

Μελέτη του Serverless Computing με το FN Project

Νικόλαος Σταματελόπουλος
Σχολή Ηλεκτρολόγων Μηχανικών και
Μηχανικών Υπολογιστών
Εθνικό Μετσόβιο Πολυτεχνείο
Αθήνα
el16138@mail.ntua.gr

Μπίτσικας Γρηγόριος
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Εθνικό Μετσόβιο Πολυτεχνείο
Αθήνα
el16693@mail.ntua.gr

Αλέξανδρος Κοντογιάννης
Σχολή Ηλεκτρολόγων Μηχανικών και
Μηχανικών Υπολογιστών
Εθνικό Μετσόβιο Πολυτεχνείο
Αθήνα
el15048@mail.ntua.gr

Σύνοψη: Σε αυτήν την εργασία μελετήθηκε η έννοια του *Serverless Computing* και έγινε υλοποίηση απλών παραδειγμάτων *serverless* εφαρμογών σε διάφορες γλώσσες προγραμματισμού με χρήση της πλατφόρμας *Fn Project*.

ενημέρωση και εκτέλεση (με ή χωρίς δεδομένα εισαγωγής) έγινε κατά κύριο με την χρήση του *terminal* που παρέχεται από το λειτουργικό σύστημα. Σε αυτό το σημείο θα ήταν καλό να επισημανθεί ότι ειδικά για την περίπτωση που θέλουμε να “τρέξουμε” κάποια συνάρτηση απουσία ή παρουσία δεδομένων εισόδου το *FN Project* παρέχει την δυνατότητα να το πράξουμε και μέσω του ειδικού γραφικού περιβάλλοντος που μας παρέχει και θα το παρουσιάσουμε ευθύς αμέσως.

I ΕΙΣΑΓΩΓΗ

Σκοπός της εργασίας μας ήταν η ανάλυση του *Serverless Computing* με την βοήθεια της πλατφόρμας *FN Project* η οποία χρησιμοποιείται για την ανάπτυξη *Serverless* εφαρμογών.

Για τον σκοπό αυτό εγκαταστήσαμε και παραμετροποιήσαμε κατάλληλα την πλατφόρμα ώστε να λειτουργεί με σωστό τρόπο στους προσωπικούς μας υπολογιστές.

Κατόπιν δημιουργήσαμε συναρτήσεις στις γλώσσες προγραμματισμού *Python*, *C#*, *Java*, *Go*, *Ruby* και *Node JS*.

Στο *FN Project* οι συναρτήσεις (functions) ομαδοποιούνται σε εφαρμογές (applications). Οπότε κάθε συνάρτηση ανήκει σε μια ή περισσότερες εφαρμογές. Βέβαια μια εφαρμογή μπορεί να περιέχει περισσότερες από μια συναρτήσεις οι οποίες δεν είναι απαραίτητο να είναι όλες στην ίδια γλώσσα προγραμματισμού.

Στην παρούσα αναφορά θα παρουσιάσουμε και θα περιγράψουμε αναλυτικά την υποδομή και το λογισμικό που χρησιμοποιήσαμε για την σύνταξη και την εκτέλεση των *serverless* συναρτήσεων και εφαρμογών που δημιουργήσαμε κατά την εκπόνηση της εργασίας αυτής. Επιπρόσθετα θα γίνει εκτεταμένη ανάλυση και επίδειξη των αποτελεσμάτων και των συμπερασμάτων τα οποία προέκυψαν από την όλη διαδικασία.

Τέλος θα επισυναφθεί και η τοποθεσία όπου βρίσκεται ο πλήρης κώδικας που παρήχθη και στον οποίο στηρίζεται η παρούσα εργασία.

II ΠΡΟΕΤΟΙΜΑΣΙΑ ΣΥΣΤΗΜΑΤΟΣ & ΠΕΙΡΑΜΑΤΩΝ

Όλα τα εργαλεία που χρησιμοποιήθηκαν εγκαταστάθηκαν σε λειτουργικό σύστημα *Linux (Ubuntu)*. Αρχικά εγκαταστήσαμε το *Docker* και κατόπιν το *FN Project* (με χρήση της εντολής από το *terminal* `curl -LSS https://raw.githubusercontent.com/fnproject/cli/master/install | sh`) κάνοντας το download από το επίσημο repository στο *GitHub*.

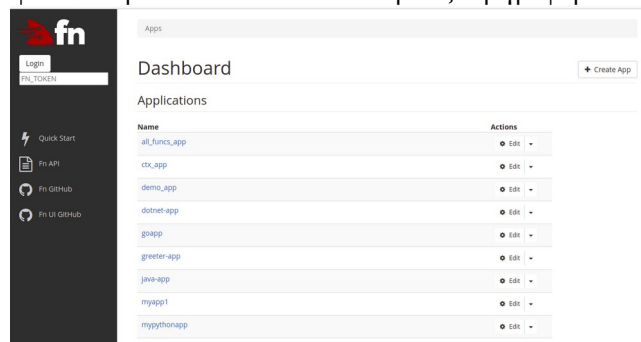
III ΥΠΟΔΟΜΗ & ΛΟΓΙΣΜΙΚΟ

Για την σύνταξη του κώδικα των συναρτήσεων χρησιμοποιήθηκε ο editor *VS Code* της *Microsoft*. Η εκτέλεση των εκάστοτε εντολών για την δημιουργία,

III.A Γραφικό Περιβάλλον (UI) του FN Project

Το *FN Project* δίνει την επιλογή στον χρήστη εκτός από τις διάφορες εντολές μέσω του *terminal* να αλληλεπιδρά με αυτό μέσω ενός απλού γραφικού περιβάλλοντος. Για αρχή πρέπει να ενεργοποιήσουμε το γραφικό περιβάλλον. Υποθέτωντας ότι ο *FN server* τρέχει τοπικά (μόνο στο δικό μας μηχάνημα δηλαδή) αυτό πραγματοποιείται με την εντολή `docker run --rm -it --link fnserver:api -p 4000:4000 -e "FN_API_URL=http://api:8080" fnproject/ui`.

Μετά από την επιτυχή εκτέλεση της παραπάνω εντολής μπορούμε να δούμε και να χρησιμοποιήσουμε το γραφικό περιβάλλον του *FN Project* από τον *Internet Browser* μας στην διεύθυνση <http://localhost:4000/>. Εκεί μπορούμε να δούμε για αρχή έναν πλήρη κατάλογο των εφαρμογών μας. Κάνοντας κλικ πάνω σε μια εφαρμογή μπορούμε να δούμε τις συναρτήσεις που ανήκουν σε αυτήν μαζί με επιπλέον πληροφορίες για την κάθε μια από αυτές. Οι πληροφορίες αυτές είναι : το όνομα της συνάρτησης, το όνομα του *docker image* που χρησιμοποιείται για να εκτελεστεί η συνάρτηση, την μέγιστη μνήμη που μπορεί να δεσμευτεί από αυτήν όταν εκτελείται τον χρόνο για το *timeout* και το *idle timeout* και τέλος παρέχεται η δυνατότητα να “τρέξουμε”, να αλλάξουμε αλλά και να διαγράψουμε την συνάρτηση που μας ενδιαφέρει. Παρακάτω φαίνονται με *screen shots* αυτά που μόλις περιγράψαμε.



demo_app

Name	Image	Memory	Timeout	Idle Timeout	Actions
add_f	fndemouser/add_f:0.0.25	256 MB	30	30	Run Function
fibonacci	fndemouser/fibo_f:0.0.23	256 MB	30	30	Edit Function Delete Function Run Function
rand_list_f	fndemouser/rand_list_f:0.0.16	256 MB	30	30	Run Function

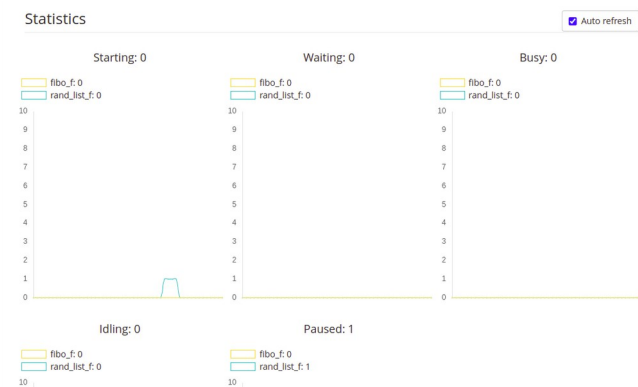
Επιπλέον κάτω από την περιοχή όπου απεικονίζονται οι συναρτήσεις με τα στοιχεία τους που περιγράψαμε παραπάνω υπάρχει η περιοχή Statistics όπου μπορούμε να δούμε πληροφορίες σχετικά με την κατάσταση που βρίσκεται η κάθε συνάρτηση της εφαρμογής μας σε πραγματικό χρόνο. Εδώ μπορούμε να δούμε ποιες και πόσες συναρτήσεις είναι σε κατάσταση εκκίνησης (starting), αναμονής (waiting), εκτέλεσης (busy), αδράνειας (idling), προσωρινής παύσης (paused).

