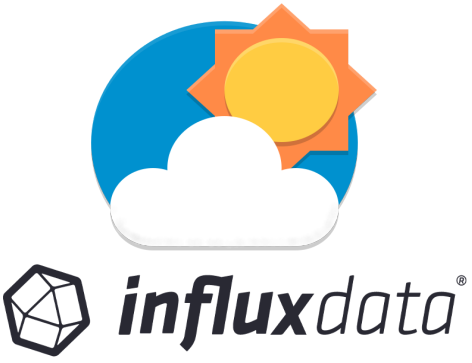
ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΔΗΜΙΟΥΡΓΙΑ ΕΦΑΡΜΟΓΗΣ ΑΝΑΛΥΣΗΣ ΚΑΙ ΠΑΡΟΥΣΙΑΣΗΣ ΜΕΤΕΩΡΟΛΟΓΙΚΩΝ ΔΕΔΟΜΕΝΩΝ ΤΟΥ OPENWEATHERMAP ΜΕ ΤΗ ΧΡΗΣΗ ΤΗΣ ΠΛΑΤΦΟΡΜΑΣ INFLUXDATA



Του φοιτητή Κωνσταντίνου Σκόκλη Αρ. Μητρώου: 144251	Επιβλέπων Ευστάθιος Αντωνίου Βαθμίδα: Associate Professor

Θεσσαλονίκη 2020

Τίτλος Π.Ε. Δημιουργία εφαρμογής ανάλυσης και παρουσίασης μετεωρολογικών δεδομένων του OpenWeatherMap, με τη χρήση της πλατφόρμας Influxdata

Κωδικός Π.Ε. 20186

Ονοματεπώνυμο φοιτητή Κωνσταντίνος Σκόκλης

Ονοματεπώνυμο εισηγητή Ευστάθιος Αντωνίου

Ημερομηνία ανάληψης Π.Ε. 16-11-2018

Ημερομηνία περάτωσης Π.Ε. 16-09-2020

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως πτυχιακή εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Πρόλογος

Στην περίοδο που διανύουμε ο όγκος των δεδομένων που μπορούμε να αντλήσουμε από το διαδίκτυο είναι αμέτρητος, ωστόσο το σημαντικότερο κομμάτι στα δεδομένα είναι η μετατροπή τους σε πληροφορία και ακόμα καλύτερα η παρουσίαση των πληροφοριών σε γραφήματα. Τα γραφήματα έχουν ως στόχο να εμφανίζουν τις πληροφορίες που συγκεντρώνονται στον χρήστη σε πιο κατανοητή μορφή και ταυτόχρονα παρέχεται η δυνατότητα να συγκρίνονται μεταξύ τους. Οι λειτουργίες και τα χαρακτηριστικά τα οποία αναφέρθηκαν παραπάνω εμπεριέχονται σε μια Single Page Application(SPA). Μία τέτοιου είδους εφαρμογή χρησιμοποιεί τις πλέον σύγχρονες τεχνολογίες και η φιλοσοφία που σχεδιάστηκε έχει σαν απόρροια το περιεχόμενο της εφαρμογής είναι δυναμικό και με έμφαση στις επιδόσεις και στο να είναι φιλικό προς τον χρήστη. Επιπλέον, αντίστοιχες εφαρμογές αναπτύσσονται και από μεγάλες εταιρείες λογισμικού σε production περιβάλλον. Τα οφέλη της Π.Ε. είναι οι γνώσεις και η εξοικείωση με μια εφαρμογή τύπου full stack. Αναλυτικότερα στη πλευρά του server υπάρχει το business logic της εφαρμογής, το οποίο δεν είναι εκτεθειμένο στον client(browser) για λόγους ασφάλειας και επιδόσεων, ύστερα ακολουθεί η βάση δεδομένων που περιέχει τις διάφορες πληροφορίες και τέλος το authentication logic, το οποίο πιστοποιεί το χρήστη ώστε να εκτελεί μόνο επιτρεπτές ενέργειες. Ακόμη υπάρχει και η πλευρά του client που αποτελείται από την παρουσίαση του U.I. και την δυναμική απεικόνιση του(render). Συμπεραίνοντας, τα οφέλη που καταγράφηκαν μπορούν να χρησιμοποιηθούν σαν εφόδια και εργαλεία για την ανάπτυξη αντιστοιχων εφαρμογών ή για την βελτίωση της παρούσας.

Περίληψη

Η παρούσα πτυχιακή εργασία έχει ως στόχο την ανάπτυξη μιας εφαρμογής για την ανάλυση και την παρουσίαση μετεωρολογικών δεδομένων με τη χρήση της InfluxDB για βάση δεδομένων και για πηγή των δεδομένων το OpenWeatherMap. Η εφαρμογή έχει σαν σκοπό να απλοποιεί την διαδικασία ανάκλησης μετεωρολογικών δεδομένων και ταυτόχρονα να οπτικοποιεί τα δεδομένα ώστε ο απλός χρήστης να μπορεί να συγκρίνει ευκολότερα. Ακόμα ο χρήστης έχει τη δυνατότητα να προσθέτει στην εφαρμογή τις πόλεις που τον ενδιαφέρουν και από την στιγμή εκείνη και έπειτα ξεκινά η συλλογή πληροφοριών. Η εφαρμογή είναι στο διαδίκτυο και ο χρήστης χρειάζεται μια σύνδεση στο διαδίκτυο. Ο server πρακτικά έχει δύο εφαρμογές η μία είναι για το front-end που εκτελείται στην δημόσια ip διεύθυνση στην πόρτα 4200 και η άλλη για το back-end στην ίδια ip αλλά στην πόρτα 3000. Ο χρήστης αλληλεπιδρά με την πρώτη και έμμεσα με τη δεύτερη. Το front-end είναι αυτο που βλέπει στο browser του και τα δεδομένα που παρουσιάζονται σε αυτό προέρχονται από το back-end. Αυτές οι δύο οντότητες συνθέτουν μια Single Page Application.

Development of an application for the analysis and presentation of OpenWeatherMap's meteorological data, using the Influxdata platform

Konstantinos Skoklis

Abstract

The present essay focuses on the development of an application for analyzing and presenting meteorological data using InfluxDB as database and OpenWeatherMap as source of information. The goal of the application is to simplify the process of recalling meteorological data and to visualize them so the user can compare them easily. On top of that, the user has the ability to add cities, which he is interested in, in the application and after that moment it starts collecting information. It is a web application and the user needs a connection to the Internet. The server has two applications, the first one runs on a public ip address on port 4200 and the other one runs on the same ip address on port 3000. The user interacts with the first one and indirectly with the second application. The front-end is what the user sees and all the data, which is presented in the UI, comes from the back-end. These two entities compose one Single Page Application.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέπων καθηγητή μου Ευστάθιο Αντωνίου, ο οποίος ήταν πάντοτε διαθέσιμος για οποιαδήποτε απορία του εξέφραζα και μου παρείχε κατευθυντήριες οδηγίες για την ανάπτυξη της εφαρμογής, ώστε να μην αποκλίνω από τον στόχο μου. Δεν θα μπορούσα να μην εκφράσω τις ευχαριστίες μου στην οικογένειά μου που στάθηκε δίπλα μου στο ψυχολογικό κομμάτι. Τέλος, προς τα μέλη της εξεταστικής επιτροπής τις θερμές μου ευχαριστίες που δέχτηκαν να αξιολογήσουν την Π.Ε. μου.

Περιεχόμενα

Κατάλογος Σχημάτων

Συντομογραφίες

Π.Ε.	Πτυχιακή Εργασία
ΔΙΠΑΕ	Διεθνές Πανεπιστήμιο Ελλάδος
IDE	Integrated Development Environment
SSH	Secure Shell
VCS	Version Control System
UI	User Interface
CSS	Cascading Style Sheets
SQL	Structured Query Language
API	Application Programming Interface
JSON	Javascript Object Notation
Π.χ.	Παραδείγματος χάρη
I/O	Input/Output
SPA	Single Page Application
cli	command line interface
URL	Uniform Resource Locator
κ.ο.κ.	και ούτω καθεξής

Κεφάλαιο 1ο: Εισαγωγή

Η παρούσα πτυχιακή εργασία έχει τον τίτλο "Δημιουργία εφαρμογής ανάλυσης και παρουσίασης μετεωρολογικών δεδομένων του OpenWeatherMap, με τη χρήση της πλατφόρμας Influxdata" και έχει σαν στόχο την διευκόλυνση του χρήστη ώστε να του παρουσιάζει και να συγκρινει τα δεδομένα που τον ενδιαφέρουν σχετικά με τις καιρικές μετρήσεις. Η πτυχιακή εργασία χωρίζεται σε επτά κεφάλαια, με το καθένα να αποτελείται από ενότητες. Το πρώτο κεφάλαιο "Εισαγωγή" διατυπώνει τον τίτλο της εργασίας όπως και τον κύριο στόχο της. Εν συνεχεία ακολουθεί μια σύντομη περιγραφή για τα κεφάλαια που έπονται. Στο δεύτερο κεφάλαιο " Εγκατάσταση προαπαιτούμενων προγραμμάτων" παρουσιάζονται όλα τα εργαλεία που χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής μαζί με τις όποιες παραμετροποιήσεις. Για κάθε εργαλείο υπάρχει και μια εισαγωγή για την κατανόηση του σκοπού. Έπειτα ακολουθεί το τρίτο κεφάλαιο "IDE & Source control" που περιγράφεται η έννοια και οι λειτουργίες που παρέχει το IDE και ύστερα πως αυτό συνδυάζεται με το source control για την διαχείριση του project. Στο τέταρτο κεφάλαιο "Ανάπτυξη UI/Front-end" υπάρχει μια αναλυτική περιγραφή για την σχεδίαση της front-end εφαρμογής και υπόλοιπων ενεργειών που υλοποιεί με το συγκεκριμένο framework. Το επόμενο κεφάλαιο "Ανάπτυξη back-end" συνεχίζει ουσιαστικά με την ίδια λογική του προηγούμενου, αλλά σε αυτό αναλύονται οι βάσεις δεδομένων που χρησιμοποιούνται και το application logic για τις λειτουργίες της εφαρμογής. Κατόπιν στο κεφάλαιο έξι "Εγχειρίδιο χρήσης" παρουσιάζεται η ορθή χρήση της εφαρμογής με τα βήματα που πρέπει να ακολουθεί ο χρήστης παρέχοντας ταυτόχρονα σχήματα. Με βάση αυτά ο χρήστης θα έρθει σε πρώτη επαφή με την εφαρμογή, ώστε αργότερα να την χρησιμοποιήσει και ο ίδιος χωρις να αντιμετωπίσει προβλήματα. Τέλος, η εργασία ολοκληρώνεται με το κεφάλαιο "Συμπεράσματα και προτάσεις βελτίωσης", όπου κατόπιν της ολοκλήρωσης της εφαρμογής διατυπώνονται τα προσωπικά συμπεράσματα που προήλθαν από τα εργαλεία που χρησιμοποιήθηκαν και καταγράφονται ιδέες για βελτίωση-εμπλουτισμο της εφαρμογής.

Κεφάλαιο 2ο: Εγκατάσταση προαπαιτούμενων προγραμμάτων

2.1 Εισαγωγή

Σε αυτό το κεφάλαιο γίνεται μια τεχνική περιγραφή σχετική με τα προγράμματα, εργαλεία που χρειάζονται πριν ξεκινήσουμε την ανάπτυξη της εφαρμογής. Επιπλέον, σε κάποια προγράμματα υπήρξαν παραμετροποιήσεις με σκοπό να λειτουργούν με τις αντίστοιχες απαιτήσεις.

2.2 InfluxDB και Telegraf

Η InfluxDB είναι μια βάση δεδομένων χρονοσειρών για να χειρίζεται πολλές εγγραφές και πολλά ερωτήματα. Ο τρόπος που έχει σχεδιαστεί παρέχει υψηλές επιδόσεις και ειδικεύεται στα δεδομένα με χρονική σήμανση(timestamps). Ένα ακόμη πλεονέκτημα αυτής της βάσης δεδομένων είναι ότι μπορεί να διαμορφωθεί ώστε να διαγράφει άχρηστα δεδομένα ύστερα από κάποιο προκαθορισμένο χρονικό διάστημα. Τέλος, προσφέρει μια γλώσσα ερωτημάτων που μοιάζει με την SQL για να υπάρχει αλληλεπίδραση με τα δεδομένα.Το telegraf είναι μια υπηρεσία από plugins που συλλέγει μετρήσεις. Τα plugins συλλέγουν μετρήσεις απευθείας από το μηχάνημα που εκτελείται η υπηρεσία διαφορετικά μπορεί να τα συλλέγει απο APIs είτε να "ακούει" για δεδομένα μέσω consumer υπηρεσιών(Kafka). Αξίζει να σημειωθεί ότι αν δεν υπάρχει κάποιο plugin στην κοινότητα της Influx τότε ο οποιοσδήποτε μπορεί να δημιουργήσει το δικό του, ώστε να ικανοποιούνται οι στόχοι του. Επιπλέον, υπάρχουν και output plugins με σκοπό να στέλνουν τις μετρήσεις σε άλλες βάσεις δεδομένων, υπηρεσίες και message queues. Την χρονική στιγμή που ξεκίνησε η ανάπτυξη της εφαρμογής έγινε εγκατάσταση των τελευταίων διαθέσιμων εκδόσεων της InfluxDB(v1.7.2) και του Telegraf(v1.9.1). Η εγκατάσταση των δύο παραπάνω εργαλειών έγινε σύμφωνα με τον οδηγό που βρίσκεται στον ιστότοπο της InfluxDB.[1],[2],[3].

2.3 Παραμετροποιήσεις σε InfluxDB και σε Telegraf

Οι παραμετροποιήσεις για την InfluxDB στη βάση δεδομένων σχετικά με το schema της, τη βάση, και του πίνακες πραγματοποιείται απο το configuration αρχείο του Telegraf. Η InfluxDB απαιτεί να τρέχει η υπηρεσία στο μηχάνημα και εφόσον υπάρχει ενεργοποιημένο το firewall να υπάρχει ο κατάλληλος κανόνας που να επιτρέπει την κίνηση. Για επαλήθευση μπορεί να εκτελεστεί η παρακάτω εντολή "systemctl status influxd" για να ελεγχθεί η κατάσταση της υπηρεσίας. Εάν η υπηρεσία εκτελείται τότε συνεχίζεται η παραμετροποίηση του Telegraf. Η κοινότητα της InfluxDB παρέχει μια μεγάλη γκάμα από plugins. Το καταλληλότερο για την συγκεκριμένη εφαρμογή είναι το "inputs.http", το οποίο συλλέγει JSON δεδομένα από API. Με τη χρήση του cli το Telegraf μαζί με συγκεκριμένα flags το --input-filter και το --output-filter δημιουργείται ένα configuration αρχείο, το οποίο πρέπει να τοποθετηθεί σε συγκεκριμένη διαδρομή. Επειδή η ανάπτυξη γίνεται σε μηχάνημα με Ubuntu 16.04 η διαδρομή που πρέπει να τοποθετηθεί είναι η εξής: "/etc/telegraf/" και εκεί τοποθετείτε το configuration αρχείο. Επίσης, σε αυτό το αρχείο ορίζουμε το interval time στα 300 δευτερόλεπτα(Σχήμα 2.1), στο output plugin ονοματίζουμε το πεδίο database με το επιθυμητό όνομα

για τη βάση δεδομένων(Σχήμα 2.2). Τέλος, εφόσον είναι γνωστή η απάντηση που στέλνει το API ορίζουμε το data_format σε JSON, τα tag_keys και τα json_string_fileds(Σχήμα 2.3).

```
[agent]
  ## Default data collection interval for all inputs
  interval = "300s"
  ## Rounds collection interval to 'interval'
  ## ie, if interval="10s" then always collect on :00, :10, :20, etc.
  round_interval = true
```

Σχήμα 2.1: Ρυθμίσεις interval time

```
# Configuration for sending metrics to InfluxDB
[[outputs.influxdb]]
  ## The full HTTP or UDP URL for your InfluxDB instance.
  ##
  ## Multiple URLs can be specified for a single cluster, only ONE of the
  ## urls will be written to each interval.
  # urls = ["unix:///var/run/influxdb.sock"]
  # urls = ["udp://127.0.0.1:8089"]
  # urls = ["http://127.0.0.1:8086"]

  ## The target database for metrics; will be created as needed.
   database = "api_response"
```

Σχήμα 2.2: Ρυθμίσεις για το όνομα της βάσης δεδομένων

```
# Read formatted metrics from one or more HTTP endpoints
[[inputs.http]]
  ## One or more URLs from which to read formatted metrics
  urls = [
          "http://api.openweathermap.org/data/2.5/weather?id=734077&units=metric&APPID=539adaf4b17a8911e564817d0d55d547",
          "http://api.openweathermap.org/data/2.5/weather?id=264371&units=metric&APPID=539adaf4b17a8911e564817d0d55d547",
          "http://api.openweathermap.org/data/2.5/weather?id=255683&units=metric&APPID=539adaf4b17a8911e564817d0d55d547",
           "http://api.openweathermap.org/data/2.5/weather?id=261743&units=metric&APPID=539adaf4b17a8911e564817d0d55d547",
           "http://api.openweathermap.org/data/2.5/weather?id=251833&units=metric&APPID=539adaf4b17a8911e564817d0d55d547",
           "http://api.openweathermap.org/data/2.5/weather?id=252664&units=metric&APPID=539adaf4b17a8911e564817d0d55d547",
           "http://api.openweathermap.org/data/2.5/weather?id=734330&units=metric&APPID=539adaf4b17a8911e564817d0d55d547",
           "http://api.openweathermap.org/data/2.5/weather?id=261604&units=metric&APPID=539adaf4b17a8911e564817d0d55d547",
           ]#add above here

  ## HTTP method
  # method = "GET"

  ## Optional HTTP headers
  # headers = {"X-Special-Header" = "Special-Value"}

  ## Optional HTTP Basic Auth Credentials
  # username = "username"
  # password = "pa$$word"

  ## Optional TLS Config
  # tls_ca = "/etc/telegraf/ca.pem"
  # tls_cert = "/etc/telegraf/cert.pem"
  # tls_key = "/etc/telegraf/key.pem"
  ## Use TLS but skip chain & host verification
  # insecure_skip_verify = false

  ## Amount of time allowed to complete the HTTP request
  # timeout = "5s"

  ## Data format to consume.
  ## Each data format has its own unique set of configuration options, read
  ## more about them here:
  ## https://github.com/influxdata/telegraf/blob/master/docs/DATA_FORMATS_INPUT.md
   data_format = "json"
```

Σχήμα 2.3: Τα επιθυμητά πεδία από την απάντηση του API

2.4 OpenWeatherMap API

Στην προηγούμενη ενότητα αναφέρθηκε ότι το Telegraf έχει την δυνατότητα να συλλέγει δεδομένα από API, οπότε η άντληση των μετεωρολογικών δεδομένων θα γίνεται από το API που μας προσφέρει δωρεάν το OpenWeatherMap και έπειτα θα αποθηκεύονται στη βάση δεδομένων. Για να ξεκινήσουν τα αιτήματα(requests) απαιτείται να έχει γίνει εγγραφή στον ιστότοπο του OpenWeatherMap(https://home.openweathermap.org/users/sign_up) για να χορηγηθεί ένα μοναδικό api key, το οποίο χρησιμοποιείται στο υπερσύνδεσμο σαν παράμετρος. Π.χ. https://samples.openweathermap.org/data/2.5/weather?q=London,uk&appid=439d4b804bc8187953eb36d2a8c26a02. Εφόσον έχει δοθεί το κλειδί, το οποίο στέλνεται στο email που έγινε η εγγραφή ύστερα από κάποια ώρα, προστίθεται στατικά μια πόλη για να επαληθευτεί ότι τα μετεωρολογικά δεδομένα αποθηκεύονται στη βάση δεδομένων. Το παραπάνω υλοποιείται ορίζοντας τον υπερσύνδεσμο μιας πόλης με το id της και το api key στο configuration αρχείο του Telegraf στο τμήμα του input plugin. Ύστερα, αφού αποθηκευτεί την αλλαγή, επανεκκινείται η υπηρεσία του Telegraf και μετά από δέκα λεπτά θα πρέπει να υπάρχουν εγγραφές με τα μετεωρολογικά δεδομένα στη InfluxDB.

```
urls = [
        "http://api.openweathermap.org/data/2.5/weather?id=734077&units=metric&APPID=539adaf4b17a8911e564817d0d55d547",
        "http://api.openweathermap.org/data/2.5/weather?id=264371&units=metric&APPID=539adaf4b17a8911e564817d0d55d547",
        "http://api.openweathermap.org/data/2.5/weather?id=255683&units=metric&APPID=539adaf4b17a8911e564817d0d55d547",
         "http://api.openweathermap.org/data/2.5/weather?id=261743&units=metric&APPID=539adaf4b17a8911e564817d0d55d547",
         "http://api.openweathermap.org/data/2.5/weather?id=251833&units=metric&APPID=539adaf4b17a8911e564817d0d55d547",
         "http://api.openweathermap.org/data/2.5/weather?id=252664&units=metric&APPID=539adaf4b17a8911e564817d0d55d547",
         "http://api.openweathermap.org/data/2.5/weather?id=734330&units=metric&APPID=539adaf4b17a8911e564817d0d55d547",
         "http://api.openweathermap.org/data/2.5/weather?id=261604&units=metric&APPID=539adaf4b17a8911e564817d0d55d547",
         ]#add above here
```

Σχήμα 2.4 OpenWeatherMap api endpoints

2.5 Node.js & Angular cli

Το Node.js είναι ανοιχτού κώδικα και ένα περιβάλλον της Javascript διαθέσιμο σε όλες τις πλατφόρμες(Windows, Unix, Mac). Μια Node.js εφαρμογή είναι μια απλή διεργασία και δεν δημιουργεί ένα καινούριο thread για κάθε αίτημα. Παρέχει ασύγχρονες συναρτήσεις που αποτρέπουν τον κώδικα να μπλοκάρεται, δηλαδή όταν εκτελειται μια I/O λειτουργία, όπως η πρόσβαση σε μια βάση δεδομένων, αντί να μπλοκάρεται και να αναμένει ο επεξεργαστής να ολοκληρωθεί, θα συνεχίσει τη λειτουργία μόλις επιστραφεί μια απάντηση από αυτήν. Αυτό το χαρακτηριστικό έχει σαν αποτέλεσμα μια Node.js εφαρμογή να επιτρέπει να διαχειρίζεται πολλαπλές ταυτόχρονες συνδέσεις χωρίς να υπάρχει το βάρος της διαχείρισης συνδέσεων, που μπορεί να οδηγήσει σε σημαντικά bugs. Τέλος, ένα πλεονέκτημα που έχει αυτή η τεχνολογία είναι πως ένας front-end developer μπορεί να αναπτύξει κώδικα στην πλευρά του server δίχως να χρειάζεται να μάθει εκ νέου μια καινούρια γλώσσα.[4] Η εγκατάσταση του Node.js έγινε από το cli(Σχήμα 2.5) και επιλέχθηκε η τελευταία διαθέσιμη έκδοση(v12.18.2). Μαζί με την εγκατάσταση του Node.js[5] γίνεται και εγκατάσταση του npm package manager, θα γίνει αναφορά σε αυτό σε άλλη ενότητα, μέσω του οποίου γίνεται και εγκατάσταση και το Angular cli.

```
# Using Ubuntu
curl -sL https://deb.nodesource.com/setup_current.x | sudo -E bash -
sudo apt-get install -y nodejs
```

Σχήμα 2.5: Οι εντολές για την εγκατάσταση του Node.js

Το Angular είναι ένα framework που ειδικεύεται στην σχεδίαση εφαρμογών και μια πλατφόρμα αναπτυξης SPA που στοχεύουν στην απόδοσή τους. Όπως και το Node.js έτσι και το Angular παρέχει σαν δυνατότητα την ανάπτυξη σε όλες τις πλατφόρμες(web, mobile, desktop). Το npm εγκαθιστά το Angular cli, το οποίο δημιουργεί τα projects μέσω του cli προγράμματος ng. Τα βήματα για εγκατάσταση βρίσκονται αναλυτικότερα στο σχήμα 2.6.

```
npm install -g @angular/cli
```

Σχήμα 2.6: Εγκατάσταση του Angular cli

2.6 MongoDB

Η MongoDB είναι μια βάση δεδομένων εγγράφων που σχεδιάστηκε για ευκολία ανάπτυξης και με δυνατότητα κλιμάκωσης. Κατηγοριοποιείται ως NoSQL βάση δεδομένων, δηλαδή μη σχεσιακή. Οι πίνακες των βάσεων δεν απαιτούν να βασίζονται σε κάποιο schema και χρησιμοποιούν JSON αντικείμενα για τις εγγραφές τους. Στην εφαρμογή επιλέχθηκε το λογισμικό του Community Server(v4.2.8). Η εγκατάσταση όπως και στα προηγούμενα εργαλεία έγινε από το cli ακολουθώντας τα βήματα όπως αναφέρονται στο [6]. Αφότου έχει ολοκληρωθεί η εγκατάσταση της MongoDB, είναι συνετό να δημιουργηθεί ένας χρήστη μέσα από το mongo shell και να οριστούν τα επιθυμητά roles(Σχήμα 2.7).

```
db.createUser({
      user: "admin",
      pwd: "password",
      roles: [
                { role: "userAdminAnyDatabase", db: "admin" },
                { role: "readWriteAnyDatabase", db: "admin" },
                { role: "dbAdminAnyDatabase",   db: "admin" }
              ]
  });
```

Σχήμα 2.7: Δημιουργία του χρήστη admin

2.7 Npm

Το Npm είναι το package manager για το Node.js. Ακόμα είναι μια δημόσια συλλογή από πακέτα/βιβλιοθήκες ανοιχτού κώδικα διαθέσιμη σε όλη τη Javascript κοινότητα. Τα δημόσια πακέτα μπορούν να εγκαθίστανται στην εφαρμογή και να είναι προσβάσιμα είτε στο Node.js είτε στο Angular. Ο καθένας μπορεί να δημοσιεύσει αυτά τα πακέτα και να επιτρέπει άλλους χρήστες να τα χρησιμοποιούν στα δικά τους projects. Σύμφωνα με την ενότητα 1.5, το Angular cli εγκαταστάθηκε μέσω του npm και έπειτα δημιουργείται το project. Με τη δημιουργία του project φτιάχνονται έμμεσα και κάποια αρχεία, ένα από αυτά είναι το package.json. Αυτό το αρχείο περιέχει οτιδήποτε πακέτα έχουν εγκατασταθεί και χρησιμοποιούνται στην εφαρμογή. Αρχικά από το Angular προσθέτονται κάποια πακέτα που είναι απαραίτητα για την λειτουργία της εφαρμογής. Καθώς το project είναι περίπλοκο η χρήση κάποιων πακέτων κρίνεται απαραίτητη. Από το cli γίνεται εγκατάσταση των παρακάτω bcryptjs, body-parser, express, highcharts, influx, jsonwebtoken, mongoose και nodemon(Σχήμα 2.8).

```
"dependencies": {
  "@angular/animations": "^8.2.2",
  "@angular/cdk": "^8.1.3",
  "@angular/common": "~8.2.0",
  "@angular/compiler": "~8.2.0",
  "@angular/core": "~8.2.0",
  "@angular/flex-layout": "^8.0.0-beta.26",
  "@angular/forms": "~8.2.0",
  "@angular/material": "^8.1.3",
  "@angular/platform-browser": "~8.2.0",
  "@angular/platform-browser-dynamic": "~8.2.0",
  "@angular/router": "~8.2.0",
  "bcryptjs": "^2.4.3",
  "body-parser": "^1.19.0",
  "express": "^4.17.1",
  "highcharts": "^8.0.0",
  "influx": "^5.5.1",
  "jsonwebtoken": "^8.5.1",
  "mongoose": "^5.6.12",
  "rxjs": "~6.4.0",
  "tslib": "^1.10.0",
  "zone.js": "~0.9.1"
},
"devDependencies": {
  "@angular-devkit/build-angular": "^0.802.2",
  "@angular/cli": "~8.2.1",
  "@angular/compiler-cli": "~8.2.0",
  "@angular/language-service": "~8.2.0",
  "@types/jasmine": "~3.3.8",
  "@types/jasminewd2": "~2.0.3",
  "@types/node": "~8.9.4",
  "codelyzer": "^5.0.0",
  "jasmine-core": "~3.4.0",
  "jasmine-spec-reporter": "~4.2.1",
  "karma": "~4.1.0",
  "karma-chrome-launcher": "~2.2.0",
  "karma-coverage-istanbul-reporter": "~2.0.1",
  "karma-jasmine": "~2.0.1",
  "karma-jasmine-html-reporter": "^1.4.0",
  "nodemon": "^1.19.1",
  "protractor": "~5.4.0",
  "ts-node": "~7.0.0",
  "tslint": "~5.15.0",
  "typescript": "~3.5.3"
}
```

Σχήμα 2.8: package.json

2.8 Επίλογος

Έχοντας ολοκληρώσει τις εγκαταστάσεις και τις παραμετροποιήσεις όλων των εργαλείων, η InfluxDB αρχίζει να καταγράφει τα μετεωρολογικά δεδομένα μέσω του Telegraf και παρακάτω θα περιγραφεί το IDE και το source control, που συμβάλλουν στην ανάπτυξη της εφαρμογής.

