες προδιαγραφές που συνήθως περιγράφονται σε ένα αρχείο διαμόρφωσης (JSON, YAML, XML). Το OpenAPI είναι μια προδιαγραφή για τη δημιουργία API, παρέχοντας έναν τυποποιημένο τρόπο περιγραφής της δομής και της συμπεριφοράς των RESTful API. Το express-openapi-validator δύναται να ενσωματωθεί σε εφαρμογές Express.js, επιτρέποντας στους προγραμματιστές να επιβάλλουν κανόνες επικύρωσης API που ορίζονται σε μια προδιαγραφή OpenAPI (Swagger.YAML). Αυτό διασφαλίζει ότι τα εισερχόμενα αιτήματα συμμορφώνονται με το καθορισμένο σχήμα και ότι οι εξερχόμενες απαντήσεις

συμμορφώνονται με την αναμενόμενη μορφή που έχει οριστεί. Επιπλέον δύναται να ενσωματωθούν χειριστές αυθεντικοποίησης (Authentication handlers) προσφέροντας authentication σε όσα αιτήματα ο χρήστης επιθυμεί μέσα στο αρχείο swagger.YAML. Ως αποτέλεσμα ένα αίτημα ή μια απόκριση που δεν πληροί τις προϋποθέσεις που έχουν οριστεί στο αρχείο τεκμηρίωσης δεν μπορεί να ολοκληρωθεί επιτυχώς, εμφανίζοντας μήνυμα σφάλματος. Με αυτό τον τρόπο, ο διακομιστής αποκτάει ένα εξωτερικό στρώμα ασφάλειας.

```
module.exports = async () => {
  await new OpenApiValidator({
    apiSpec: "./swagger.yaml",
    validateSecurity: {
      handlers: {
        BearerAuth: bearerAuthenticator,
        BasicAuth: basicAuthenticator
      },
    },
  }).install(app);
```

Οι χειριστές είναι απλά συναρτήσεις γραμμένες σε JavaScript, οι οποίες λαμβάνουν το αίτημα και ελέγχουν εάν οι επικεφαλίδες (headers) είναι έγκυρες:

```
const basicAuthenticator = (req, scopes, schema) => {
  try {
    const header_basic_auth = (req.headers.authorization || "").split("
")[1] || "";
    const [headerUsername, headerPassword] =
Buffer.from(header_basic_auth, "base64")
      .toString()
      .split(":");

    const server_basic_auth = process.env.BASIC_AUTH;
    const [serverUsername, serverPassword] =
Buffer.from(server_basic_auth, "base64")
      .toString()
      .split(":");


    // Verify login and password are set and correct
    if (headerUsername && headerPassword && headerUsername ===
serverUsername && headerPassword === serverPassword) {
      return true;
    } else throw new AuthError("Invalid Token");
  } catch (e) {
    throw new AuthError("Invalid Token");
  }
};
```

6 Εφαρμογές μηχανικής μάθησης ως μέρος των υπηρεσιών δικτύου

Η μηχανική μάθηση αποτελεί στοιχείο ορόσημο στον τομέα της τεχνητής νοημοσύνης, αντιπροσωπεύοντας ένα δυναμικό πεδίο όπου οι υπολογιστές εκμεταλλεύονται την ικανότητά τους να μαθαίνουν από δεδομένα και να βελτιώνουν την απόδοσή τους. Μέσω της εφαρμογής προηγμένων αλγορίθμων και στατιστικών μοντέλων, τα συστήματα μηχανικής μάθησης ανιχνεύουν επιδέξια πρότυπα, προβλέπουν, και αυτοματοποιούν τις διαδικασίες λήψης αποφάσεων. Αυτή η τεχνολογία διεισδύει σε διάφορους τομείς, από την επεξεργασία φυσικής γλώσσας και την αναγνώριση εικόνας έως τα συστήματα συστάσεων και τα αυτόνομα οχήματα. Η επαναληπτική διαδικασία εκμάθησης επιτρέπει στις μηχανές να προσαρμόζονται και να εξελίσσονται, καθιστώντας τη μηχανική μάθηση ένα εξαιρετικό εργαλείο για την επίλυση πολύπλοκων προβλημάτων και την ανάκτηση πληροφοριών από έκτακτα σύνολα δεδομένων.