Το FN Project χρωματίζει διαφορετικά τα στατιστικά που αφορούν την κάθε συνάρτηση παρέχοντας έτσι διευκόλυνση στον τρόπο παρατήρησης των στατιστικών που αφορούν τις συναρτήσεις οι οποίες ανήκουν στην εφαρμογή.

Η περιοχή Statistics υπάρχει και στο τέλος της καρτέλας Applications όπου μπορούμε να δούμε πληροφορίες σε πραγματικό χρόνο για το πλήθος των συναρτήσεων που είναι στην ουρά αναμονής και προετοιμάζονται για εκτέλεση (Queued), πόσες εκτελούνται (Running), και πόσες έχουν ολοκληρώσει την εκτέλεση τους.

Όπως για κάθε εφαρμογή ξεχωριστά έτσι και εδώ το FN Project παρουσιάζει εκτός από τα στατιστικά νούμερα και απεικονίσεις τους με γραφήματα.

Περιοχή Statistics για μια εφαρμογή:



Περιοχή Statistics στην καρτέλα Applications:



Από το γραφικό περιβάλλον μπορούμε όπως προαναφέρθηκε να εκτελέσουμε μια συνάρτηση με ή χωρίς δεδομένα εισόδου.

Για τον σκοπό αυτό επιλέγουμε στο πεδίο Actions της συνάρτησης που θέλουμε την εντολή "Run Function". Κατόπιν στην καρτέλα Run Function στο πεδίο Payload μπορούμε είτε να εισάγουμε τα δεδομένα με τα οποία θέλουμε να τρέξει η συνάρτηση είτε να το αφήσουμε κενό οπότε και η συνάρτηση θα τρέξει χωρίς δεδομένα εισόδου. Κάτω από το προαναφερθέν πεδίο βλέπουμε το πεδίο curl command στο οποίο φαίνεται η εντολή που τρέχει στο παρασκήνιο προκειμένου να εκτελεστεί η συνάρτηση. Αυτό άλλωστε μας θυμίζει ότι όλες οι εντολές του FN Project είναι εντολές που χρειαζόμαστε το terminal για να τις εκτελέσουμε. Η ύπαρξη του γραφικού περιβάλλοντος δεν καταργεί βεβαίως αυτές τις εντολές απλά μεταφέρει την ύπαρξη και την εκτέλεση τους στο παρασκήνιο.

Αφού εκτελέσουμε την συνάρτηση παρατηρούμε ότι εμφανίζεται το πεδίο Output όπου παρουσιάζεται η έξοδος της συνάρτησης που μόλις εκτελέσαμε. Σε περίπτωση σφάλματος κατά την διάρκεια της εκτέλεσης εμφανίζεται το αντίστοιχο μήνυμα σφάλματος στο πάνω δεξιά μέρος της οθόνης μέσα σε κόκκινο πλαίσιο. Επιπλέον εμφανίζεται και αντίστοιχη ειδοποίηση στο terminal στο οποίο τρέξαμε την εντολή η οποία ενεργοποίησε το γραφικό περιβάλλον του FN Project.

Επιπρόσθετα πάντα στο αριστερό μέρος του γραφικού περιβάλλοντος υπάρχει ένα menu με τις εξής επιλογές:

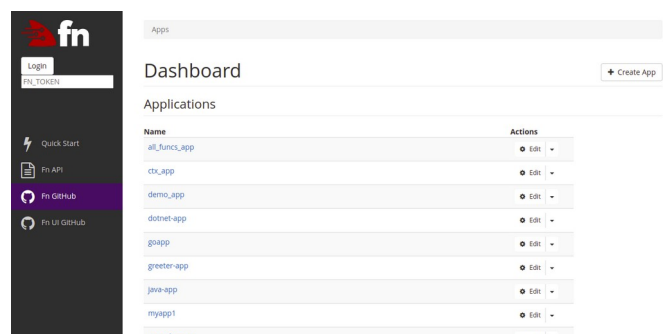
Quick Start: Μετάβαση στο GitHub Repository για το FN Project στο αρχείο readme στην υποενότητα Quick Start όπου παρέχονται οδηγίες για την εγκατάσταση του εργαλείου, για την εκκίνηση του FN Project Server και για την δημιουργία και εκτέλεση μιας απλής συνάρτησης και μιας εφαρμογής στην οποία φυσικά θα ανήκει αυτή η συνάρτηση.

FN API: Μετάβαση στο γραφικό περιβάλλον του API του FN Project.

FN GitHub: Μετάβαση στο επίσημο Repository του FN Project στο GitHub.

FN UI GitHub: Μετάβαση στο επίσημο Repository του FN Project UI στο GitHub.

Παρακάτω φαίνεται και σε screenshot το menu με τις επιλογές που μόλις περιγράψαμε:



Και τα μηνύματα που εμφανίζονται στο terminal αν γίνει κάποιο σφάλμα κατά την διάρκεια της εκτέλεσης:

```
qibgg@Lenovo-ideapad-100-15180:~$ docker run --rm -it --link fnserver:api -p 4000:4000 -e "FN_API_URL=http://api:8080" fnproject/ui

> FunctionsUI@0.0.39 start /app
> node server

WARNING: NODE_ENV value of 'production' did match any deployment config file names.
WARNING: See https://github.com/lorenwest/node-config/wiki/Strict-Mode
info: Using API url: api:8080
info: Server running on port 4000
error: Error. Api responded with 502 message=function failed
error: Error. Api responded with 502 message=function failed
error: Error. Api responded with 502 message=function failed
```

III.B FN Project & Terminal

Πέραν του γραφικού περιβάλλοντος χρησιμοποιήθηκε ιδιαίτερα και το terminal για την αλληλεπίδραση με το εργαλείο μέσω των εντολών που παρουσιάζονται στο documentation. Η αλήθεια είναι αυτός ο τρόπος αποτελεί τον βασικό τρόπο παραμετροποίησης και εκτέλεσης τόσο των συναρτήσεων όσο και των εφαρμογών της πλατφόρμας αυτής. Αυτό επιβεβαιώνεται και από το γεγονός ότι στην επίσημη ιστοσελίδα του εργαλείου στο τμήμα όπου παρέχονται οδηγίες για μια πρώτη εξοικείωση του χρήστη με την σύνταξη και εκτέλεση συναρτήσεων οι εντολές μέσω terminal αποτελούσαν και τον μόνο τρόπο για να επιτευχθεί ο σκοπός αυτός.

Για τον σκοπό αυτό κρίνουμε σκόπιμο να παρουσιάσουμε μερικές βασικές εντολές των οποίων κάναμε εκτενή χρήση κατά την διάρκεια της εργασίας.