Κεφάλαιο 3ο: IDE & Source control

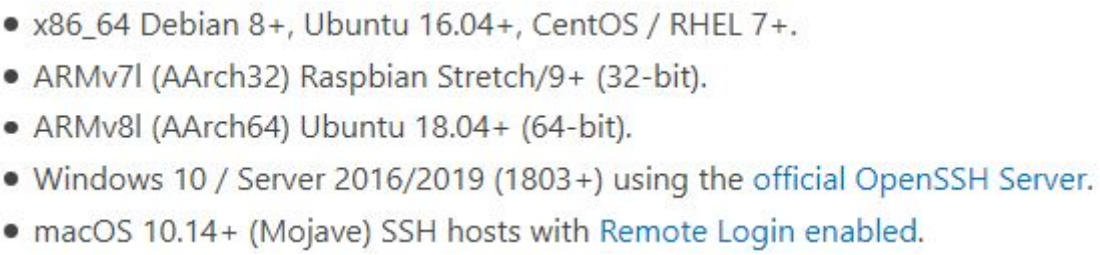
3.1 Εισαγωγή

Κατόπιν της εγκατάστασης των προγραμμάτων, για να ξεκινήσει η ανάπτυξη της εφαρμογής χρειάζεται ένα IDE για την συγγραφή του λογισμικού. Σε αυτό το κεφάλαιο θα σχολιαστεί το IDE που χρησιμοποιήθηκε και το source control .

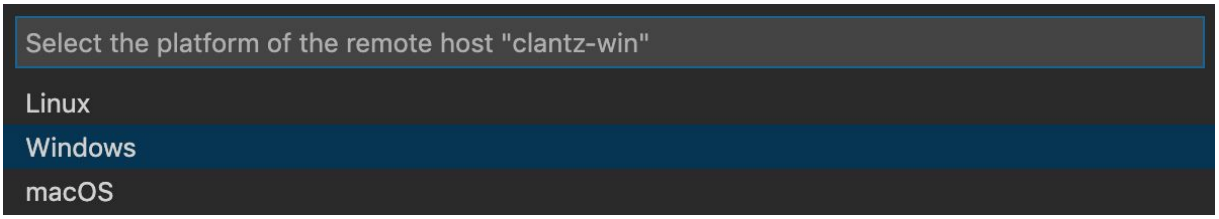
3.2 IDE

Το IDE ή το Ολοκληρωμένο Περιβάλλον Ανάπτυξης, επιτρέπει στους προγραμματιστές να ενοποιήσουν τις διάφορες πτυχές της σύνταξης ενός προγράμματος υπολογιστή. Τα IDE αυξάνουν την παραγωγικότητα συνδυάζοντας κοινές δραστηριότητες σύνταξης λογισμικού σε μία μόνο εφαρμογή, την επεξεργασία του πηγαίου κώδικα, την κατασκευή εκτελέσιμων και τον εντοπισμό σφαλμάτων. Στο IDE από την κατάληξη του κάθε αρχείου γνωρίζει τη γλώσσα σύνταξης και διαφοροποιεί χρωματικά τις λέξεις κλειδιά. Αυτό έχει σαν αποτέλεσμα να διευκολύνει την ανάγνωση του κώδικα διευκρινίζοντας οπτικά διαφορετικά στοιχεία της γλώσσας. Επίσης, όταν το IDE γνωρίζει τη γλώσσα προγραμματισμού μπορεί να προβλέψει τι πρόκειται να πληκτρολογήσετε. Για παράδειγμα στη Javascript αντί να χρειάζεται να πληκτρολογήσετε το “console.log()”, γράφοντας “log” δίνεται η επιλογή για την αυτόματη σύνταξη όλου του μηνύματος. Αυτό εξοικονομεί πατήματα πλήκτρων και χρόνο, ώστε ο προγραμματιστής να μπορεί να εστιάσει στη λογική του κώδικα που θέλει να αναπτύξει. Τα IDE παρέχουν επίσης συμβουλές κατά την διάρκεια ανάπτυξης του κώδικα για την αποφυγή σφαλμάτων πριν από τη σύνταξη. Ακόμα παρέχεται η δυνατότητα εκτέλεσης κώδικα, αλλά στην περίπτωση αυτή το build του project γίνεται στον server και εκτελείται μέσω του cli. Κανένας προγραμματιστής δεν μπορεί να αποφεύγει να γράψει με κάποιο σφάλμα στα προγράμματα. Για αυτό το λόγο τα IDE παρέχουν εργαλεία εντοπισμού σφαλμάτων που επιτρέπουν στους προγραμματιστές να εξετάζουν διαφορετικές μεταβλητές και να ελέγχουν τον κώδικά τους με σκόπιμο τρόπο[8]. Από την πληθώρα των επιλογών που υπάρχουν διαθέσιμα στο διαδίκτυο επιλέχθηκε το Visual Studio Code. Το VS Code είναι ένα ελαφρύ, δωρεάν και ισχυρό πρόγραμμα επεξεργασίας κώδικα που είναι διαθέσιμο για Windows, macOS και Linux. Έρχεται με ενσωματωμένη υποστήριξη για JavaScript, TypeScript και Node.js και διαθέτει πλούσιο οικοσύστημα επεκτάσεων για άλλες γλώσσες (όπως C ++, C #, Java, Python, PHP, Go). Με την εγκατάσταση extensions μπορεί να υποστηρίξει πρόσθετες γλώσσες, θέματα, προγράμματα εντοπισμού σφαλμάτων, εντολές και άλλα. Επειδή η ανάπτυξη έγινε απευθείας στον server απαιτείται μια επέκταση. Κατάλληση για αυτήν την περίσταση είναι το Visual Studio Code Remote - SSH extension, που επιτρέπει να ανοίγει ένας απομακρυσμένος φάκελος σε οποιονδήποτε απομακρυσμένο μηχάνημα, εικονική μηχανή ή container με SSH server που εκτελείται. Μόλις συνδεθεί ο τοπικός υπολογιστής σε έναν server, μπορεί να αλληλεπιδρά με αρχεία, φακέλους και το λειτουργικό οπουδήποτε στο απομακρυσμένο σύστημα αρχείων και να επωφεληθεί πλήρως από το σύνολο δυνατοτήτων του VS Code. Για να είναι εφικτή η επικοινωνία των δύο υπολογιστών χρειάζεται το τοπικό μηχάνημα να έχει εγκατεστημένο ένα SSH client και το απομακρυσμένο ένα SSH server. Σαν remote host υποστηρίζονται συγκεκριμένες εκδόσεις λειτουργικών(Σχήμα 3.1). Αφού έχουν εγκατασταθεί όλα όσα αναφέρθηκαν παραπάνω , στο Remote SSH host επιλέγεται το “Connect to Host” και μετά το “Add New SSH Host”. Εισάγεται το “ssh user@ip_address” και πατήστε το πλήκτρο Enter. Έπειτα επιλέγεται το config αρχείο που γραφονται οι πληροφορίες που εισήχθησαν και επιλέγεται το λειτουργικό σύστημα του server(Σχήμα 3.2). Τέλος απαιτείται ο κωδικός πρόσβασης

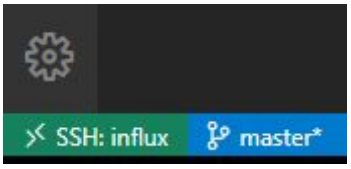
του χρήστη που δηλώθηκε παραπάνω και η σύνδεση εκτελείται. Αν όλα πήγαν καλά στην κάτω αριστερη γωνία του IDE θα εμφανίζεται η SSH σύνδεση(Σχήμα 3.3). Πλέον είναι δυνατόν να ανοίξετε οποιοδήποτε φάκελο ή χώρο εργασίας στο απομακρυσμένο μηχάνημα χρησιμοποιώντας File > Open ή File > Open Workspace όπως θα κάνατε τοπικά.



Σχήμα 3.1: Υποστηριζόμενα λειτουργικά συστήματα



Σχήμα 3.2: Επιλογή λειτουργικού

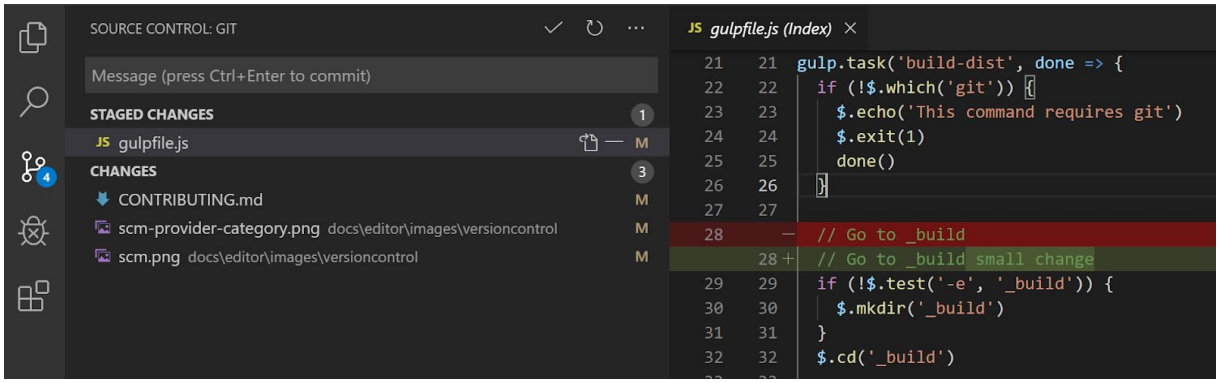


Σχήμα 3.3: Status bar για τη SSH σύνδεση

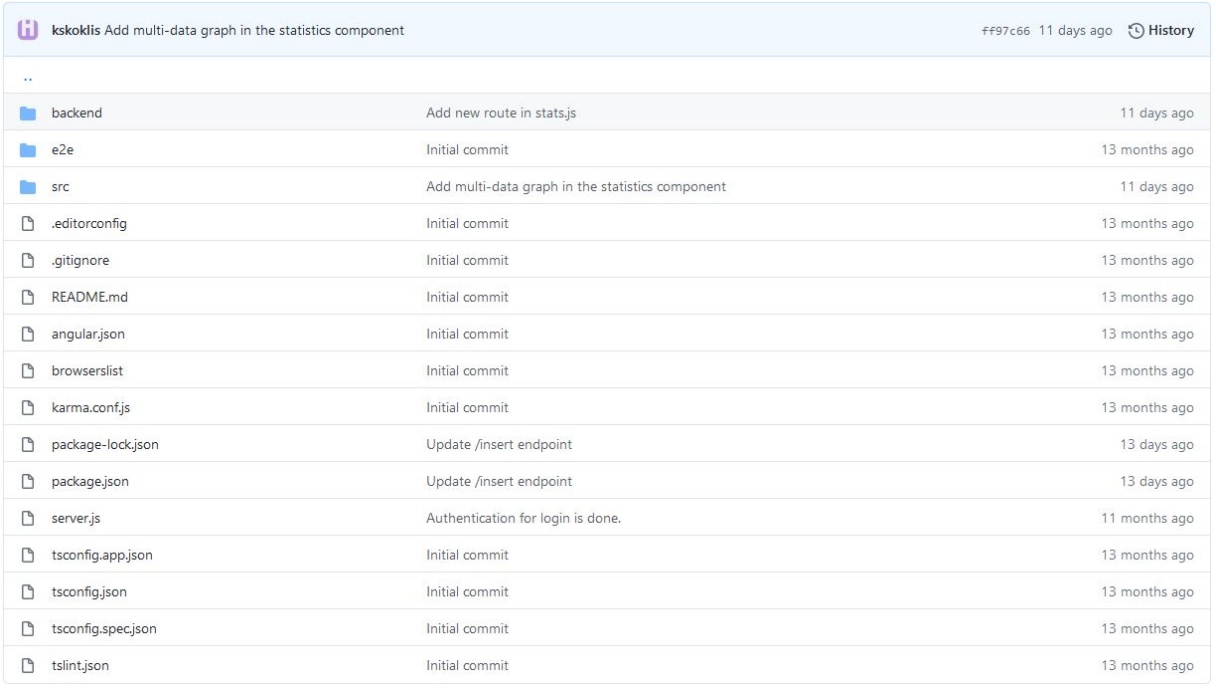
3.3 Version control

Το version control είναι ένα σύστημα που καταγράφει τις αλλαγές σε ένα αρχείο ή ένα σύνολο αρχείων στη διάρκεια του χρόνου, ώστε να μπορούν να ανακαλούνται συγκεκριμένες εκδόσεις αργότερα. Αυτό επιτρέπει να επαναφέρετε τα επιλεγμένα αρχεία σε προηγούμενη κατάσταση, να επαναφέρετε ολόκληρο το project σε προηγούμενη κατάσταση, να συγκρίνετε τις αλλαγές με την πάροδο του χρόνου, να δείτε ποιος τροποποίησε τελευταία κάτι που μπορεί να προκαλεί πρόβλημα, ποιος ανέφερε ένα ζήτημα και πότε και πολλά άλλα. Η χρήση ενός VCS γενικά σημαίνει ότι αν κάτι πάει λάθος ή χάσετε αρχεία, μπορείτε εύκολα να ανακτήσετε[9]. Το Visual Studio Code έχει ενσωματωμένο version control και περιλαμβάνει υποστήριξη του Git. Πολλοί άλλοι πάροχοι version control είναι διαθέσιμοι μέσω επεκτάσεων στο VS Code Marketplace. Το VS Code έρχεται με Git source control manager (SCM). Το εικονίδιο στα αριστερά(Σχήμα 3.4) θα δείχνει πάντα τον αριθμό των αλλαγών που υπάρχουν αυτή τη στιγμή στο project. Κάνοντας κλικ σε αυτό, θα εμφανιστούν οι λεπτομέρειες των τρέχων αλλαγών: changes, staged changes and merge changes. Κάνοντας κλικ σε κάθε αρχείο θα εμφανιστούν λεπτομερώς οι αλλαγές του[10]. Το project αυτό είναι συνδεδεμένο σε κάποιο remote branch στο Github(Σχήμα 3.5). Το VS Code προσφέρει χρήσιμες ενέργειες για push,

pull και sync για το branch που αναπτύσσεται η εφαρμογή, ώστε οι αλλαγές να κρατούνται σε ένα κεντρικό server και να μπορεί όποιος έχει πρόσβαση σε αυτό να λαμβάνει αλλά και να προσφέρει αλλαγές.



Σχήμα 3.4: Το UI για το Source control



Σχήμα 3.5: Github repo

3.4 Επίλογος

Ανακεφαλαιόνοντας, έγινε αναφορά στο IDE που χρησιμοποιήθηκε για την ανάπτυξη της εφαρμογής παρουσίασης μετεωρολογικών δεδομένων και για το source control ώστε να υπάρχει καταγραφή των αλλαγών και ασφάλεια για να μην συμβεί κάποια απώλεια στα αρχεία. Αφού όλα είναι εγκατεστημένα και ρυθμισμένα όπως προτείνεται, στα επόμενα κεφάλαιο θα γίνει ανάλυση για το front-end και το back-end της εφαρμογης

Κεφάλαιο 4ο: Ανάπτυξη UI/Front-end

4.1 Εισαγωγή

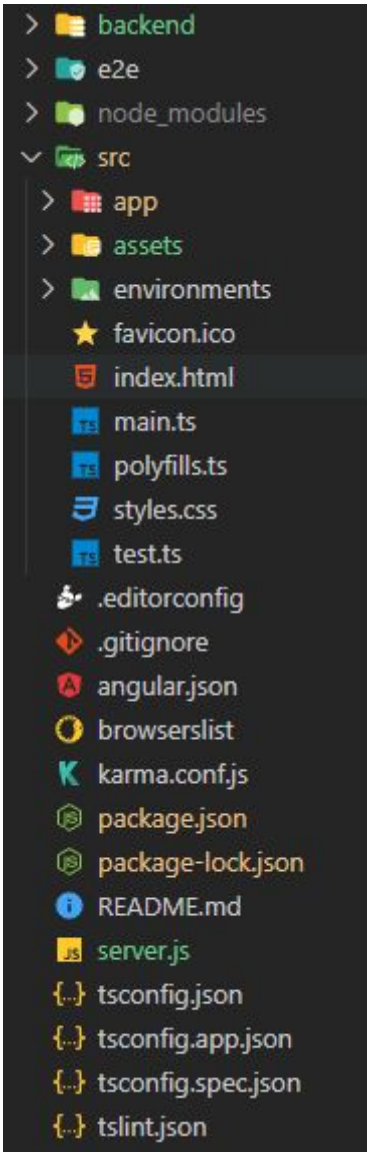
Σε αυτό το κεφάλαιο θα γίνει αναφορά στο UI της εφαρμογής και στα επιμέρους components που το αποτελούν.

4.2 Αρχική δομή του project

Ύστερα από τη δημιουργία του project από το Angular cli(Σχήμα 4.1) όπου στο my-first-project αντικαθίσταται με το weather-influx και μεταβαίνουμε στο φάκελο του και να αρχίσει η ανάπτυξη του βασικού σκελετού του UI(Σχήμα 4.2).

```
ng new my-first-project
cd my-first-project
ng serve
```

Σχήμα 4.1: Εντολές για τη δημιουργία νέου project

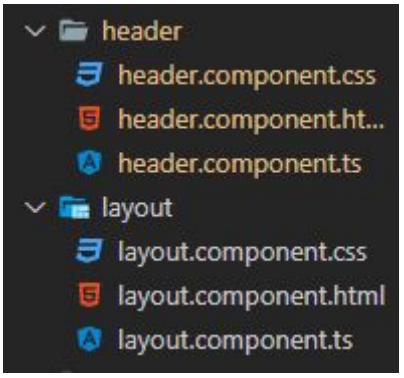


Σχήμα 4.2: Αρχική δομή του project

Μέσα στο φάκελο που ονομάζεται app θα γίνει η δημιουργία των components και οτιδήποτε περιέχεται ήδη μέσα στα αρχεία app.component.html, app.component.css, app.component.ts πρέπει να αφαιρεθεί διότι από default υπάρχει έτοιμος κώδικας. Όπως είχε αναφερθεί και στον πρόλογο, η εφαρμογή είναι SPA οπότε ο βασικός κορμός θα παραμένει ως έχει και θα αλλάζει το περιεχόμενο ανάλογα με την σελίδα και την αυθεντικοποίηση του χρήστη. Τα components στο Angular ελέγχουν ένα μέρος της οθόνης που ονομάζεται view. Τα components καθορίζουν τη λογική τους μέσα στην κλάση τους, η οποία αλληλεπιδρά με το view μέσω ενός API που αποτελείται από properties και μεθόδους. Μια τέτοια κλάση περιέχει κάποια metadata για να την χαρακτηρίζουν σαν component class όπως είναι το decorator @Component(Σχήμα 4.3). Αφού είχε προηγηθεί έρευνα και μελέτη, η δημιουργία ενός καινούριου component γίνεται απο το cli με την παρακάτω εντολή “ng generate component component_name”. Δημιουργούνται δύο components για τον βασικό κορμό, το ένα είναι το header που εμπεριέχει τις σελίδες και το δεύτερο, το layout, εμπεριέχει το κύριο περιεχόμενο και θα αλλάζει δυναμικά ανάλογα την σελίδα που περιηγείται ο χρήστης(Σχήμα 4.4).

```
@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
```

Σχήμα 4.3: Component metadata



Σχήμα 4.4: Τα δύο βασικά components του UI

Κάθε component όταν δημιουργείται με βάση το όνομα που του ορίζεται ονομάζεται αντίστοιχα και το selector του, με το οποίο είναι δυνατόν να το χρησιμοποιείται στο project. Ακόμα το app.component ονομάζεται app-root και τοποθετείτε στο index.html αρχείο(Σχήμα 4.5) όπου ο μηχανισμός του Angular κάνει render τα components στον περιηγητή(browser) του χρήστη.

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Weather Influx</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
  <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

Σχήμα 4.5: Index.html

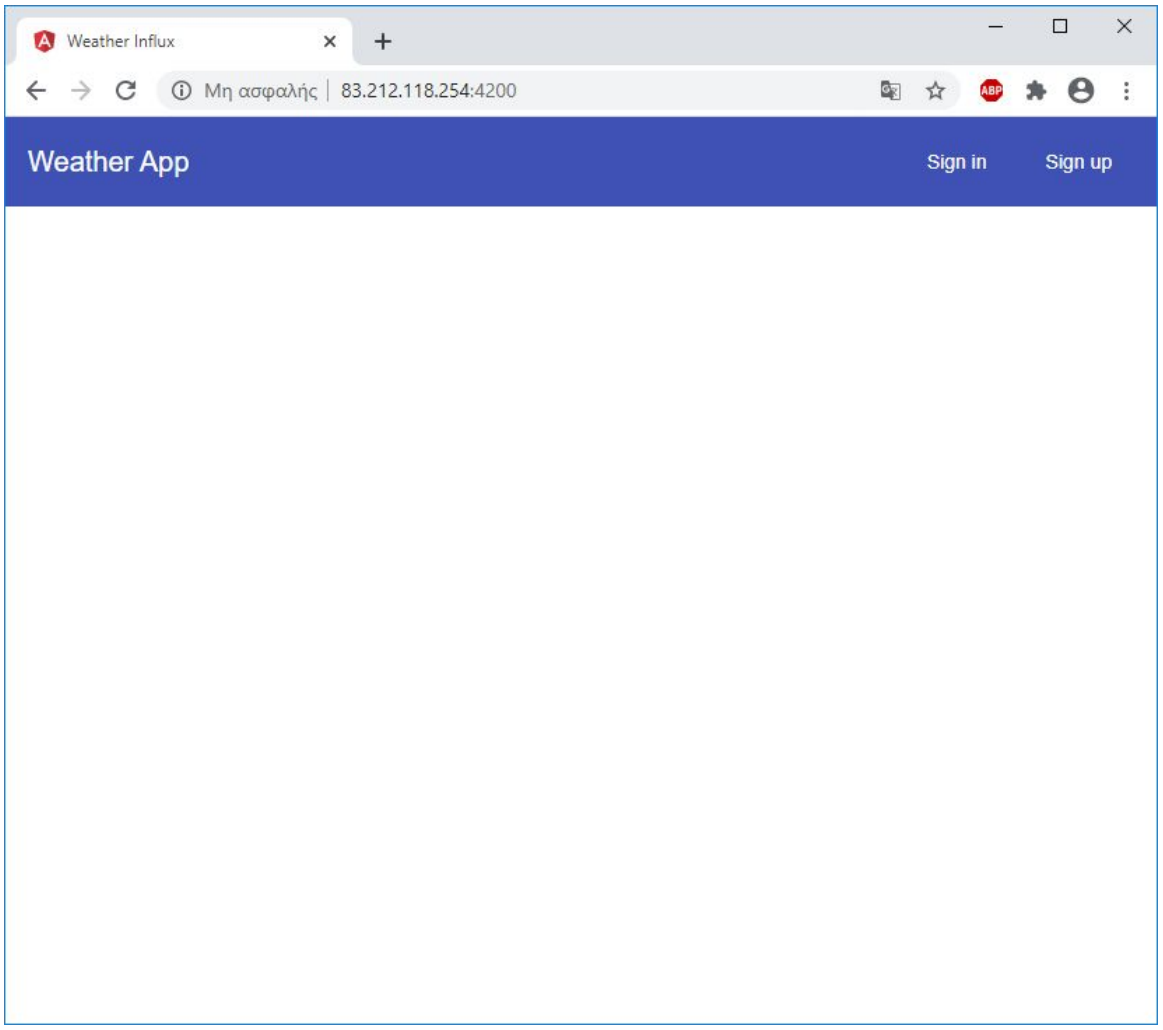
Για να είναι ορατά τα δυο components που δημιουργήθηκαν παραπάνω πρέπει να προστεθούν με το αντίστοιχο selector στο app.component.html(Σχήμα 4.6). Το tag με όνομα router-outlet λειτουργεί σαν

μπαλαντέρ και κάθε φορά κάνει render την αντίστοιχη σελίδα που επιλέγει ο χρήστης. Με αυτό τον τρόπο θα απεικονίζονται τα components όταν το project γίνει build.

```
<app-layout>
  <app-header></app-header>
  <main>
    <router-outlet></router-outlet>
  </main>
</app-layout>
```

Σχήμα 4.6: app.component.html

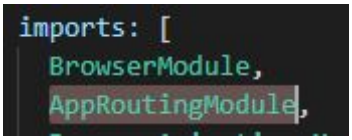
Τέλος, για να γίνει build το project εκτελείται την εντολή “ng server --host ip_address” όπου στο flag host γράφεται η ip διεύθυνση του μηχανήματος(Σχήμα 4.7).