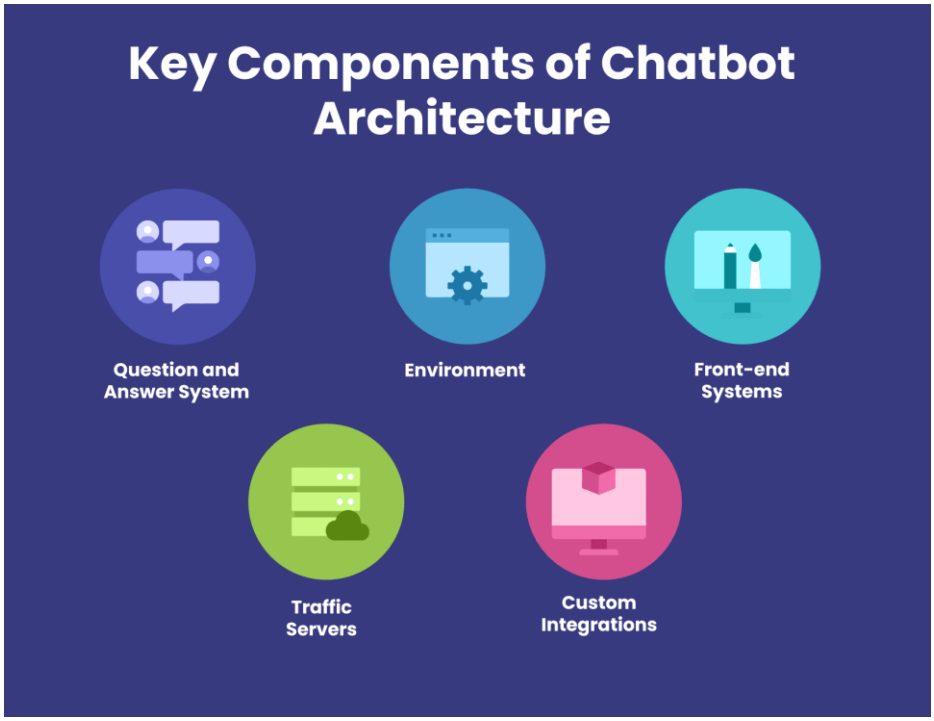
Η σχέση μεταξύ μηχανικής μάθησης και των REST APIs ενισχύει περαιτέρω τις δυνατότητες των ευφυών συστημάτων. Τα μοντέλα μηχανικής μάθησης, με την ικανότητά τους να αναλύουν και να ερμηνεύουν δεδομένα, αποτελούν σημαντικό κομμάτι των εφαρμογών που αξιοποιούν τα REST APIs. Αυτά τα APIs διευκολύνουν την απρόσκοπτη επικοινωνία και ανταλλαγή δεδομένων μεταξύ διαφορετικών συστημάτων λογισμικού, επιτρέποντας στους αλγόριθμους μηχανικής μάθησης να έχουν πρόσβαση και να επεξεργάζονται πληροφορίες από διαφορετικές πηγές. Χαρακτηριστικό παράδειγμα αποτελούν τα chatbots που συχνά συναντώνται σε ιστοσελίδες και εφαρμογές, έχοντας ως σκοπό να εξυπηρετήσουν επισκέπτες και να επιλύσουν προβλήματα χωρίς να απαιτείται ανθρώπινη παρέμβαση, εξοικονομώντας τόσο χρόνο όσο και πόρους. Αυτοί οι εικονικοί βοηθοί αξιοποιούν αλγορίθμους μηχανικής μάθησης για να κατανοήσουν κείμενο φυσικής γλώσσας και να ανταποκριθούν δημιουργώντας περιεχόμενο ως απάντηση στο εισερχόμενο αίτημα. Η απάντησή τους βασίζεται στη γνώση που έχουν αποκτήσει μέσα από τη βάση γνώσεων που είναι συνδεμένα. Συνήθως όταν δεν είναι ικανό να απαντήσει σε κάποιο ερώτημα, απαιτείται ανθρώπινη παρέμβαση για διαχείριση αυτού του σεναρίου, με το bot να μαθαίνει μέσα από αυτή τη διαδικασία, αποκτώντας την ικανότητα να ανταποκριθεί σε ένα παρόμοιο μελλοντικό αίτημα. Όλοι αυτοί οι αλγόριθμοι που είναι υπεύθυνοι να λάβουν ένα εισερχόμενο αίτημα, να το αναγνωρίσουν, αναλύσουν και να επιστρέψουν το αντίστοιχο αποτέλεσμα συνήθως είναι υλοποιημένοι σε μία πλατφόρμα, όπως AWS, Google κ.α. Έτσι οι υπηρεσίες διαδικτύου χρησιμοποιούνται ως μεσάζοντας, αναλαμβάνοντας τη μεταφορά του αιτήματος και της απόκρισης.

6.1 Chatbot Architecture

Σύμφωνα με το άρθρο **How do Chatbots Work? A Guide to Chatbot Architecture** υπάρχουν 5 βασικά στοιχεία από τα οποία απαρτίζεται ένα chatbot:

- Question and Answer System (σύστημα διαχείρισης ερωτημάτων και δημιουργίας απαντήσεων): Είναι υπεύθυνο να ανταποκρίνεται στις ερωτήσεις των χρηστών, απαντώντας κατάλληλα σύμφωνα με τη βάση γνώσεων. Αποτελείται από τα ακόλουθα στοιχεία:
    - Manual Training (χειροκίνητη – μη αυτόματη εκπαίδευση): Συντάσσεται μια λίστα με συνηθισμένες – αναμενόμενες ερωτήσεις, μαζί με τις αντίστοιχες απαντήσεις για κάθε αίτημα. Βασικό πλεονέκτημα αυτής της μεθόδου, είναι ο άμεσος εντοπισμός

