

BACKEND

THE UI OF THE PROGRAMMERS

apostolis anastasiou

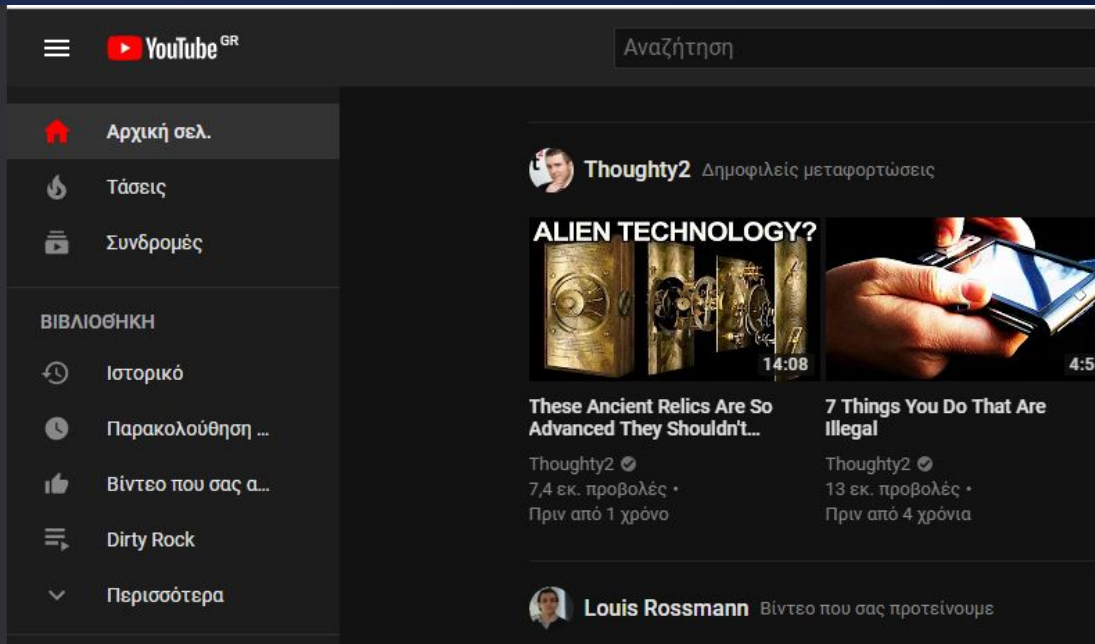
INTRODUCTION

TI **EINAI** TO **BACKEND**

INTRODUCTION

BACKEND vs FRONTEND

Το **Frontend** αποτελεί ο **καμβάς** που εμφανίζεται στα μάτια του χρήστη.



Μια σελίδα δηλαδή λέγεται **Front-end**.

Frontend
διότι ανήκει στο **μπροστινό**
μέρος του προγράμματος

INTRODUCTION

BACKEND vs FRONTEND

Το **Backend** είναι για τον **προγραμματιστή** ότι είναι το **Frontend** για τον **απλό χρήστη**.

Backend - User Interface **Προγραμματιστή**

Frontend - User Interface **Απλού Χρήστη**

INTRODUCTION

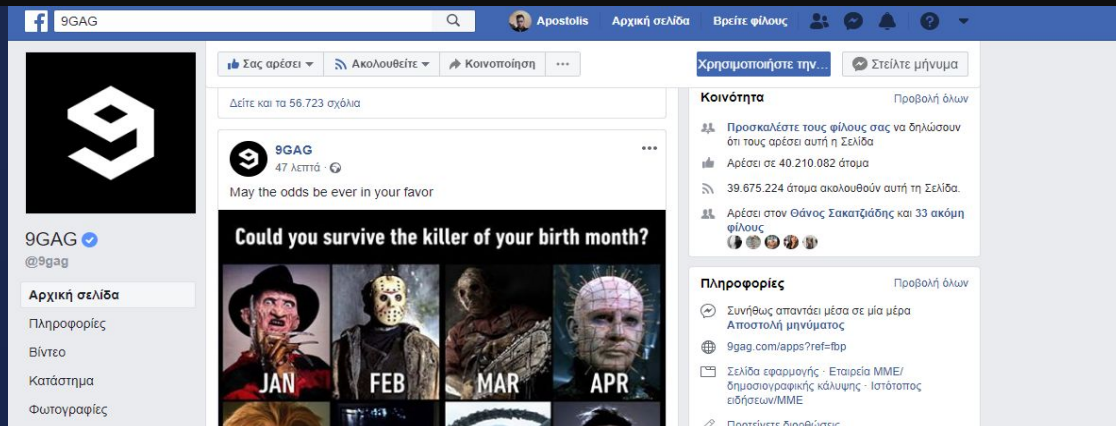
BACKEND vs FRONTEND

Backend = Λήψη + Διαχείριση Πληροφοριών + Υπηρεσίες =

- στοιχεία χρηστών (Όνομα χρήστη, Φίλοι)
- φωτογραφίες / χάρτες (photo-profile, google maps)
- ιστορικό χρήστη (Likes χρήστη, σχόλια)
- Voicell (Skype Call, Discord Call)
- **Και ότι υπάρχει σε μορφή δεδομένων, διάδρασης και υπηρεσιών**