Σχήμα 4.7: Αρχικό layout

4.3 Angular router

Σε μια SPA εφαρμογή αλλάζει το περιεχόμενο που βλέπει ο χρήστης εμφανίζοντας ή αποκρύπτοντας κάποια κομμάτια που ανταποκρίνονται σε συγκεκριμένα components αντί να ζητάει ο browser από τον server κάθε φορά μια καινούρια σελίδα. Καθώς ο χρήστης εκτελεί διάφορες λειτουργίες στην εφαρμογή, πρέπει να μετακινείται ανάμεσα στα διαφορετικά views που έχουν οριστεί. Σχετικά με τα components θα γίνει ανάλυση στην επόμενη ενότητα. Για την διαχείριση της πλοήγησης από το ένα view στο άλλο γίνεται χρήση του Angular router[11]. Το router επιτρέπει την πλοήγηση διερμηνεύοντας το URL του browser για να αλλάζει το view. Κατά τη δημιουργία του project επιλέγεται η επιλογή του routing να είναι ενεργοποιημένη όπως επίσης υπάρχουν επιλογές για CSS ή κάποιο CSS processor. Για να χρησιμοποιηθεί πρακτικά το Angular router απαιτούνται τουλάχιστον δύο components για να είναι δυνατό να πλοηγείται ο χρήστης από το ένα στο άλλο. Αρχικά πρέπει το αρχείο app-routing.module.ts να εισαχθεί στο app.module.ts στον πίνακα imports(Σχήμα 4.8). Έπειτα στο πρωτο αρχείο στον πίνακα Routes ορίζονται τα routes του UI(Σχήμα 4.9), αυτά τα routes διαφέρουν από τα back-end routes που θα περιγραφούν στο επόμενο κεφάλαιο. Τέλος, στα components όπου στο html αρχείο περιέχουν κάποιο “a” tag δηλώνεται το routerLink που είναι το ίδιο με το path property του πίνακα Routes. Να σημειωθεί ότι η σειρά των routes είναι πολύ σημαντική διότι χρησιμοποιείται η στρατηγική “first-match”. Για παράδειγμα αν σε ένα path υπάρχει το wildcard που ταιριαζει με όλα, τότε αυτό πρέπει να δηλωθεί τελευταίο για να ελέγχονται τα υπόλοιπα πρώτα και αν δεν ταιριάξει κανένα τότε επιλέγεται αυτό. Το default route ορίζεται ως κενό string και σαν 404 page το “**”, που πρέπει να είναι τελευταίο στον πίνακα που αναφέρθηκε παραπάνω. Τα paths της Angular εφαρμογής είναι πέντε και αντιστοιχούν σε ένα component το καθένα. Παρακάτω θα γίνει ανάλυση για αυτά.



Σχήμα 4.8: Ο πίνακας imports

```
const routes: Routes = [
  { path: '', component: WeatherInfoComponent},
  { path: 'login', component: LoginComponent},
  { path: 'register', component: RegisterComponent},
  { path: 'stats', component: StatisticsComponent, canActivate: [AuthGuard]},
  { path: 'search', component: SearchCitiesComponent, canActivate: [AuthGuard]},
];
```

Σχήμα 4.9: Ο πίνακας Routes

4.4 Υλοποίηση των σελίδων

Για την υλοποίηση των υπόλοιπων έγινε χρήση του Angular Material. Για να γίνει εγκατάσταση πρέπει να βρισκόμαστε στο φάκελο του project και στο cli εκτελείται την παρακάτω εντολή “ng add @angular/material”. Τώρα είναι διαθέσιμα κάποια βασικά components όπως τα buttons αλλά με

διαφορετικό style και υπάρχουν κάποια πιο σύνθετα components. Κάθε material component έχει το δικό του module και για να προστεθεί στο project χρειάζεται να γίνει import στο αρχείο app.module.ts(Σχήμα 4.10) και να τα εισάγετε στον πίνακα imports στο προαναφερθέν αρχείο(Σχήμα 4.11). Αξίζει να σημειωθεί πως κάθε component του project γίνεται και αυτό import(Σχήμα 4.12) αλλά προτίθεται στον πίνακα declarations(Σχήμα 4.13).

```
import { MatToolbarModule, MatButtonModule, MatCardModule, MatInputModule, MatIconModule,
         MatProgressSpinnerModule, MatFormFieldModule, MatSelectModule, MatMenuModule,
         MatDatepickerModule, MatNativeDateModule, MatGridListModule, MatDividerModule,
         MatRadioModule, MatSnackBarModule, MatDialogModule } from '@angular/material/';
```

Σχήμα 4.10: Imported modules from angular material

```
imports: [
  BrowserModule,
  AppRoutingModule,
  BrowserAnimationsModule,
  FlexLayoutModule,
  MatToolbarModule,
  MatButtonModule,
  MatCardModule,
  MatInputModule,
  MatIconModule,
  MatProgressSpinnerModule,
  FormsModule,
  HttpClientModule,
  MatFormFieldModule,
  ReactiveFormsModule,
  MatSelectModule,
  MatMenuModule,
  MatDatepickerModule,
  MatNativeDateModule,
  MatGridListModule,
  MatDividerModule,
  MatRadioModule,
  MatSnackBarModule,
  MatDialogModule,
],
```

Σχήμα 4.11: Ο πίνακας imports

```
import { LayoutComponent } from './layout/layout.component';
import { LoginComponent } from './auth/login/login.component';
import { RegisterComponent } from './auth/register/register.component';
import { HeaderComponent } from './header/header.component';
import { StatisticsComponent } from './statistics/statistics.component';
import { AuthInterceptor } from "./auth/auth-interceptor";
import { WeatherInfoComponent } from './weather-info/weather-info.component';
import { SearchCitiesComponent } from './search-cities/search-cities.component';
import { ErrorInterceptor } from './error-interceptor';
import { ErrorComponent } from './error/error.component';
```

Σχήμα 4.12: Angular components

```
declarations: [
  AppComponent,
  LayoutComponent,
  LoginComponent,
  RegisterComponent,
  HeaderComponent,
  StatisticsComponent,
  WeatherInfoComponent,
  SearchCitiesComponent,
  ErrorComponent,
],
```

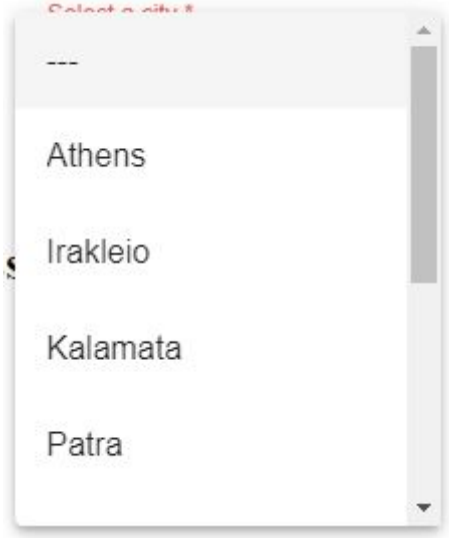
Σχήμα 4.13: Ο πίνακας declarations

4.4.1 Αρχική σελίδα

Στην αρχική σελίδα υπάρχουν οι πιο πρόσφατες πληροφορίες σχετικά με τον καιρό(Σχήμα 4.14), όπως θερμοκρασία, κατάσταση καιρού και σχετικό εικονίδιο του καιρού, σε όσες πόλεις έχουμε διαθέσιμες στην εφαρμογή(Σχήμα 4.15). Οι μετεωρολογικές πληροφορίες ανανεώνονται κάθε φορά που αλλάζει η πόλη στο mat-select με το interpolation[12]. Το interpolation επιτρέπει να ενσωματώνονται δεδομένα μεταξύ html tags π.χ. <h3>Current customer: {{ currentCustomer }}</h3> ή <img src="{{itemImageUrl}}"> . Έχει προκαθοριστεί ότι το interpolation χρησιμοποιεί σαν οριοθέτη τα "{{" και "}}". Το κείμενο που υπάρχει ανάμεσα από τους οριοθέτες μπορεί να αναφέρεται σε property του component, ως συνέπεια αλλάζοντας την τιμή του property γίνεται και η αλλαγή στο html στοιχείο. Το κείμενο ενδιάμεσα στις αγκύλες είναι ένα template expression, το οποίο το Angular πρώτα το αξιολογεί και έπειτα το μετατρέπει σε string. Τα template expressions χρησιμοποιούνται και για μαθηματικούς υπολογισμούς και μπορούν να καλέσουν μεθόδους από το component που βρίσκεται. Το mat-select έχει γίνει bind με το event "selectiononChange", το οποίο ενεργοποιείται κάθε φορά που αλλάζει η τιμή του select και τότε εκτελείται ένα get request όπου αντλεί από την InfluxDB τα πιο πρόσφατα δεδομένα και τα ορίζει στα template expressions, με αποτέλεσμα το περιεχόμενο να είναι δυναμικό και να ενημερώνεται σε κάθε πόλη. Σε επόμενη ενότητα θα γίνει ανάλυση της μεθοδολογίας που χρησιμοποιείται για να εκτελούνται τα http requests με τον server.

```
<mat-spinner *ngIf="isLoading"></mat-spinner>
<span *ngIf="!isLoading">
  <div class="city">
    <h1>Select a city</h1>
    <mat-form-field>
      <mat-select placeholder="Select a city" [formControl]="cityControl" (selectionChange)="onChange($event)" required>
        <mat-option>---</mat-option>
        <mat-option *ngFor="let city of cities" value="{{city.id}}">
          {{ city.name }}
        </mat-option>
      </mat-select>
    </mat-form-field>
  </div>
  <div class="location">
    <h2 class="location-timezone">{{ city }}, {{ country }}</h2>
    <img class="location-icon" src="{{ imgUrl }}">
  </div>
  <div class="temperature">
    <div class="degree-section">
      <h2 class="temperature-degree"> {{ temperature }}</h2>
      <p>°C</p>
    </div>
    <div class="temperature-description">{{ description }}</div>
  </div>
</span>
```

Σχήμα 4.14: Αρχική σελίδα



Σχήμα 4.15: Διαθέσιμες πόλεις

4.4.2 Σελίδα login και register

Αυτές οι δύο σελίδες ακολουθούν παρόμοια λογική στην σχεδίασή τους και στην λειτουργία τους. Σχετικά με την σχεδίαση, οι σελίδες αποτελούνται από μια φόρμα για σύνδεση και για εγγραφή αντίστοιχα(Σχήμα 4.16, Σχήμα 4.17). Ταυτόχρονα περιέχει και ένα mat-spinner, το οποίο το χρησιμοποιείται από το Angular material σαν έτοιμο component, για να γίνει η διεπαφή του χρήστη πιο φιλική μέχρι να ολοκληρωθεί η επιβεβαίωση των στοιχείων που έχουν εισαχθεί. Όλα τα πεδία που περιέχονται στις φόρμες επαληθεύονται και σε περίπτωση λάθος εμφανίζεται αντίστοιχο μήνυμα προς τον χρήστη, ώστε να το διορθώσει και να προχωρήσει έπειτα σε υποβολή της φόρμας. Τα δεδομένα

της φόρμας, εφόσον είναι έγκυρα, αποστέλλονται στον server μέσω ενός post request και στην περίπτωση της σύνδεσης(login) αν είναι όλα σωστά ο χρήστης μεταβαίνει στην αρχική σελίδα. Στην περίπτωση της εγγραφής(register) εάν όλα είναι σωστά ο χρήστης μεταβαίνει στην σελίδα της σύνδεσης. Τα εικονίδια που βρίσκονται δίπλα στο κείμενο των κουμπιών είναι Material icons και έχουν γίνει import στο αρχείο app.module.ts(Σχήμα 4.18, 4.19). Υπάρχει μια μεγάλη ποικιλία εικονιδίων, τα οποία χωρίζονται σε κατηγορίες και διευκολύνουν τον developer από την σχεδίαση custom εικονιδίων.

```
<div fxLayoutAlign="center center" >
  <mat-card>
    <mat-spinner *ngIf="isLoading"></mat-spinner>
    <mat-card-content>
      <form (submit)="onLogin(loginForm)" #loginForm="ngForm" *ngIf="!isLoading">
        <mat-form-field  >
          <mat-label>Username</mat-label>
          <input  matInput  placeholder="Username" ngModel #uname="ngModel" name="username" required>
          <mat-error *ngIf="uname.invalid">Please enter a valid email.</mat-error>
        </mat-form-field>
        <mat-form-field  >
          <mat-label>Password</mat-label>
          <input  matInput type="password"  placeholder="Password" ngModel #passwd="ngModel" name="password" required>
          <mat-error *ngIf="passwd.invalid">Please enter a valid password.</mat-error>
        </mat-form-field>
        <div fxLayoutAlign="center center">
          <button mat-raised-button fxLayout="row" *ngIf="!isLoading" color="primary">LOGIN  <mat-icon>send</mat-icon></button>
        </div>
      </form>
    </mat-card-content>
  </mat-card>
</div>
```

Σχήμα 4.16: Σελίδα login

```
<div fxLayoutAlign="center center">
  <mat-card>
    <mat-spinner *ngIf="isLoading"></mat-spinner>
    <mat-card-content>
      <form class="my-form" (submit)="onSignup(signupForm)" #signupForm="ngForm" *ngIf="!isLoading">
        <mat-form-field class="full-width">
          <mat-label>Username</mat-label>
          <input  matInput type="text" placeholder="Username" name="username" ngModel #uname="ngModel" required>
          <mat-error *ngIf="uname.invalid">Please enter a valid username.</mat-error>
        </mat-form-field>
        <mat-form-field class="full-width">
          <mat-label>Email</mat-label>
          <input  matInput type="email" placeholder="Email"  name="email" ngModel #mail="ngModel" email>
        </mat-form-field>
        <mat-form-field class="full-width">
          <mat-label>Password</mat-label>
          <input  matInput type="password" placeholder="Password"  name="password" ngModel #password="ngModel" required>
          <mat-error *ngIf="password.invalid">Please enter a valid password.</mat-error>
        </mat-form-field>
        <mat-form-field class="full-width">
          <mat-label>Retype Password</mat-label>
          <input  matInput type="password" placeholder="Retype Password"  name="retypepassword" ngModel #passwordretype="ngModel" required>
          <mat-error *ngIf="passwordretype.invalid">Please enter a valid password.</mat-error>
        </mat-form-field>
        <div fxLayoutAlign="center center">
          <button mat-raised-button fxLayout="row" type="submit" *ngIf="!isLoading" color="primary">REGISTER  <mat-icon>person_add</mat-icon></button>
        </div>
      </form>
    </mat-card-content>
  </mat-card>
</div>
```

Σχήμα 4.17: Σελίδα register

Σχήμα 4.18: Material icon στο login component

Σχήμα 4.19: Material icon στο register component

4.4.3 Σελιδα Statistics

Στην τρέχων σελίδα στο πάνω μέρος υπάρχει ένα mat-grid-list, το οποίο προσφέρει μια ευέλικτη δομή στην διεπαφή και χωρίζεται σε κελιά. Ανάμεσα από τα elements υπάρχουν mat-dividers με σκοπό να διατηρείται η λογική συνοχή και για να είναι ξεκάθαρη η διεπαφή με τις ενέργειες που κάνει ο χρήστης. Ακόμα στο δεύτερο mat-grid-tile υπάρχουν δυο mat-datepickers, και αυτά είναι έτοιμα components από το Angular material. Σε αυτή τη σελίδα όταν έχουν επιλέξει τα κατάλληλα στοιχεία τότε δημιουργείται ένα γράφημα ανάλογο των δεδομένων που έχουν εισαχθεί(Σχήμα 4.20,4.21). Σε περίπτωση που κάποιο πεδίο είναι εσφαλμένο τότε εμφανίζεται μήνυμα λάθους προς τον χρήστη για να προβεί σε διόρθωση. Το γράφημα δημιουργείται με τη χρήση του πακέτου highcharts που έγινε εγκατάσταση από το npm. Να αναφερθεί ότι πριν επιλεγεί αυτό για την εφαρμογή, είχε προηγηθεί μελέτη και έρευνα για αναζήτηση πακέτων/βιβλιοθηκών που ειδικεύονται σε γραφήματα. Αυτά τα πακέτα που προτείνονται για τη δικιά μας εφαρμογή, δηλαδή να παρουσιάζουν δεδομένα που σχετίζονται με χρονοσειρές, είναι το plotlyjs, το rickshaw και το highcharts. Να αναφερθεί ότι πριν επιλεγεί ένα συγκεκριμένο πακέτο είχαν δοκιμαστεί και δυο ακόμα Angular wrappers του highcharts, αλλά αντιμετωπίστηκαν προβλήματα στην ανάπτυξη και δεν κατέστει δυνατόν να βρεθεί κάποια επίλυση σε αυτά. Αρχικά στη σελίδα statistics κάνουμε import τη βιβλιοθήκη για να υπάρχει πρόσβαση σε αυτήν και στις μεθόδους της. Το πακέτο highcharts με τη μέθοδο chart δημιουργεί ένα γράφημα με βάση τις επιλογές που έχουν ορισθεί. Οι επιλογές του γραφήματος είναι ένα JSON αντικείμενο(Σχήμα 4.22), στο οποίο ορίζουμε τον τίτλο του, τον τύπο του, τον άξονα X, του άξονα Y και τα δεδομένα του άξονα Y. Στον άξονα X αποδίδουμε τις ημερομηνίες στο εύρος που έχουν επιλεγεί. Αντίστοιχα στον άξονα Y αποδίδουμε τα μετεωρολογικά δεδομένα. Υπάρχουν περαιτέρω παραμετροποιήσεις που μπορούν να συμβούν στο γράφημα φυσικά με τα αντίστοιχα πεδία στις επιλογές.

```
<mat-grid-list cols="1" rowHeight="6rem">
  <!-- First form -->
  <form (submit)="onFirstSubmit(firstForm)" #firstForm="ngForm">

    <mat-grid-tile>
      <mat-form-field class="towns">
        <mat-select placeholder="Select a city"  name="cityId" [(ngModel)]="cityId" required >
          <mat-option>---</mat-option>
          <mat-option *ngFor="let city of cities" value="{{city.id}}">
            {{ city.name }}
          </mat-option>
        </mat-select>

      </mat-form-field>

      <mat-divider [vertical]="true"></mat-divider>

        <mat-radio-group aria-label="Select an option"  name="aggragateFunction" required [(ngModel)]="aggragateFunction">
          <mat-radio-button value="Max" >Max</mat-radio-button>
          <mat-radio-button value="Mean" >Mean</mat-radio-button>
          <mat-radio-button value="Min" >Min</mat-radio-button>
          <mat-radio-button value="All" >All</mat-radio-button>
          <br>
          <label class="error-label" *ngIf="invalidRadioButton">Please choose an option! </label>
        </mat-radio-group>

      <mat-divider [vertical]="true"></mat-divider>

      <mat-form-field>
      <mat-select placeholder="Select a measurement" name="size" [(ngModel)]="size" required>
        <mat-option>---</mat-option>
        <mat-option value="main_temp">Temperature</mat-option>
        <mat-option value="main_humidity">Humidity</mat-option>
        <mat-option value="wind_speed">Wind</mat-option>
        <mat-option value="clouds_all">Clouds</mat-option>
      </mat-select>
      </mat-form-field>
      <mat-divider [vertical]="true"></mat-divider>
      <h4>Predifined time values:</h4>
      <div class="example-button-row">
        <button mat-raised-button color="primary" type="submit" (click)='getTimePeriod($event)'>Last hour</button>
        <button mat-raised-button color="primary" type="submit" (click)='getTimePeriod($event)'>Last day</button>
        <button mat-raised-button color="primary" type="submit" (click)='getTimePeriod($event)'>Last week</button>
        <button mat-raised-button color="primary" type="submit" (click)='getTimePeriod($event)'>Last month</button>
      </div>
    </mat-grid-tile>
  </form>
```

Σχήμα 4.20: Πρώτη φόρμα στη σελίδα statistics

```
  <mat-grid-tile>
    <label id="example-radio-group-label">Custom time values:</label>  <br>
    <form  #secondForm="ngForm">
      <mat-form-field class="calendar">
        <input matInput  [matDatepicker]="picker" required [(ngModel)]="startDate" name="startDate"   placeholder="Choose a date">
        <mat-datepicker-toggle matSuffix [for]="picker"></mat-datepicker-toggle>
        <mat-datepicker #picker startView="year" [startAt]="startDatee"></mat-datepicker>
      </mat-form-field>

      <mat-form-field class="calendar">
        <input matInput [matDatepicker]="picker2" required [(ngModel)]="endDate" name="endDate" placeholder="Choose a date">
        <mat-datepicker-toggle matSuffix [for]="picker2"></mat-datepicker-toggle>
        <mat-datepicker #picker2 startView="year" [startAt]="startDate"></mat-datepicker>
      </mat-form-field>
      <label class="error-label" *ngIf="invalidDatePicker">Please choose a date for each calendar!</label>
    </form>
    <button mat-raised-button color="primary"  (click)="onSecondSubmit(firstForm,secondForm)">Apply</button>
  </mat-grid-tile>

</mat-grid-list>

<div id="container"></div>
```

Σχήμα 4.21: Δεύτερη φόρμα στη σελίδα statistics

```
this.chartOptions = {
  chart: {
    type: 'spline'
  },
  title: {
    text: graphTitle
  },
  xAxis: {
    categories: this.timeArray
  },
  yAxis: {
    title:{
      text: graphText,
      rotation: 0
    }
  },
  series: [
    {
      name: this.stats[0].city,
      data: this.statsArray
    }
  ]
};
```

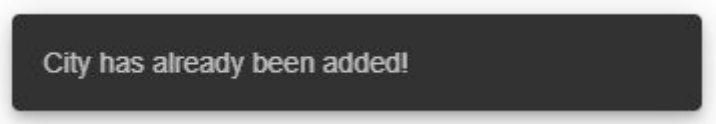
Σχήμα 4.22: Επιλογές γραφήματος

4.4.4 Σελίδα search

Σε αυτήν την σελίδα υπάρχει ένα input το οποίο εμπεριέχεται σε μια φόρμα η οποία το επαληθεύει για κάθε τιμή που εισάγεται από τον χρήστη. Επίσης στο κουμπί υπάρχει και ένα Material εικονίδιο, όπως είχε αναφερθεί σε προηγούμενη ενότητα. Επειδή η λειτουργία αυτής της σελίδας απαιτεί λίγο χρόνο για να ολοκληρώσει τις ενέργειες, έχει προστεθεί ένα matSnackBar προγραμματιστικά μέσα στο search-cities.component.ts από το Angular material. Κάθε φορά που ο χρήστης κλικάρει το κουμπί Add εκτελείται ένα post request και μόλις το front-end λάβει την απάντηση από τον server τότε εμφανίζεται στην διεπαφή το matSnackBar, το οποίο εμπεριέχει το αντίστοιχο μήνυμα(Σχήμα 4.24).

```
<mat-spinner *ngIf="isLoading"></mat-spinner>
<span *ngIf="!isLoading" >
  <form (submit)="addCity(addForm)" #addForm="ngForm">

    <mat-form-field class="example-full-width">
      <input matInput placeholder="Insert a new city"  name="cityName" ngModel #cityName="ngModel" required>
      <mat-error *ngIf="cityName.invalid">Please enter a city name.</mat-error>
    </mat-form-field>

    <button mat-raised-button color="primary">Add <mat-icon>add_circle_outline</mat-icon></button>
  </form>
</span>
```

Σχήμα 4.23: Σελιδα search



Σχήμα 4.24: Το mat-SnackBar με το μήνυμα

4.4.5 Σελίδα Error

Το error component(Σχήμα 4.25) δεν αποτελεί μια σελίδα, διότι είναι το περιεχόμενο ενός Mat-dialog που παρέχεται από το Angular material και εμφανίζει στον χρήστη κάποιο μήνυμα που του γνωστοποιεί τι ακριβώς πήγε λάθος σχετικά με την ενέργεια που εκτέλεσε. Το μήνυμα αυτό αλλάζει ανάλογα με το σφάλμα που συμβαίνει και προέρχεται από την απάντηση του back-end. Ακόμα σε αυτό το component η εφαρμογή δεν το κανει render ούτε από το selector ούτε από το router όπως συμβαίνει με τα υπόλοιπα. Στα metadata της κλάσης ορίζεται το templateUrl που ορίζεται το html αρχείο και το styleUrls που δηλώνεται το css αρχείο(Σχήμα 4.26). Το component γίνεται render προγραμματιστικά και το mat-dialog εμφανίζεται με τα περιεχόμενα του component. Γι' αυτό το λόγο το component πρέπει δηλωθεί εκτός από τον πίνακα declarations του NgModule και στον πίνακα entryComponents(Σχήμα 4.27). Αυτό για το Angular σημαίνει ότι ένω δεν το βλέπει στην εφαρμογή με αυτή την προσθήκη είναι δυνατόν να το χρησιμοποιήσει, διαφορετικά παράγεται σφάλμα. Έπειτα όταν είναι να εμφανιστεί στην διεπαφή το mat-dialog τότε σαν όρισμα περνάει το component μαζί με το μήνυμα λάθους στη μέθοδο open.

```
<h1 mat-dialog-title>An error occured!</h1>
<div mat-dialog-content>
    <p class="mat-body-1">{{ data.message }}</p>
</div>
<div mat-dialog-actions>
    <button mat-button mat-dialog-close>Ok</button>
</div>
```

Σχημα 4.25: Σελίδα Error

```
@Component({
  templateUrl: './error.component.html',
  styleUrls: ['./error.component.css']
})
```

Σχήμα 4.26: Error component metadata

```
entryComponents: [ErrorComponent]
```

Σχήμα 4.27: Ο πίνακας entryComponents στο αρχείο app.module.ts

4.5 Observable, Observers

Τα Observables παρέχουν υποστήριξη για να περνάνε μηνύματα ανάμεσα στην εφαρμογή[13],[14]. Χρησιμοποιούνται συχνά στο Angular και ενδείκνυνται για την διαχείριση events, ασύγχρονες λειτουργίες και την διαχείριση πολλαπλών τιμών. Το observer pattern είναι ένα design pattern λογισμικού στο οποίο ένα αντικείμενο που καλείται subject, διατηρεί μια λίστα από εξαρτώμενους, τους observers και τους ειδοποιεί αυτόματα μόλις υπάρξει αλλαγή της κατάστασης. Αυτό το pattern μοιάζει αρκετά με το publish. Στα Observables δηλώνεται μια συνάρτηση για να δημοσιεύονται-γνωστοποιούνται νέες τιμές, αλλά δεν εκτελείται μέχρις ότου κάποιος consumer(καταναλωτής) εγγραφεί(subscribe) σε αυτήν. Ο εγγεγραμμένος καταναλωτής τότε λαμβάνει ειδοποιήσεις ως την ολοκληρωση της μεθόδου ή όταν καταργηθεί η εγγραφή(unsubscribe). Το observable μπορεί να παραδώσει πολλαπλές τιμές οποιουδήποτε τύπου είτε σύγχρονα είτε ασύγχρονα. Επειδή όλη η λογική του observable είναι έτοιμη, ο κώδικας της εφαρμογής χρειάζεται μόνο για την εγγραφή σε τιμές του καταναλωτή και για την ολοκλήρωση ή την κατάργηση της εγγραφής. Εξαιτίας αυτών των πλεονεκτημάτων χρησιμοποιούνται στο Angular και προτείνονται για την ανάπτυξη εφαρμογών. Για παράδειγμα, δημιουργείτε ένα αντικείμενο τύπου observable το οποίο ορίζει μια μέθοδο εγγραφής(subscriber function). Αυτή η μέθοδος εκτελείται όταν ένας observer καλέσει τη μέθοδο subscribe. Η μέθοδος εγγραφής ορίζει πως θα αποκτά τιμές ή μηνύματα ώστε να γνωστοποιηθούν. Όταν το observer χρησιμοποιήσει τη μέθοδο subscribe αυτή επιστρέφει ένα αντικείμενο τύπου Subscription το οποίο έχει μια μέθοδο unsubscribe για να σταματάει να λαμβάνει ειδοποιήσεις.Το Observer interface τρία notification type το next, το error και το complete. Το next είναι απαραίτητο για τη λήψη ειδοποιήσεων και ενεργοποιείται κάθε φορά που υπάρχει μια νέα τιμή. Το error είναι προαιρετικό και σταματάει την εκτέλεση του observable μόλις συμβεί κάποιο λάθος. Τέλος το complete είναι και αυτό προαιρετικό και ενεργοποιείται όταν η εκτέλεση του observable ολοκληρωθεί. Το HttpClient από το Angular επιστρέφει ένα observable από τις HTTP μεθόδους. Για παράδειγμα το “http.get(“/api”)” γυρίζει ένα αντικείμενο τύπου observable με κάποια πλεονεκτήματα. Τα Observables δεν μεταλλάσσουν την απάντηση από το server αλλά με διάφορες μεθόδους μετατρέπουμε τις τιμές στη μορφή που χρειάζεται. Τα http requests μπορούν να ακυρωθούν με τη συνάρτηση unsubscribe(κατάργηση εγγραφής) και να επιστρέφουν την πρόοδο από τα events. Τέλος αν κάποιο αίτημα αποτύχει μπορεί πολύ εύκολα να δοκιμαστεί εκ νέου.