- `fn start` για έναρξη του τοπικού `fn server`
- `fn start --log-level DEBUG` έναρξη του τοπικού `fn server` σε Debug mode για διευκόλυνση της αποσφαλμάτωσης.
- `fn init --runtime <language> <function name>` δημιουργία συνάρτησης σε γλώσσα προγραμματισμού `"language"` με όνομα `"function name"`.
- `fn create app <app-name>` δημιουργία εφαρμογής (application) για ομαδοποίηση συναρτήσεων με όνομα `"app-name"`.
- `fn --verbose deploy --app <app-name> --local` προσθήκη συνάρτησης στην εφαρμογή με το όνομα `app-name`. Το `option --verbose` θα εμφανίσει στο terminal όλες τις λεπτομέρειες της διαδικασίας προσθήκης (όπως κατέβασμα των κατάλληλων docker images για την σωστή εκτέλεση του κώδικα, εκτέλεση και παρουσίαση αποτελεσμάτων test functions αν υπάρχουν κλπ.). Το `option --local` προκαλεί την προσθήκη της συνάρτησης στον τοπικό FN server.
- `fn list apps` παρουσιάζει στο terminal μια λίστα με όλες τις εφαρμογές που έχουμε δημιουργήσει μαζί με την μοναδική ταυτότητα (ID) που έχει αποκτήσει η κάθε μια.
- `fn list functions <app-name>` παρουσιάζει στο terminal μια λίστα με όλες τις συναρτήσεις που έχουμε δημιουργήσει και σχετίζονται με την εφαρμογή `app-name` μαζί με την μοναδική ταυτότητα (ID) που έχει αποκτήσει η κάθε μια.
- `fn invoke <app-name> <function-name>` εκτελεί την συνάρτηση `function-name` που ανήκει στην εφαρμογή `app-name`.
- `echo -n '{"name":"Bob"}' | fn invoke <app-name> <function-name> --content-type application/json` εισάγει δεδομένα σε JSON μορφή (`{"name":"Bob"}`) και κατόπιν εκτελεί την συνάρτηση με τα παραπάνω δεδομένα εισόδου.
- `fn inspect function <app-name> <function-name>` παρουσιάζει στο terminal πληροφορίες για την συνάρτηση `"function-name"` που ανήκει στην εφαρμογή `"app-name"`. Μια αρκετά σημαντική πληροφορία από αυτές είναι το URL μέσω του οποίου μπορούμε να καλέσουμε την συνάρτηση (το FN Project μας δίνει αυτήν την επιπλέον δυνατότητα και αυτό το url ονομάζεται `invoke endpoint`).

- `curl -X "POST" -H "Content-Type: application/json" <invoke-Endpoint>`
Εκτέλεση της συνάρτησης μέσω HTTP. Η συνάρτηση μπορεί να εκτελεστεί από το `<invoke-endpoint>` το οποίο της έχει ανατεθεί από τον FN server.
- `curl -X "POST" -H "Content-Type: application/json" -d '{"name":"Bob"}' <invoke-Endpoint>` Εισαγωγή δεδομένων σε JSON μορφή και άμεση εκτέλεση της συνάρτησης μέσω HTTP με τον ίδιο τρόπο όπως στην προηγούμενη εντολή.

Επιπλέον το εργαλείο μας παρέχει την δυνατότητα να φτιάχνουμε για κάθε συνάρτηση ένα url πριν την αναθέσουμε σε μια εφαρμογή. Αυτό το url ονομάζεται `trigger` και μπορούμε να το δημιουργήσουμε προσθέτοντας το `option --trigger` κατά στην εντολή `fn init`. Ως `argument` του `option --trigger` βάζουμε το `http` και έτσι ορίζουμε ότι το url που δημιουργήσαμε θα είναι προσβάσιμο μέσω `http` πρωτοκόλλου.

Με την εντολή `fn list triggers <app-name>` μπορούμε να δούμε τα `triggers` που έχουν δημιουργηθεί για τις συναρτήσεις που αποτελούν μέλη της εφαρμογής με το όνομα `"app-name"`. Επιπλέον μπορούμε να δούμε το μοναδικό αναγνωριστικό του `trigger` (ID) τον τύπο του (πάντα `http`) αλλά και το ίδιο το `trigger` (`source endpoint`) μέσω του οποίου μπορούμε να εκτελέσουμε την συνάρτηση.

Ακολουθούν μερικές εικόνες με κάποιες από τις εντολές που περιγράφηκαν παραπάνω και τα αποτελέσματά τους:

`fn inspect` για την εφαρμογή `java-app` και την συνάρτηση `javacon`:

```
gb@gb-Lenovo-Ideapad-100-15180:~/fn_funcs$ fn inspect function java-app javacon
{
  "annotations": {
    "fnproject.io/fn/invokeEndpoint": "http://localhost:8080/invoke/01GCKXBjNBWNG8G00GZj00000007"
  },
  "app_id": "01G8H0DNHNG8G00GZj00000001",
  "created_at": "2022-09-14T06:39:28.124Z",
  "id": "01GCKXBjNBWNG8G00GZj00000007",
  "idle_timeout": 30,
  "image": "fndemouser/javacon:0.0.21",
  "memory": 128,
  "name": "javacon",
  "timeout": 30,
  "updated_at": "2022-09-14T08:17:21.427Z"
}
```

`fn list triggers` για την εφαρμογή `demo_app` η οποία περιλαμβάνει τις συναρτήσεις `add_f`, `fibonacci_f`, `rand_list_f`:

```
gb@gb-Lenovo-Ideapad-100-15180:~/fn_funcs$ fn list triggers demo_app
FUNCTION NAME ID TYPE SOURCE ENDPOINT
add_f add_f 01GCKNAK6PNC8G00GZj00000028 http /add_f http://localhost:8080/t/demo_app/add_f
fibonacci_f fibonacci_f 01GCKRJVAS3NG8G00GZj00000014 http /fibonacci_f http://localhost:8080/t/demo_app/fibonacci_f
rand_list_f rand_list_f 01GCKRPHC3SNG8G00GZj0000002A http /rand_list_f http://localhost:8080/t/demo_app/rand_list_f
```

`fn list functions java-app` όπου βλέπουμε τις συναρτήσεις `java_list`, `java_ser`, `javacon`, `javafn` που αποτελούν το σύνολο των μελών της εφαρμογής `java-app`.

Παρατηρούμε ότι εμφανίζεται το `docker image` που χρησιμοποιήθηκε για κάθε μια από αυτές (στήλη `IMAGE`) αλλά και το μοναδικό αναγνωριστικό τους (στήλη `ID`).

```
gb@gb-Lenovo-Ideapad-100-15180:~/fn_funcs$ fn list functions java-app
NAME IMAGE ID
java_list fndemouser/java_list:0.0.13 01GCKKZD17NG8G00GZj0000000V
java_ser fndemouser/java_ser:0.0.11 01GCKWGSKWNG8G00GZj00000001
javacon fndemouser/javacon:0.0.21 01GCKXBjNBWNG8G00GZj00000007
javafn fndemouser/javafn:0.0.13 01G8H0VJMNNG8G00GZj00000002
```

IV ΑΠΟΤΕΛΕΣΜΑΤΑ ΚΑΙ ΣΥΜΠΕΡΑΣΜΑΤΑ

Σε αυτό το κεφάλαιο της αναφοράς θα παρουσιαστούν και θα σχολιαστούν τα αποτελέσματα της εκτέλεσης αρκετών από τις συναρτήσεις που παρήχθησαν.

Οι συναρτήσεις που δημιουργήσαμε είναι στις γλώσσες Python, C#, Java, Go, Ruby και Node JS. Ιδιαίτερη βαρύτητα δόθηκε στην Python και την Java λόγω μεγαλύτερης εξοικείωσης μας με αυτές τις δύο γλώσσες. Εξού και το μεγαλύτερο πλήθος συναρτήσεων και εφαρμογών στις γλώσσες αυτές. Παρόλα αυτά όπως προαναφέρθηκε δημιουργήθηκαν συναρτήσεις σε όλο το εύρος των γλωσσών προγραμματισμού που υποστηρίζεται από το FN Project.

Κάθε φορά που δημιουργούμε μια συνάρτηση στον τοπικό server το FN Project παράγει ένα αρχείο με κώδικα στην γλώσσα που ορίσαμε κατά την δημιουργία της συνάρτησης το οποίο επιτελεί μια απλή λειτουργία που ποικίλλει από γλώσσα σε γλώσσα. Επιπρόσθετα δημιουργεί ένα αρχείο με το όνομα "func.yaml" στο οποίο αναφέρονται κάποιες ιδιότητες της συναρτήσεως μας όπως την έκδοση του fn project που χρησιμοποιήθηκε (schema version), όνομα συνάρτησης (name), έκδοση (version, η οποία αρχίζει με την τιμή 0.0.1 και αυξάνεται κατά 1 αυτόματα κάθε φορά που κάνουμε deploy την συνάρτηση στον τοπικό fn server.), γλώσσα συνάρτησης (runtime), όνομα εκτελέσιμου αρχείου που χρησιμοποιείται κάθε φορά που καλούμε την συνάρτηση (entrypoint). Επιπρόσθετα στην περίπτωση που υπάρχουν triggers εμφανίζονται αντίστοιχα πεδία στο εν λόγω αρχείο (trigger name, trigger type, trigger source).

Τέλος επισημαίνεται ότι στις συναρτήσεις οι εισόδοι πρέπει να έχουν αυστηρά την ίδια μορφή όπως στα παραδείγματα που θα παρουσιαστούν παρακάτω.