INTRODUCTION

BACKEND vs FRONTEND



GET /users

GET /users/15/orders

GET /video/yut54324refwedf

POST /users/15/orders

payload: { "food" : "pizza margaritta" }

VS

INTRODUCTION

BACKEND vs FRONTEND

Μπορώ να έχω μια σελίδα/εφαρμογή που τρέχει μόνο Frontend;

Αν είναι **στατική** – Δηλαδή αν δεν αλλάζει το περιεχόμενο της σελίδας

Ή

Αν **παίρνεις** και διαχειρίζεσαι **δεδομένα** από άλλα **Backend** (APIs)

Ή

Αν **παίρνεις** το **περιεχόμενο** άλλων **σελίδων** (πχ. iFrames)

INTRODUCTION

BACKEND vs FRONTEND

Στατικές Σελίδες

- Portfolio
- Land Page

Μη Στατικές Σελίδες/Εφαρμογές

- | | |
|------------|--|
| - Facebook | - Spotify |
| - Youtube | - e-shop |
| - Google | - Skrutz |
| - Discord | - Gmail |
| - Skype | - Ότι άλλο χρειάζεται δεδομένα και υπηρεσίες |

INTERACTION

ΠΩΣ ΝΑ **ΕΠΙΚΟΙΝΩΝΗΣΕΙΣ** ΜΕ ΤΟ **BACKEND**

INTERACTION

ΠΩΣ ΠΑΙΡΝΩ ΜΙΑ **ΠΛΗΡΟΦΟΡΙΑ** ΑΠΟ ΤΟ **BACKEND**

ΜΕ **HTTP REQUESTS**

ΒΑΣΙΚΟΙ ΤΥΠΟΙ **HTTP REQUEST**

- **POST** (CREATE)
- **GET** (READ)
- **PUT** (UPDATE)
- **DELETE** (DELETE)

Για να εκτελούμε βασικές εντολές **CRUD** (**C**reate **R**ead **U**ppdate **D**elete)

INTERACTION

HTTP **GET** REQUEST

ΘΕΛΩ ΝΑ ΠΑΡΩ ΟΛΟΥΣ ΤΟΥΣ ΧΡΗΣΤΕΣ

GET /users/ (<http://api.facebook.com/users>)

```
[ { id: "1", name: "jack" }, { id: "2", name: "bob" } ]
```

ΘΕΛΩ ΝΑ ΠΑΡΩ ΤΟΝ ΧΡΗΣΤΗ ΜΕ "ID = 1"

GET /users/1 (<http://api.facebook.com/users/1>)

```
[ { id: "1", name: "jack" } ]
```

INTERACTION

HTTP **POST** REQUEST

ΘΕΛΩ ΝΑ ΔΗΜΙΟΥΡΓΗΣΩ ΕΝΑ **ΝΕΟ ΧΡΗΣΤΗ**

POST /users/ (<http://api.facebook.com/users>)

```
fetch(  
  request="POST",  
  url="http://api.facebook.com/users",  
  data={ name: "jack", password: "1234" } )
```

ΔΗΜΙΟΥΡΓΗΣΩ **ΝΕΑ ΠΑΡΑΓΓΕΛΙΑ**

POST /orders/ (<http://api.e-food.com/orders>)

```
fetch(request="POST", url="http://api.e-food.com/orders", data=["pizza"])
```

INTERACTION

HTTP **RESPONSE**

ΤΑ ΠΙΟ ΣΥΝΗΘΙΣΜΕΝΑ **HTTP CODES**

SUCCESS CODES **(2XX)**

200 - Ok
201 - Created
204 - No content

CLIENT ERROR **(4XX)**

400 - Bad Request
401 - Unauthorized
403 - Forbidden
404 - Not Found
410 - Gone

SERVER ERROR (5XX)

500 - Internal Error
503 - Service Unavailable

REDIRECT CODES (3XX)

304 - *Not Modified*

INTERACTION

HTTP **RESPONSE**

ΤΟ ΠΟΛΥ **8-10** ΔΙΑΦΟΡΕΤΙΚΑ **HTTP CODES**



K.I.S.S. - Keep It Simple Stupid

Great advice... Hurts my feelings every time

INTERACTION

HTTP **REQUEST** + **RESPONSE**

```
fetch(  
  "PUT",  
  "http://localhost:3000/users",  
  {  
    userId: "5",  
    name: "nick",  
    email: "nick@gmail.com"  
  }  
)  
// Updates the fields `name` and  
`email` of user with ID=5
```

```
PUT /users  
HTTP RESPONSE - 200  
Payload:  
{  
  "response": {  
    "msg": "Updated user",  
    "userId": "5",  
    "updatedFields": {  
      "name": "nick",  
      "email": "nick@gmail.com"  
    }  
  }  
}
```

INTERACTION

HTTP GET QUERIES

ΘΕΛΩ ΝΑ ΠΑΡΩ ΤΟΝ ΧΡΗΣΤΗ ΜΕ ID=1

ΧΩΡΙΣ URI QUERIES

```
GET /users/1
```

ΜΕ URI QUERIES

```
GET /users?id=1
```

ΘΕΛΩ ΝΑ ΠΑΡΩ ΤΟΝ ΧΡΗΣΤΗ ΜΕ EMAIL=nick23@gmail.com ΚΑΙ NAME=nick

```
GET /users?email="nick23@gmail.com"&name="nick"
```