4.6 Angular services

Το Angular service είναι μια κλάση που εκτελεί κάποιες συγκεκριμένες ενέργειες και είναι διαθέσιμη στα components του project παρέχοντας σε αυτά εύκολη πρόσβαση σε δεδομένα[15]. Το Angular διαχωρίζει τα components απο τα services για να αυξήσει επαναχρησιμοποίηση του κώδικα και το modularity, δηλαδή η λειτουργία του καθενός να είναι ανεξάρτητη και εξειδικευμένη σε ένα ζήτημα μόνο. Η λειτουργικότητα των components σχετίζεται με το view(UI) και διαχωρίζεται από άλλους είδους επεξεργασίες με σκοπό να είναι λιτά και αποδοτικά. Το component αναθέτει κάποιες εργασίες στα services, όπως να φέρνει δεδομένα από τον server, να επαληθεύει τα στοιχεία που εισάγει ο χρήστης. Για να είναι διαθέσιμο ένα service στα components χρειάζεται να γίνεται inject(Dependency injection). Το dependency injection είναι συνδεδεμένο με το Angular framework και χρησιμοποιείται οπουδήποτε για να παρέχει σε νέα components τα services ή ο,τι χρειάζονται. Για να οριστεί μια κλάση ως service πρέπει να χρησιμοποιηθεί το decorator "@Injectable" μαζί με τα metadata, ώστε να επιτρέψει το Angular να την κάνει inject σε component σαν dependency. Παρόμοιο decorator προστίθεται και σε κλάσεις όταν υπάρχει κάποιο dependency. Σε κάθε component του project υπάρχει ένα service τα οποία κατά κύριο λόγο φέρνει δεδομένα από τον server μαζί με άλλες λειτουργίες.

4.6.1 Επικοινωνία με το backend χρησιμοποιώντας το Http

Οι περισσότερες εφαρμογές χρειάζεται να επικοινωνούν με κάποιον server με το πρωτόκολλο Http, ώστε να κατεβάζουν (download) ή να ανεβάζουν(upload) δεδομένα και να υπάρχει πρόσβαση σε άλλα backend services[16]. Το Angular παρέχει ένα απλό Http client για τις εφαρμογές, το HttpClient[17] service και γίνεται import απο το "@angular/common/http" στο αρχείο app.module.ts(Σχήμα 4.28). Το HttpClient υλοποιεί όλες τις μεθόδους για να εκτελεί Http requests, οι οποίες είναι οι εξής: request, delete, get, head, jsonp, options, patch, post, και put. Αυτές οι μέθοδοι είναι generic και δίνεται η δυνατότητα να οριστεί τι τύπου απάντηση θα λάβει από τον server. Κατά κύριο λόγο θα χρησιμοποιηθούν οι μέθοδοι get και post. Η μέθοδος HTTPClient.get() φέρνει δεδομένα από τον server ασύγχρονα, αφού πρώτα στείλει ένα HTTP request και επιστρέφει ένα Observable, το οποίο εκπέμπει-μεταδίδει τα αποτελέσματα μόλις έρθει η απάντηση. Αυτή η μέθοδος δέχεται 2 παραμέτρους, το endpoint URL και ένα αντικείμενο με options για να παραμετροποιηθεί το request. Το endpoint είναι το URL στο οποιο μπορεί να έχει πρόσβαση μια client εφαρμογή. Πιο συγκεκριμένα τώρα, στη μέθοδο get ορίζουμε τι τύπο response θα στείλει ο server, μετά βάζουμε το αντίστοιχο endpoint URL αλλά χωρίς κάποιο option(Σχήμα 4.29). Το Angular επιστρέφει ένα Observable που δημοσιεύει τα νέα δεδομένα μόλις σταλει μία απάντηση(response) από τον server, όπου για να τα "ακούμε" χρησιμοποιούμε τη μέθοδο subscribe. Καλώντας τη μέθοδο subscribe σε ένα Observable αντικείμενο περνάει ένας observer για να λαμβάνει τις καινούριες ειδοποιήσεις. Με την αντίστοιχη φιλοσοφία λειτουργεί και η μέθοδος HTTPClient.post() με μια σημαντική διαφορά. Οι παράμετροι που δέχεται η μέθοδος post είναι το endpoint URL, το request body το οποίο περιέχει τα δεδομένα που θα σταλούν στον server και για την εφαρμογή είναι JSON αντικείμενα και τέλος τα options για το HTTP request(Σχήμα 4.30).

```
import { HttpClientModule, HTTP_INTERCEPTORS } from '@angular/common/http';
```

Σχήμα 4.28: Εισαγωγή του HttpClientModule στο app.module.ts

```
getCities(){
  this.http.get<{message: string, result: Cities[]}>('http://83.212.118.254:3000/api/weather/towns')
  .subscribe((citiesData) => {
    this.cities = citiesData.result;
    this.citiesUpdated.next([...this.cities]);
  });
}
```

Σχήμα 4.29: Παράδειγμα HTTPClient.get

```
createUser(username: string, email: string, password: string ){
  const authdata: AuthData = {username: username, email: email, password: password};
  this.http.post("http://83.212.118.254:3000/api/user/signup", authdata)
  .subscribe(response => {
    this.router.navigate(['/login']);
    console.log(response);
  }, error => {
    this.authStatusListener.next(false);
  });
}
```

Σχήμα 4.30: Παράδειγμα HTTPClient.post

4.6.2 Service αρχικής σελίδας

Σε προηγούμενη ενότητα έγινε ανάλυση της αρχικής σελίδας σχετικά με το UI, τη σχεδίαση του component και την λειτουργία της. Τα δεδομένα της σελίδας γίνονται fetch μέσω του weather.service.ts, το οποίο βρίσκεται στον ίδιο φάκελο με το αντίστοιχο component. Σε αυτή την κλάση για να θεωρείται σαν service υπάρχει το κατάλληλο decorator(Σχήμα 4.31) και ταυτόχρονα έχουν γίνει import τα απαραίτητα πακέτα ή μοντέλα(Σχήμα 4.32). Θα γίνει αναφορά σε μοντέλα σε επόμενη ενότητα. Υπάρχουν τρεις μέθοδοι που εκτελούν και οι τρεις ένα get request. Η getWeatherInfo() χρησιμοποιεί την HttpClient.get() στο endpoint 'http://83.212.118.254:3000/api/weather/inf' και σαν generic type ορίζεται το αντικείμενο "{message: string, result: any}". Έπειτα αφού έχουν έρθει τα μετεωρολογικά δεδομένα, πριν αποθηκευτούν με τη χρήση της μεθόδου subscribe, χρειάζεται να τροποποιηθούν διότι δεν συμβαδίζουν τα ονόματα των πεδίων της απάντησης με αυτά του Weather model. Για να γίνει η μετατροπή των δεδομένων χρησιμοποιείται η μέθοδος pipe, που είναι πολύ απλή, δέχεται σαν είσοδο τιμές και τις επιστρέφει διαμορφωμένες[18]. Μπορεί να μετατρέπει strings, νομίσματα, ημερομηνίες και άλλα δεδομένα. Το Angular επίσης παρέχει έτοιμες μεθόδους για τυπικές μετατροπές όπως DatePipe, UpperCasePipe, LowerCasePipe, CurrencyPipe, DecimalPipe και PercentPipe. Αφού τα μετεωρολογικά δεδομένα είναι μέσα σε πίνακα, εκτός από την χρήση της μεθόδου pipe χρησιμοποιείται και η μέθοδος map, η οποία καλείται από έναν πίνακα και επιστρέφει πίσω έναν νέο πίνακα διαμορφωμένο. Στην δική μας περίπτωση ο πίνακας περιέχει JSON αντικείμενα και στην μέθοδο map ορίζουμε τα πεδία του κάθε αντικειμένου με τα ονόματα που έχει το model που έχουμε δημιουργήσει. Ύστερα από την μετατροπή, κάνουμε subscribe στα νέα δεδομένα τα αποθηκεύουμε σε ένα private πίνακα και τέλος μέσω ενός

Subject δημοσιεύουμε τις αλλαγές(Σχήμα 4.33). Το Subject στο Angular είναι ένας ειδικός τύπος Observable που του επιτρέπει να δημοσιεύει τιμές σε πολλούς Observers. Ενώ τα απλά Observables δημοσιεύουν τις τιμές σε Observers που έχουν κάνει subscribe. Αυτές οι τιμές για να περάσουν στο component χρησιμοποιείται μια μέθοδος που επιστρέφει αυτό το αντικείμενο Subject με μια ειδική μέθοδο την asObservable ώστε να μην είναι δυνατόν κάποιος εκτός του service να δημοσιεύει νέες τιμές(Σχήμα 4.34). Στην ίδια κλάση υπάρχει μία αντίστοιχη μέθοδος αλλα δέχεται σαν παράμετρος το id της πόλης και επιστρέφει τα πιο πρόσφατα μετεωρολογικά της δεδομένα. Τέλος, υπάρχουν δύο συναρτήσεις η μία επιστρέφει από τον server τις πόλεις που έχουν αποθηκευτεί στην InfluxDB σε έναν πίνακα Cities και μετά ακολουθείται η ίδια μεθοδολογία με την προηγούμενη μέθοδο, η δεύτερη επιστρέφει το Subject με τη συνάρτηση asObservable.

```
@Injectable({
  providedIn: 'root'
})
```

Σχήμα 4.31: Το @Injectable decorator

```
import { Injectable } from "@angular/core";
import { HttpClient } from "@angular/common/http";
import { Subject } from "rxjs";
import { map } from "rxjs/operators";
import { Weather } from "./weather.model";
import { Cities } from "./cities.model";
```

Σχήμα 4.32: Service imports

```
this.http.get<{message: string, result: any}>('http://83.212.118.254:3000/api/weather/inf')
.pipe(map((weatherData) => {
  return weatherData.result.map(data => {
    return {
      id: data.id,
      city: data.name,
      country: data.sys_country,
      icon: data.weather_0_icon,
      temperature: data.last,
      description: data.weather_0_description
    };
  });
}))
.subscribe(transformedWeatherData => {
  this.weather = transformedWeatherData;
  this.weatherUpdated.next([...this.weather]);
});
```

Σχήμα 4.33: Μέθοδος getWeatherInfo

```
private weatherUpdated = new Subject<Weather[]>();
getWeatherUpdateListener(){
  return this.weatherUpdated.asObservable();
}
```

Σχήμα 4.34: Το Subject και η μέθοδος που το επιστρέφει

4.6.3 Auth service

Το Auth service περιέχει όλη τη λογική για την αυθεντικοποίηση του χρήστη της εφαρμογής στο front-end και χρησιμοποιείται σε μεγάλο βαθμό από τα components register και login. Αρχικά, αυτή η κλάση περιέχει τη μέθοδο createUser(Σχήμα 4.35), η οποία δέχεται σαν παραμέτρους τα username, email, password. Και για αυτήν την κλάση έχει δημιουργηθεί ένα μοντέλο που ονομάζεται auth-data.model.ts. Οι παράμετροι ορίζουν τις τιμές στο αντικείμενο auth-data και ύστερα πραγματοποιείται το post request για τη δημιουργία ενός νέου χρήστη. Όπως έχει αναφερθεί γίνεται subscribe στο HTTPClient.post για να λάβει την απάντηση από τον server και με τη χρήση του service Router που παρέχεται από το Angular ο χρήστης πλοηγείται στη σελίδα login. Σε περίπτωση λάθους ένα αντικείμενο τύπου Subject δημοσιεύει ότι συνέβη κάποιο λάθος και έπειτα εμφανίζεται κατάλληλο μήνυμα. Η μέθοδος login(Σχήμα 4.36) δέχεται δύο παραμέτρους το username και το password έπειτα δημιουργεί ένα αντικείμενο τύπου auth-data και εκτελεί ένα post request με generic type το "{token: string, expiresIn: number}". Γίνεται subscribe σε αυτό το Observable που επιστρέφει το request και αναμένουμε το token, αν το λάβει το service τότε εκτελούμε τη μέθοδο setAuthTimer, η οποία μετά από μία ώρα κάνει αποσύνδεση τον χρήστη από την εφαρμογή. Επιπλέον μια boolean μεταβλητή δέχεται τη τιμή true, διότι ο χρήστης είναι συνδεδεμένος και εισήγαγε τα σωστά στοιχεία. Ένας listener εκπέμπει ότι το authentication status είναι true και δημιουργείται η ημερομηνία και η ώρα που λήγει το token. Τέλος, το token και η ώρα που λήγει αυτό αποθηκεύονται στο local storage ο χρήστης μεταβαίνει στην αρχική σελίδα. Υπάρχει η μέθοδος logout που όσες μεταβλητές ορίσαμε στη login τώρα ορίζονται σε null, false, το local storage και το timer της μιας ώρας καθαρίζονται και ο χρήστης μεταβαίνει πάλι στην αρχική σελίδα. Επιπλέον έχουμε τη συνάρτηση autoAuthUser(Σχήμα 4.37) που ελέγχει αν υπάρχουν οι μεταβλητές token και expirationDate και υπολογίζει τη διαφορά από το expirationDate με την τρέχων ώρα. Αν η διαφορά είναι θετικός αριθμός τότε ορίζεται το token, το isAuthenticated, καλεί τη μέθοδο setAuthTimer που θέτει ένα χρονικό όριο και ενημερώνει με το Subject τους listeners. Υπάρχουν τρεις μέθοδοι που αποθηκεύουν, ορίζουν και διαγράφουν τις τιμές στο local storage, που έχουν δηλωθεί ως private και καλούνται από τις προηγούμενες που αναλύσαμε. Τέλος για τα πεδία token, isAuthenticated και το Subject authStatusListener υλοποιούνται οι getters τους.

```
createUser(username: string, email: string, password: string ){
  const authdata: AuthData = {username: username, email: email, password: password};
  this.http.post("http://83.212.118.254:3000/api/user/signup", authdata)
  .subscribe(response => {
    this.router.navigate(['/login']);
    console.log(response);
  }, error => {
    this.authStatusListener.next(false);
  });
}
```

Σχήμα 4.35: Μέθοδος createUser

```
login(username: string, password: string){
  const authdata: AuthData = {username: username, email: null, password: password};
  this.http.post<{token: string, expiresIn: number}>("http://83.212.118.254:3000/api/user/login", authdata)
  .subscribe(response => {
    console.log(response.token);
    const token = response.token;
    this.token = token;
    if (token){
      const expiresInDuration = response.expiresIn;
      this.setAuthTimer(expiresInDuration);
      this.isAuthenticated = true;
      this.authStatusListener.next(true);
      const now = new Date();
      const expirationDate = new Date(now.getTime() + expiresInDuration * 1000);
      console.log(expirationDate);

      this.saveAuthData(token, expirationDate);
      this.router.navigate(['/']);
    }
  }, error => {
    this.authStatusListener.next(false);
  });
}
```

Σχήμα 4.36: Μέθοδος login

```
autoAuthUser() {
  const authInformation = this.getAuthData();
  if (!authInformation) {
    return;
  }
  const now = new Date();
  const expiresIn = authInformation.expirationDate.getTime() - now.getTime();
  if (expiresIn > 0) {
    this.token = authInformation.token;
    this.isAuthenticated = true;
    this.setAuthTimer(expiresIn / 1000);
    this.authStatusListener.next(true);
  }
}
```

Σχήμα 4.37: Μέθοδος authAuthUser

4.6.4 Statistics service

Το statistics service σχετίζεται με τη ομώνυμη σελίδα και περιέχει τη λογική για τα γραφήματα. Η μέθοδος getWeatherStats(Σχήμα 4.38) που έχει έξι παραμέτρους τις fun, size, city, start, end και period. Εκτελείται ένα get request με generic type το"{message: string, result: any}" και στο endpoint URL περνάνε όλες οι παράμετροι. Στην απάντηση που λαμβάνει το front-end τα δεδομένα ποικίλουν ανάλογα με τη συνάρτηση που έχει επιλέξει ο χρήστης, δηλαδή max, mean και min. Οι αλλαγές των δεδομένων στην μορφή που έχει οριστεί το μοντέλο statistics γίνονται με τη μέθοδο pipe. Ακόμη επειδή υπάρχουν πίνακες η χρήση της συνάρτησης map είναι καθοριστική. Αφού οριστεί σωστά ο πίνακας αντικειμένων με τις μετατροπές τότε στον πίνακα statistics περνάνε αυτά και με το Subject statsUpdated γίνεται push ο πίνακας statistics για να εκπέμπουν οι αλλαγές. Επίσης από τα αρχικά δεδομένα δημιουργείται ένας πίνακας που περιέχει το όνομα της πόλης ώστε αργότερα να εμφανίζεται στο γράφημα. Τέλος, υπάρχει μια συνάρτηση που επιστρέφει το Subject statsUpdated με τη μέθοδο asObservable για να μην μπορούν να δημοσιεύουν νέες τιμές από άλλες κλάσεις.

```
getWeatherStats(fun, size, city, start, end, period){
  this.http.get<{message: string, result: any}>("http://83.212.118.254:3000/api/stats/" + fun + "/" + size + "/" + city + "/" + start + "/" + end + "/" + period)
    .pipe(map((statsData) => {
      let cityArr = statsData.result[1].map(data => {
        return data.name  ;
      });

      return statsData.result[0].map(stat => {
        if (stat.mean) {
          return {
            time: stat.time,
            statistic_value: stat.mean,
            city: cityArr[0]
          };
        }
        else if (stat.min) {
          return {
            time: stat.time,
            statistic_value: stat.min,
            city: cityArr[0]
          };
        }
        else if (stat.max) {
          return {
            time: stat.time,
            statistic_value: stat.max,
            city: cityArr[0]
          };
        }
      });
    }))
    .subscribe(transformedStatsData => {
      console.log(transformedStatsData);
      this.statistics = transformedStatsData;
      this.statsUpdated.next([...this.statistics])
    });

}
```

Σχήμα 4.38: Μέθοδος getWeatherStats

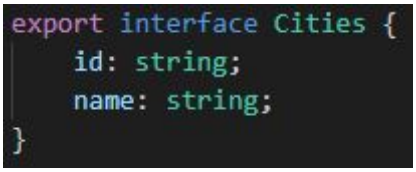
4.6.5 Search-cities service

Η λογική αυτού του service είναι να βρίσκει το id της πόλης που εισήγαγε ο χρήστης. Εκτελείται ένα HttpClient.get με generic type να ορίζεται το “<message: string, cities: Cities[]>”. Εκτελούμε ένα post request στο backend μας και σαν request body στέλνουμε σε JSON μορφή το όνομα της πόλης. Έπειτα κάνουμε subscribe για να ενημερωθούμε για το Observable που επιστρέφεται και σε αυτή την περίπτωση κάνουμε push το μήνυμα του backend ώστε να ενημερωθεί ο χρήστης για την ενέργεια που εκτέλεσε. Αυτή ήταν η μέθοδος setCitiesIds(Σχήμα 4.39) και υπάρχει μια μέθοδος η CityUpdateListener που επιστρέφει το Subject που αναφέραμε παραπάνω με την asObservable συνάρτηση. Η μέθοδος getUserCities εκτελεί ένα get request με generic το “<{message: string, cities: Cities[]}>” σε ένα endpoint URL που επιστρέφει τις πόλεις που έχει εισάγει ο χρήστης που είναι συνδεδεμένος. Οι πόλεις αποθηκεύονται σε έναν πίνακα Cities, είναι το ίδιο μοντέλο που χρησιμοποιείτε σε προηγούμενο service και μετά ενημερώνονται για τις αλλαγές με τη συνάρτηση next του Subject citiesUpdated. Τέλος, όπως αναφέραμε με τη CityUpdateListener έχουν πρόσβαση οι υπόλοιπες κλάσεις στις πόλεις που επιστρέφει ο server.

Σχήμα 4.39: Μέθοδος setCitiesIds

4.7 Models

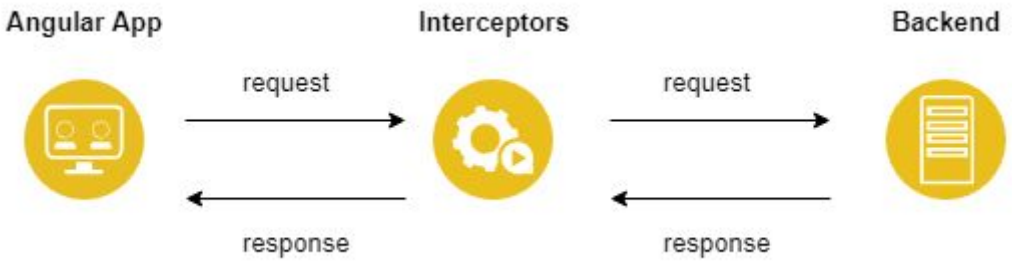
Το Angular framework χρησιμοποιεί σαν γλώσσα την Typescript όπου αυτή με τη σειρά της προσφέρει στους developers την ικανότητα να δημιουργούν σαφής τύπους στις μεταβλητές. Για παράδειγμα “let name: string”, δηλαδή η μεταβλητή name είναι τύπου string. Σε περίπτωση όμως που είναι επιθυμητό μπορεί να γίνεται χρήση κάποιων πιο σύνθετων δομών δεδομένων τότε αυτό που χρειάζεται είναι το μοντέλο[19]. Όταν διατυπώνεται ένα μοντέλο μπορεί να επαναχρησιμοποιηθεί σε όλη την εφαρμογή. Ενώ η Typescript έχει interfaces, τα οποία παρέχουν αυτή τη λειτουργικότητα, το Angular προτείνει να υπάρχει μια απλή κλάση που ορίζονται σε αυτήν οι μεταβλητές για μια οντότητα για χρήση σε services, components, directives ή pipes(Σχήμα 4.40). Τα μοντέλα που έχει η εφαρμογή είναι τα εξής: auth-data.model.ts, statistics.model.ts, weather.model.ts και όλα ακολουθούν την ίδια μεθοδολογία με το μοντέλο Cities που υπάρχει στο παρακάτω σχήμα.



Σχήμα 4.40: Cities model

4.8 Http Interceptor

Το HttpInterceptor εισήχθη στο Angular από την έκδοση 4.3 και παρέχει έναν τρόπο να “αναχαιτίζει” ή να σταματάει τα Http requests ή responses να τα μετατρέπει ή να τα χειρίζεται και ύστερα να αποδεσμεύει συνεχίζοντας την πορεία τους(Σχήμα 4.41)[20]. Αν και το interceptor είναι ικανό να μεταλλάσει τα requests και τα responses τα properties από τα HttpRequest και HttpResponses είναι μόνο για ανάγνωση(read-only) με αποτέλεσμα να τα καθιστά σε μεγάλο βαθμό αμετάβλητα. Αυτό συμβαίνει επειδή αν χρειαστεί να δοκιμάσουμε ξανά ένα request το οποίο δεν επέτυχε, η αμεταβλητότητα διαβεβαιώνει ότι όλη η διαδικασία μπορεί να εκτελεστεί ξανά με το ίδιο request πολλές φορές. Επιπλέον, υπάρχει η δυνατότητα να χρησιμοποιηθούν πολλά interceptors, αλλά το Angular διατηρεί την σειρά με την οποία παρέχονται. Για παράδειγμα υλοποιούνται τα interceptors Α μετά Β και τέλος Γ, τα requests ακολουθούν τη φορά Α -> Β -> Γ και τα responses την ανάποδη Γ -> Β -> Α. Εφόσον οριστεί μια σειρά δεν γίνεται να αλλάξει ούτε να διαγράφεται ένα interceptor. Μπορεί να ενεργοποιείται ή να απενεργοποιείται δυναμικά μέσα στη μέθοδό του. να πω για τα assets.