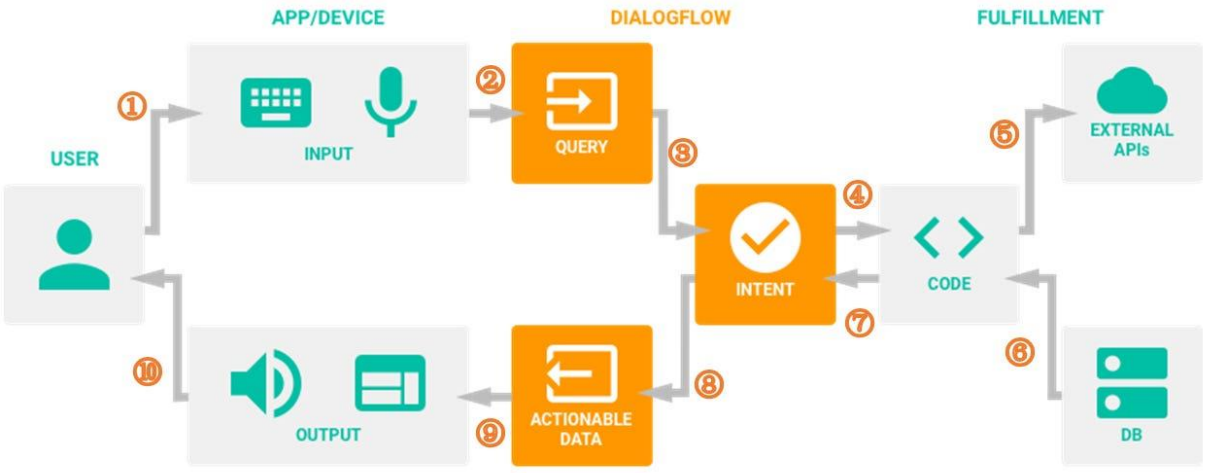
απαντήσεων χωρίς να χρειάζονται πολλοί πόροι και χρόνος για την εκπαίδευση του bot και την επεξεργασία του αιτήματος.
    - Automated Training (Αυτόματη εκπαίδευση): Συνήθως εισάγονται έγγραφα κειμένου, καθοδηγώντας το bot πως να εκπαιδευτεί μόνο του. Μέσα από αυτά τα αρχεία θα δημιουργηθεί τόσο μια λίστα με υποψήφιες ερωτήσεις όσο και μια λίστα με υποψήφιες απαντήσεις.
- Environment (Περιβάλλον): Εφαρμόζει επεξεργασία φυσική γλώσσας (NLP – Natural Language Process) με σκοπό να ενοποιήσει συμφραζόμενα από τα μηνύματα του χρήστη. Η επεξεργασία φυσικής γλώσσας αποτελεί τον πυρήνα της αρχιτεκτονικής ενός chatbot. Ερμηνεύει το κείμενο του χρήστη και εφαρμόζοντας προηγμένους μεθόδους και αλγορίθμους μηχανικής μάθησης, μετατρέπει αυτό το κείμενο σε οργανωμένες και αναγνωρίσιμες εισόδους, που το σύστημα μπορεί να αναλύσει και να επεξεργαστεί. Αποτελείται από δύο σημαντικά στοιχεία:
    - Intent Classifier (ταξινομητής πρόθεσης): Ένας ταξινομητής πρόθεσης αντιστοιχίζεται μεταξύ αυτού που ζητά ο χρήστης και του τύπου της ενέργειας που εκτελείται από το λογισμικό.
    - Entity Extractor (Εργαλείο εξαγωγής οντοτήτων): Ο εξαγωγέας οντοτήτων είναι υπεύθυνος για τον προσδιορισμό λέξεων-κλειδιών από το ερώτημα του χρήστη που βοηθά στον προσδιορισμό του τι αναζητά ο χρήστης.
- Front – End Systems: Πρόκειται για το μέρος στο οποίο ο χρήστης αλληλοεπιδρά με το bot. Μπορεί να είναι μια γραφική διεπαφή συνομιλίας, η οποία προσφέρεται μέσα από μια ιστοσελίδα, μια εφαρμογή δικτύου ή μια εφαρμογή για κινητές συσκευές. Messenger, Slack, Skype είναι γνωστά παραδείγματα τέτοιων συστημάτων.
- Traffic Server (Διακομιστής κυκλοφορίας): Πρόκειται για το διακομιστή που είναι υπεύθυνος για τη διαχείριση των αιτημάτων και της μεταφοράς της πληροφορίας ανάμεσα στο front end σύστημα και στη πλατφόρμα που είναι υλοποιημένο το chatbot. Μπορεί να είναι ένας node server, όπου κάνοντας χρήση της βιβλιοθήκης Express, καθορίζει routes και controllers για τη διαχείριση και δρομολόγηση των αιτημάτων και αποκρίσεων.
- Custom Integrations (Προσαρμοσμένες ενσωματώσεις): Ένα chatbot δύναται να ενσωματωθεί με ήδη υπάρχοντα συστήματα όπως CRMs, βάσεις δεδομένων, εξωτερικά APIs (Weather, Geolocation) προσφέροντας πρόσβαση σε υπηρεσίες χωρίς να υπάρχει ανάγκη υλοποίησης τους από το μηδέν



Εικόνα 24 - Chatbot Architecture

6.2 Google DialogFlow

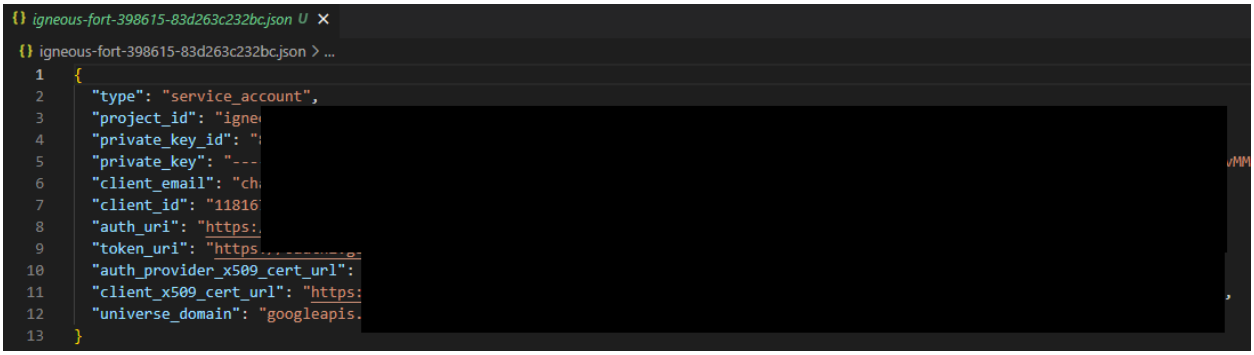
Το Dialogflow είναι μια πλατφόρμα επεξεργασίας φυσικής γλώσσας (NLP) που αναπτύχθηκε από την Google. Επιτρέπει στους χρήστες να δημιουργούν ελκυστικές και δια δραστικές εμπειρίες συνομιλίας κατανοώντας και αναλύοντας δεδομένα φυσικής γλώσσας.



Εικόνα 25 - DialogFlow

Το DialogFlow είναι διαθέσιμο μέσα από το GCP (Google Cloud Platform) και προκειμένου να χρησιμοποιηθεί πρέπει να ακολουθηθούν τα παρακάτω βήματα:

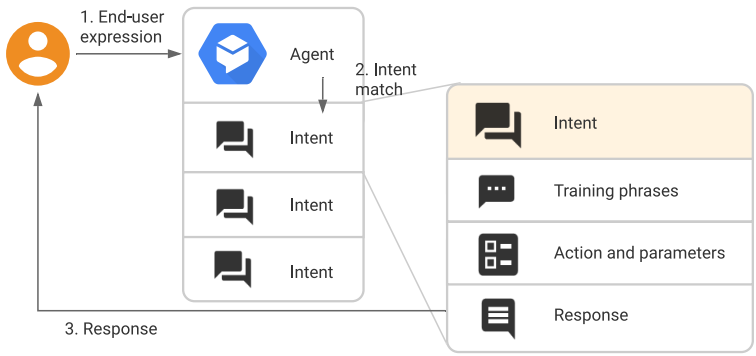
- Δημιουργία νέου project
- Δημιουργία service account για το επιλεγμένο project
- Δημιουργία ιδιωτικού κλειδιού (API key), το οποίο θα χρησιμοποιηθεί για την επικοινωνία ανάμεσα στο διακομιστή και την πλατφόρμα.
- Εξαγωγή κλειδιού μέσω ενός JSON αρχείου