INTERACTION

RESPONSE **DATA TYPES**

ΕΝΑ **BACKEND** ΜΠΟΡΕΙ ΝΑ ΔΩΣΕΙ ΜΙΑ ΠΛΗΡΟΦΟΡΙΑ ΜΕ ΔΙΑΦΟΡΕΤΙΚΑ **FORMAT**

JSON

```
{“name”: “nick”, “balance”: 2}
```

XML

```
<user>  
  <name>nick</name>  
  <balance>2</balance>  
</user>
```

YAML

```
name: nick  
balance: 2
```

STACK

ΣΥΝΟΛΟ ΕΡΓΑΛΕΙΩΝ ΓΙΑ **ΑΝΑΠΤΥΞΗ** & **ΛΕΙΤΟΥΡΓΙΑ** ΛΟΓΙΣΜΙΚΟΥ

STACK = ΠΑΚΕΤΟ ΤΕΧΝΟΛΟΓΙΩΝ ΓΙΑ ΤΗΝ ΑΝΑΠΤΥΞΗ ΚΑΙ ΣΥΝΤΗΡΗΣΗ ΜΙΑΣ ΕΦΑΡΜΟΓΗΣ

ΓΕΝΙΚΑ ΣΕ ΕΝΑ **STACK** ΠΕΡΙΛΑΜΒΑΝΟΝΤΑΙ ΟΛΑ ΤΑ

- ΕΡΓΑΛΕΙΑ
- ΣΥΣΤΗΜΑΤΑ
- FRAMEWORKS
- ΓΛΩΣΣΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ
- ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ

ΚΑΙ ΟΤΙΔΗΠΟΤΕ ΑΛΛΟ ΣΗΜΑΝΤΙΚΟ ΑΠΑΙΤΕΙΤΑΙ ΓΙΑ ΤΗΝ **ΣΤΑΘΕΡΗ ΛΕΙΤΟΥΡΓΙΑ** ΚΑΙ **ΑΝΑΠΤΥΞΗ** ΤΟΥ ΠΛΗΡΟΦΟΡΙΑΚΟΥ ΣΥΣΤΗΜΑΤΟΣ

STACK

ΣΥΝΟΛΟ ΕΡΓΑΛΕΙΩΝ ΓΙΑ **ΑΝΑΠΤΥΞΗ** & **ΛΕΙΤΟΥΡΓΙΑ** ΛΟΓΙΣΜΙΚΟΥ

πχ.

MERN

MongoDB - **E**xpressJS - **R**eactJS - **N**odeJS

MongoDB = **Database**


ExpressJS = **Web Server** library for HTTP Communication

ReactJS = **Scripting Language** for frontend

NodeJS = **Operating System & Scripting Language & Web Server**

STACK

ΣΥΝΟΛΟ ΕΡΓΑΛΕΙΩΝ ΓΙΑ **ΑΝΑΠΤΥΞΗ** & **ΛΕΙΤΟΥΡΓΙΑ** ΛΟΓΙΣΜΙΚΟΥ



Airbnb ✓


Tech Stack

Application and Data

Utilities

DevOps

Business Tools



Uber ✓


Tech Stack

Application and Data

Utilities

DevOps

Business Tools



Dropbox ✓

Tech Stack

Application and Data

Utilities

DevOps

Business Tools

ARCHITECTURE

ΠΩΣ ΝΑ ΣΧΕΔΙΑΣΕΙΣ ΤΟ BACKEND

ARCHITECTURE

3 ΣΤΑΔΙΑ ΠΡΟΓΡΑΜΜΑΤΟΣ

- DEV
- STAGING
- PRODUCTION

DEV ΤΟ **ΣΤΑΔΙΟ** ΣΤΟ ΟΠΟΙΟ **ΠΡΟΓΡΑΜΜΑΤΙΖΕΙΣ** ΤΗΝ ΛΕΙΤΟΥΡΓΙΑ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ

STAGING ΟΤΑΝ ΤΟ ΠΡΟΓΡΑΜΜΑ ΕΙΝΑΙ **ΕΤΟΙΜΟ** ΝΑ ΑΝΕΒΕΙ ΣΕ **ΔΟΚΙΜΑΣΤΙΚΗ** ΛΕΙΤΟΥΡΓΙΑ

PRODUCTION ΜΟΝΟ ΟΤΑΝ ΕΙΝΑΙ ΕΤΟΙΜΟ ΤΟ ΠΡΟΓΡΑΜΜΑ ΝΑ ΜΠΕΙ ΣΕ **ΠΑΡΑΓΩΓΗ** - ΑΝ ΥΠΑΡΧΕΙ ΚΑΠΟΙΟ ΣΗΜΑΝΤΙΚΟ **BUG** ΑΠΛΑ ΠΡΟΣΕΥΧΗΣΟΥ

ARCHITECTURE

3 ΣΤΑΔΙΑ ΠΡΟΓΡΑΜΜΑΤΟΣ

ΔΙΑΦΟΡΕΤΙΚΑ **DOMAINS** ΓΙΑ ΚΑΘΕ ΛΕΙΤΟΥΡΓΙΑ ΠΡΟΓΡΑΜΜΑΤΟΣ

facebook.com

localhost:3001
staging.facebook.com
facebook.com

#dev - δηλαδή **local**
#staging
#production

api.facebook.com

localhost:3002
staging.api.facebook.com
api.facebook.com

#dev - δηλαδή **local**
#staging
#production

ARCHITECTURE

HTTP **REQUEST HANDLING**

ΕΞΑΓΩ ΤΟ **PATH** (`/orders/users/{id}`) - ΑΝΑΛΟΓΑ ΜΕ ΤΟ **PATH** ΕΠΙΛΕΓΩ ΤΟ ΑΝΑΛΟΓΟ **ROUTE**