Σχήμα 4.41: Γραφική αναπαράσταση των Interceptors

4.8.1 Auth Interceptor

Στο φάκελο auth βρίσκεται το Auth Interceptor το οποίο υλοποιεί τη μέθοδο intercept από το interface HttpInterceptor. Σε αυτή τη μέθοδο σαν παράμετρος υπάρχει τη req που είναι ένα HttpRequest και τη next που είναι ένα HttpHandler όπου με αυτή το request συνεχίζει την πορεία προς κάποιο back-end. Η λογική σε αυτό το interceptor είναι πως όσα requests γίνονται προς το back-end μας πρέπει στα Http headers να συμπεριλαμβάνεται το "Authorization" και του ορίζεται σαν τιμή το "Bearer " και μετά ακολουθεί το token. Για να έχουμε πρόσβαση στο token που ορίζεται στο AuthService είναι απαραίτητο να γίνει inject το service που αναφέραμε για να έχουμε πρόσβαση στη μέθοδο getToken. Πριν προσθέσουμε αυτή την πληροφορία στο header πρέπει να κλωνοποιήσουμε το request με τη μέθοδο clone η οποία επιτρέπει ταυτόχρονα να το τροποποιήσουμε. Σε περίπτωση που οι αλλαγές γινόντουσαν απευθείας στο request θα δημιουργούνταν πρόβλημα στην εφαρμογή και στον handler που διαχειρίζεται τα requests. Για να περαστούν οι αλλαγές στο request χρησιμοποιούμε τη μέθοδο set η οποία προσθέτει κάποιο header εάν δεν υπάρχει, διαφορετικά αν υπάρχει ήδη ορίζεται αυτό που δηλώθηκε, οποιαδήποτε headers υπήρχαν παραμένουν και δεν επηρεάζονται. Στη δική μας εφαρμογή προσθέτουμε το header "Authorization" και σαν τιμή του ορίζουμε το token.Τέλος, για να συνεχίσει την πορεία του το request προωθούμε αυτό που μόλις κλωνοποιήσαμε με τη μεταβλητή next και τη μέθοδο handle όπου ορίζουμε σαν παράμετρο(Σχήμα 4.42). Για να μπορεί πλέον να χρησιμοποιείται το interceptor από την εφαρμογή πρέπει να γίνει inject αλλά όχι με τον τρόπο που αναφέρεται σε προηγούμενη ενότητα με τα metadata στην αρχή της κλάσης, αλλά κάνοντας την import στο αρχείο app.module.ts. Στον πίνακα providers προσθέτουμε ένα νέο αντικείμενο με properties τα provide που περιέχει το token "HTTP_INTERCEPTORS" που γίνεται import από το Angular, useClass που αναφέρεται στην κλάση του interceptor η οποία πρέπει με τη σειρά της να γίνει import και το multi που ορίζεται true για να κάνει γνωστό στο Angular ότι μπορούν να υπάρχουν πολλά interceptors και να μην γίνουν overwrite όσα προϋπάρχουν(Σχήμα 4.43).

```
@Injectable()
export class AuthInterceptor implements HttpInterceptor {
    constructor(private authService: AuthService){}

    intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
        const authToken = this.authService.getToken();
        const authRequest = req.clone({
            headers: req.headers.set('Authorization', "Bearer " + authToken)
        });
        return next.handle(authRequest);
    }
}
```

Σχήμα 4.42: Auth Interceptor κλάση

```
providers: [
  {provide: HTTP_INTERCEPTORS, useClass: AuthInterceptor, multi: true},
```

Σχήμα 4.43: Ο πίνακας provides στο app.module.ts

4.8.2 Error Interceptor

Το error-interceptor δεν θα μεταβάλλει το εξερχόμενο request αλλά θα "παρακολουθεί" εάν το response που θα λάβει περιέχει κάποιο σφάλμα, ώστε να ενεργοποιηθεί και να εμφανίζει στον χρήστη το αντίστοιχο μήνυμα λάθους στο matDialog που υπάρχει το error component. Σε περίπτωση που βρεθεί κάποιο λάθος στο back-end και δεν υπάρχει κατάλληλο μήνυμα που να εξηγεί τι συνέβη τότε χρησιμοποιείται ένα προκαθορισμένο μήνυμα και έχει γενικό χαρακτήρα. Για να είναι δυνατόν να "παρατηρούμε" τι απάντηση θα λάβει το front-end χρησιμοποιούμε τη μέθοδο pipe με το operator catchError, το οποίο διαχειρίζεται σφάλματα και συγκεκριμένα τα Http. Μεσα στον handler ορίζουμε μία συνάρτηση που θα έχει το σφάλμα με τύπο HttpErrorResponse. Στη μέθοδο ορίζεται στη μεταβλητή errorMessage ένα γενικευμένο μήνυμα, ενώ αν η παράμετρος error έχει ορισμένο το πεδίο error.error.message, το οποίο είναι ένα JSON αντικείμενο, τότε στη μεταβλητή errorMessage ορίζεται αυτό το πεδίο. Για να εμφανιστεί στο UI αυτό το μήνυμα έχει γίνει inject το matDialog που έγινε import από το Angular Material και καλείται το dialog.open ορίζοντας το ErrorComponet και ένα αντικείμενο με δεδομένα τη μεταβλητή errorMessage. Επειδή το response πρέπει να συνεχίσει την πορεία και να πάει προς τα services που έχουν δημιουργηθεί η μεταβλητή error που ορίσαμε στην catchError οφείλει να επιστρέφει ένα Observable και να περνάει το error που προέκυψε. Αυτό υλοποιείται με τη μέθοδο throwError, η οποία χρειάζεται να γίνει και αυτή import και έχοντας παράμετρο τη μεταβλητή error(Σχήμα 4.44). Αντίστοιχα και αυτό το interceptor πρέπει να οριστεί στον πίνακα providers του αρχείου app.module.ts.

```
@Injectable()
export class ErrorInterceptor implements HttpInterceptor {

    constructor(private dialog: MatDialog) {}

    intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
        return next.handle(req).pipe(
            catchError((error: HttpErrorResponse) => {
                let errorMessage = 'An unknow error occured!';
                if(error.error.message){
                    errorMessage = error.error.message;
                }
                this.dialog.open(ErrorComponent, {data: {message: errorMessage}});
                return throwError(error);
            })
        );
    }
}
```

Σχήμα 4.44: Error Interceptor

4.9 Route guards

Τα route guards είναι interfaces από το Angular τα οποία ενημερώνουν το router αν θα πρέπει να επιτρέψει ή όχι στην προσπέλαση στο αιτούμενα route[21]. Αυτή η απόφαση λαμβάνεται από μια επιστρεφόμενη τιμή που είναι boolean και υλοποιείται από μια κλάση. Υπάρχουν πέντε διαφορετικά είδη από guards και καλείται το καθένα ύστερα από μια συγκεκριμένη ακολουθία. Αυτά είναι τα canActivate, canActivateChild, canDeactivate, canLoad και Resolve. Σε προηγούμενη ενότητα έγινε αναφορά στο Authentication της εφαρμογής ότι ελέγχεται το token και η ώρα λήξης του. Σε αυτή τη περίπτωση η απόφαση που πρέπει να παρθεί είναι αν ο χρήστης είναι συνδεδεμένος και ανάλογα την απάντηση αν είναι αληθής ή όχι, το router θα κάνει render το route που επιλέχθηκε. Αύτη η λειτουργία υφίσταται ώστε να μην μπορεί οποιοσδήποτε χρήστης να έχει πρόσβαση σε ορισμένα resources στο backend. Αν προσπαθήσει δηλαδή κάποιος να το προσπελάσει απευθείας τότε θα οδηγείται στην σελίδα του login ώστε να υποχρεούται να συνδέεται. Το ίδιο θα μπορούσε να εφαρμοστεί και σε περίπτωση που οι χρήστες είχαν κάποιους ρόλους στο προφίλ τους και η πρόσβαση σε μερικές λειτουργίες απαιτούσε το ρόλο του διαχειριστή. Στο φάκελο auth που περιέχει τα components login και register υπάρχει και η κλάση auth.guard.ts(Σχήμα 4.45) που είναι ένα εξειδικευμένο service. Στο service αυτό υλοποιείται το guard canActivate που δέχεται δύο παραμέτρους το route και το state. Το route που προσπαθεί ο χρήστης να έχει πρόσβαση και το state που είναι ένα snapshot με την κατάσταση του route εκείνη τη στιγμή. Αυτή η μέθοδος επιστρέφει μια τιμη boolean ή ένα Observable ή ένα Promise που αποφέρουν και αυτά μια boolean τιμή. Έπειτα εισάγεται το authService για να χρησιμοποιηθεί η μέθοδος isAuthenticated και αποθηκεύεται η επιστρεφόμενη τιμή. Αν η τιμή αυτή είναι αληθής τότε επιστρέφεται από το route guard η boolean μεταβλητή, διαφορετικά το router οδηγεί τον χρήστη στην σελίδα login. Τέλος, απαιτείται να δηλωθεί σε ποια front-end routes θα υπάρχει το guard(Σχήμα 4.46). Αφού εισάγεται, στο NgModule προστίθεται ένα ακόμα πεδίο ο πίνακας providers

και μέσα ορίζεται το AuthGuard. Επίσης στα routes που επιλέγονται ύστερα από το component εισάγεται το AuthGuard.

```
@Injectable()
export class AuthGuard implements CanActivate {
    constructor(private authService: AuthService, private router: Router) {}
    canActivate(
        route: ActivatedRouteSnapshot,
        state: RouterStateSnapshot):
        boolean | Observable<boolean> | Promise<boolean> {
        const isAuth = this.authService.getIsAuthenticated();
        if (!isAuth) {
            this.router.navigate(['/login'])
        }
        return isAuth;
    }
}
```

Σχήμα 4.45: Η κλάση Auth.guard.ts

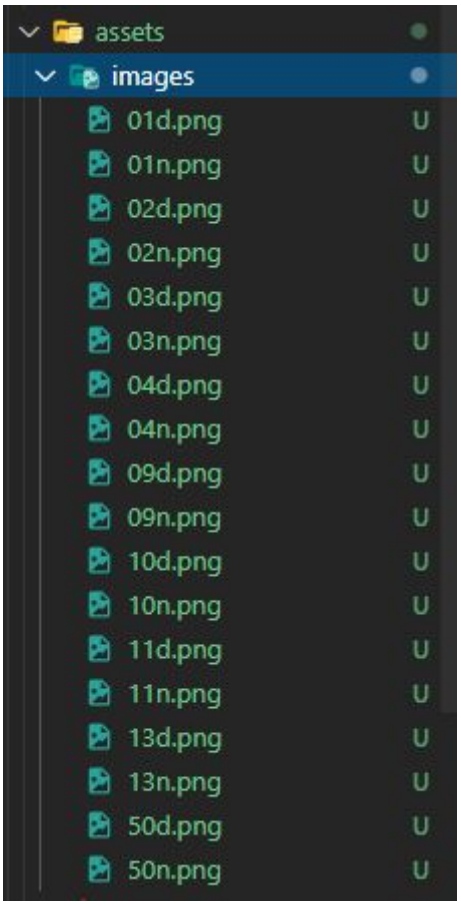
```
const routes: Routes = [
  { path: '', component: WeatherInfoComponent},
  { path: 'login', component: LoginComponent},
  { path: 'register', component: RegisterComponent},
  { path: 'stats', component: StatisticsComponent, canActivate: [AuthGuard]},
  { path: 'search', component: SearchCitiesComponent, canActivate: [AuthGuard]},
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule],
  providers: [AuthGuard]
})
export class AppRoutingModule { }
```

Σχήμα 4.46: Το App-routing.module.ts

4.10 Assets

Στο UI της αρχικής σελίδας στην εφαρμογή μέσα στις πληροφορίες που εμφανίζονται είναι και κάποια εικονίδια που απεικονίζουν την κατάσταση του καιρού τη δεδομένη στιγμή. Το Angular cli όταν δημιουργείται ένα νέο project φτιάχνει και το φάκελο assets, ο οποίος έχει τον συγκεκριμένο σκοπό να αποθηκεύει φωτογραφίες και να είναι προσβάσιμα τα περιεχόμενα του σε όλη την εφαρμογή(Σχήμα 4.47). Το response του OpenWeatherMap api επιστρέφει εκτός από τα μετεωρολογικά δεδομένα και εικονίδια, τα οποία χρησιμοποιούνται στην αρχική σελίδα με σκοπό να είναι πιο εύκολα κατανοητή η κατάσταση του καιρού στον χρήστη και να είναι πιο όμορφο το UI. Εφόσον περαστούν όλες οι φωτογραφίες που παρέχονται από τον ιστότοπο του OpenWeatherMap στο project τότε στην html

σελίδα της αρχικής είναι δυνατόν το img tag στο όρισμα src να περαστεί η τοπική διαδρομή του αρχείου png. Επειδή τα δεδομένα αλλάζουν ανάλογα την επιλεγμένη πόλη η διαδρομή αυτή είναι δυναμική, δηλαδή το path “assets/images” παραμένει σταθερό και από το response από το back-end θα οριστεί το όνομα της εικόνας. Όταν η εφαρμογή γίνεται build τότε ο φάκελος assets μετακινείται στο φάκελο dist αλλά οι διαδρομές συνεχίζουν να λειτουργούν στο component.



Σχήμα 4.47: Οι εικόνες στον φάκελο assets

4.11 Επίλογος

Σε αυτό το κεφάλαιο έγινε ανάλυση των στοιχείων που χρησιμοποιήθηκαν από το Angular framework για να αναπτυχθεί το front-end της εφαρμογής, ξεκινώντας απο τα components ύστερα στα services και στα interceptors. Αυτή η εφαρμογή για να καθίσταται σιγά σιγά χρήσιμη απαιτούνται και τα δεδομένα τα οποία θα παρουσιάζει στον περιηγητή του χρήστη και θα ενημερώνεται για τον καιρό και θα μπορεί να ανατρέξει σε παλαιότερα στατιστικά δεδομένα που αφορούν τον καιρό. Αυτά τα δεδομένα θα προέρχονται από τον server(back-end) και θα μιλήσουμε περαιτέρω για αυτό στον επόμενο κεφάλαιο.

Κεφάλαιο 5ο: Ανάπτυξη back-end

5.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα γίνει ανάλυση της μεθοδολογίας που χρησιμοποιήθηκε για την ανάπτυξη του back-end με χρήση του Node.js. Το back-end στην εφαρμογή είναι ένα RESTful API, δηλαδή Representational State Transfer. Το REST API είναι stateless, πιο συγκεκριμένα δεν ενδιαφέρεται για τους clients που επικοινωνούν με αυτόν παρά μόνο έχει προσβάσιμα κάποια endpoints προς τους clients και αλληλεπιδρά με αυτούς μέσα από το http πρωτόκολλο στέλνοντας responses. Επειδή οποιοσδήποτε μπορεί να επικοινωνήσει με τον server επιλέγεται ποια endpoints επιστρέφουν μια απάντηση και με τι δεδομένα και ποια endpoints χρειάζονται να έχουν αυθεντικοποίηση του χρήστη. Στο Node.js υπάρχουν διαδρομές(paths) που εμπεριέχουν τα endpoints που αναφέρονται παραπάνω και υποστηρίζονται διαφορετικές Http μεθόδους όπως get,post,delete, put patch. Η Angular εφαρμογή π.χ. στέλνει ένα post αίτημα(request) με το service HttpClient στο path /users, ώστε να δημιουργήσει ένα νέο χρήστη στη βάση. Η λειτουργία αυτή υλοποιείται στην πλευρά του server και τελικά θα επιστραφεί μια απάντηση η οποία θα επηρεάσει το front-end. Τέλος να σημειωθεί ότι η επικοινωνία του client με τον server γίνεται με JSON δεδομένα, που είναι Javascript αντικείμενα που είναι και για το Node.js εύκολα διαχειρίσιμα και προσπελάσιμα.Υπάρχουν εναλλακτικά και άλλα τύπου δεδομένα να χρησιμοποιηθούν για την επικοινωνία όπως XML,URLEncoded αλλά δεν είναι προτιμώνται.

5.2 Server.js

Το Node.js για να εκτελείται σαν server και να μπορεί να είναι προσβάσιμο με τους clients, πρέπει να δημιουργήσουμε ένα javascript αρχείο και λειτουργεί ως αυτός. Σε αντίθεση με τη php που απαιτεί ένα λογισμικό οπως το apache ή το nginx για να τρέχει-σερβιρει, εδώ το αρχείο έχει τον παραπάνω ρόλο. Στον root φάκελο του project δημιουργείται το server.js αρχείο(Σχήμα 5.1), όπου εκτελείται από το cli node server.js. Αρχικά θα γίνει import το πακέτο http και θα αποθηκευτεί σε μια μεταβλητή, μόνο που στην παρούσα έκδοση της Node γίνεται ως εξής “require(‘package_name’)” και κατόπιν το πακέτο που χρειάζεται. Το πακέτο http δεν χρειάζεται εγκατάσταση, διότι είναι default και γίνεται εγκατάσταση μαζί με τη Node. Διαφορετικά, σε άλλα πακέτα για την εγκατάστασή τους γίνεται χρήση του npm που αναφερεται στο κεφάλαιο 1. Ύστερα από τα απαιτούμενα require, δημιουργούνται κάποιες μέθοδοι για τις ρυθμίσεις του server, η πρώτη normalizePort ελέγχει την τιμή για το port ώστε να είναι έγκυρη, η δεύτερη onError ελέγχει τις μεταβλητές addr και port και σε περίπτωση λάθος εμφανίζεται το αντίστοιχο μήνυμα και τερματίζεται η εκτέλεση του αρχείου. Η τρίτη ορίζει την τιμή addr και τυπώνει στην κονσόλα σε ότι τρέχει ο server στο συγκεκριμένο port. Επιπλέον, ορίζουμε το port στην εφαρμογή και δημιουργείται ο server με τη συνάρτηση http.createServer και παράμετρο τις ρυθμίσεις που ορίστηκαν. Εν τέλη, δηλώνουμε τους listeners που αναλύσαμε παραπάνω και ξεκινάει ο server να “ακούει” τα requests.

```
const app = require("./backend/app");
const debug = require("debug")("node-angular");
const http = require("http");

const normalizePort = val => {
  var port = parseInt(val, 10);

  if (isNaN(port)) {
    // named pipe
    return val;
  }

  if (port >= 0) {
    // port number
    return port;
  }

  return false;
};

const onError = error => {
  if (error.syscall !== "listen") {
    throw error;
  }
  const bind = typeof port === "string" ? "pipe " + port : "port " + port;
  switch (error.code) {
    case "EACCES":
      console.error(bind + " requires elevated privileges");
      process.exit(1);
      break;
    case "EADDRINUSE":
      console.error(bind + " is already in use");
      process.exit(1);
      break;
    default:
      throw error;
  }
};

const onListening = () => {
  const addr = server.address();
  const bind = typeof port === "string" ? "pipe " + port : "port " + port;

  debug("Listening on " + bind);
};

const port = normalizePort(process.env.PORT || "3000");
app.set("port", port);

const server = http.createServer(app);
server.on("error", onError);
server.on("listening", onListening);
server.listen(port,"83.212.118.254");
```

Σχήμα 5.1: server.js

5.3 App.js

Αφού δημιουργήθηκε ο server και εκτελέστηκε, παρατηρήθηκε ότι οι αλλαγές στα javascript αρχεία απαιτούσαν τον τερματισμό εκτέλεσης του αρχείου και της επανεκτέλεσής του. Γι' αυτό το λόγο έγινε εγκατάσταση ενός πακέτου, το οποίο κατόπιν των αλλαγών που συνέβησαν στα αρχεία του back-end θα επανακκινούσε το server.js. Με τη χρήση του npm εγκαθίσταται το nodemon(Σχήμα 5.2) και στο package.json αρχείο στο property scripts εισάγεται μια γραμμή ώστε εκτελώντας το "npm run start:server" στην ουσία θα εκτελείται το "nodemon server.js"(Σχήμα 5.3). Έπειτα, δημιουργούμε στον root φάκελο του project το φάκελο backend που περιέχει το app.js αρχείο. Αυτό το αρχείο είναι ουσιαστικά η express εφαρμογή στην πλευρά του server και ορίζονται όλα τα routes ή paths όπως αναφέρονται παραπάνω. Ακόμα γίνεται σύνδεση με τις βάσεις, mongodb και influxdb, προστίθεται στην εφαρμογή το πακέτο bodyParser, διότι "γεμίζει" το req.body ανάλογα με τα headers που έχουν οριστεί και τα αναλύει(parse).Ύστερα ορίζονται τα headers για τα requests που θα στέλνονται από τους clients και πρέπει να υπάρχουν για να μπορούν να εκτελούνται(Σχήμα 5.4). Τέλος, για να είναι προσβάσιμη η κλάση και σε άλλες στη χρησιμοποιείται το "module.exports = app;" και μπορεί να γίνει import στο server.js.

```
npm install --save-dev nodemon
```

Σχήμα 5.2: Εγκατάσταση nodemon σαν development dependency

```
"scripts": {
  "ng": "ng",
  "start": "ng serve",
  "build": "ng build",
  "test": "ng test",
  "lint": "ng lint",
  "e2e": "ng e2e",
  "start:server": "nodemon server.js"
},
```

Σχήμα 5.3: Το αντικείμενο scripts στο package.json

```
app.use((req,res,next) => {
  res.setHeader("Access-Control-Allow-Origin", "*");
  res.setHeader("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept, Authorization");
  res.setHeader("Access-Control-Allow-Methods", "GET, POST, PATCH, PUT, DELETE, OPTIONS");
  next();
});
```

Σχήμα 5.4: Headers

5.4 Βάσεις δεδομένων

Το back-end της express εφαρμογής συνδέεται με δύο βάσεις δεδομένων. Η πρώτη είναι η InfluxDB και η δεύτερη η MongoDB. Στο 1ο κεφάλαιο έγινε μια μικρή αναφορά και στις δύο εξίσου. Αν και είναι ουσιαστικά το ίδιο πράγμα, ωστόσο διαφέρουν μεταξύ τους καθως η πρώτη βασίζεται στις χρονοσειρές και η δεύτερη είναι μη σχεσιακή. Η InfluxDB λόγω της ιδιαιτερότητας της έχει σε κάθε measurement μία στήλη του ονομάζεται time και αποθηκεύει το τρέχων timestamp(Σχήμα 5.5). Ακόμα μπορεί οι υπόλοιπες στήλες να χωρίζονται σε fields και tags[22]. Τα fields αποτελούνται από το field key και το field value. Το field key είναι string και εξηγούν τι ακριβώς είναι οι τιμές που αντιστοιχούν στα field values. Τα field values είναι τα δεδομένα τα οποία μπορεί να είναι αριθμοί, strings ή booleans και πάντα συσχετίζονται με ένα timestamp. Τα fields είναι απαραίτητο κομμάτι της InfluxDB και δεν γίνεται να υπάρχουν δεδομένα χωρίς fields. Επίσης, τα fields δεν είναι indexed, δηλαδή όταν εκτελείται ένα query με field values πρέπει να ελεγχθούν όλες οι τιμές που ταιριάζουν με τα κριτήρια του, που σημαίνει ότι δεν θα είναι με υψηλή επίδοση και γρήγορο στην εκτέλεση. Από την άλλη υπάρχουν τα tags, τα οποία χωρίζονται και αυτά σε tag keys και tag values. Και τα δύο αποθηκεύουν string και metadata των εγγραφών και σε αντιθεση με τα fields τα tags είναι indexed και προτείνεται να είναι πεδία που χρησιμοποιούνται συχνά στα query ώστε να είναι πιο αποδοτικά και γρήγορα. Τα tags σε ένα measurement της InfluxDB είναι προεραιτικά αλλά χρήσιμα οπως περιγράψαμε παραπάνω. Θα ήταν ωφέλιμο στην σχεδίαση ενός νέου schema να βελτιστοποιείται και τα ορίζονται ως tags τα κατάλληλα πεδία, διότι σε περίπτωση ενός query που το select clause αποτελείται από fields θα πρέπει να σκαναριστουν όλες οι τιμές του πρώτου field ύστερα του δεύτερου κ.ο.κ. Ακόμα στην InfluxDB είναι το measurement, το οποίο περιέχει tags, fields και τη στήλη time και ένα όνομα που περιγράφει τα δεδομένα που αποθηκεύει. Το measurement είναι ο αντίστοιχος πίνακας στο SQL και μπορεί να ανήκει σε κάποιο retention policy. Το retention policy περιγράφει το πόσο χρονικό διάστημα θα κρατούνται τα δεδομένα στη βάση πριν διαγραφούν οριστικά και το πόσα αντίγραφα των δεδομένων υπάρχουν στο cluster. Στην παρούσα εφαρμογή το measurement δεν έχει retention policy διότι είναι απαραίτητο να υπάρχουν δεδομένα από το παρελθόν για να δημιουργούνται τα γραφήματα. Η δεύτερη βάση δεδομένων είναι και αυτή με διαφορετική φιλοσοφία από την SQL. Η MongoDB αποθηκεύει τα δεδομένα σαν documents, τα οποία συγκεντρώνονται όλα μαζί σε collections[23]. Μια βάση δεδομένων αποθηκεύει ένα η πολλά collections από documents. Δεν χρειάζεται να δημιουργηθεί η βάση δεδομένων, αλλά την πρώτη εισαγωγή δεδομένων γίνεται αυτόματα. Μια εγγραφή στη MongoDB είναι ένα document, το οποίο είναι δομημένο από πεδία και τις τιμές του. Αυτό είναι παρόμοιο με ένα JSON αντικείμενο και μέσα να περιέχει άλλα documents ή πίνακες ή πίνακες από documents. Το πλεονέκτημα των documents είναι πως τα native types των δεδομένων ανταποκρίνονται σε πολλές γλώσσες προγραμματισμού, τα ενσωματωμένα documents μειώνουν τις πολύπλοκες συσχετίσεις και τα δυναμικά schemas υποστηρίζουν τον πολυμορφισμό. Έπειτα αυτά αποθηκεύονται σε collections, που είναι ανάλογα των πινάκων στις σχεσιακές βάσεις δεδομένων. Τα collections υποστηρίζουν views και indexing για γρηγορότερη εκτέλεση των queries. Το indexing επιτυγχάνεται με κάποια κλειδιά από τα ενσωματωμένα documents. Στην βάση δεδομένων της εφαρμογής το κλειδί για indexing είναι το πεδίο _id. Η MongoDB παρέχει μια πλούσια query language η οποία αποτελείται από μεθόδους με τη μορφή “db.collection.function”(Σχήμα 5.6). Όλες οι λειτουργίες εγγραφής γίνονται σε ατομικό επίπεδο, δηλαδή επηρεάζεται ένα document την κάθε φορά[24]. Οι υπόλοιπες λειτουργίες όπως read,update και delete λειτουργούν με την ίδια λογική.

```
name: http
time                mean
----                ----
1599593400000000000
1599594000000000000 28.03
1599594600000000000 27.71
1599595200000000000 27.665
1599595800000000000 27.62
1599596400000000000 27.57
1599597000000000000 27.54

name: http
time                name   id
----                ----   --
1576748700000000000 Athens 264371
```

Σχήμα 5.5: Παράδειγμα εκτέλεσης query στη InfluxDB

```
> db.cities.find().limit(5)
{ "_id" : ObjectId("5f20510c7ea3652e6d89424b"), "id" : 261743, "name" : "Irákleio", "__v" : 0 }
{ "_id" : ObjectId("5f2051c8ee751414ed43e4ec"), "id" : 251833, "name" : "Volos" }
{ "_id" : ObjectId("5f20525e4b3be59950263b5b"), "id" : 734077, "name" : "Thessaloniki" }
{ "_id" : ObjectId("5f2052974b3be59950263b5c"), "id" : 255683, "name" : "Pátrai" }
{ "_id" : ObjectId("5f2052b34b3be59950263b5d"), "id" : 264371, "name" : "Athens" }
```

Σχήμα 5.6: Παράδειγμα εκτέλεσης query στη MongoDB

5.5 Routes

Σε προηγούμενη ενότητα γίνεται αναφορά στα routes που ορίζονται στο app.js. Τα server side routes διαφέρουν από τα client side. Τα πρώτα διαχειρίζονται εισερχόμενα αιτήματα(requests) και επιστρέφουν πίσω κάποια δεδομένα ανάλογα του αιτήματος. Τα client side routes αναλύουν το τρέχων URL και κάνουν render το αντίστοιχο component.Ένα server side route είναι ένα τμήμα της express εφαρμογής που συσχετίζει μια HTTP μέθοδο, ένα URL path και μια μέθοδο που καλείται όταν υπάρχει το κατάλληλο αίτημα και το διαχειρίζεται[25]. Για να δημιουργηθούν τα routes χρησιμοποιείται το express.Router middleware και μπορούν να ομαδοποιηθούν σε διαφορετικές κλάσεις. Η κάθε κλάση χειρίζεται ένα διαφορετικό κομμάτι του application logic και έχει ένα κοινό πρόθεμα στο path για μπορείς να τα προσπελάσεις. Να σημειωθεί ότι μπορεί να υπάρχουν ίδια URL paths αλλά με διαφορετικές Http μεθόδους που να εκτελούν τελείως διαφορετικές λειτουργίες. Το URL path καθορίζει σε ένα route το endpoint, στο οποίο τα clients αποστέλνουν τα αιτήματά τους. Το URL path είναι string και είναι δυνατόν να χρησιμοποιηθούν regex ή μεταβλητές ώστε να ταιριάζει με τα πιθανά αιτήματα που θα λαμβάνει. Οι μεταβλητές ορίζονται σε συγκεκριμένα σημεία του με το σύμβολο “:” να υπάρχει πριν από αυτές. Ακόμα αποθηκεύονται στο αντικείμενο req.params και η μέθοδος που διαχειρίζεται το route μπορεί να τις αξιοποιήσει για τις ενέργειες που απαιτούνται. Για να είναι αποδεκτό το όνομα μιας μεταβλητής πρέπει να μόνο γράμματα, πεζά και κεφαλαία, αριθμους και την κάτω παύλα. Στο project έχει δημιουργηθεί ο φάκελος routes κάτω από το backend που τα περιέχει όλα.