Εικόνα 26 - API Key

Επόμενο βήμα αποτελεί η επιλογή της έκδοσης του DialogFlow που θα χρησιμοποιηθεί καθώς προσφέρονται 2 επιλογές:

1. DialogFlow CX: Αποτελεί μια *out of the box* επιλογή καθώς βασίζεται σε διαγράμματα ροής και είναι ιδανική για απλά σενάρια όπου προσφέρεται δυνατότητα επιλογής στο χρήστη μέσα από μια λίστα επιλογών. Δεν είναι ιδανικό για διαχείριση ελεύθερου κειμένου.
2. DialogFlow ES (Essentials): Μια πιο σύνθετη επιλογή, ιδανική για διαχείριση ελεύθερου κειμένου με περισσότερες δυνατότητες από το CX.



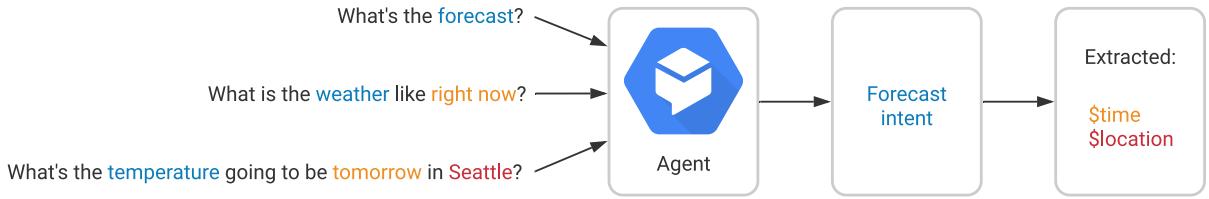
Εικόνα 27 - DialogFlow ES

6.2.1 Agent

Πρώτο βήμα αποτελεί η δημιουργία ενός agent, όπου agent είναι μια ενότητα κατανόησης φυσικής γλώσσας που σχεδιάζεται και δημιουργείται για τη διαχείριση συνομιλιών με τους τελικούς χρήστες. Λειτουργεί ως εικονικός βοηθός ή chatbot που αλληλοεπιδρά με τους χρήστες και επεξεργάζεται τα αιτήματά τους. Μπορεί να δημιουργηθεί ένας custom από την αρχή, να χρησιμοποιηθεί ένας υπάρχον (prebuilt) ή ένας συνδυασμός αυτών των επιλογών. Συνήθως οι prebuilt agents βασίζονται σε επικοινωνία με επιμέρους οντότητες όπως εξωτερικά συστήματα ή third party APIs, καθιστώντας τις επιλογές για πιο έμπειρους χρήστες της πλατφόρμας.

6.2.2 Intents

Ένα intent αντιπροσωπεύει αυτό που ο χρήστης θέλει να επιτύχει ή να επικοινωνήσει. Για κάθε νέο agent, υπάρχουν δύο διαθέσιμα από προεπιλογή (Welcome, Fallback), προσφέροντας παράλληλα τη δυνατότητα δημιουργίας ενός νέου.



Εικόνα 28 - Intents

Κάθε intent προσφέρει τις ακόλουθες επιλογές:

- Context: Διατηρεί την κατάσταση μιας συνομιλίας, επιτρέποντας στα intents να συνδέονται με ουσιαστικό τρόπο, βοηθώντας έτσι το chatbot να κατανοεί πιο εύκολα το περιεχόμενο της συζήτησης, βελτιώνοντας την ακρίβεια αντιστοίχισης.
- Events: Ένας τρόπος αναγνώρισης εισόδου του χρήστη, χρησιμοποιώντας προκαθορισμένα event της πλατφόρμας, χωρίς να χρειάζεται ανάλυση και αντιστοίχιση του κειμένου εισόδου.
- Training Phrases: Συγκεκριμένες φράσεις – προτάσεις που μπορεί να γράψει ο χρήστης, έτσι ώστε να ενεργοποιήσει κάποιο intent. Αυτή η λειτουργικότητα, βοηθάει το chatbot να εντοπίζει και να αναγνωρίζει διαφορετικούς τρόπους με τους οποίους μπορεί να εκφραστεί το ίδιο intent.
- Actions and Parameters: Οι παράμετροι χρησιμοποιούνται για να διευκρινιστεί πληροφορία σε κάποιο intent. Για παράδειγμα σε μια πρόταση όπου ο χρήστης γράφει “what is the weather in Tripoli”, η λέξη Τρίπολη μπορεί να χαρακτηριστεί ως μια παράμετρος με όνομα *city*, η οποία μπορεί να λάβει διάφορες τιμές, χωρίς να τροποποιείται η υπόλοιπη πρόταση. Αντίστοιχα ένα action αντιπροσωπεύει την εργασία ή τη λειτουργία που πρέπει να εκτελεστεί όταν ενεργοποιείται ένα intent.
- Responses: Τα μηνύματα ή οι ενέργειες που το chatbot θα πρέπει να επιστρέψει πίσω στον χρήστη μόλις γίνει αντιστοίχιση του intent. Μπορεί να είναι απλό κείμενο, ηχητικό μήνυμα ή κάποια ενέργεια.

- Fulfillment: Κώδικας διαθέσιμος σε κάποιο εξωτερικό σύστημα, ο οποίος είναι υπεύθυνος να διαχειριστεί το intent και να προσφέρει δεδομένα στο χρήστη μέσα από κάποια υπηρεσία δικτύου.