ΕΞΑΓΩ ΤΗΝ **METHOD** (**GET**, **POST**, **PUT** κλπ)

ΕΞΑΓΩ ΤΑ **HEADERS** (**Content-Type**, **Cache-Control**, **Expires** και πολλά άλλα)

ΕΞΑΓΩ ΤΑ **QUERIES** (`/orders?item="pizza"` - Αν είναι **GET** request)

ΕΞΑΓΩ ΤΟ **PAYLOAD** (Τα **data** από ένα **request**)

ARCHITECTURE

ROUTES

ΤΟ ΚΑΘΕ **ROUTE** ΑΝΤΙΣΤΟΙΧΕΙ ΣΕ ΔΙΑΦΟΡΕΤΙΚΗ ΣΥΝΑΡΤΗΣΗ ΚΑΙ ΛΕΙΤΟΥΡΓΙΑ

πχ. Το **ROUTE** με **path** `/users` αντιστοιχεί σε διαφορετικές **λειτουργίες** και **πληροφορίες** ή **υπηρεσίες** από το **ROUTE** με **path** `/orders`.

ΕΝΑ **ROUTE** ΠΑΙΡΝΕΙ ΤΗΝ **METHOD** ΚΑΙ ΧΡΗΣΙΜΟΠΟΙΕΙ ΤΗΝ ΚΑΤΑΛΛΗΛΗ ΣΥΝΑΡΤΗΣΗ ΣΥΜΦΩΝΑ ΜΕ ΑΥΤΗΝ

πχ. Το **ROUTE** `/users` με method **GET** θα εκτελεί διαφορετικές λειτουργίες από το ίδιο **ROUTE** με method **POST**

ARCHITECTURE

ROUTES

ΚΑΘΕ **ΛΕΙΤΟΥΡΓΙΑ**

ΔΕΝ ΠΡΕΠΕΙ

ΝΑ ΕΠΗΡΕΑΖΕΙ

ΤΙΠΟΤΑ ΑΛΛΟ

ΕΚΤΟΣ ΑΠΟ ΤΗΝ

ΒΑΣΙΚΗ ΤΗΣ ΛΕΙΤΟΥΡΓΙΑ

πχ. Η **GET** όσες φορές και την καλέσεις **ΔΕΝ ΠΡΕΠΕΙ** να **ΕΚΤΕΛΕΣΕΙ** ή να **ΑΛΛΑΞΕΙ**
ΤΙΠΟΤΑ ΕΣΩΤΕΡΙΚΑ - παρά μόνο να δίνει την πληροφορία που χρειάζεται ο χρήστης

ARCHITECTURE

WORKERS

ΕΝΑΣ **WORKER** ΕΚΤΕΛΕΙ **BACKGROUND TASKS** ΑΝΑ ΚΑΠΟΙΑ ΟΡΙΣΜΕΝΗ ΩΡΑ ΚΑΙ ΓΕΝΙΚΟΤΕΡΑ ΕΚΤΕΛΕΙ **ΣΥΝΕΧΟΜΕΝΑ TASKS**.

πχ. **LOGGING WORKER** - **ΑΠΟΘΗΚΕΥΕΙ** το **output** από το πρόγραμμα για **μελλοντική χρήση** και **debugging**