5.5.1 City routes

Στο τρέχων route υπάρχουν δυο endpoints(Σχήμα 5.7), το /insert που δέχεται post requests μαζί με το middleware checkAuth που ελέγχει αν το αίτημα προέρχεται από χρήστη που είναι επαληθευμένος και το controller function, η οποία εκτελεί όλες τις ενέργειες για να εισαχθεί μια καινούρια πόλη στο configuration αρχείο του telegraf. Το δεύτερο endpoint έχει σαν URL path το cities, δέχεται get αιτήματα, παρόμοια με το παραπάνω υπάρχει το checkAuth και υπάρχει η συνάρτηση που επιστρέφει τις πόλεις που έχει προσθέσει ο χρήστης στο αρχείο που αναφέραμε στο πρώτο endpoint. Αυτές οι πληροφορίες για τις πόλεις που έχει εισάγει ο κάθε χρήστης αποθηκεύονται στη βάση δεδομένων MongoDB. Για τα endpoints γίνεται χρήση του express.Router και γίνονται import τα απαιτούμενα αρχεία. Τέλος, σαν url path prefix έχει οριστεί το “api/city”.

```
const express = require('express');
const router = express.Router();
const StatsController = require('../controllers/stats');
const checkAuth = require('../middleware/check-auth');


// get mean, max, min
router.get("/one/:fun/:size/:id/:start/:end/:period", checkAuth, StatsController.getOneAggragateFunction);

//get all
router.get("/all/:size/:id/:start/:end/:period", checkAuth, StatsController.getAllFunctions);

module.exports = router;
```

Σχήμα 5.7: City routes

5.5.2 Influxdb routes

Τα routes του InfluxDB είναι τρία http get requests(Σχήμα 5.8). Τα δυο πρώτα εκτελούν παρόμοια λειτουργία με διαφορά ότι το δεύτερο έχει και ένα route parameter στο URL path. Οι συναρτήσεις του controller επιστρέφουν τις πιο πρόσφατες πληροφορίες για τον καιρό στην πόλη που επιλέχθηκε. Το τρίτο endpoint, το “/towns” επιστρέφει όλες τις πόλεις που είναι αποθηκευμένες στη βάση δεδομένων InfluxDB. Σε αυτό το route το prefix είναι το “api/weather”.

```
const express = require('express');
const router = express.Router();
const InfluxdbController = require('../controllers/influxdb');

router.get("/inf", InfluxdbController.getThessWeatherInfo);

router.get("/inf/:id", InfluxdbController.getCityWeatherInfo);

router.get("/towns", InfluxdbController.getInfluxCities);

module.exports = router;
```

Σχήμα 5.8: Influxdb routes

5.5.3 Stats routes

Το αρχείο stats αποτελείται από δύο routes(Σχήμα 5.9), το ένα εκτελεί ένα query με μία από τις max, mean, min, επιστρέφοντας τα αποτελέσματα με βάση τα route parameters που δέχεται και είναι τα fun, size, id, start, end και period. Το fun αναφέρεται στην aggregate μέθοδο που θα χρησιμοποιηθεί στο query και είναι κάποια από τις παραπάνω. Το size σχετίζεται με τη τιμή που θα χρησιμοποιήσει η aggregate μέθοδος. Το id είναι το id της πόλης που είναι αποθηκευμένο στη βάση δεδομένων, το start και end αφορά τη χρονική περίοδο των μετεωρολογικών δεδομένων που θα επιστραφούν και τέλος το period επηρεάζει την ομαδοποίηση αυτών, δηλαδή ανά πόσο χρονικό διάστημα θα επιστρέφεται μια τιμή ανάμεσα στα χρονικά όρια του start και end. Η διαφορά του δεύτερου route από το προηγούμενο είναι ότι επιστρέφονται πληροφορίες που εκτελούν και τα τρία functions(max, mean, min) για τις αντίστοιχες παραμέτρους που στέλνονται. Αυτά τα δύο routes είναι protected από το checkAuth middleware και το route prefix είναι το “api/stats”.

```
const express = require('express');
const router = express.Router();
const StatsController = require('../controllers/stats');
const checkAuth = require('../middleware/check-auth');


// get mean, max, min
router.get("/one/:fun/:size/:id/:start/:end/:period", checkAuth, StatsController.getOneAggragateFunction);

//get all
router.get("/all/:size/:id/:start/:end/:period", checkAuth, StatsController.getAllFunctions);

module.exports = router;
```

Σχήμα 5.9: Stats routes

5.5.4 User routes

Το αρχείο user αποτελείται από δύο routes και έχει σχέση με την αυθεντικοποίηση των χρηστών της εφαρμογής(Σχήμα 5.10). Το πρώτο έχει το URL “path /signup”, ενώ το δεύτερο έχει το URL “path /login”. Και τα δύο “ακούνε” μόνο http post αιτήματα. Οι συναρτήσεις του κάθε route αντίστοιχα αποθηκεύουν ή ελέγχουν τα στοιχεία του εκάστοτε χρήστη στη MongoDB βάση δεδομένων. Στην περίπτωση της εγγραφής αποθηκεύονται τα στοιχεία και στην άλλη περίπτωση ανακτώνται και παράγουν ένα token αν είναι ορθά. Τέλος, το prefix route είναι το “api/user”.

```
const express = require('express');
const UserController = require("../controllers/user");

const router = express.Router();

router.post("/signup", UserController.createUser);

router.post("/login", UserController.userLogin);

module.exports = router;
```

Σχήμα 5.10: User routes

5.6 Controllers

Στα routes που αναφερθήκαμε στην προηγούμενη, έχουν σαν τελευταία παράμετρο μια μέθοδο που ονομάζεται callback function, δηλαδή εκτελείται πάντα μόλις κάποιο αίτημα σταλεί στο συγκεκριμένο endpoint. Αυτές οι μέθοδοι σε όλα τα routes στην express εφαρμογή υπάρχουν σε ξεχωριστά αρχεία και ονομάζονται Controllers. Στα αρχεία που περιέχουν τα routes αυτό που συμβαίνει είναι ότι η μέθοδος δεν ορίζεται και υλοποιείται εκεί, αλλά περνάει μια αναφορά για αυτήν. Στο φάκελο controllers υπάρχουν τα javascript αρχεία που εκεί γίνονται export οι συναρτήσεις αυτές για να είναι προσβάσιμες παντού. Ο κώδικας στα callBack function είναι σε μεγάλο βαθμό με promises, πιο συγκεκριμένα είναι ένα αντικείμενο που αντιπροσωπεύει την ενδεχόμενη επιτυχία ή αποτυχία μιας ασύγχρονης λειτουργίας. Το promise είναι το επιστρεφόμενο αντικείμενο που συνδέονται σ' αυτό callbacks[26]. Το επιστρεφόμενο αντικείμενο έχει κάποιες εγγυήσεις, η callback μέθοδος δεν θα καλεστεί ποτέ πριν ολοκληρωθεί η ασύγχρονη διαδικασία, η ολοκλήρωση του θα έχει δύο ενδεχόμενα είτε θα επιτύχει είτε θα αποτύχει. Σε περίπτωση επιτυχίας θα εκτελεστεί η μέθοδος που υπάρχει στο then() διαφορετικά θα εκτελεστεί η μέθοδος που ορίζεται στο catch(). Η τελευταία εγγύηση είναι ότι μπορεί να προστεθούν πολλά callbacks και το ένα κατόπιν του άλλου και η σειρά εκτέλεσής τους είναι ίδια, δηλαδή αφού τελειώσει το πρώτο θα συνεχίσει το δεύτερο και ούτω καθεξής. Ακόμα χρησιμοποιούνται μέθοδοι με τη λέξη-κλειδί το async που επιτρέπουν τη χρήση για το await. Αυτές οι δύο λέξεις-κλειδιά είναι και αυτές ιδανικές για ασύγχρονες λειτουργίες με πιο απλό και κατανοητό τρόπο σε σύγκριση με τα promise chains που περιγράψαμε παραπάνω. Όταν υπάρχει το await μέσα σε μια μέθοδο ουσιαστικά αυτό που συμβαίνει είναι να σταματάει η εκτέλεσή της μέχρι να ολοκληρωθεί ή να απορριφθεί μια ασύγχρονη ενέργεια και τότε συνεχίζει η async μέθοδος μέχρι να τελειώσει[27]. Ο κώδικας που έπεται από ένα await μπορεί να θεωρηθεί ως το αντίστοιχο then() callback με τελικό βήμα την επιστροφή μιας τιμής που γίνεται με λιγότερο κώδικα.

5.6.1 City controller

Αρχικά γίνονται τα απαραίτητα imports σε πακέτα και στο μοντέλο City που θα αναφερθούμε σε αυτά στην επόμενη ενότητα(Σχήμα 5.11). Αυτός ο controller περιέχει δύο μεθόδους, όσα και τα endpoints, με το exports και το όνομα της μεταβλητής γίνονται export και μπορούν να καλεστούν εκτός αυτού του αρχείου. Η πρώτη συνάρτηση επιστρέφει τις πόλεις που αντιστοιχούν με το χρήστη που είναι συνδεδεμένος στην εφαρμογή και της έχει επιλέξει ο ίδιος. Με βάση το μοντέλο εκτελείται η μέθοδος find και σε ένα javascript αντικείμενο περνιέται το πεδίο userId. Το userId δεν το δέχεται σαν παράμετρο από το route αλλά αντλείται από το αίτημα του front-end. Η συνάρτηση find είναι ασύγχρονη, διότι απαιτείται κάποιος χρόνος μέχρι να βρεθούν οι εγγραφές στη βάση και χρησιμοποιείται το callback then για την περίπτωση που επιτύχει η λειτουργία και επιστρέφονται οι πόλεις από τη βάση σε μορφή JSON. Το req και res αναφέρονται στο αίτημα(request) και την απάντηση(response) έπειτα από τον πίνακα αντικειμένων με τις πόλεις επιστρέφεται η απάντηση προς τον client. Αυτή επιστρέφει με http status διακόσια αφού ήταν επιτυχές και σε JSON περιέχεται ένα μήνυμα και οι πόλεις. Σε περίπτωση αποτυχίας υπάρχει το callback catch που επιστρέφει σαν απάντηση ένα μήνυμα που περιγράφει το λάθος(Σχήμα 5.12). Επιπλέον, υπάρχει και η άλλη συνάρτηση που δηλώνεται ως async και δέχεται σαν route παράμετρο το όνομα μιας πόλης. Αφού έγινε μελέτη για το πως λειτουργεί το search engine του OpenWeatherMap (https://openweathermap.org/find) και μέσω τον developer tools από τον browser παρατηρούνται τα http requests. Καθώς βρέθηκε το endpoint που επιστρέφει τη λίστα με τις διαθέσιμες πόλεις σε σχέση

με την τιμή που εισάγει ο χρήστης. Η απάντηση περιέχει και κάποια δεδομένα που δεν χρειάζονται για την εφαρμογή όπως οι συντεταγμένες της πόλης, ο καιρός της, το timestamp σε epoch μορφή, πληροφορίες για τον άνεμο, την χώρα που ανήκει , και εξειδικευμένες πληροφορίες για τη βροχή, το χιόνι και τα σύννεφα. Τα πεδία που μας ενδιαφέρουν από τη λίστα είναι τα id και name. Με το πακέτο axios εκτελείται ένα get request στο OpenWeatherMap API για να ελεγχθεί πρωτίστως αν υπάρχει η πόλη και κατά δεύτερον το id της. Το URL είναι δυναμικό καθώς εμπεριέχει το όνομα της πόλης που μπορεί να αλλάζει σε κάθε request. Η ασύγχρονη λειτουργία διαχειρίζεται με τη λέξη-κλειδί await και αποθηκεύει το αποτέλεσμα του αιτήματος σε μια μεταβλητή που θα χρειαστεί στον υπόλοιπο κώδικα(Σχήμα 5.13). Ύστερα από κάποιες μετατροπές στην μεταβλητή τροποποιείται ώστε να είναι JSON αντικείμενο και να είναι πιο εύκολα διαχειρίσιμο(Σχήμα 5.14). Αυτές οι μέθοδοι είναι οι indexOf, όπου βρίσκει τη θέση του χαρακτήρα που δίνεται σαν παράμετρος και επιστρέφεται η θέση του και η substring, η οποία δέχεται ως παραμέτρους δυο αριθμους και με βάση αυτούς επιστρέφεται ένα μέρος του αρχικού string. Αρχικά βρίσκουμε τις θέσεις των συγκεκριμένων χαρακτήρων που θέλουμε να διαγραφούν και κατόπιν με διπλή χρήση της συνάρτησης substring αφαιρούμε τον τελευταίο χαρακτήρα και ένα κομμάτι στην αρχή. Αφού μένει στην string μεταβλητή μόνο η λίστα μετατρέπεται σε JSON. Αφού είναι πλέον JSON μπορούν να ελεγχθούν κάποια πεδία, όπως το response status ώστε να επιστρέφεται μήνυμα και να τερματίζεται η μέθοδος(Σχήμα 5.15), μετά το property list που περιέχει τις πόλεις που ταιριάζουν με το όνομα που δόθηκε. Ελέγχεται το μήκος της λίστας, εάν είναι μηδέν επιστρέφει κατάλληλη απάντηση στον client διαφορετικά ακολουθεί μια σειρά ελέγχων. Αν ο χρήστης έχει εισάγει ήδη αυτήν την πόλη τότε επιστρέφεται μια απάντηση με http status 200 για να τον ενημερώσει ότι έχει γίνει αυτή η ενέργεια. Αν όχι τότε ελέγχεται αν υπάρχει στον πίνακα από κάποιον άλλον χρήστη γιατί όταν είναι καινούρια εγγραφή πρέπει να τροποποιηθεί και το configuration αρχειο του service Telegraf και να προστεθεί στη λίστα με τις υπόλοιπες πόλεις που υπάρχουν. Το διάβασμα και το γράψιμο του αρχείου του Telegraf γίνεται με τη χρήση του πακέτου fs (Σχήμα 5.16). Ακόμα για να εφαρμοστούν οι αλλαγές του configuration στο Telegraf service απαιτείται να επανεκκινηθεί με το child_process. Να σημειωθεί ότι για να διαβαστεί το αρχείο και μετέπειτα για να γραφτουν οι αλλαγές ορίζεται η κωδικοποίηση του σε “utf8”. Έχει προβλεφθεί ότι αν το service δεν καταφέρει να επανεκκινηθεί τότε επαναφέρεται το προηγούμενο configuration και επανεκκινείται ξανά ώστε να συνεχίσουν να αποθηκεύονται τα μετεωρολογικά δεδομένα των άλλων πόλεων(Σχήμα 5.17). Να σημειωθεί ότι αν η πόλη υπάρχει στη mongoDB βάση τότε απλά δημιουργείται μια νέα εγγραφή σε αυτήν με το id της πόλης και το userId(Σχήμα 5.18). Αυτες οι δύο λειτουργίες που αναφέρθηκαν παραπάνω δηλαδή η επεξεργασία αρχείων και service είναι ασύγχρονες και υπάρχει αντίστοιχο callback για να χειρίζεται το λάθος εφόσον συμβεί.

```
const fs = require('fs');
const axios = require('axios');
const {exec} = require('child_process');
const mongoose = require('mongoose');
const City = require('../models/city');
```

Σχήμα 5.11: Imports

```
exports.getCities = (req, res, next) =>{
    City.find({userId: mongoose.Types.ObjectId(req.userData.userId)})
    .then(cities => {
        res.status(200).json({
            message: "Cities fetched successfully!",
            cities: cities
        });
    })
    .catch(err => {
        res.status(500).json({
            message: "Failed to fetch cities!"
        });
    });
}
```

Σχήμα 5.12: Η συνάρτηση getCities

```
let apiResponse = await axios.get(`https://openweathermap.org/data/2.5/find?callback=
jQuery19107484579824848949_1589908196548
&q=${req.body.name}&type=like&sort=population&cnt=30
&appid=439d4b804bc8187953eb36d2a8c26a02&_=1589908196549`);
```

Σχήμα 5.13: Το get request με το πακέτο axios

```
let pos = apiData.indexOf("(");
pos++;
resposeData = apiData.substring(0, apiData.length-1).substring(pos);
let obj = JSON.parse(resposeData);
```

Σχήμα 5.14: Οι μετατροπές της απάντησης για να πάρει JSON μορφή

```
if (apiResponse.status != 200) {
    return res.status(500).json({
        message: apiResponse.statusText
    });
}
```

Σχήμα 5.15: Έλεγχος του http status

```
configData =  fs.readFileSync('/etc/telegraf/telegraf.conf', { encoding: 'utf8', flag: 'r'});

fs.writeFile('/etc/telegraf/telegraf.conf', configData, 'utf8', (err) => {
    if (err) return console.log(err);
});
```

Σχήμα 5.16: Οι μέθοδοι του πακέτου fs για την επεξεργασία αρχείων

```
exec("systemctl restart telegraf", (error, stdout, stderr) => {
    if (error) {
        //undo changes
        fs.writeFile('/etc/telegraf/telegraf.conf', old_config, 'utf8', (err) => {
            if (err) console.log(err);
        });
        exec("systemctl restart telegraf");
        return res.status(500).json({
            message: "Adding a city failed!"
        });
    }
    if (stderr) {
        //undo changes
        fs.writeFile('/etc/telegraf/telegraf.conf', old_config, 'utf8', (err) => {
            if (err) console.log(err);
        });
        exec("systemctl restart telegraf");
        return res.status(500).json({
            message: "Adding a city failed!"
        });
    }
    console.log(`stdout: ${stdout}`);
});
```

Σχήμα 5.17: Επανεκκίνηση του service με το πακέτο child_process

```
const city = new City({
    id:obj.list[0].id,
    name: obj.list[0].name,
    userId: req.userData.userId
});
city.save()
.then(result => {
    res.status(201).json({
        message: "City was added successfully!",
        result: result
    });
})
.catch(err =>{
    res.status(500).json({
        message: "Adding a city failed!"
    });
});
```

Σχήμα 5.18: Αποθήκευση της πόλης και επιστροφή μιας απάντησης

5.6.2 Influxdb controller

Σε αυτό το controller γίνεται import το πακέτο της Influx και δημιουργείται μια καινούρια σύνδεση με τη βάση δεδομένων InfluxDB. Σαν παράμετροι δίνονται η ip που εκτελείται το service της και το όνομα της βάσης δεδομένων(Σχήμα 5.19). Έπειτα εξετάζεται αν υπάρχει εναλλακτικά εμφανίζεται ένα μήνυμα λάθους(Σχήμα 5.20). Εν συνεχεία ορίζονται και οι τρεις μέθοδοι του controller. Η πρώτη εκτελεί ένα query(Σχήμα 5.21) στη βάση και αν είναι επιτυχές στέλνεται μια απάντηση με http status διακόσια και σε JSON μορφή ένα μήνυμα και τα πιο πρόσφατα μετεωρολογικά δεδομένα για την πόλη της Θεσσαλονίκης. Αν προκύψει κάποιο λάθος επιστρέφει μια απάντηση με http status πεντακόσια και ένα μήνυμα πως κάτι έχει αποτύχει. Παρόμοια λογική έχει και η επόμενη μέθοδος μόνο που το id της πόλης είναι route parameter και χρησιμοποιείται στο query(Σχήμα 5.22). Η χρήση της μεθόδου last στο πεδίο main_temp έχει ως στόχο να επιστρέφεται από τη βάση. Η τελευταία μέθοδος του controller εκτελεί ένα ερώτημα στη βάση και επιστρέφει όλα τα ονόματα των πόλεων που έχει αποθηκεύσει(Σχήμα 5.23). Σε όλα τα queries στο πεδίο name χρησιμοποιούνται διπλά εισαγωγικά διότι είναι λέξη-κλειδί στην InfluxQL και με αυτόν τον τρόπο η εκτέλεση είναι επιτυχής και επιστρέφει αποτέλεσμα. Τα ονόματα αυτά σχετίζονται με τη λειτουργία του προηγούμενου controller και πιο συγκεκριμένα με τη συνάρτηση που έγραφε στο configuration αρχείο του Telegraf. Όσες πόλεις υπάρχουν σε αυτό το αρχείο θα βρίσκονται στο αποτέλεσμα του query.

```
const influxClient = new influx.InfluxDB({
    host: '127.0.0.1',
    database: 'api_response'
});
```

Σχήμα 5.19: Δημιουργία ενός influxDB αντικειμένου

```
influxClient.getDatabaseNames()
.then(names => {
    if(names.includes('api_response')){
        console.log("Connected to InfluxDB");
    }
})
.catch(err => {
    console.error("Error database doesn't exists!!");
});
```

Σχήμα 5.20: Έλεγχος για την ύπαρξη της βάσης

```
influxClient.query(`select "name", sys_country, weather_0_icon,
last(main_temp), weather_0_description, id from http where id=734077`)
```

Σχήμα 5.21: Το query της πρώτης μεθόδου

```
influxClient.query(`select "name", sys_country, weather_0_icon,
last(main_temp), weather_0_description, id from http where id=`+ req.params.id)
```

Σχήμα 5.22: Το query με route parameter

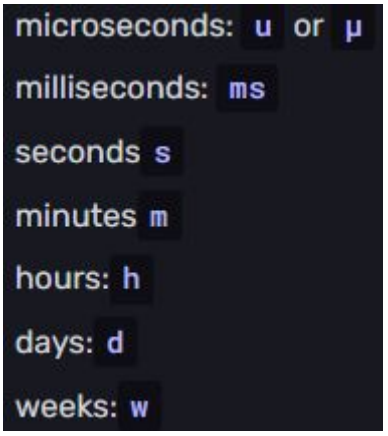
```
influxClient.query(`select distinct("id") as id from http group by "name"`)
```

Σχήμα 5.23: Το query της μεθόδου getInfluxCities

5.6.3 Stats controller

Όπως το προηγούμενο controller έτσι και αυτό θα χρησιμοποιήσει τη βιβλιοθήκη του Influx για το Node.js. Δημιουργείται ένα νέο instance για να συνδεθεί με τη βάση και επακολουθούν οι κατάλληλοι έλεγχοι. Παρακάτω υπάρχουν δύο συναρτήσεις που γίνονται export. Η πρώτη εκτελεί ένα ερώτημα στη βάση, το οποίο είναι δυναμικό και "χτίζεται" ανάλογα με τα route parameters του endpoint. Όταν το req.param.end είναι "99" τότε στο where clause του query γίνεται το now() - req.param.start. Αυτή η συνθήκη είναι αληθής όταν ο χρήστης κλικάρει κάποιο από τα τέσσερα κουμπιά (Last hour, Last day, Last week και Last month) στο UI του component statistics. Στη InfluxDB είναι εφικτό στο where να φιλτράρει δεδομένα που είναι είτε fileds είτε tags είτε timestamps. Κάθε ένα από τα τέσσερα κουμπιά ουσιαστικά περιορίζουν το πλήθος των δεδομένων που θα εμφανίσει η βάση εκτελώντας το query. Το φιλτράρισμα σε μια βάση χρονοσειρών θα γίνει με τον χρόνο (timestamp) που γίνεται από default σε κάθε νέα εγγραφή. Το timestamp παρότι είναι σε epoch μορφή η συνθήκη στο where είναι σε μορφή κατανοητή στον άνθρωπο και η InfluxDB κάνει τις μετατροπές στον χρόνο. Ακόμα παρέχεται η μέθοδος now που επιστρέφει την τρέχουσα ώρα του server και βοηθάει στο να υπολογίζεται πολύ εύκολα η σχετική ώρα. Πιο συγκεκριμένα υποστηρίζονται οι αριθμητικές πράξεις της πρόσθεσης και της αφαίρεσης, κάποιες σταθερές(Σχήμα 5.24). Για παράδειγμα το "time > now() - 1h" επιστρέφει τα δεδομένα που έγιναν εισαγωγή στη βάση μέσα στην τελευταία ώρα και τα κενά ανάμεσα στο "-" είναι απαραίτητα. Επιπλέον για να μην έρχεται μεγάλος όγκος δεδομένων χρησιμοποιείται το group by για να ομαδοποιούνται με βάση τον χρόνο. Όπως υπολογίζεται η σχετική διαφορά της ώρας σε σχέση με την τωρινή έτσι γίνεται και η ομαδοποίηση ανάλογα με την τιμή του req.param.start, όσο μεγαλύτερη τόσο και η τιμή req.param.period. Οι σταθερές στο σχήμα 5.24 ισχύουν και για το group by clause. Επειδή στην περίπτωση για την τελευταία ώρα το interval για να γράφονται τα δεδομένα είναι ανά δέκα λεπτά υπήρχε περίπτωση στο τελευταίο δεκάλεπτο της ώρας να μην επιστρέφεται τιμή και αυτό να προκαλεί σφάλμα στο front-end για τη εμφάνιση του γραφήματος συμπληρώνεται στο query η μέθοδος "fill(none)". Αυτή η μέθοδος αντικαθιστά τις κενές τιμές με την επιλογή που ορίζεται σαν παράμετρος. Το option none αφαιρεί τελείως το timestamp και την τιμή ή τιμές του select. Τέλος για να εμφανίζεται στο γράφημα ποιας πόλης είναι τα μετεωρολογικά δεδομένα εκτελείται και ένα query που επιστρέφει το όνομα και το id της με το limit 1 για να γυρίσει μια μόνο εγγραφή(Σχήμα 5.25). Εναλλακτικά αν η τιμή της req.param.end δεν είναι ενενήντα εννιά τότε εκτελείται άλλο query στη βάση και στο where clause υπάρχει ένα λογικό "και" και στο time ορίζονται τα όρια από το κατώτερο ως το ανώτερο(Σχήμα 5.26). Επίσης για να είναι πιο ευανάγνωστο το γράφημα και καλύτερες οι επιδόσεις του server ελέγχεται η διαφορά των ημερομηνιών στις μεταβλητές req.params.start και req.params.end. Αν είναι μεγαλύτερο των 25 ημερών τότε αλλάζει η σταθερά στο group by σε "1d"(Σχήμα 5.27). Η σύγκριση γίνεται αρχικά με

μετατροπή των string σε date και αποθηκεύεται η διαφορά των απόλυτων τιμών. Κατόπιν διαιρείται η διαφορά με τη σταθερά και με τη μέθοδο ceil στρογγυλοποιείται το αποτέλεσμα. Η άλλη συνάρτηση ακολουθεί την ίδια λογική αλλά το query υπολογίζει ταυτόχρονα για τη μέτρηση που επιλέγει ο χρήστης το max, mean και min(Σχήμα 5.28). Να αναφερθεί ότι αυτές οι συναρτήσεις στην InfluxDB μπορούν να εμπεριέχουν μόνο fields.