IV.A rubyapp

Περιλαμβάνει την συνάρτηση rubyfn. Η rubyfn χωρίς δεδομένα εισόδου έχει έξοδο {"message":"Hello World!"} (JSON μορφή) ενώ αν της δώσουμε εισόδο της μορφής {"name":"Bob"} παράγει έξοδο της μορφής {"message":"Hello Bob!"}. Η γλώσσα της συνάρτησης είναι η ruby.

IV.B demo_app

Μέλη της εφαρμογής αποτελούν οι συναρτήσεις add_f, fibo_f, οι οποίες έχουν όλες γραφεί σε Python.

- add_f: Δέχεται εισόδο σε JSON μορφή μια λίστα με ακέραιους αριθμούς και επιστρέφει το άθροισμα των στοιχείων της λίστας, το μέγιστο στοιχείο της, τον αριθμό των στοιχείων της λίστας, και τον μέσο όρο των στοιχείων της λίστας. Για παράδειγμα με εισόδο {"nums":[1, 2]} έχει έξοδο {"sum":3, "max":2, "number of elements":2, "median of elements":1.5}.
- fibo_f: Δέχεται εισόδο σε JSON μορφή {"N":n} έναν ακέραιο αριθμό και επιστρέφει τον n-οστό αριθμό Fibonacci. Ο υπολογισμός του αριθμού αυτού γίνεται με την χρήση του τύπου που συνδέει τον n-οστό αριθμό Fibonacci με την χρυσή τομή ($\varphi = (1+\sqrt{5})/2$). Δηλαδή αν Fn ο n-οστός Fibonacci αριθμός τότε είναι $F_n = (\varphi^n - (1-\varphi)^n) / \sqrt{5}$. Για παράδειγμα με εισόδο {"N":3} έχει έξοδο {"result":2}.

IV.C dotnet_app

Περιλαμβάνει την συνάρτηση dotnetfn η οποία είναι γραμμένη σε γλώσσα C#. Η συνάρτηση με κενή είσοδο ({}) έχει έξοδο {"message":"Hello World!"} Ενώ με είσοδο της μορφής {"name":"Dotenet"}, έχει έξοδο {"message":"Hello Dotenet!"}. Εδώ αξίζει να σημειωθεί ότι κατά την διαδικασία δημιουργίας της συνάρτησης το FN Project δημιουργεί αυτόματα test functions τα οποία και εκτελούνται κάθε φορά που ανεβάζουμε την συνάρτηση στον τοπικό fn server. Αυτό μας δίνει την δυνατότητα να δούμε με έναν εύκολο και αποτελεσματικό τρόπο το πώς συμπεριφέρεται η συνάρτηση μας για διάφορες εισόδους και να εντοπίσουμε τυχόν σφάλματα.

IV.D go_app

Περιλαμβάνει την συνάρτηση gofn (σε γλώσσα go) η οποία με κενή είσοδο έχει έξοδο {"message":"Hello World!"} και με είσοδο της μορφής {"name":"Nickie"} έχει έξοδο της μορφής {"message":"Hello Nickie!"}.

IV.E greeter-app

Περιλαμβάνει τις συναρτήσεις hello (γλώσσα go), goodbye (γλώσσα go) και greeter-app (γλώσσα node JS).

Εδώ βλέπουμε ότι σε μια εφαρμογή δεν είναι υποχρεωτικό να ανήκουν συναρτήσεις μόνο από μια γλώσσα. Κάθε εφαρμογή μπορεί να περιλαμβάνει συναρτήσεις που έχουν συνταχθεί σε διάφορες γλώσσες.

- hello: Με κενή είσοδο έχει έξοδο {"message":"Hello World!"} και με είσοδο της μορφής {"name":"Bob"} έχει έξοδο της μορφής {"message":"Hello Bob!"}.
- goodbye: Με κενή είσοδο έχει έξοδο {"message":"Goodbye World!"} και με είσοδο της μορφής {"name":"Kiki"} έχει έξοδο της μορφής {"message":"Goodbye Kiki!"}.
- greeter-app: Με κενή είσοδο έχει έξοδο {"message":"Hello from root"} και με είσοδο της μορφής {"name":"John"} έχει έξοδο {"message":"Hello John"}

Ακολουθούν μερικά screenshots από την εκτέλεση των συναρτήσεων της εφαρμογής greeter-app:

Εκτέλεση της greeter-app (nodeJS) μέσω terminal με δεδομένα εισόδου {"name":"Mike"}:

```
ib09gb-Lenovo-Ideapad-100-151BD:~/fn_funcs/greeter-app$ echo -n '{"name":"Mike"}'|fn invoke greeter-app greeter-app
{"message":"Hello Mike"}
```

Εκτέλεση της hello (go) μέσω του γραφικού περιβαλλοντος χωρίς δεδομένα εισόδου :

Run Function

App: greeter-app

Function Name: hello

Payload: {}

cURL command: curl -X POST -d '{}' http://api:8080/invoke/016CK1KPXDNG8G06ZJ000000P

Output: { "message": "Hello World" }

Run

IV.F java-app

Σε αυτήν την εφαρμογή έχουμε τις συναρτήσεις `java_list`, `java_ser`, `javacon`, `javafn`. Όλες αυτές οι συναρτήσεις έχουν συνταχθεί σε γλώσσα Java.

Εδώ κρίνεται σκόπιμο να αναφερθεί ότι όπως και στην περίπτωση της γλώσσας C# έτσι και για την Java το FN Project κατά την διάρκεια δημιουργίας μιας συνάρτησης παράγει αυτόματα ένα αρχείο με test function για την συνάρτηση αυτή. Προφανώς σε περίπτωση που εμείς αλλάξουμε την αρχική μορφή της βασικής συνάρτησης που παρέχεται από το FN Project (με την εντολή `fn init`) πρέπει να κάνουμε και ανάλογες προσαρμογές στην αντίστοιχη test function. Σε αντίθετη περίπτωση η συνάρτηση μας δεν θα γίνει deploy στον τοπικό FN server διότι κάθε φορά που εκτελείται αυτή η διαδικασία το εργαλείο αυτόματα εκτελεί τις test functions που σχετίζονται με την συνάρτηση που θέλουμε να “ανεβάσουμε” στον τοπικό FN server και αποτυχία στις δοκιμές αυτές δημιουργεί τελικά αποτυχία στην προσθήκη της συνάρτησης στον διακομιστή.

- `java_list`: Δέχεται είσοδο σε JSON μορφή μια λίστα με ακέραιους αριθμούς και επιστρέφει την λίστα αυτή με τα στοιχεία της ταξινομημένα σε αύξουσα σειρά. Για παράδειγμα με είσοδο `{"list": [2, 1]}` έχει έξοδο `{"f_str": "Sorted List is:", "sorted": [1, 2]}`
- `java_ser`: Δέχεται είσοδο σε JSON μορφή τον πρώτο όρο, την διαφορά και το πλήθος των όρων(n) μιας αριθμητικής προόδου και επιστρέφει το πλήθος των όρων, ένα μήνυμα που επεξηγεί την έξοδο, τον n-οστό όρο και το άθροισμα των n πρώτων όρων της αριθμητικής πρόοδου. Για παράδειγμα με είσοδο `{"a1": 1, "n": 2, "d": 2}` έχουμε έξοδο `{"n": 2, "message": "Nth term and Sum of n terms are:", "a_n": 3, "sum_of_n_terms": 4}`
- `javacon`: Δέχεται είσοδο σε JSON μορφή δύο συμβολοσειρές και επιστρέφει την σύνδεση τους και το μήκος αυτής. Για παράδειγμα με είσοδο `{"str1": "abcd", "str2": "efg"}` επιστρέφει `{"f_str": "Concatanated string is abcdefg", "length": "7"}`
- `javafn`: Δέχεται είσοδο σε JSON μορφή. Με κενή είσοδο (`{}`) επιστρέφει `{"salutation": "Hello World!"}` ενώ με είσοδο `{"name": "Luke"}` επιστρέφει `{"salutation": "Hello Luke !"}`

Ακολουθούν screenshots από την εκτέλεση της `java_list`:

Εκτέλεση της `java_list` μέσω terminal:

```
sb@gb-Lenovo-ideapad-100-151BD:~/fn_funcs/java-app/java_list$ echo -n '{"list": [98, 76, 5, 14]}' | fn invoke java-app java_list
{"f_str": "Sorted List is: ", "sorted": [5, 14, 76, 98]}
```

Εκτέλεση της `java_list` μέσω γραφικού περιβάλλοντος:

The screenshot shows the 'Run Function' dialog for the 'java-app' application. The 'Function Name' is set to 'java_list'. The 'Payload' field contains the JSON object `{"list": [8, 1, 7]}`. Below the dialog, the 'cURL command' is shown as `curl -X POST -d '{"list": [8, 1, 7]}' http://api:8080/invoke/01GCKXKDZ17NG8G00GZJ000000V`. The 'Output' section displays the result: `{ "f_str": "Sorted List is: ", "sorted": [1, 7, 8] }`.