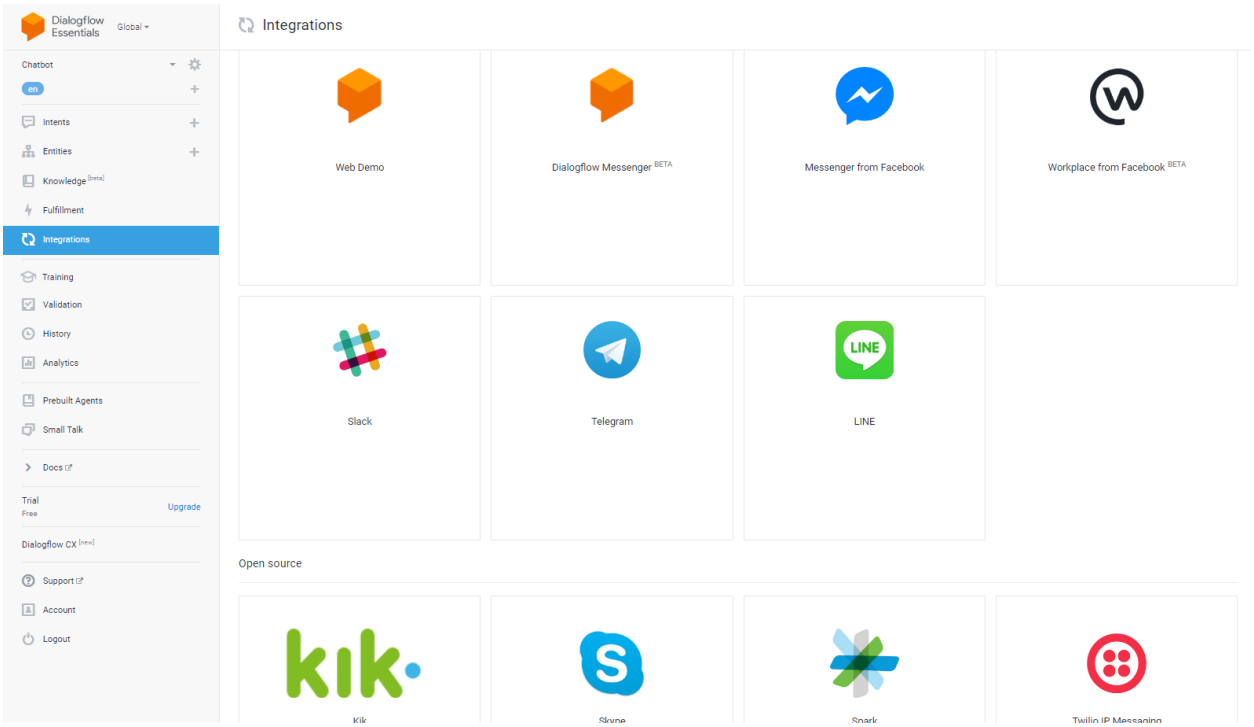
6.2.3 Entities

Μια οντότητα αντιπροσωπεύει μια έννοια ή ένα αντικείμενο που σχετίζεται με την είσοδο ενός χρήστη. Χρησιμοποιούνται για την εξαγωγή συγκεκριμένων τμημάτων πληροφοριών από το κείμενο που δόθηκε από το χρήστη, βοηθώντας στην ακριβέστερη κατανόηση της πρόθεσης του. Χωρίζονται σε δύο κατηγορίες:

- System entities: Υπάρχουσες οντότητες, χρήσιμες για απλά και γνωστά ζητήματα όπως η εξαγωγή μιας ημερομηνίας ή ενός χρώματος από το κείμενο που δόθηκε ως είσοδος. Χαρακτηριστικά system entities είναι τα *@sys.date, @sys.number*
- Custom entities: Οντότητες δημιουργημένες από το χρήστη ώστε να ικανοποιήσουν τις ανάγκες της δικής του υλοποίησης.

6.2.4 Integrations

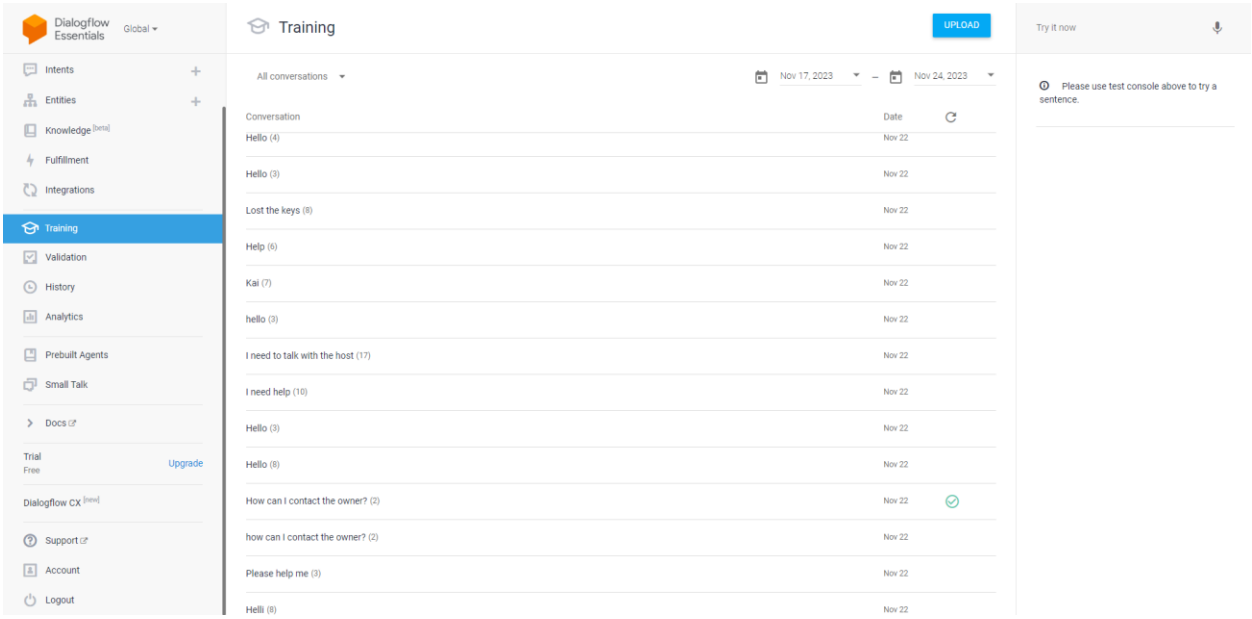
Τα integrations αποτελούν ένα από τα βασικά στοιχεία της αρχιτεκτονικής των chat bots, όπου προσφέρεται δυνατότητα ενσωμάτωσης τους σε υπάρχοντα συστήματα όπως Facebook, Messenger, Slack.