Σχήμα 5.24: Σταθερές της InfluxDB

```
influxClient.query(`select ` + req.params.fun + `(` + req.params.size + `)
from http where id = ` + req.params.id +
` and time > now() - ` + req.params.start + ` group by time(` + req.params.period + `)
 fill(none) ; select "name", id from http  where id =` + req.params.id + ` limit 1`)
```

Σχήμα 5.25: Το query για τα στατιστικά του καιρού

```
influxClient.query(`select ` + req.params.fun + `(` + req.params.size + `)
from http where id = ` + req.params.id + `
and time>='` + req.params.start + `'
and time<='` + req.params.end +
`' group by time(` + req.params.period + `) fill(none) ;
select "name", id from http where id =` + req.params.id + `  limit 1`)
```

Σχήμα 5.26: Το query με όρια στο πεδίο time

```
let start = Date.parse(req.params.start);
let end = Date.parse(req.params.end);
let diffTime = Math.abs(end - start);
const diffDays = Math.ceil(diffTime / (1000 * 60 * 60 * 24));
if (diffDays > 25 ) {
    req.params.period = "1d";
}
```

Σχήμα 5.27: Έλεγχος των μεταβλητών start και end

```
influxClient.query(`select  Max(` + req.params.size + `), Mean(` + req.params.size + `),
Min(` + req.params.size + `) from http where id = ` + req.params.id + `
and time>='` + req.params.start + `' and time<='` + req.params.end +
`' group by time(` + req.params.period + `) fill(none) ;
select "name", id from http where id =` + req.params.id + `  limit 1`)
```

Σχήμα 5.28: Το query επιστρέφει max, mean, min

5.6.4 User controller

Σε αυτό το controller υπάρχει η λογική για την αυθεντικοποίηση και υλοποιούνται οι μέθοδοι για το login και το register. Εισάγονται οι βιβλιοθήκες bcryptjs και jsonwebtoken, η πρώτη κάνει hash το password για να μην αποθηκεύεται στη βάση MongoDB ως απλό κείμενο για λόγους ασφάλειας. Η δεύτερη έχει τη δυνατότητα να δημιουργεί tokens τα οποία χρησιμοποιούνται για τα αιτήματα των χρηστών στο back-end και την αυθεντικοποίηση τους για τα protected routes αλλά και στα components στο front-end που έχουν route guards. Μετά τα imports εισάγεται και το μοντέλο User και ακολουθεί η μέθοδος createUser. Αρχικά από τη βιβλιοθήκη bcryptjs γίνεται χρήση της συνάρτησης hash με παραμέτρους το password που στέλνεται από το request και το μήκος για το hash που θα παραχθεί. ‘Υστερα δημιουργείται ένα νέο javascript αντικείμενο τύπου User, όπως το μοντέλο, και σε κάθε πεδίο ορίζεται η αντίστοιχη τιμή. Αυτό το αντικείμενο αποθηκεύεται στη βάση δεδομένων MongoDB και ανάλογα με το αποτέλεσμα της ασύγχρονης λειτουργίας επιστρέφεται μια απάντηση(Σχήμα 5.29). Η δεύτερη συνάρτηση ονομάζεται userLogin(Σχήμα 5.30), η οποία με το μοντέλο User γίνεται αναζήτηση στη βάση με το username του αιτήματος. Αν δεν υπάρξει κάποια εγγραφή τότε επιστρέφεται μήνυμα λάθους με κωδικο το “401”, εναλλακτικά αποθηκεύεται το αποτέλεσμα σε μια μεταβλητή και συγκρίνεται το password του αιτήματος με το hash που έχει η μεταβλητή, δηλαδή της βάσης. Η μέθοδος compare της bcryptjs επιστρέφει ένα promise για την σύγκριση των δύο τιμών. Όπως και πριν αν δεν συμπίπτουν τότε η εκτέλεσης της συνάρτησης τερματίζει και επιστρέφεται μήνυμα με http status 401 λόγω της αποτυχημένης αυθεντικοποίησης. Στην περίπτωση όμως που συμπίπτουν ο κωδικός με το hash δημιουργείται ένα JSON web token με τη συνάρτηση sign που παίρνει ένα payload, ένα secret και options. Το payload είναι JSON και περιέχει τις πληροφορίες του χρήστη που το δημιουργεί. Το secret είναι ένα string σαν password με βάση αυτό δημιουργείται το hash του token και αργότερα με αυτό θα γίνεται το verify. Το secret κρατείται μόνο στον server και πρέπει να είναι μεγάλο σε μήκος. Η τελευταία παράμετρος είναι το options για έξτρα παραμετροποιήσεις, παρόλα αυτά θα χρησιμοποιηθεί μόνο το expiresIn στο οποίο ορίζεται σε πόση ώρα λήγει το token, όπως “1h” και δεν θα είναι έγκυρο. Αφού ολοκληρωθεί όλη αυτή η διαδικασία του token τότε επιστρέφεται στο client για να χρησιμοποιηθεί σε επόμενα requests. Ακόμα στην απάντηση στέλνεται στο πεδίο “expireIn” ο χρόνος σε δευτερόλεπτα. Αυτό το πεδίο είναι απαραίτητο στο Angular για να κάνει αυτόματα logout τον χρήστη κατόπιν το πέρας της μίας ώρας, αυτό έχει υλοποιηθεί στο auth service στο προηγούμενο κεφάλαιο.

```
exports.createUser = (req, res, next) => {
    bcrypt.hash(req.body.password, 10)
    .then(hash => {
        const user = new User({
            username: req.body.username,
            email: req.body.email,
            password: hash
        });
        user.save()
        .then(result => {
            res.status(201).json({
                message: "User created!",
                result: result
            });
        })
        .catch(err => {
            res.status(500).json({
                message: "Invalid authentication credentials!"
            });
        });
    });
}
```

Σχήμα 5.29: Μέθοδος createUser

```
exports.userLogin = (req, res, next) => {
    let fetchedUser;
    User.findOne({ username: req.body.username})
        .then(user => {
            if (!user) {
                return res.status(401).json({
                    message: "Auth failed!"
                });
            }
            fetchedUser = user;
            return bcrypt.compare(req.body.password, user.password)
                .then(result => {
                    if (!result) {
                        return res.status(401).json({
                            message: "Auth failed!"
                        });
                    }
                    const token = jwt.sign({username: fetchedUser.username, userId: fetchedUser._id},
                    'secret_for_web_token',
                    { expiresIn: "1h" });
                    res.status(200).json({
                        token: token,
                        expiresIn: 3600
                    });
                })
                .catch(err => {
                    return res.status(401).json({
                        message: "Invalid authentication credentials!"
                    });
                });
    });
```

Σχήμα 5.30: Η μέθοδος userLogin

5.7 Middleware

Το express είναι ένα routing και middleware framework, το οποίο έχει ελάχιστη λειτουργικότητα από μόνο του. Τα middlewares είναι συναρτήσεις που έχουν πρόσβαση στο request, response και next αντικείμενα. Το next είναι μια συνάρτηση του express router που όταν καλείται εκτελεί το επόμενο middleware από το τρέχων. Αν στο middleware δεν ολοκληρωθεί ο κύκλος με του request-response, δηλαδή δεν επιστραφεί κάποια απάντηση προς το client τότε πρέπει να καλείται η μέθοδος next για να συνεχίσει η εκτέλεση των υπόλοιπων middlewares(Σχήμα 5.31).

```
app.use(function (req, res, next) {
  console.log('Time:', Date.now())
  next()
});
```

Σχήμα 5.31: Παράδειγμα με κλήση της μεθόδου next

5.7.1 Check auth middleware

Στην εισαγωγή είχε σημειωθεί ότι το RESTful API είναι stateless και είναι σαφές ότι κάποιες λειτουργίες στο endpoints δεν πρέπει να είναι διαθέσιμες για όλους τους χρήστες. Στην ενότητα που αφορά τα routes και είναι χωρισμένα με βάση την λειτουργικότητα τους είχε αναφερθεί ότι κάποια

από αυτά είναι protected και δίχως αυθεντικοποίηση ή εξουσιοδότηση δεν εκτελείται η συνάρτηση του controller. Τα protected routes έχουν μετά το URL path το middleware checkAuth(Σχήμα 5.32), το οποίο αναλύει το εισερχόμενο αίτημα ελέγχοντας το πεδίο "Authorization" και κατά συνέπεια το token που περιέχει. Είναι πιθανό να υπάρχει κάποιο token ενσωματωμένο στο request αλλά να μην είναι έγκυρο, γι' αυτό το λόγο πρέπει να επαληθευτεί ώστε να συνεχίσει η εκτέλεση του αιτήματος ή να απορριφθεί επιστρέφοντας κάποιο μήνυμα. Αρχικά εισάγεται το πακέτο "jsonwebtoken" και για να μπορεί να χρησιμοποιείται το middleware στα άλλα αρχεία γίνεται export. Το req αντικείμενο μέσω του Express δίνει πρόσβαση στα headers και στο "Authorization". Αποθηκεύεται σε μια μεταβλητή το token, το οποίο αρχικά έχει τη μορφή "Bearer token". Από το πακέτο JsonWeBToken χρησιμοποιείται η συνάρτηση verify που δέχεται το token και το secret ως παραμέτρους. Το secret είναι το ίδιο string που ορίζεται και στο controller στη μέθοδο userLogin(βλέπε σχήμα 5.30). Έπειτα στο request αντικείμενο προστίθεται ένα πεδίο που είναι και αυτό με τη σειρά του javascript object και περιέχει το username και το userId του χρήστη που στέλνει το αίτημα. Αυτές οι τιμές είναι διαθέσιμες επειδή το token η μέθοδος verify το αποκρυπτογραφεί. Θυμίζεται πως όταν δημιουργείται ένα token έχει ένα payload με τα εξής πεδία. Σε περίπτωση όμως που το token δεν επαληθευτεί τότε γυρνάει το res αντικείμενο στο client με status code το "401" και το μήνυμα του. Αφού ολοκληρωθούν επιτυχώς οι έλεγχοι και η αλλαγή καλείται η μέθοδος next για να εκτελεστεί το επόμενο middleware, δηλαδή η μέθοδος του controller από το αντίστοιχο endpoint.

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

module.exports = (req, res, next) => {
    try {
        const token = req.headers.authorization.split(" ")[1];
        const decodedToken = jwt.verify(token, 'secret_for_web_token');
        req.userData = {
            username: decodedToken.username,
            userId: decodedToken.userId
        };
        next();
    } catch (error) {
        res.status(401).json({ message: "You are not authenticated!"});
    }
}
```

Σχήμα 5.32: Check-auth middleware

5.8 Models

Η MongoDB δεν είναι σχεσιακή όπως αναφέρθηκε πολλές φορές, αλλά μπορεί να συνδυαστεί με ένα Object Data Model ή Object Relation Model[28]. Το mongoose είναι ORM και μπορεί να δημιουργήσει Javascript αντικείμενα για τα documents της βάσης δεδομένων. Το πλεονέκτημα της χρήσης ORM είναι ότι το μοντέλο μπορεί να αλλάζει ανάλογα τις απαιτήσεις της εφαρμογής και ο προγραμματιστής να σκέφτεται μόνο με τους όρους της Javascript, χωρίς τις συσχετίσεις της βάσης δεδομένων. Το μοντέλο που δημιουργείται από το Mongoose είναι ένα σχεδιάγραμμα για το αντικείμενο που θα αποθηκεύεται στη βάση ως collection. Ο συνδυασμός του Node.js της MongoDB και του Mongoose είναι αρκετά δημοφιλής επειδή η αποθήκευση και η εκτέλεση ερωτημάτων

λειτουργούν με JSON. Επιπλέον, ένα όφελος είναι ότι τα δεδομένα μπορούν να γίνουν εύκολα επαλήθευση και να αποθηκεύονται στη βάση χωρίς λάθη. Στην παρούσα εφαρμογή υπάρχουν δύο μοντέλα , το City(Σχήμα 5.33) και το User(Σχήμα 5.34). Στο αρχείο city που βρίσκεται στο φάκελο models γίνεται import το πακέτο mongoose και δημιουργείται το citySchema που είναι σε JSON μορφή, με πεδία το id, το name και το userId. Το id είναι αριθμός, αντιστοιχίζεται με το id του OpenWeatherMap API. Το name είναι string και περιέχει το όνομα της πόλης. Τέλος το πεδίο userId είναι ObjectId τύπος από το mongoose και έχει αναφορά στο άλλο μοντέλο, το User. Όλα τα πεδία είναι required που σημαίνει ότι για να αποθηκευτεί μια νέα εγγραφή αυτού του μοντέλου πρέπει αυτό να είναι παρών και να έχει τιμή. Το δεύτερο μοντέλο που ονομάζεται User αποτελείται από τα πεδία username, email και password που είναι όλα τύπου string. Το password είναι required, ενώ τα άλλα δύο πεδία είναι unique. Το unique διασφαλίζει ότι σε αυτό το πεδίο δεν θα αποθηκεύσει διπλές τιμές. Σε περίπτωση προσπάθειας εισαγωγής δεδομένων με διπλή τιμή, η διαδικασία προκαλεί σφάλμα και διατηρείται ο περιορισμός της μοναδικότητας. Τέλος, αν και σε κανένα από τα δύο μοντέλα δεν έχει οριστεί το πεδίο _id η MongoDB για κάθε νέο collection το προσθέτει και από default είναι index. Αυτό βοηθάει στην γρηγοροτερη αναζήτηση εγγραφών στο κάθε collection.

```
const mongoose = require('mongoose');

const citySchema = mongoose.Schema({
    id: {type: Number, required: true },
    name: {type: String, required: true },
    userId: {type: mongoose.Schema.Types.ObjectId, ref: "User", required: true}
});

module.exports = mongoose.model("City", citySchema);
```

Σχήμα 5.33: City model

```
const mongoose = require('mongoose');

const userSchema = mongoose.Schema({
  username: {type: String, required: true, unique: true},
  email: {type: String, unique: true},
  password: {type: String, required: true}
});