Στις συναρτήσεις `java_list`, `java_ser` και `javacon` ήταν απαραίτητο να παραμετροποιήσουμε κατάλληλα το αρχείο με τα test functions έτσι ώστε η σύνταξη μας να περνάει επιτυχώς τις δοκιμές.

Κάθε φορά που κάνουμε μια αλλαγή στον κώδικα και κατόπιν κάνουμε deploy την συνάρτησή μας στον τοπικό FN server στην αντίστοιχη εφαρμογή (java-app στην προκειμένη περίπτωση) το εργαλείο κατεβάζει τα αντίστοιχα docker images ώστε να γίνει σωστά η εκτέλεση του κώδικα και τον μεταγλωττίζει. Σε περίπτωση επιτυχίας εμφανίζεται σχετικό μήνυμα στο terminal ([INFO] Build Success). Κατόπιν εκτελεί τα test functions που βρίσκονται στον φάκελο `src/test`. Πριν ξεκινήσει η εκτέλεσή τους υπάρχει ενημέρωση με σχετικό μήνυμα στο terminal ([INFO] T E S T S) και έπειτα εμφανίζονται και τα αποτελέσματά τους συνοδευόμενα από σχετικά μηνύματα. Για παράδειγμα αν είχαμε ορίσει ακριβώς μια test function για την συνάρτηση μας θα είχαμε τα παρακάτω μηνύματα σε περίπτωση επιτυχίας των δοκιμών: [INFO] : Results, [INFO] : Tests run 1, Failures: 0, Errors: 0, Skipped: 0.

Τέλος εφόσον η μεταγλώττιση και τα test functions έχουν ολοκληρωθεί απρόσκοπτα εμφανίζεται μήνυμα που μας ενημερώνει για την συνολική επιτυχία [INFO] BUILD SUCCESS και μπορούμε να προχωρήσουμε στα επόμενα βήματα.

Ακολουθούν screenshots με αποτελέσματα εκτέλεσης κάποιων συναρτήσεων της java-app.

Εκτέλεση της `java_ser` μέσω terminal με δεδομένα εισόδου `{"a1": 1, "n": 3, "d": 3}`:

```
sb@gb-Lenovo-ideapad-100-151BD:~/fn_funcs/java-app/java_ser$ echo -n '{"a1": 1, "n": 3, "d": 3}' | fn invoke java-app java_ser
{"n": 3, "message": "Nth term and Sum of n terms are:", "a_n": 7, "sum_of_n_terms": 12}
```

Εκτέλεση της `javacon` μέσω γραφικού περιβάλλοντος με δεδομένα εισόδου `{"str1": "gr", "str2": "eg"}`:

The screenshot shows the 'Run Function' dialog for the 'java-app' application. The 'Function Name' is set to 'javacon'. The 'Payload' field contains the JSON object `{"str1": "gr", "str2": "eg"}`. Below the dialog, the 'cURL command' is shown as `curl -X POST -d '{"str1": "gr", "str2": "eg"}' http://api:8080/invoke/01GCXBJNBWNG8G00GZJ0000007`. The 'Output' section displays the result: `{ "f_str": "Concatanated string is greg", "length": 4 }`.

Εκτέλεση της `javafn` μέσω terminal με δεδομένα εισόδου `{"name": "Alex"}`:

```
sb@gb-Lenovo-ideapad-100-151BD:~/fn_funcs/java-app/javafn$ echo -n '{"name": "Alex"}' | fn invoke java-app javafn
{"salutation": "Hello Alex !"}
```

IV.G myapp1

Περιλαμβάνει την συνάρτηση function1 για της οποίας την σύνταξη έχει χρησιμοποιηθεί η java.

Η συνάρτηση αυτή όταν εκτελεστεί χωρίς δεδομένα εισόδου επιστρέφει "Hello World!" ενώ με οποιαδήποτε συμβολοσειρά ή αριθμό επιστρέφει την συμβολοσειρά "Hello " και τους χαρακτήρες που έλαβε για είσοδο μαζί με ένα "!" στο τέλος. Οι χαρακτήρες της συμβολοσειράς εξόδου είναι κίτρινου χρώματος.

Για παράδειγμα με είσοδο την συμβολοσειρα "ddd" επιστρέφει Hello, ddd !. Ενώ απουσία εισόδου επιστρέφει Hello, World!.

Εδώ πρέπει να τονιστεί ότι αυτή η συνάρτηση μπορεί να δώσει την αναμενόμενη έξοδο μόνο στην περίπτωση εκτέλεσής της μέσω terminal. Στην εκδοχή εκτέλεσης μέσω του γραφικού περιβάλλοντος του εργαλείου παρατηρείται αδυναμία εμφάνισης του κίτρινου χρώματος στην έξοδο. Αντί αυτού τυπώνονται οι ειδικοί χαρακτήρες που αντιπροσωπεύουν το χρώμα αλλά δεν χρωματίζονται και οι χαρακτήρες όπως ήταν το αναμενόμενο.

Προφανώς έπρεπε να γίνουν και οι κατάλληλες αναπροσαρμογές που αφορούν τους χρωματισμένους χαρακτήρες εξόδου στο αρχείο κώδικα με τα test functions προκειμένου να μπορέσουμε να κάνουμε με ορθό τρόπο προσθήκη της νέας έκδοσης της συνάρτησής μας στην εφαρμογή και στον FN server.

Ακολουθούν screensots που αφορούν την εκτέλεση της παραπάνω συνάρτησης.

Εκτέλεση με και χωρίς δεδομένα εισόδου.

```
gb@gb-Lenovo-Ideapad-100-15IBD:~/fn_funcs/function1$ echo -n ddd | fn invoke myapp1 function1
Hello, ddd!
gb@gb-Lenovo-Ideapad-100-15IBD:~/fn_funcs/function1$ fn invoke myapp1 function1
Hello, world!
```

Εκτέλεση μέσω του γραφικού περιβάλλοντος όπου παρατηρείται η αδυναμία χρωματισμού της εξόδου η οποία παρουσιάστηκε αναλυτικά παραπάνω.

Η συνάρτηση εκτελείται με είσοδο την συμβολοσειρά "adad".

Run Function

App

myapp1

Function Name

function1

Payload

adad

cURL command

curl -X POST -d 'adad' http://api:8080/invoke/01GCK4E21BNG8G00GZJ000001R

Output

[33mHello, "adad"!0m

Run

IV.H mypythonapp

Μοναδικό μέλος της εφαρμογής αυτής αποτελεί η συνάρτηση mypythonfunction της οποίας η γλώσσα είναι η python.

Η συνάρτηση αυτή δέχεται ως είσοδο ένα συγκεκριμένο url το οποίο οδηγεί σε ένα αρχείο excel (.csv) και κατόπιν ανοίγει αυτό το αρχείο και παρουσιάζει τμήμα του περιεχομένου του στο terminal ή στην κονσόλα εξόδου τπυ γραφικού περιβάλλοντος (αναλογά με το που λαμβάνει χώρα η εκτέλεση της συνάρτησης).

Το αρχείο csv είναι ένας κατάλογος με προσωπικά στοιχεία αθλητών μπάσκετ στις ΗΠΑ. Τα στοιχεία αυτά είναι το όνομα τους, η ομάδα που αγωνίζονται, το νούμερο με το οποίο αγωνίζονται, η θέση στην οποία παίζουν, η ηλικία, το ύψος, το βάρος, το κολλέγιο στου οποίου την ομάδα ανήκουν καθώς και ο μισθός τους.

Η συνάρτηση μεταβαίνει στο url που της παρέχεται ως δεδομένο εισόδου κατόπιν εντοπίζει, ανοίγει και διαβάζει το αρχείο csv. Κατόπιν μετατρέπει όλα τα δεδομένα του σε συμβολοσειρές και τα εμφανίζει με JSON μορφή στην έξοδο.

Παρακάτω θα παρουσιαστούν screenshots από την εκτέλεση της συνάρτησης αυτής τόσο από το γραφικό περιβάλλον όσο και από το terminal.