Εικόνα 29 – Integrations

6.2.5 Training

Μέσα από την καρτέλα Training, υπάρχει η δυνατότητα να εκπαιδευτεί ο Agent είτε ανεβάζοντας κάποιο αρχείο, είτε εξετάζοντας και αναλύοντας προηγούμενες συζητήσεις. Ο πρώτος τρόπος αποτελεί την αυτόματη εκπαίδευση που αναφέρθηκε στην αρχιτεκτονική των chatbots, όπου το chatbot εκπαιδεύεται αυτόνομα βάση ενός αρχείου κειμένου που περιέχει αναμενόμενες ερωτήσεις και απαντήσεις. Αποτελεί ένα πιο χρονοβόρο τρόπο αλλά με πιο υποσχόμενα αποτελέσματα. Από την άλλη μεριά, υπάρχει η δυνατότητα μη αυτόματης εκπαίδευσης του chatbot, όταν δεν μπορεί να ανταποκριθεί σε κάποια είσοδο του χρήστη, απαιτείται χειροκίνητη παρέμβαση από το διαχειριστή του agent, ώστε να αποδώσει το δοθέν κείμενο σε κάποιο υπάρχον intent ή να δημιουργήσει ένα νέο, υπεύθυνο να διαχειριστεί αυτή την περίπτωση.



Εικόνα 30 – Training

6.3 Front – End System

Όπως αναφέρθηκε στην αρχιτεκτονική των chatbots, απαραίτητο στοιχείο αποτελεί η διεπαφή της συνομιλίας. Μπορεί να είναι μια ιστοσελίδα, μια εφαρμογή ή ένα υπάρχον σύστημα όπου θα ενσωματωθεί τελικά το bot. Για τη συγκεκριμένη περίπτωση επιλέχθηκε η δημιουργία μιας απλής εφαρμογής χρησιμοποιώντας **react native**, ένα framework για την ανάπτυξη υβριδικών εφαρμογών για κινητές συσκευές. Αρχικά απαιτείται εγκατάσταση του πακέτου **react-native-gifted-chat** δίνοντας την ακόλουθη εντολή στο τερματικό του project:

```
$ npm install react-native-gifted-chat
```

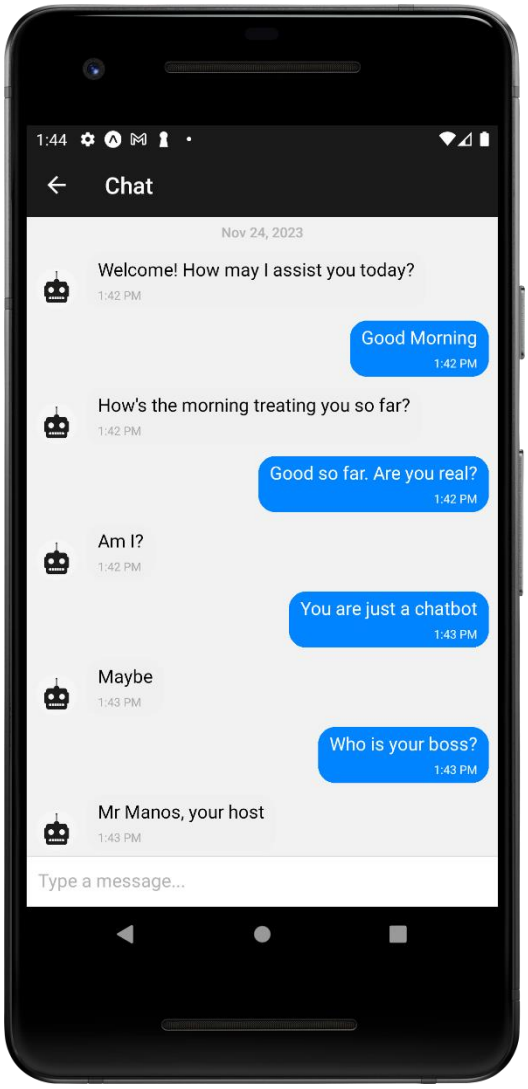
Σε πρώτο βήμα εισάγονται οι απαραίτητες βιβλιοθήκες και στοιχεία. Το *useState* αποτελεί hook του React και χρησιμοποιείται για να ελέγξει την κατάσταση μιας μεταβλητής, των μηνυμάτων που ανταλλάσσονται στο συγκεκριμένο παράδειγμα. Το *GiftedChat* είναι το στοιχείο που θα χρησιμοποιηθεί για τη διαχείριση της συνομιλίας και προβολή της διεπαφής, το οποίο βρίσκεται στο πακέτο που εγκαταστάθηκε προηγουμένως. Το dialogFlowRequest αποτελεί μια ασύγχρονη συνάρτηση, γραμμένη σε διαφορετικό αρχείο, η οποία χρησιμοποιείται για να σταλεί το αίτημα του χρήστη στο διακομιστή. Η συνάρτηση *onSend()* αποτελεί το χειριστή του κουμπιού send που υπάρχει στη διεπαφή της συνομιλίας και ουσιαστικά καλεί τη συνάρτηση DialogFlowIntegration(userMessage) στέλνοντας ως παράμετρο το κείμενο που έχει πληκτρολογήσει ο χρήστης στο chat. Η ασύγχρονη συνάρτηση *DialogFlowRequest(text)* στέλνει ένα HTTP αίτημα με το κείμενο του χρήστη, τη μεταβλητή text δηλαδή που έλαβε από τη συνάρτηση *DialogFlowIntegration(text)*. Τέλος στο view model, το οποίο είναι γραμμένο σε JSX, μια συντακτική επέκταση της JavaScript, η οποία συχνά ταυτίζεται με τη βιβλιοθήκη React υπάρχει διαθέσιμη η διεπαφή της συνομιλίας χρησιμοποιώντας το component *GiftedChat.*

```
import {useState} from "react";
import {View} from "react-native";
import {GiftedChat} from "react-native-gifted-chat";
import {dialogFlowRequest} from "../api/DialogFlowRequest";

function DialogFlowScreen() {
  const [messages, setMessages] = useState([]);
  const [chatbotResponse, setChatBotResponse] = useState("");

  const onSend = (newMessages = []) => {
    setMessages((previousMessages) =>
      GiftedChat.append(previousMessages, newMessages)
    );
    const userMessage = newMessages[0].text;
    dialogFlowIntegration(userMessage);
  };

  async function dialogFlowIntegration(text) {
    try {
      const res = await dialogFlowRequest(text);
      setChatBotResponse(res.response);
      const botMessage = {
        _id: Math.random().toString(36).substring(7),
        text: res.response,
        createdAt: new Date(),
        user: {
          _id: 2,
          name: "Chatbot",
        },
      };
      setMessages((previousMessages) =>
        GiftedChat.append(previousMessages, [botMessage])
      );
    } catch (error) {console.log(error);}
  }
  return (
    <View>
      <GiftedChat
        messages={messages}
        onSend={(newMessages) => onSend(newMessages)}
        user={{
          _id: 1,
        }}
      />
    </View>
  );
}
export default DialogFlowScreen;
```