module.exports = mongoose.model("User", userSchema);
```

Σχήμα 5.34: User model

5.9 Επίλογος

Σε αυτό το κεφάλαιο περιγράφηκε η υλοποίηση ενός RESTful API με χρήση της Node.js και με δύο βάσεις δεδομένων, όπου η μία περιέχει τα μετεωρολογικά δεδομένα και η δεύτερη πληροφορίες για τους χρήστες της εφαρμογής. Το API που δημιουργήθηκε είναι συμβατό με διάφορα front-end pages και η επικοινωνία πρέπει να γίνεται μέσω του HTTP πρωτοκόλλου. Κάθε request και response που

ανταλλάσουν οι δύο οντότητες (front-end & back-end) γίνεται με JSON αντικείμενα και κάθε απάντηση επιστρέφει με ένα http status για να κατανοεί ο προγραμματιστής το αποτέλεσμα του request. Το μεγαλύτερο μέρος του application logic υλοποιείται στην πλευρά του server όπου κρατούνται και τα δεδομένα της εφαρμογής για περισσότερη ασφάλεια. Το μόνο που είναι προσβάσιμο από το back-end σε μια εφαρμογή είναι τα endpoints. Μέσω αυτών εκτελούνται οι λειτουργίες στον server αλλά και πάλι κάποια από αυτά προστατεύονται και δεν είναι προσβάσιμα σε όλους τους χρήστες. Η αυθεντικοποίηση των χρηστών γίνεται με token που κάνει encrypt τα στοιχεία του εκάστοτε χρήστη. Η δομή του back-end είναι παρόμοια και με άλλες γλώσσες προγραμματισμού και με βάση αυτή μπορεί να δημιουργηθούν και άλλα projects διαφορετικού σκοπού.

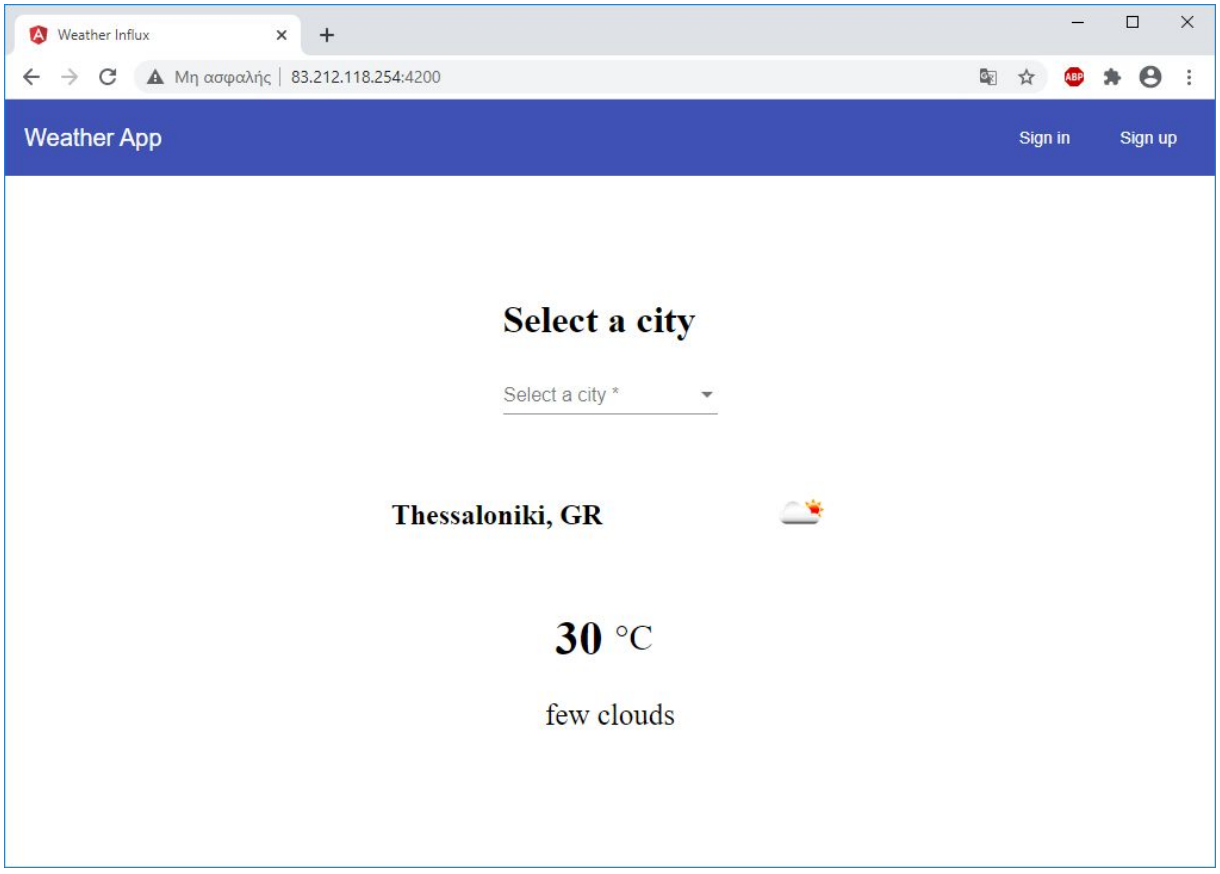
Κεφάλαιο 6ο: Εγχειρίδιο χρήσης

6.1 Εισαγωγή

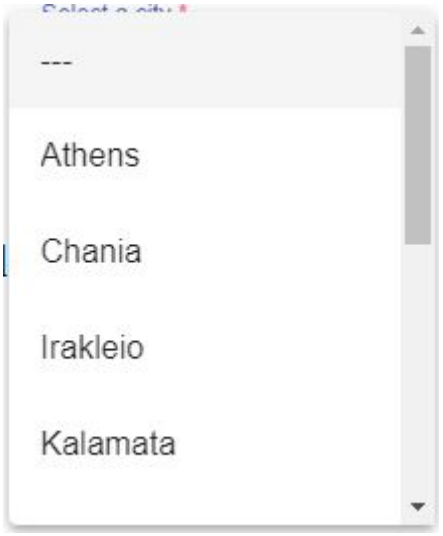
Σε αυτό το κεφάλαιο θα παρουσιαστούν βήμα-βήμα οι ενέργειες που πρέπει να εκτελεί ο χρήστης για να χρησιμοποιήσει την εφαρμογή ορθά.

6.2 Ροή λειτουργίας

Αρχικά, ανοίγοντας ένα browser πληκτρολογώντας στο URL το "http://83.212.118.254:4200/" ο χρήστης πλοηγείται στην αρχική σελίδα(Σχήμα 6.1), όπου εμφανίζονται οι πιο πρόσφατες πληροφορίες για τον καιρό της Θεσσαλονίκης. Υπάρχει η επιλογή να διαλέξει μια άλλη πόλη που υπάρχει ήδη στο σύστημα και να του εμφανίζει τις αντίστοιχες πληροφορίες(Σχήμα 6.2). Διαφορετικά ο χρήστης μπορεί να συνδεθεί στην εφαρμογή ή να εγγραφεί αν δεν έχει ήδη λογαριασμό. Αυτές οι λειτουργίες μπορούν να εκτελεστούν να κλικαριστεί ένα από τα δύο κουμπιά πάνω δεξιά.

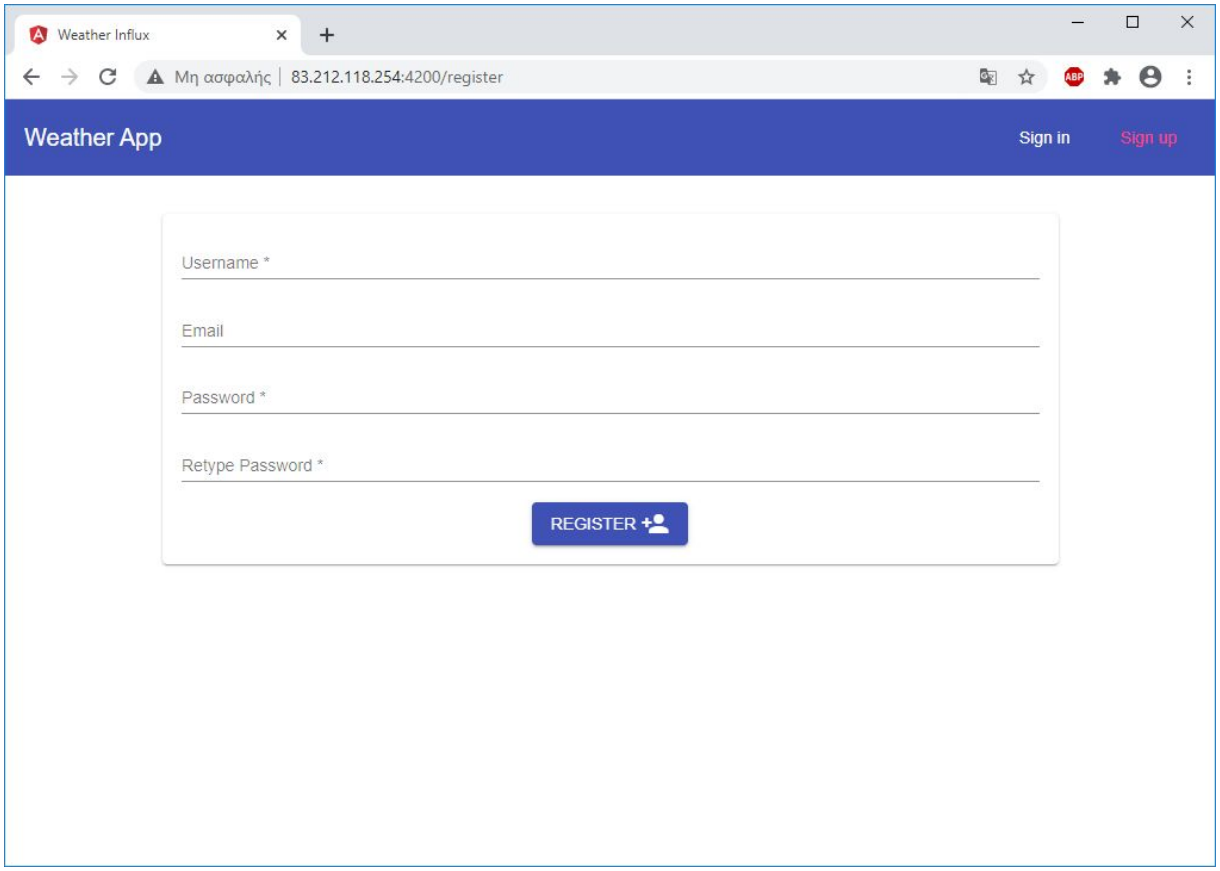


Σχήμα 6.1: Αρχική σελίδα



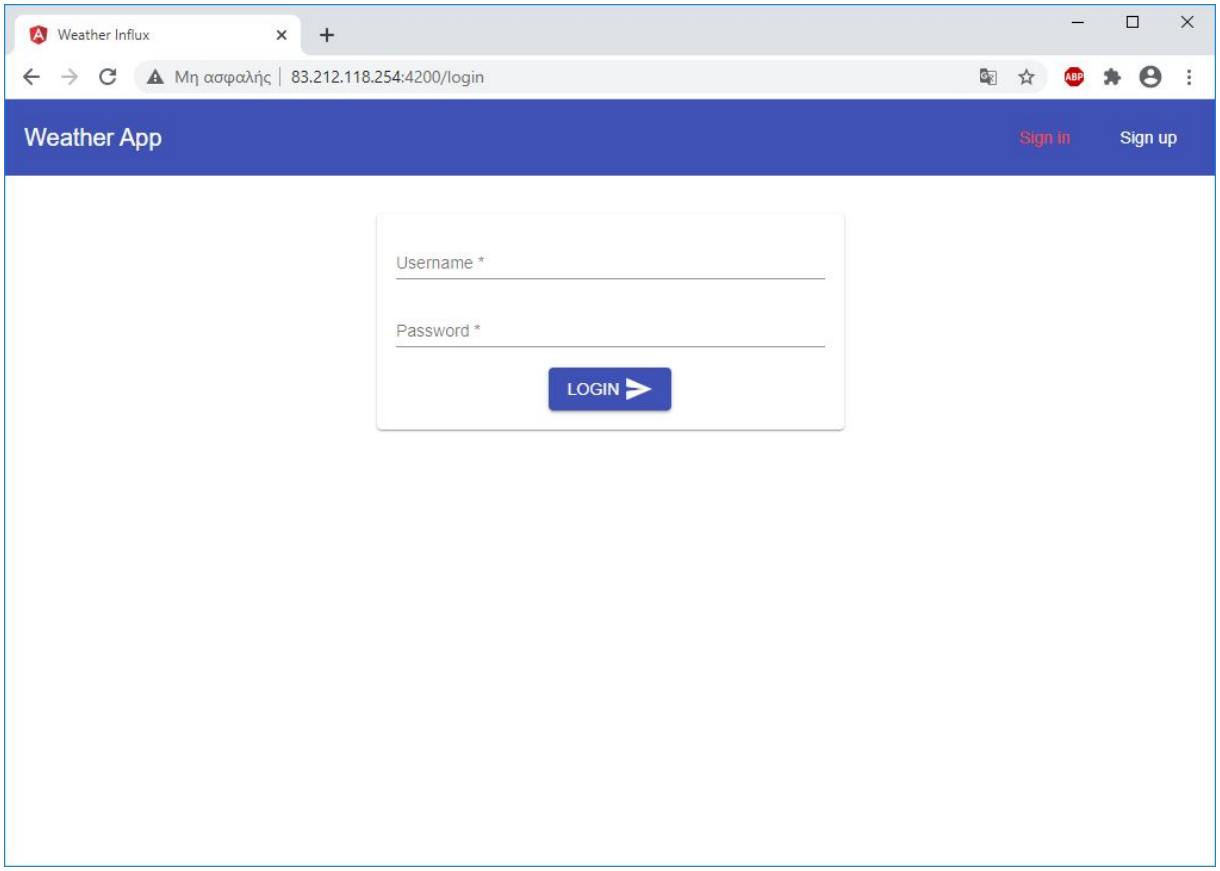
Σχήμα 6.2: Υπόλοιπες επιλογές πόλεων

Για τη δεύτερη περίπτωση που κλικαριστεί το “sign up” ο χρήστης μεταβαίνει στην σελίδα εγγραφής(Σχήμα 6.3), που πρέπει να συμπληρώσει τα απαιτούμενα πεδία(τα πεδία με τον αστερίσκο) και επιλέγει το κουμπί “register”. Αυτό το βήμα πρέπει να εκτελεστεί μία μόνο φορά και αν υπάρχει το username σε χρήση ήδη από κάποιον άλλον τότε εμφανίζεται σφάλμα. Τα usernames είναι μοναδικά. Αφού ολοκληρωθεί η εγγραφή ο χρήστης μεταβαίνει στη σελίδα σύνδεσης(login) που θα αναλυθεί παρακάτω.



Σχήμα 6.3: Σελίδα εγγραφής

Στη σελίδα του login τώρα μπορεί ο χρήστης να μεταβεί από την αρχική σελίδα μέσω του κουμπιού "sign in" είτε αν έχει κάνει μόλις εγγραφή μεταφέρεται αυτόματα. Εκεί πρέπει να πληκτρολογούνται τα στοιχεία εισόδου και έπειτα να επιλέγεται το κουμπί "login". Τότε αν τα στοιχεία που εισήγαγε ο χρήστης είναι σωστά, τότε μεταφέρεται πάλι στην αρχική σελίδα πλέον οι επιλογές πάνω στο header είναι διαφορετικές. Υπάρχει το "Statistics", το "Cities" και δεξιά το κουμπί "logout" για την αποσύνδεση του χρήστη. Τα δύο πρώτα είναι δύο διαφορετικές σελίδες και θα αναλυθούν παρακάτω για την λειτουργία που εκτελούν. Να σημειωθεί ότι ο χρήστης από την στιγμή που κάνει login στην εφαρμογή μένει συνδεδεμένος μέχρι τη μία ώρα, έκτος αν νωρίτερα έχει πατήσει το κουμπί logout.

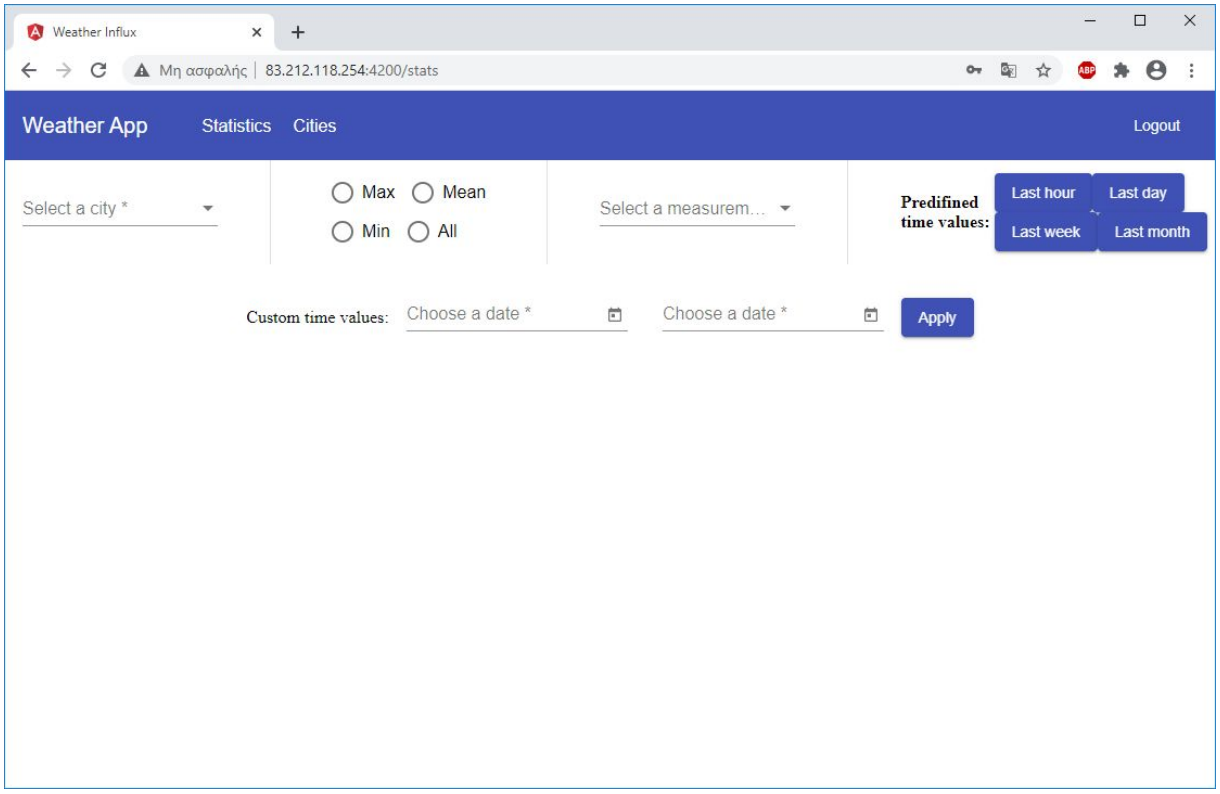


Σχήμα 6.4: Σελίδα login

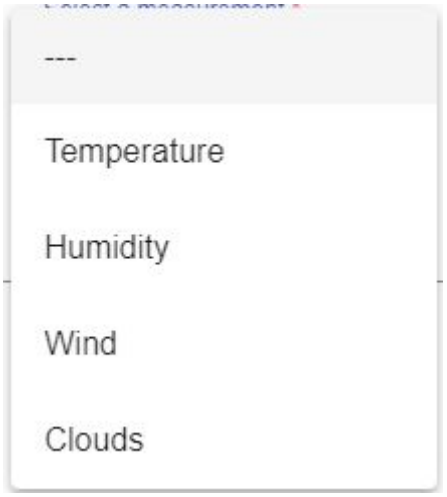
Σχήμα 6.5: Το header όταν ο χρήστης είναι authenticated

Η σελίδα Statistics(Σχήμα 6.6), εφόσον την έχει επιλέξει ο χρήστης, περιέχει από αριστερά προς τα δεξιά ένα dropdown select με πόλεις που υπάρχουν στην εφαρμογή, ένα radio button με τις διαθέσιμες συναρτήσεις, ένα άλλο dropdown select με τα μεγέθη που συλλέγονται(Σχήμα 6.7), τέσσερα κουμπιά και από κάτω υπάρχουν δύο datepickers και ένα κουμπί "Apply" δίπλα τους. Εδώ υπάρχουν δύο επιλογές για τον χρήστη. Η πρώτη είναι να επιλέξει μια πόλη από το dropdown select, μια μέθοδο από τα radio buttons, μία μέτρηση από το dropdown select και τέλος ένα από τα τέσσερα κουμπιά(Last hour, Last day, Last week, Last month). Τότε κάτω από τα datepickers θα εμφανιστεί ένα γράφημα με βάση τα στοιχεία που επιλέχθηκαν. Το γράφημα σε κάθε περίπτωση στην στήλη X θα εμφανίσει τις ημερομηνίες από το χρονικό εύρος του κουμπιού που πατήθηκε και στον άξονα Y η μονάδα μέτρησης. Ακόμα αν και ο χρήστης δεν εισήγαγε κάποια τιμή στα datepickers αυτά συμπληρώνονται αυτόματα με τις σωστές ημερομηνίες(Σχήμα 6.8). Η δεύτερη επιλογή διαφοροποιείται από την πρώτη στο τελευταίο βήμα που αντί για κάποιο από τα τέσσερα κουμπιά, ορίζονται από τον χρήστη οι ημερομηνίες στα datepickers(Σχήμα 6.9) και τέλος πρέπει κλικάρετε το κουμπί "Apply" για να

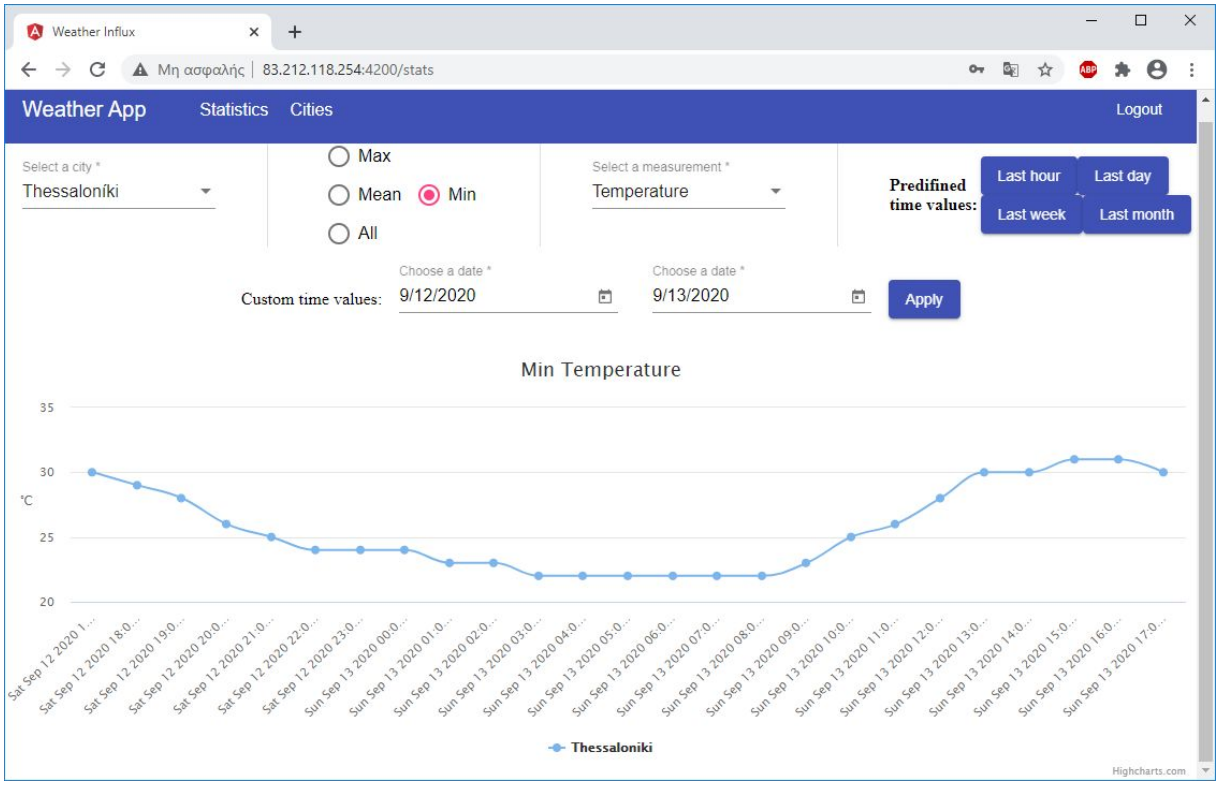
εμφανιστεί το γράφημα. Η ημερομηνία για την αρχή των μετεωρολογικών δεδομένων είναι η 24 Δεκεμβρίου 2019 για τις τρεις πόλεις(Αθήνα, Θεσσαλονίκη, Πάτρα).



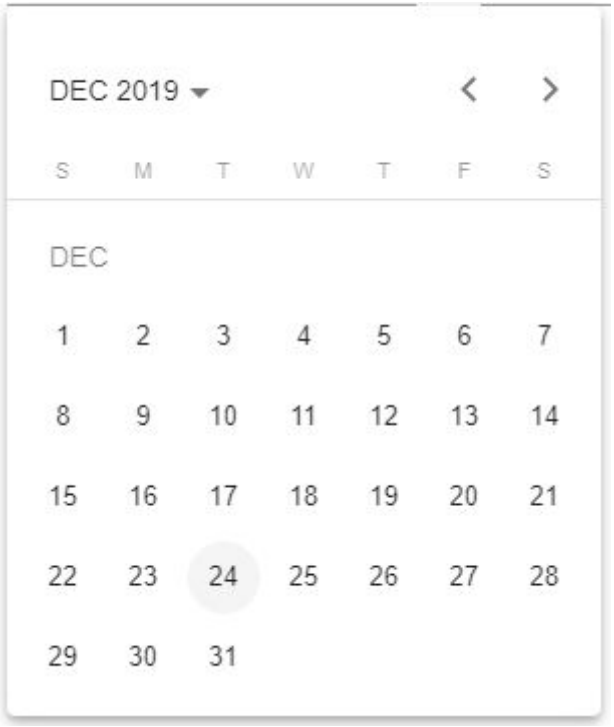
Σχήμα 6.6: Σελίδα statistics



Σχήμα 6.7: Οι διαθέσιμες μετρήσεις

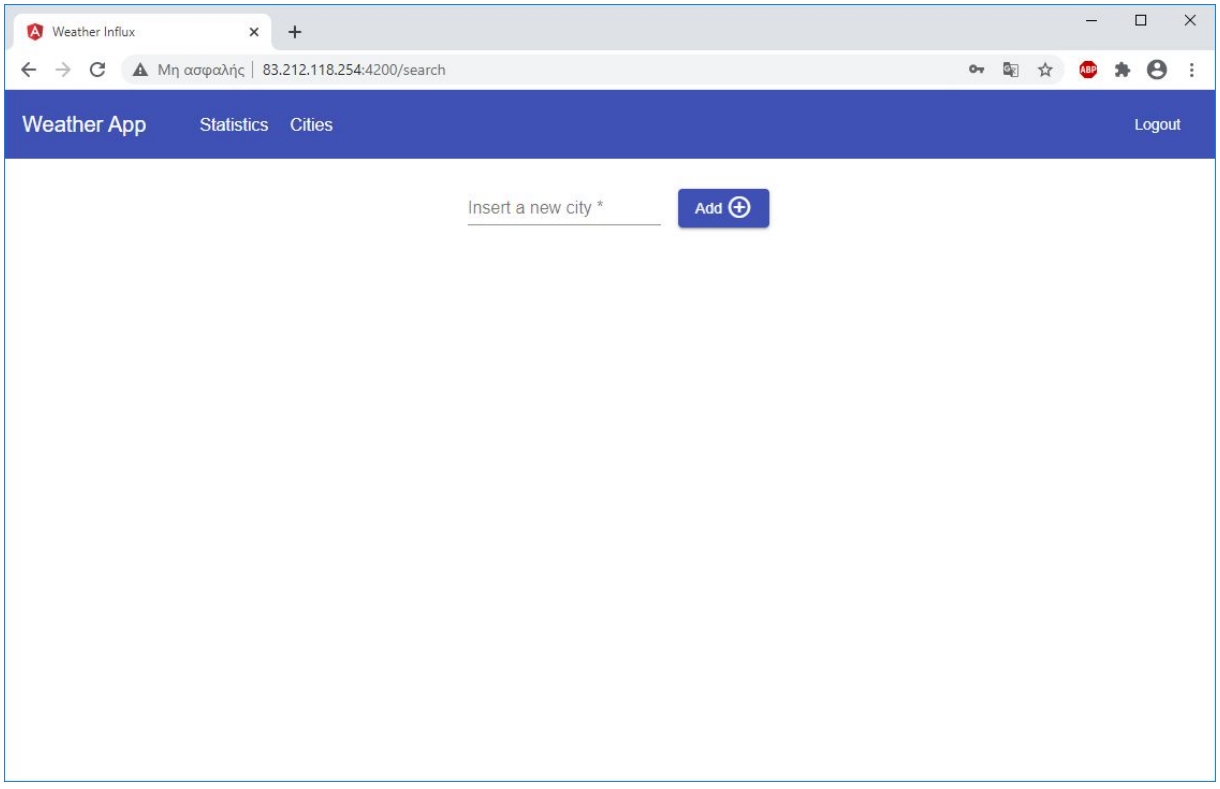


Σχήμα 6.8: Πρώτη επιλογή



Σχήμα 6.9: Datepicker

Τέλος, υπάρχει η σελίδα Cities, η οποία κάνει αρχικά αναζήτηση την πόλη που εισάγεται στο πεδίο από τον χρήστη και ανάλογα με το αποτέλεσμα της αναζήτησης εμφανίζεται αντίστοιχο μήνυμα για δύο δευτερόλεπτα. Από τη στιγμή που η εισαγωγή της πόλης είναι σωστή τότε μέσα σε διάστημα δέκα λεπτών θα έχει αρχίσει η καταγραφή μετεωρολογικών δεδομένων. Τότε αν ο χρήστης μεταβεί στη σελίδα statistics στο dropdown select των πόλεων θα εμφανίζεται και αυτή η πόλη, όπως και στην αρχική σελίδα στο αντίστοιχο dropdown select. Τέλος ο χρήστης μπορεί οποιαδήποτε στιγμή να πατήσει το κουμπί "logout" και ύστερα να κλείσει τον browser για να τερματίσει τη λειτουργία.



Σχήμα 6.10: Σελίδα cities

Κεφάλαιο 7ο: Συμπεράσματα και προτάσεις βελτίωσης

Με το πέρας αυτής της Π.Ε βγήκαν ως συμπέρασμα ότι η Influx είναι μια πανίσχυρη βάση δεδομένων που παρέχει wrappers για πολλές γλώσσες προγραμματισμού ώστε να διευκολύνει τους προγραμματιστές να αναπτύσσουν εφαρμογές και με πληθώρα έτοιμων μεθόδων ώστε να αφοσιώνονται στο project τους. Ακόμα η Node.js είναι μια πολύ εύχρηστη, εύκολη για να την μάθει κάποιος και η σύνδεση με το front-end ήταν επίσης πολύ εύκολη. Το Angular framework από την άλλη πλευρά είναι πιο σύνθετο και απαιτεί περισσότερο χρόνο από τον προγραμματιστή για να το μάθει. Μολονότι είναι αρκετά σύνθετο παρέχει πάρα πολλά εργαλεία για να μην χρειάζεται να τα υλοποιεί ο προγραμματιστής. Αν και η javascript με τη typescript είναι διαφορετικές σε μεγάλο βαθμό έχουν ίδιες συναρτήσεις και η εκμάθηση της typescript δεν είναι δύσκολη. Τέλος, η MongoDB σε συνδυασμό με το ORM είναι απλά στη χρήση και η δυνατότητα να χρησιμοποιείται παντού τα JSON αντικείμενα ήταν αρκετά βολικό για την διαχείριση των δεδομένων. Σαν προτάσεις βελτίωσης υπάρχουν κάποιες. Πρώτα από όλα, αν και το project είναι λειτουργικό υπάρχουν κάποια σημεία που μπορεί να γίνει refactor ο κώδικας όπως το αρχείο στο component statistics. Επιπλέον, στην αρχική σελίδα μπορούν να προστεθούν οι πληροφορίες για τον άνεμο, την υγρασία και την αίσθηση θερμοκρασίας. Στους τρόπους σύνδεσης μελλοντικά μπορεί να υπάρχει και η επιλογή να συνδέεται ο χρήστης και με το Google λογαριασμό, δίχως να χρειάζεται να δημιουργεί καινούριο. Τέλος, θα ήταν δυνατό να προστεθεί μια σελίδα παραπάνω για να παρουσιάζει τις προβλέψεις του καιρού για τις μέρες που έπονται, διότι υπάρχουν στο API του OpenWeatherMap και αυτές οι πληροφορίες.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Internet Site

[1] Influx Data [Online]. Available: https://www.influxdata.com/time-series-platform/

[2] Influx Data Documentation [Online]. Available: https://docs.influxdata.com/influxdb/v1.7/introduction/installation/

[3] Influx Data Documentation [Online]. Available: https://docs.influxdata.com/telegraf/v1.9/introduction/installation/

[4] Nodejs [Online]. Available: https://nodejs.dev/learn

[5] Nodejs [Online]. Available: https://nodejs.org/en/download/current/

[6] MongoDB Documentation [Online]. Available: https://docs.mongodb.com/v4.2/tutorial/install-mongodb-on-ubuntu/

[7] Npmjs [Online]. Available: https://www.npmjs.com/about

[8] Codecademy[Online]. Available: https://www.codecademy.com/articles/what-is-an-ide

[9] Git [Online]. Available: https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

[10] Visual Studio Code [Online]. Available: https://code.visualstudio.com/docs/editor/versioncontrol

[11] Angular [Online]. Available: https://angular.io/guide/router

[12] Angular [Online]. Available: https://angular.io/guide/interpolation

[13] Angular [Online]. Available: https://angular.io/guide/comparing-observables

[14] Angular [Online]. Available: https://angular.io/guide/observables-in-angular

[15] Angular [Online]. Available: https://angular.io/guide/architecture-services

[16] Angular [Online]. Available: https://angular.io/guide/http

[17] Angular [Online]. Available: https://angular.io/api/common/http/HttpClient

[18] Angular [Online]. Available: https://angular.io/guide/pipes

[19] Thinkster [Online]. Available: https://thinkster.io/tutorials/typescript-models-angular-2

[20] Medium [Online]. Available: https://medium.com/angular-in-depth/top-10-ways-to-use-interceptors-in-angular-db450f8a62d6

[21] Medium [Online]. Available: https://medium.com/@ryanchenkie_40935/angular-authentication-using-route-guards-bf7a4ca13ae3

[22] Influx Data Documentation [Online]. Available: https://docs.influxdata.com/influxdb/v1.8/concepts/key_concepts/

[23] MongoDB Documentation [Online]. Available: https://docs.mongodb.com/manual/introduction/

[24] MongoDB Documentation [Online]. Available: https://docs.mongodb.com/manual/crud/

[25] Mozilla Developer [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes

[26] Mozilla Developer [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises

[27] Mozilla Developer [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function

[28] Mozilla Developer [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/mongoose