Εκτέλεση μέσω terminal:

```
gb@gb-Lenovo-Ideapad-100-15IBD:~/fn_funcs/mypythonapp/mypythonfunction$ echo -n '{"url":"https://media.geeksforgeeks.org/wp-content/uploads/nba.csv"}' | fn invoke mypythonapp mypythonfunction --content-type application/json
{"DataFrame.describe": "<bound method NDFrame.describe of\nme\nTeam\nCollege\nSalary\nAvery Bradley\nBoston Celtics\nTexas\n7730337.0\n1\nJae Crowder\nBoston Celtics\nMarquette\n6796117.0\n2\nJohn Holland\nBoston Celtics\nBoston University\nNaN\n3\nR.J. Hunter\nBoston Celtics\nGeorgia State\n1148640.0\n4\nJonas Jerebko\nBoston Celtics\nNaN\n500000.0\n...\n...\n453\nShelvin Mack\nUtah Jazz\nButler\n243333.0\n454\nRaul Neto\nUtah Jazz\nNaN\n900000.0\n455\nTibor Pleiss\nUtah Jazz\nNaN\n2900000.0\n456\nJeff Withey\nUtah Jazz\nKansas\n947276.0\n457\nNaN\nNaN\nNaN\n[458 rows x 9 columns]>"}

```

Εκτέλεση μέσω γραφικού περιβάλλοντος:

App

mypythonapp

Function Name

mypythonfunction

Payload

{"url":"https://media.geeksforgeeks.org/wp-content/uploads/nba.csv"}

cURL command

curl -X POST -d '{"url":"https://media.geeksforgeeks.org/wp-content/uploads/nba.csv"}' http://api:8080/invoke/01GCK7TGJKN8G00GZJ000003G

Output

{\n "DataFrame.describe": "<bound method NDFrame.describe of\nName\nTeam\nCollege\nSalary\nAvery Br\nadley\nBoston Celtics\nTexas\n7730337.0\n1\nJae C\nrowder\nBoston Celtics\nMarquette\n6796117.0\n2\nJohn\nHolland\nBoston Celtics\nBoston University\nNaN\n3\nR.\nJ. Hunter\nBoston Celtics\nGeorgia State\n1148640.0\n4\nJon\nas Jerebko\nBoston Celtics\nNaN\n500000.0\n...\n...\n453\nShelvin\nMack\nUtah Jazz\nButler\n243333.0\n454\nRaul\nNeto\nUtah Jazz\nNaN\n900000.0\n455\nTibor P\nleiss\nUtah Jazz\nNaN\n2900000.0\n456\nJeff\nWithey\nUtah Jazz\nKansas\n947276.0\n457\nNaN\nNaN\nNaN\n[458 rows x\n9 columns]>"\n}

Μπορούμε εύκολα να παρατηρήσουμε ότι τόσο κατά την εκτέλεση μέσω γραφικού περιβάλλοντος όσο και κατά την εκτέλεση μέσω terminal δεν εμφανίζεται το σύνολο των γραμμών και των στηλών του αρχείου στην έξοδο. Ο λόγος είναι ότι το μέγεθος τους είναι αρκετά μεγάλο και υπερβαίνει τις δυνατότητες αναπαράστασης και των δύο τρόπων. Πάντως βλέπουμε ότι και στις δύο περιπτώσεις αναγράφεται πόσες γραμμές και στήλες έχει συνολικά το csv αρχείο. Δηλαδή 458 γραμμές και 9 στήλες. Παρατηρούμε επίσης την ύπαρξη ειδικών χαρακτήρων οι οποίοι δεν μπορούν να αναπαρασταθούν με ορθό τρόπο από το terminal ή το UI.

Η έξοδος όπως προαναφέραμε είναι σε JSON μορφή και ξεκινάει με την συμβολοσειρά "Data Frame.describe" και κατόπιν ακολουθούν τα δεδομένα του csv αρχείου σε μορφή συμβολοσειράς.

IV.I nodeapp

Περιλαμβάνει την συνάρτηση nodefn η οποία είναι γραμμένη σε node JS.

Η συνάρτηση αυτή μπορεί να εκτελεστεί με ή χωρίς δεδομένα εισόδου.

Εάν υπάρχει εισόδος τότε αυτή πρέπει να είναι της μορφής (JSON) {"name": "Kiki", "year": 1998} και έχουμε έξοδο (JSON) {"message": "Hello Kiki !", "Your age is": 24, "The length of your name word is": 4}. Δηλαδή η συνάρτηση αυτή δέχεται σε μορφή JSON το όνομα και την ημερομηνία γέννησης ενός ανθρώπου και επιστρέφει ένα μήνυμα που περιλαμβάνει το όνομά του, την ηλικία του, και το πλήθος των χαρακτήρων που έχει η λέξη του ονόματός του.

Σε περίπτωση απουσίας εισόδου ή κενής εισόδου τότε επιστρέφεται η εξής έξοδος σε JSON μορφή {"message": "Hello World !", "Your age is": 0, "The length of your name word is": 0} όπου παρατηρούμε ότι τα πεδία για την ηλικία και το μήκος του ονόματος είναι 0 όπως αναμενόταν αφού δεν πήραμε κάποιο όνομα και χρονολογία γέννησης ως εισόδο.

Ακολουθούν κάποια screenshot από την εκτέλεση της nodefn:

Εκτέλεση της συνάρτησης με δεδομένα εισόδου {"name": "John", "year": 1987} μέσω terminal.

```
gb@gb-Lenovo-Ideapad-100-151B0:~/fn_funcs/nodefn$ echo -n '{"name": "John", "year": 1987}' | fn invoke nodeapp nodefn
{"message": "Hello John !", "Your age is": 35, "The length of your name word is": 4}
```

Εκτέλεση της συνάρτησης χωρίς δεδομένα εισόδου μέσω του γραφικού περιβάλλοντος.

Run Function

App

nodeapp

Function Name

nodefn

Payload

{}

cURL command

curl -X POST -d '{}' http://api:8080/invoke/01G8MZ9A08NG8G00GZJ000000F

Output

```
{
  "message": "Hello World !",
  "Your age is": 0,
  "The length of your name word is": 0
}
```

Run

IV.J. pythonapp

Η εφαρμογή αυτή φιλοξενεί την συνάρτηση pythofn η οποία είναι δυνατόν να εκτελεστεί με ή χωρίς δεδομένα εισόδου.

Η συνάρτηση αυτή λαμβάνει σε JSON μορφή το όνομα (συμβολοσειρά), το βάρος (ακέραιος ή δεκαδικός αριθμός) και το ύψος (ακέραιος ή δεκαδικός αριθμός) ενός ανθρώπου και επιστρέφει τον δείκτη μάζας σώματος του (ΔΜΣ, BMI) με ακρίβεια δύο δεκαδικών ψηφίων. Κατόπιν εκτυπώνει μήνυμα σε JSON μορφή με το όνομα του προσώπου, το ΔΜΣ του και μια γνωμάτευση για το αν είναι λιπόσαρκος, υπέρβαρος ή παχύσαρκος.

Στην περίπτωση κενής εισόδου η συνάρτηση επιστρέφει σε μορφή JSON το εξής μήνυμα: {"message": "Hello Welcome", "Your bmi is": 0.0, "You are": "Please insert values"}

Για εισόδο της μορφής {"name": "Susan", "weight": 65.9, "height": 1.69} επιστρέφει την εξής έξοδο: {"message": "Hello Susan", "Your bmi is": "23.07", "You are": "Normal"}.

Ενώ για εισόδο της μορφής {"name": "John", "weight": 98, "height": 1.72} επιστρέφει την εξής έξοδο: {"message": "Hello John", "Your bmi is": "33.13", "You are": "Overweight. You should lose weight."}.

Παρακάτω θα παρουσιάσουμε κάποια screenshots από την εκτέλεση της pythofn μέσω γραφικού περιβάλλοντος και μέσω terminal.

Εκτέλεση της συνάρτησης χωρίς δεδομένα εισόδου μέσω terminal:

```
gb@gb-Lenovo-Ideapad-100-151B0:~/fn_funcs/pythonfn$ fn invoke pythonapp pythonfn
{"message": "Hello Welcome", "Your bmi is": "0.0", "You are": "Please insert values"}
```

Εκτέλεση της συνάρτησης με εισόδο {"name": "Mike", "weight": 67.9, "height": 1.87} μέσω terminal.

```
gb@gb-Lenovo-Ideapad-100-151B0:~/fn_funcs/pythonfn$ echo -n '{"name": "Mike", "weight": 67.9, "height": 1.87}' | fn invoke pythonapp pythonfn
{"message": "Hello Mike", "Your bmi is": "19.42", "You are": "Normal"}
```

Εκτέλεση της συνάρτησης με εισόδο {"name": "John", "weight": 105, "height": 1.72} μέσω γραφικού περιβάλλοντος του εργαλείου.