Ακολούθως φαίνεται η ασύγχρονη συνάρτηση *dialogFlowReuqest(input)*, η οποία είναι διαθέσιμη σε ξεχωριστό αρχείο για λόγους οργάνωσης. Πρόκειται για μια μέθοδο POST, στέλνοντας ως body το κείμενο εισόδου. Επειδή το response θα είναι σε μορφή JSON, ρυθμίζονται κατάλληλα τα headers θέτοντας το **content-type: "application/json"**. Ακολούθως το αίτημα θα σταλεί στη διαδρομή */{host}/data/api/dialogFlow* χρησιμοποιώντας το **Fetch API**. Σε περίπτωση επιτυχίας επιστρέφεται το response, διαφορετικά το status code της ανταπόκρισης.

```
import { host } from "../constants/host";

export async function dialogFlowRequest(input) {
  const url = `${host}/data/api/dialogFlow`;

  let data = {
    text: input,
  };

  const requestOptions = {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify(data),
  };

  try {
    const response = await fetch(url, requestOptions);
    const res = await response.json();

    if (res) return res;
    else return res.status
  } catch (error) {
    console.error("dialogFlow request error: ", error);
    return null;
  }
}
```

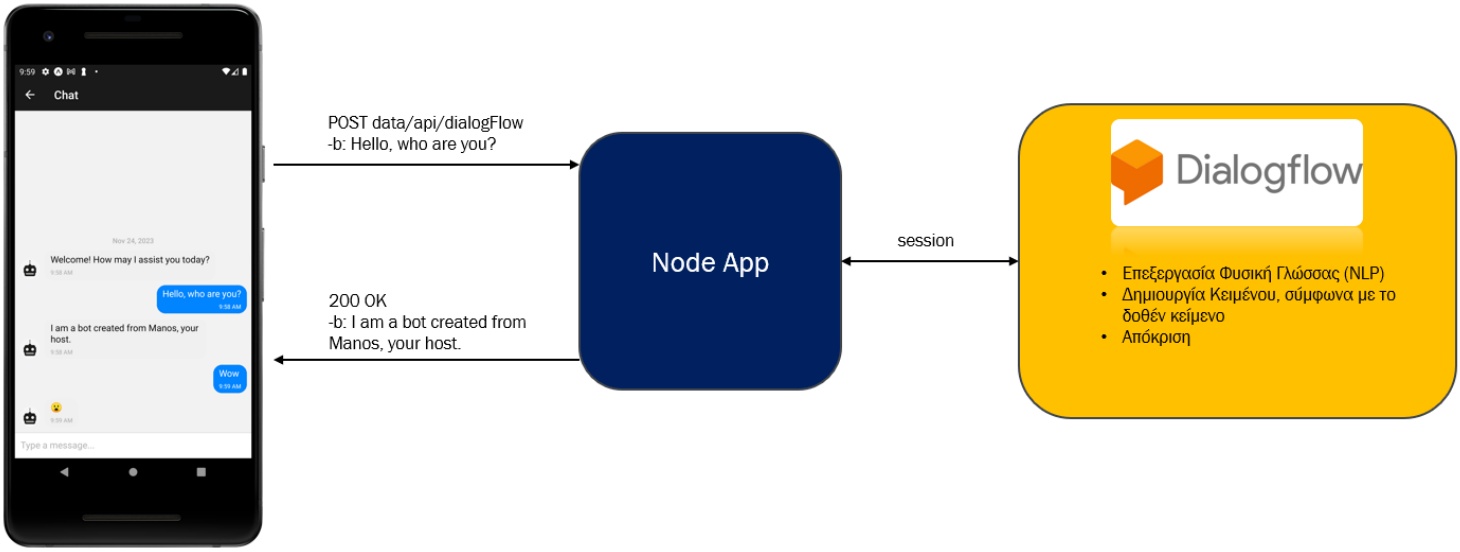
Ως αποτέλεσμα του κώδικα που περιεγράφηκε προηγουμένως είναι το ακόλουθο chat:



Εικόνα 31 - Front End System

6.4 Traffic Server

Το αίτημα που στάλθηκε προηγουμένως από το Front End System καταλήγει στο διακομιστή, ο οποίος λαμβάνει το κείμενο του χρήστη, μπορεί προαιρετικά να το επεξεργαστεί – φιλτράρει και στη συνέχεια το προωθεί στο dialogFlow, στο GCP όπου θα πραγματοποιηθεί ανάλυση του κειμένου με τους τρόπους που προαναφέρθηκαν και στη συνέχεια θα επιστραφεί πίσω στο διακομιστή η απόκριση του αρχικού μηνύματος. Τέλος ο διακομιστής θα ανταποκριθεί στο Front – End System, στο αρχικό αίτημα, στέλνοντας το κείμενο που δημιουργήθηκε από το bot. Ο διακομιστής δηλαδή λειτουργεί ως ένα κανάλι επικοινωνίας ανάμεσα στο GCP και στο react native app όπως φαίνεται στην παρακάτω εικόνα:



Εικόνα 32 - Traffic Server

Χρησιμοποιώντας ένα διακομιστή node και τη βιβλιοθήκη Express, πρέπει να δημιουργηθεί ένα νέο μονοπάτι (/data/api/dialogFlow) για να δρομολογηθεί σωστά το αίτημα και ένας νέος controller για να το διαχειριστεί. Πριν τη δημιουργία του route και του controller όμως θα πρέπει να ενημερωθεί σωστά το αρχείο τεκμηρίωσης (swagger.YAML) όπως φαίνεται παρακάτω:

```
/data/api/dialogFlow:
  post:
    tags:
      - dialogFlow
    summary: Interact with DialogFlow
    description: Send a text to DialogFlow and receive a response.
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            properties:
              text:
                type: string
                description: The text to send to DialogFlow.
                example: "Hello, how are you?"
            required:
              - text
    responses:
      "200":
        description: Successful response
      "400":
        description: Bad request
      "500":
        description: Internal server error
```

Αντίστοιχα ο router θα είναι:

```
const express = require("express");
const botController = require("../controllers/bot");
const router = express.Router();

router.post("/dialogFlow",botController.getBotResponse);

module.exports = router;
```

Για τον controller απαιτείται η εγκατάσταση του πακέτου *dialogflow*, μέσω του οποίου επιτυγχάνεται η σύνδεση με το GCP, ανοίγοντας ένα τερματικό και δίνοντας την εντολή:

```
$ npm install dialogflow
```

Στη συνέχεια θα χρησιμοποιηθεί το JSON αρχείο, το οποίο δημιουργήθηκε στο GCP, παρέχοντας το API key για την ομαλή επικοινωνία του διακομιστή με τη πλατφόρμα, δημιουργώντας ένα κανάλι επικοινωνίας με σκοπό την ανταλλαγή μηνυμάτων:

```
const { SessionsClient } = require("dialogflow");
const path = require("path");

const sessionClient = new SessionsClient({
  keyFilename: path.join(__dirname, "key.json"),
});

exports.getBotResponse = async (req, res, next) => {
  const sessionPath = sessionClient.sessionPath(<project name>);
  const request = {
    session: sessionPath,
    queryInput: {
      text: {
        text: req.body.text
      },
    },
  };

  try {
    const response = await sessionClient.detectIntent(request);
    res.status(200).json({
      response: response[0].queryResult.fulfillmentText,
    });
  } catch (error) {
    res.status(500).json({
      error: "Internal Server Error",
    });
  }
};
```

7 Συμπεράσματα

Συνοπτικά, οι υπηρεσίες δικτύου φαίνεται να παίζουν ένα καταλυτικό ρόλο στην επίτευξη της επικοινωνίας και ανταλλαγής δεδομένων μεταξύ συστημάτων στο διαδίκτυο. Ειδικότερα, η αρχιτεκτονική REST, βασισμένη στο πρωτόκολλο HTTP, προσφέρει ένα ευέλικτο πλαίσιο για την αποτελεσματική ανταλλαγή πληροφοριών. Η ασύγχρονη φύση του Node.js κρίνεται ιδιαίτερα χρήσιμη και αποτελεσματική στον χειρισμό πολλαπλών και ταυτόχρονων αιτημάτων που σχετίζονται με τα RESTful APIs, καθιστώντας το μια από τις δημοφιλέστερες επιλογές για την ανάπτυξη εφαρμογών που χρησιμοποιούν υπηρεσίες δικτύου. Κατά το σχεδιασμό ενός API, είναι ζωτικής σημασίας η συμμόρφωση με τις βέλτιστες πρακτικές, δίνοντας έμφαση στην ασφάλεια, οργάνωση και απόδοση των εφαρμογών που δημιουργούνται.

Πρόκειται για ένα κλάδο που εξελίσσεται διαρκώς, επομένως η παρακολούθηση των εξελίξεων στον τομέα των υπηρεσιών δικτύου είναι αναγκαία για την προσαρμογή σε νέες τεχνολογίες και πρότυπα. Λόγω της αδιάκοπης αυτής εξέλιξης, οι υπηρεσίες δικτύου βρίσκουν απήχηση σε ολοένα και περισσότερους κλάδους της τεχνολογίας, παραδείγματα των οποίων αποτελούν η τεχνητή νοημοσύνη και οι αρχές μηχανικής μάθησης, διευκολύνοντας τη μεταφορά πληροφοριών και βελτιστοποιώντας την επικοινωνία ανάμεσα σε διάφορα συστήματα που συμμετέχουν.

8 Πηγές – Βιβλιογραφία

[1] What's the Difference Between JSON and XML. Ανάκτηση December 24, 2023 από aws.amazon.com: https://aws.amazon.com/compare/the-difference-between-json-xml/

[2] Best practices for REST API design. Ανάκτηση December 27, 2023 από stackoverflow.blog.com: https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design/

[3] REST API URI Naming Conventions and Best Practices. Ανάκτηση February 11, 2023 από restfulapi.net: https://restfulapi.net/resource-naming/

[4] How to strengthen the security of your APIs to counter the most common attacks. Ανάκτηση December 28, 2023 από vaadata.com: https://www.vaadata.com/blog/how-to-strengthen-the-security-of-your-apis-to-counter-the-most-common-attacks/

[5] How do Chatbots Work? A Guide to Chatbot Architecture. Ανάκτηση January 3 2024 από marutitech.com https://marutitech.com/chatbots-work-guide-chatbot-architecture/

[6] *Brajesh De (2017) - API Management_ An Architect's Guide to Developing and Managing APIs for Your Organization, Apress*