Και συνήθως φροντίζει να τα **ΣΥΜΠΙΕΖΕΙ** τα **logs** μετά από κάποιες ημέρες

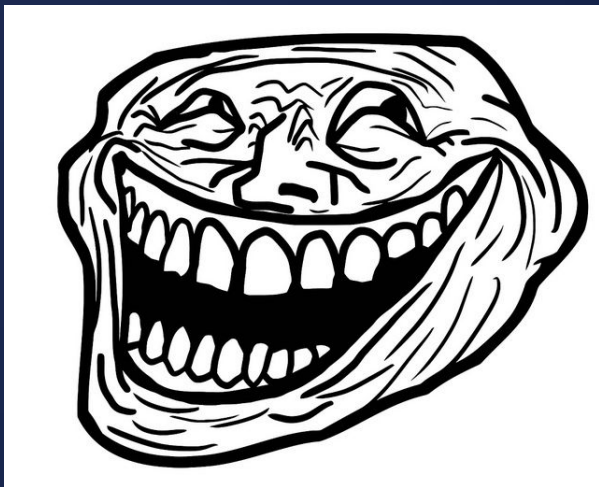
ΕΝΑΣ **WORKER** ΜΠΟΡΕΙ ΝΑ ΕΞΥΠΗΡΕΤΕΙ ΚΑΙ ΥΠΗΡΕΣΙΕΣ ΧΡΗΣΤΗ - ΟΧΙ ΜΟΝΟ ΣΥΣΤΗΜΑΤΟΣ

ARCHITECTURE

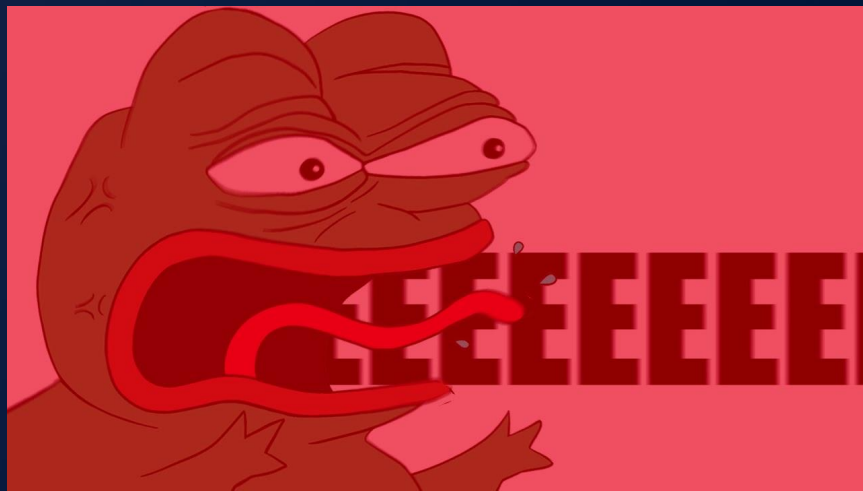
ERROR HANDLING

ΕΝΑ **ERROR** ... **ΔΕΝ ΠΡΕΠΕΙ ΝΑ ΔΙΑΚΟΠΤΕΙ ΤΕΛΕΙΩΣ ΤΟ ΠΡΟΓΡΑΜΜΑ** - ΜΟΝΟ ΣΕ ΕΞΑΙΡΕΤΙΚΕΣ ΠΕΡΙΠΤΩΣΕΙΣ Ή ΣΕ ΛΕΙΤΟΥΡΓΙΑ **DEV / STAGING**

ERROR CRASHING SERVER



CLIENT



ARCHITECTURE

ERROR HANDLING

ΕΣΥ



ARCHITECTURE

ERROR HANDLING

ΟΤΑΝ ΕΝΑ **REQUEST**

- ΒΓΑΖΕΙ **ERROR**
- Ο ΧΡΗΣΤΗΣ ΔΕΝ ΕΧΕΙ ΔΩΣΕΙ ΤΑ **REQUIRED ARGUMENTS**

ΤΟ **RESPONSE** ΘΑ ΠΡΕΠΕΙ ΝΑ **ΠΛΗΡΟΦΟΡΕΙ** ΤΟΝ ΧΡΗΣΤΗ ΣΧΕΤΙΚΑ ΜΕ ΤΟ ΣΦΑΛΜΑ ΟΣΟ ΠΙΟ **ΕΠΕΞΗΓΗΜΑΤΙΚΑ** ΚΑΙ **ΟΜΟΡΦΑ** ΓΙΝΕΤΑΙ

```
{
  "title"           : "Invalid Arguments",
  "detail"          : "Invalid arguments were given to the request",
  "statusCode"      : 400,
  "url"             : "http://api.application.com/error/invalid-args",
  "instance"        : "http://api.application.com/error/instance/23wdf2TS2qs",
  "invalid-params": [
    "email" : { "reason" : "Should be string" }
  ]
}
```

ARCHITECTURE

AUTHORIZATION

```
if pass=="walter" then:  
    authorize()
```



ARCHITECTURE

AUTHORIZATION

AUTHORIZATION STEPS

1. **FRONTEND ENCRYPTS** password
2. **FRONTEND PASSES** hashed password to **BACKEND**
3. **BACKEND CHECKS** hashed password to see
if its equal with the hashed password in the **DATABASE**

ARCHITECTURE

DATABASES

ΜΙΑ **DATABASE** ΠΡΕΠΕΙ ΝΑ ΣΕ **ΕΞΥΠΗΡΕΤΕΙ ΣΩΣΤΑ**

NoSQL Databases - Ευέλικτες, πολύ καλές για **Startups** και για **Databases** που αλλάζουν ή υπάρχει περίπτωση να αλλάξει η μορφή της και τα δεδομένα της

SQL Databases - Γρήγορες & Light αλλά δεν είναι ευέλικτες

Διαλέγεις τα σωστά εργαλεία για τις σωστές δουλειές
Για να χρησιμοποιείται κάτι, πιθανόν κάπου είναι πιο χρήσιμο από τα υπόλοιπα

ARCHITECTURE

MONOLITHIC VS **MICROSERVICES**

MONOLITHIC



One Ring to rule them all.

ARCHITECTURE

MONOLITHIC VS MICROSERVICES

MONOLITHIC

ΕΝΑ **MONOLITHIC** BACKEND ΕΧΕΙ ΟΛΕΣ ΤΙΣ ΛΕΙΤΟΥΡΓΙΕΣ ΚΑΙ ΔΙΑΧΕΙΡΙΖΕΤΑΙ ΟΛΑ ΤΑ REQUESTS ΑΠΟ ΕΝΑΝ CLIENT

ΑΠΑΡΤΙΖΕΤΑΙ ΑΠΟ ΜΙΑ **ΚΟΙΝΗ ΓΛΩΣΣΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ** ΚΑΙ **ΙΔΙΑ ΕΡΓΑΛΕΙΑ** ΣΤΟ ΣΥΝΟΛΟ ΤΗΣ

ARCHITECTURE

MONOLITHIC VS MICROSERVICES

MICROSERVICES

ΤΑ **MICROSERVICES** ΕΙΝΑΙ ΜΙΑ **ΑΡΧΙΤΕΚΤΟΝΙΚΗ BACKEND** ΠΟΥ ΑΠΑΡΤΙΖΕΤΑΙ ΑΠΟ ΠΟΛΛΕΣ ΕΦΑΡΜΟΓΕΣ ΠΟΥ ΕΠΙΚΟΙΝΩΝΟΥΝ ΜΕΤΑΞΥ ΤΟΥΣ