Run Function

App

pythonapp

Function Name

pythofn

Payload

{"name": "John", "weight": 105, "height": 1.72}

cURL command

curl -X POST -d '{"name": "John", "weight": 105, "height": 1.72}' http://api:8080/invoke/01G8G91WEWNG8G00GZJ0000002

Output

```
{
  "message": "Hello John",
  "Your bmi is": "35.49",
  "You are": "Overweight. You should lose weight."
}
```

Run

IV.K cipher_app

Περιλαμβάνει τις συναρτήσεις `decrypt_f`, `encrypt_f` οι οποίες κρυπτογραφούν και αποκρυπτογραφούν ένα κείμενο αντίστοιχα. Και οι δύο συναρτήσεις έχουν γραφεί σε `python`.

- `encrypt_f`: Δέχεται είσοδο σε JSON μορφή το κείμενο που θα κρυπτογραφηθεί (`text`), τον αλγόριθμο κρυπτογράφησης (οι υποστηριζόμενες μορφές είναι Caesar, Shift Cipher και Vigenere) και το κλειδί κρυπτογράφησης (`key`) για τους αλγορίθμους Shift Cipher (ακέραιος αριθμός) και για τον Vigenere (μια λέξη χωρίς ειδικούς χαρακτήρες). Για παράδειγμα με είσοδο `{"text": "I am Rhaenyra Targaryen", "cipher": "Caesar"}` έχει έξοδο `{"ciphertext": "FXJOEXBKVOXQXODXOVBK"}`
- `decrypt_f`: Δέχεται είσοδο σε JSON μορφή το κείμενο που θα αποκρυπτογραφηθεί (`text`), τον αλγόριθμο κρυπτογράφησης (οι υποστηριζόμενες μορφές είναι Caesar, Shift Cipher και Vigenere) και το κλειδί κρυπτογράφησης (`key`) για τους αλγορίθμους Shift Cipher (ακέραιος αριθμός) και για τον Vigenere (μια λέξη χωρίς ειδικούς χαρακτήρες). Για παράδειγμα με είσοδο `{"text": "YQEDSZ", "cipher": "Vigenere", "key": "foo"}` έχει έξοδο `{"plaintext": "DESIGN"}`.

Ακολουθούν screenshots με στιγμιότυπα εκτέλεσης των συναρτήσεων για διάφορες εισόδους.

Εκτέλεση της `encrypt_f` μέσω terminal με είσοδο `{"text": "dragon", "cipher": "Shift Cipher", "key": 9}`:

```
gb@gb-Lenovo-ideapad-100-15IBD:~/fn_funcs/cipher_app/encrypt_f$ echo -n '{"text": "dragon", "cipher": "Shift Cipher", "key": 9}' | fn invoke cipher_app encrypt_f '{"ciphertext": "UIRXFE"}'
```

Εκτέλεση της `encrypt_f` μέσω γραφικού περιβάλλοντος με είσοδο `{"text": "I know you", "cipher": "Vigenere", "key": "low"}`:

Run Function

App

cipher_app

Function Name

encrypt_f

Payload

```
{"text": "I know you", "cipher": "Vigenere", "key": "low"}
```

cURL command

```
curl -X POST -d '{"text": "I know you", "cipher": "Vigenere", "key": "low"}' http://api:8080/invoke/016D7X6QVCN6G6G0GZJ0000004
```

Output

```
{
  "ciphertext": "XWRDIDCG"
}
```

Run

Εκτέλεση μέσω terminal της `decrypt_f` για αποκρυπτογράφηση της συμβολοσειράς που προέκυψε από το προηγούμενο παράδειγμα:

```
gb@gb-Lenovo-ideapad-100-15IBD:~/fn_funcs/cipher_app/decrypt_f$ echo -n '{"text": "XWRDIDCG", "cipher": "Vigenere", "key": "low"}' | fn invoke cipher_app decrypt_f '{"plaintext": "IKNOWYOU"}'
```

Όπου μπορούμε να δούμε ότι η αποκρυπτογράφηση έγινε με ορθό τρόπο.

IV.L Επισήμανση για τις συναρτήσεις σε Python

Αν παρατηρήσουμε τα περιεχόμενα του φακέλου με την σύνταξη που δημιουργεί το FN Project για την περίπτωση της Python θα δούμε ότι εκτός από το αρχείο με τον πηγαίο κώδικα (`func.py`) και το αρχείο `func.yaml` (του οποίου έχουμε σκιαγραφήσει παραπάνω την δομή) δημιουργείται και το αρχείο `requirements.txt` το οποίο περιέχει την έκδοση του `fdk` (`fn project development kit`) που χρησιμοποιείται και τυχόν επιπλέον βιβλιοθήκες που θα χρειαστούν στον κώδικα μας ώστε να αναζητηθούν από το εργαλείο κατά την διάρκεια της διαδικασίας που η συνάρτηση γίνεται `deploy` οι κατάλληλες `docker images` για να γίνει η εισαγωγή αυτών των επιπλέον βιβλιοθηκών και να εκτελεστεί με σωστό τρόπο ο κώδικάς μας.

Κάθε φορά λοιπόν που κάνουμε εισαγωγή μιας βιβλιοθήκης η οποία δεν περιλαμβάνεται στην βασική βιβλιοθήκη της `python3` που χρησιμοποιούμε πρέπει εκτός από την εντολή `import` στο αρχείο πηγαίου κώδικα να έχει προστεθεί και το όνομα της βιβλιοθήκης στο αρχείο `requirements.txt`.

Κατά την διάρκεια που προσθέτουμε την συνάρτησή μας στον τοπικό `fn server` και στην εφαρμογή που επιθυμούμε (ενόσω εκτελείται η εντολή `deploy` δηλαδή) το FN Project θα μας ενημερώσει με σχετικά μηνύματα ότι εντόπισε ότι χρειάζεται η εγκατάσταση επιπλέον βιβλιοθήκης και ξεκινάει την αναζήτηση του κατάλληλου `docker image`.

Απαραίτητη προϋπόθεση για να γίνει σωστά το “ανέβασμα” της εφαρμογής στον τοπικό `fn server` είναι η βιβλιοθήκη που προσθέσαμε στο αρχείο `requirements.txt` να μην συμπεριλαμβάνεται στις βιβλιοθήκες που είναι προεγκατεστημένες στην `python3`. Σε αντίθετη περίπτωση η διαδικασία δεν μπορεί να ολοκληρωθεί και το εργαλείο μας ενημερώνει με μήνυμα σφάλματος στο terminal όπως φαίνεται και στο παρακάτω screenshot.

```
Collecting fdk>=0.1.46
  Downloading fdk-0.1.47-py3-none-any.whl (78 kB)
ERROR: Could not find a version that satisfies the requirement webbrowser (from
-r requirements.txt (line 2)) (from versions: none)
ERROR: No matching distribution found for webbrowser (from -r requirements.txt (
line 2))
The command '/bin/sh -c pip3 install --target /python/ --no-cache --no-cache-di
r -r requirements.txt && rm -fr ~/.cache/pip /tmp* re
quirements.txt func.yaml Dockerfile .venv && chmod -R o+r
/python' returned a non-zero code: 1

Fn: error running docker build: exit status 1
```

Στο παραπάνω screenshot φαίνεται το αποτέλεσμα όταν προσπαθούμε να εγκαταστήσουμε την βιβλιοθήκη `webbrowser` ή οποία όμως περιλαμβάνεται στις προεγκατεστημένες βιβλιοθήκες της `python3`. Για αυτόν τον λόγο η διαδικασία αποτυγχάνει και τερματίζεται. Προκειμένου να ολοκληρωθεί με ορθό τρόπο το `deploy` πρέπει να αφαιρέσουμε την δήλωση της βιβλιοθήκης από το αρχείο `requirements.txt`.

IV.M all_funcs_app

Αυτή η εφαρμογή περιλαμβάνει όλες τις συναρτήσεις που περιγράψαμε παραπάνω. Η ύπαρξη αυτής της εφαρμογής επιβεβαιώνει για ακόμη μια φορά ότι στο FN Project συναρτήσεις διαφορετικών γλωσσών προγραμματισμού μπορούν να ανήκουν σε μια συνάρτηση χωρίς να παρουσιάζεται κάποιο πρόβλημα.