ΚΑΘΕ ΜΙΑ ΕΦΑΡΜΟΓΗ **ΜΠΟΡΕΙ ΝΑ ΓΡΑΦΤΕΙ** ΣΕ **ΔΙΑΦΟΡΕΤΙΚΗ ΓΛΩΣΣΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ** ΚΑΙ ΜΕ **ΔΙΑΦΟΡΕΤΙΚΑ ΕΡΓΑΛΕΙΑ** Ή **ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ**, ΑΡΚΕΙ ΝΑ **ΕΞΥΠΗΡΕΤΕΙ ΤΟΝ ΑΡΧΙΚΟ ΣΚΟΠΟ**

ARCHITECTURE

MONOLITHIC VS MICROSERVICES

MICROSERVICES

Πχ Facebook Microservices

- Login Handling *Service*
- Messenger *Service*
- Timeline *Service*
- Like / Reaction *Service*

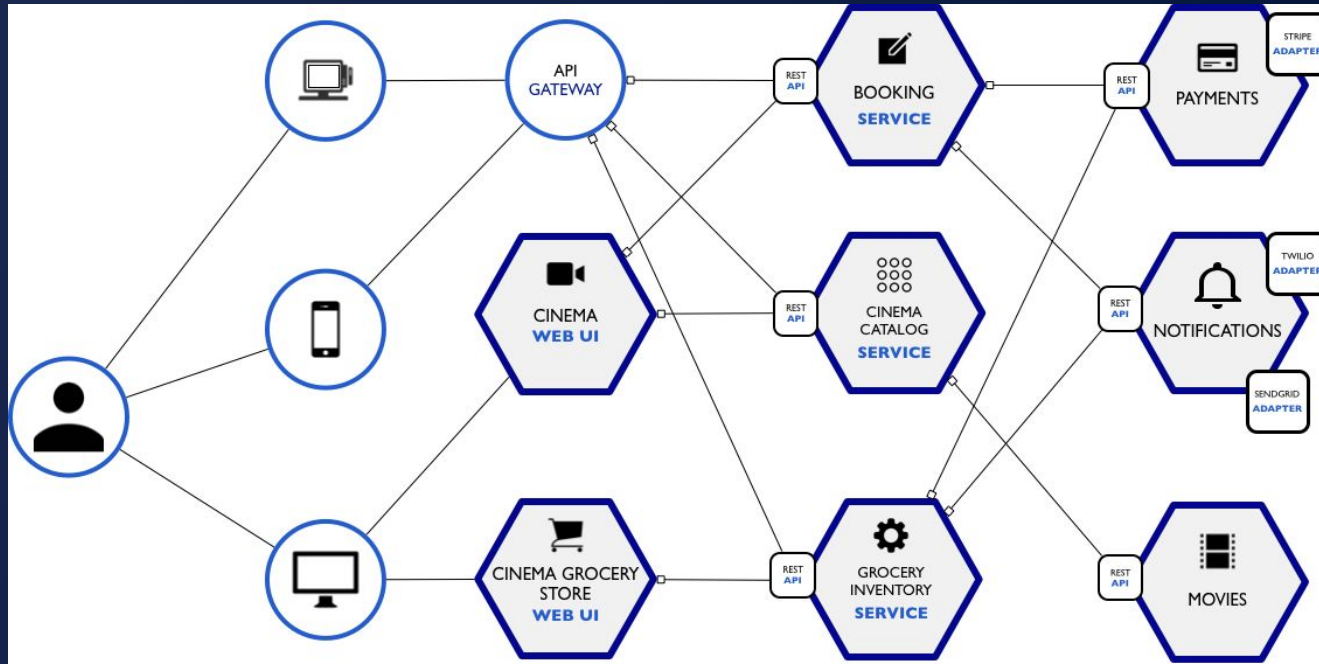
Πχ Steam Microservices

- Login Handling *Service*
- Item Purchase *Service*
- Gift *Service*
- Community *Service*
- User Messaging *Service*

ARCHITECTURE

MONOLITHIC VS MICROSERVICES

MICROSERVICES



DEPLOYMENT

ΠΩΣ ΑΝΕΒΑΖΩ ΤΟ **BACKEND** ΣΤΟ **SERVER**

DEPLOYMENT

SSH

Secure Shell - ΧΡΗΣΙΜΟΠΟΙΕΙΤΑΙ ΓΙΑ ΤΗΝ **ΑΣΦΑΛΗ ΣΥΝΔΕΣΗ** ΓΕΝΙΚΑ ΣΕ **Unix** ΣΥΣΤΗΜΑΤΑ
ΜΠΟΡΕΙ ΕΠΙΣΗΣ ΝΑ ΧΡΗΣΙΜΟΠΟΙΗΘΕΙ ΚΑΙ ΣΕ **Windows**

ΧΡΗΣΙΜΟΠΟΙΕΙΤΑΙ ΤΥΠΙΚΑ ΓΙΑ **ΑΠΟΜΑΚΡΥΣΜΕΝΗ ΕΚΤΕΛΕΣΗ ΕΝΤΟΛΩΝ**

```
$ ssh nick@132.125.94.1 -p 22
```


DEPLOYMENT

NGINX / PROXY

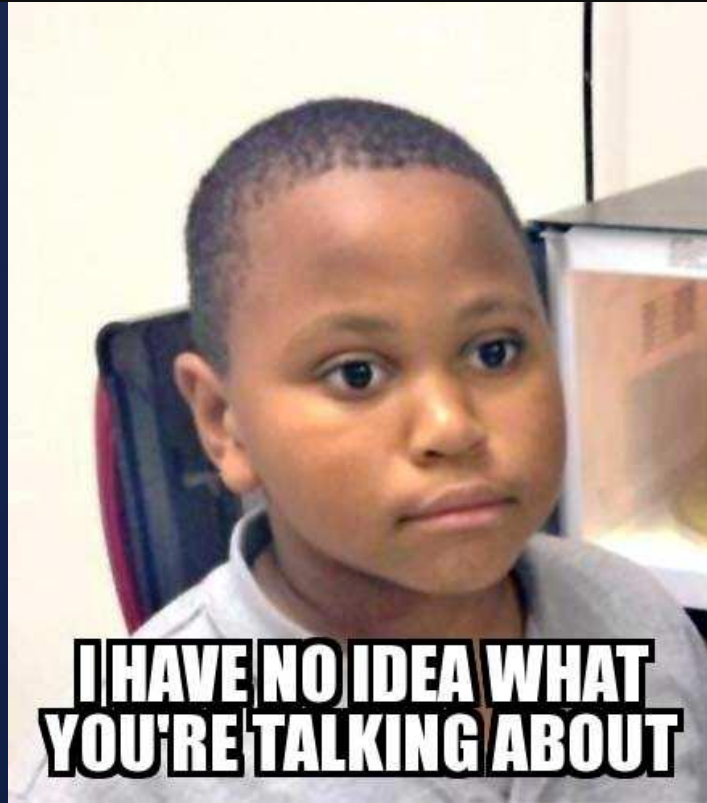
NGINX ΕΙΝΑΙ ΕΝΑΣ **WEBSERVER** Ο ΟΠΟΙΟΣ ΕΠΙΣΗΣ ΜΠΟΡΕΙ ΝΑ ΧΡΗΣΙΜΟΠΟΙΗΘΕΙ ΚΑΙ ΩΣ

- **REVERSE PROXY,**
- **LOAD BALANCER,**
- **MAIL PROXY** ΚΑΙ
- **HTTP CACHE**

The Nginx logo is displayed in a large, bold, green font. The letters are stylized with a modern, sans-serif typeface. The 'i' in 'Nginx' has a unique design with a dot that is a small circle. The 'x' is composed of two intersecting lines that form a cross shape.

DEPLOYMENT

NGINX / PROXY



DEPLOYMENT

NGINX / PROXY

“

NGINX ΕΙΝΑΙ ΕΝΑΣ **WEBSERVER** ... wdyw mate?

”

WEBSERVER είναι ο “**γραμματέας**” μιας εφαρμογής από την οποία θα ζητήσεις αρχεία

- Στείλε μου το **index.html**
- Στείλε μου το αρχείο JSON με πληροφορίες για τον **user** με **id=1**

DEPLOYMENT

SSL / TLS

ΤΟ **SSL** ΣΗΜΑΙΝΕΙ **Secure Socket Layer**

ΤΟ **TLS** ΣΗΜΑΙΝΕΙ **Transport Secure Layer** ΚΑΙ ΕΙΝΑΙ ΤΟ ΙΔΙΟ ΜΕ ΤΟ **SSL** ΑΠΛΑ ΠΙΟ **ΑΣΦΑΛΗΣ** ΚΑΙ **ΑΝΑΒΑΘΜΙΣΜΕΝΟ**

ΤΙΣ ΠΕΡΙΣΣΟΤΕΡΕΣ ΦΟΡΕΣ ΟΤΑΝ ΛΕΝΕ **SSL** ΕΝΝΟΕΙΤΑΙ ΟΤΙ ΜΙΛΑΜΕ ΓΙΑ **TLS**, ΑΦΟΥ ΤΟ **SSL** ΑΠΟΤΕΛΕΙ **DEPRECATED** ΤΕΧΝΟΛΟΓΙΑ

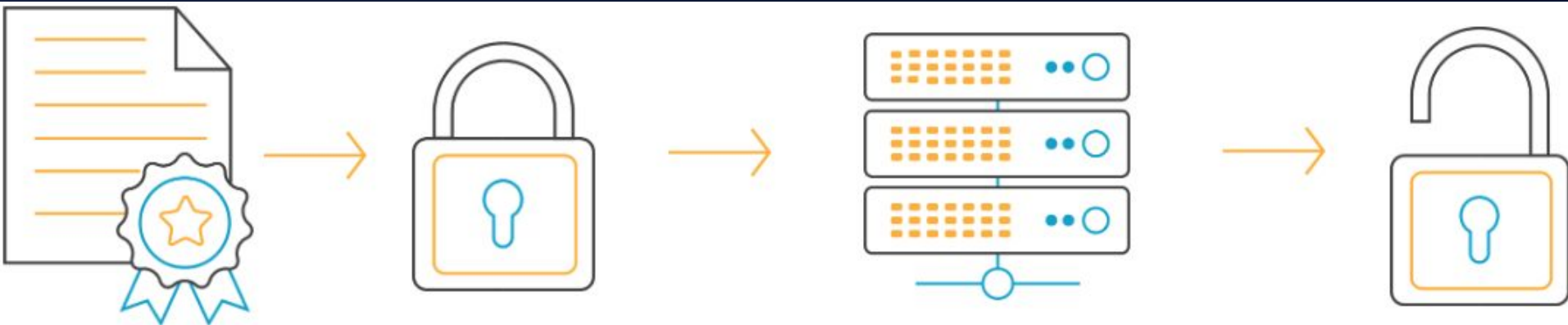
SSL ΣΗΜΑΙΝΕΙ ΟΤΙ ΧΡΗΣΙΜΟΠΟΙΕΙΣ **HTTPS** ΑΝΤΙ ΓΙΑ **HTTP**

ΤΟ **HTTPS** ΔΕΝ ΜΠΟΡΕΙ ΝΑ ΕΠΙΚΟΙΝΩΝΗΣΕΙ ΜΕ ΤΟ **HTTP**

DEPLOYMENT

SSL / TLS

ΤΟ **SSL** ΔΙΑΣΦΑΛΙΖΕΙ ΟΤΙ ΤΑ ΔΕΔΟΜΕΝΑ **ΣΤΕΛΝΟΝΤΑΙ** ΚΑΙ **ΛΑΜΒΑΝΟΝΤΑΙ** **ΚΡΥΠΤΟΓΡΑΦΗΜΕΝΑ** (ΜΕ **ΡΡΚ ΑΛΓΟΡΙΘΜΟΥΣ** ΟΠΩΣ Ο **RSA**)



DEPLOYMENT

SSL / TSL

ΛΟΓΟΙ ΓΙΑ ΝΑ ΒΑΛΕΙΣ **SSL** ΣΤΗΝ ΕΦΑΡΜΟΓΗ ΣΟΥ

- ΓΙΑΤΙ **ΔΕΝ** ΜΠΟΡΕΙΣ **ΝΑ ΕΠΙΚΟΙΝΩΝΗΣΕΙΣ** ΜΕ ΑΛΛΟΥΣ **HTTPS SERVERS**
(ΟΙ ΠΕΡΙΣΣΟΤΕΡΟΙ ΚΑΙ ΟΙ ΠΙΟ ΔΙΑΣΗΜΟΙ ΕΧΟΥΝ ΜΟΝΟ **HTTPS**)
- ΓΙΑΤΙ ΤΟ ΝΑ ΕΧΕΙΣ **HTTP ΕΙΝΑΙ ΓΙΑ ΗΛΙΘΙΟΥΣ**
- ΠΟΛΥ ΜΕΓΑΛΑ **SECURITY VULNURABILITIES !!!**
- ΧΤΙΖΕΙΣ ΕΜΠΙΣΤΕΥΤΙΚΟΤΗΤΑ ΜΕ ΤΟΝ ΠΕΛΑΤΗ/ΧΡΗΣΤΗ
- Η **GOOGLE ΒΑΘΜΟΛΟΓΕΙ ΚΑΛΥΤΕΡΑ** ΤΑ **SITES** ΜΕ **SSL**

DEPLOYMENT

SSL / TSL

ΠΡΩΤΑ ΠΡΕΠΕΙ ΝΑ ΠΑΡΕΙΣ **PUBLIC KEY** ΚΑΙ **PRIVATE KEY** (CERTIFICATE) ΑΠΟ ΕΝΑ ΠΑΡΟΧΟ **SSL/TSL CERTIFICATE**

ΔΩΡΕΑΝ - Lets Encrypt

ΤΑ **ΠΛΗΡΩΜΕΝΑ CERTIFICATES** ΔΕΝ ΣΟΥ ΔΙΝΟΥΝ ΤΙΠΟΤΑ ΑΛΛΟ ΕΚΤΟΣ ΑΠΟ ΑΠΟΖΗΜΙΩΣΗ ΣΕ ΠΕΡΙΠΤΩΣΗ **CERTIFICATE LEAK** (ΚΑΙ ΚΑΝΕΝΑ GREEN BAR - ΣΗΜΑΝΤΙΚΟ ΓΙΑ ΜΙΑ ΕΤΑΙΡΙΑ)

DEPLOYMENT

SERVER SETUP

ΠΟΥ ΒΡΙΣΚΩ SERVER ;

- Το **PC** σου μπορεί να είναι ένας **Server**
- Μπορείς να **νοικιάσεις Server**
 - **Dedicated**
 - **VPS** (Virtual Private Server)
 - **Cloud**

ΠΩΣ ΣΥΝΔΕΟΜΑΙ REMOTELY ;

SSH !

ΤΙ ΔΙΑΦΟΡΑ ΕΧΕΙ ΑΠΟ ΕΝΑ ΑΠΛΟ LINUX/WINDOWS PC;

ΚΑΜΙΑ - Μπορείς να τον χειριστείς σαν να ήταν ο καθημερινός σου υπολογιστής.

Επί μηνιαία πληρωμή.

ΒΙΒΛΙΟΓΡΑΦΙΚΕΣ ΑΝΑΦΟΡΕΣ

https://en.wikipedia.org/wiki/Transport_Layer_Security

<https://www.websecurity.symantec.com/security-topics/what-is-ssl-tls-https>

<https://tools.ietf.org/html/rfc7568>

<https://stackshare.io/stacks>

<https://martinfowler.com/articles/microservices.html>

<http://blog.christianposta.com/microservices/the-hardest-part-about-microservices-data/>