Προκειμένου να δημιουργήσουμε με μια εντολή μια εφαρμογή που θα περιλαμβάνει περισσότερες από μια συναρτήσεις πρέπει στον υπολογιστή μας όλοι οι φάκελοι που περιλαμβάνουν τα αρχεία των συναρτήσεων να είναι ομαδοποιημένοι σε έναν φάκελο ο οποίος συνίσταται (χωρίς

βέβαια να είναι απαραίτητο να φέρει το όνομα της συνάρτησης που θέλουμε να φτιάξουμε.). Σε αυτόν τον φάκελο απαραίτητο είναι να υπάρχει ένα αρχείο `yaml` με την ονομασία `“app.yaml”` το οποίο θα καθορίζει το όνομα της εφαρμογής που φτιάχνουμε.

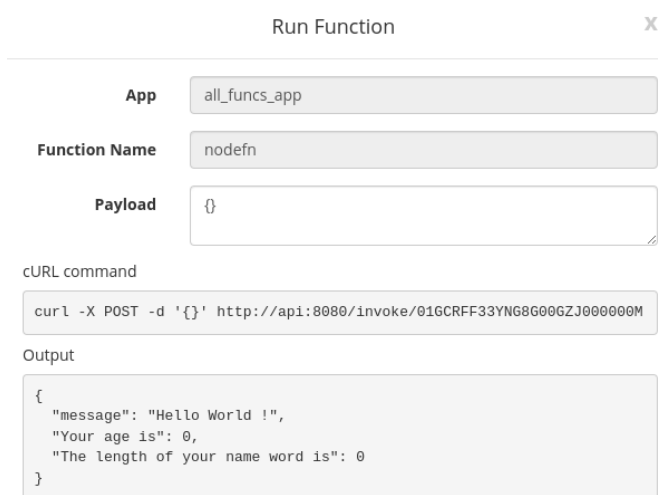
Στο εν λόγω παράδειγμα στο αρχείο `app.yaml` έχουμε την καταχώρηση `“name: all_funcs_app”` καθώς η εφαρμογή μας θέλουμε να έχει το όνομα `“all_funcs_app”`.

Κάνοντας χρήση της εντολής `fn -verbose deploy --all --app all_funcs_app --local` προσθέτουμε όλες τις συναρτήσεις που υπάρχουν στον φάκελο στην εφαρμογή `all_funcs_app`. Παρατηρούμε ότι η διαδικασία εκτέλεσης απαιτεί συνολικά αρκετά περισσότερο χρόνο από ότι στις προηγούμενες εφαρμογές. Αυτό είναι αναμενόμενο διότι εδώ προσθέτουμε έναν μεγαλύτερο αριθμό συναρτήσεων σε μια εφαρμογή από ότι στις προηγούμενες περιπτώσεις. Στο terminal μπορούμε να δούμε ότι το FN Project εκτελεί ξεχωριστά την διαδικασία για κάθε συνάρτηση η οποία προφανώς διαφέρει από γλώσσα σε γλώσσα. Για παράδειγμα όταν θα κάνει `deploy` μια συνάρτηση `java` είναι απαραίτητο εκτός από το αρχείο πηγαίου κώδικα να εκτελέσει και τα `test functions` τα οποία το συνοδεύουν. Ως εκ τούτου ο χρόνος εκτέλεσης της εντολής `deploy` και κατ’ επέκταση της δημιουργίας της εφαρμογής `all_funcs_app` είναι αυξημένος σε σχέση με τις προηγούμενες περιπτώσεις.

Κατόπιν της δημιουργίας της εφαρμογής αυτής είμαστε σε θέση να τρέξουμε χωριστά την κάθε συνάρτηση-μέλος τόσο μέσω terminal όσο και μέσω γραφικού περιβάλλοντος όπως σε όλες τις προηγούμενες περιπτώσεις.

Παρακάτω ακολουθούν screenshots από την εκτέλεση κάποιων από τις συναρτήσεις που ανήκουν στην `all_funcs_app`.

Εκτέλεση της `nodefn` χωρίς δεδομένα εισόδου μέσω γραφικού περιβάλλοντος:



Run Function

App: all_funcs_app

Function Name: nodefn

Payload: {}

cURL command

```
curl -X POST -d '{}' http://api:8080/invoke/01GCRFF33YNG8G00GZJ000000M
```

Output

```
{
  "message": "Hello World!",
  "Your age is": 0,
  "The length of your name word is": 0
}
```

Εκτέλεση της συνάρτησης `hello` μέσω terminal χωρίς δεδομένα εισόδου.

```
gb@gb-Lenovo-Ideapad-100-151BD:~/fn_funcs$ fn invoke all_funcs_app hello
{"message": "Hello World!"}
```

Εκτέλεση της συνάρτησης `goodbye` μέσω terminal με δεδομένα εισόδου `{“name”:”Hiryu”}`.

```
gb@gb-Lenovo-Ideapad-100-151BD:~/fn_funcs$ echo -n '{"name": "Hiryu"}' | fn invoke all_funcs_app goodbye
{"message": "Goodbye Hiryu!"}
```

V. ΛΟΙΠΕΣ ΠΑΡΑΤΗΡΗΣΕΙΣ ΚΑΙ ΣΥΜΠΕΡΑΣΜΑΤΑ

Γενικά το FN Project αποτελεί μια αξιόπιστη και εύκολη στην χρήση πλατφόρμα `serverless computing`. Η διαδικασία προετοιμασίας του συστήματος του υπολογιστή και της εγκατάστασης είναι εξαιρετικά απλές καθώς όπως περιγράψαμε και παραπάνω απαιτούνται μόνο μερικές εντολές μέσω του terminal. Η λοιπή παραμετροποίηση της πλατφόρμας η οποία έπεται της εγκατάστασης δεν παρουσιάζει ιδιαίτερη δυσκολία και σε κάθε περίπτωση οι οδηγίες που παρέχονται στον επίσημο ιστότοπο του εργαλείου και περιγράφουν τα βήματα για τις διαδικασίες αυτές είναι πολύ χρήσιμες και βοηθητικές.

Επιπλέον στον ιστότοπο αυτόν υπάρχει μια μεγάλη ποικιλία από παραδείγματα τα οποία παρουσιάζουν και καλύπτουν την ανάπτυξη συναρτήσεων και εφαρμογών σε διάφορες γλώσσες τις οποίες υποστηρίζει το FN Project.

Τα παραδείγματα αυτά είναι λεπτομερή και κατανοητά και αποτελούν ιδανικό τρόπο για κάποιον αρχάριο χρήστη προκειμένου να αποκτήσει μια πρώτη ιδέα για τις δυνατότητες του εργαλείου και να αποκτήσει βασικές ικανότητες χειρισμού και αξιοποίησης των δυνατοτήτων της πλατφόρμας.

Μετά την παρουσίαση όλων των ανωτέρω παραδειγμάτων επίδειξης συναρτήσεων και εφαρμογών γίνεται κατανοητό το πλεονέκτημα που μας προσφέρει η αναδυόμενη δυνατότητα του `serverless computing`. Όπως παρατηρήσαμε σε όλες τις παραπάνω περιπτώσεις μας δίνεται η δυνατότητα να εκτελέσουμε κώδικα χωρίς να απαιτείται να έχουμε εγκατεστημένο στον τοπικό μας υπολογιστή εργαλεία και εξαρτήσεις για τις διάφορες γλώσσες προγραμματισμού στις οποίες συντάσσουμε την εκάστοτε συνάρτηση. Το FN Project φροντίζει κάθε φορά να κατεβάσει κάθε τι που είναι απαραίτητο για την σωστή εκτέλεση μέσω `docker images`.

Επιπλέον σημαντική είναι και η δυνατότητα αυτόματης δημιουργίας και εκτέλεσης `test functions` που παρέχει το εργαλείο για εφαρμογές σε γλώσσες όπως η `java` και η `C#`. Με τον τρόπο αυτό είμαστε σε θέση να έχουμε με άμεσο τρόπο μια αξιόπιστη εικόνα για το αν η συνάρτησή μας έχει την επιθυμητή έξοδο για διάφορους συνδυασμούς εισόδων και να εντοπίσουμε τυχόν λάθη που κάναμε κατά την διαδικασία σύνταξης της.

Πολύ βοηθητική, για τους προγραμματιστές που ασχολούνται με το εργαλείο, είναι η δυνατότητα έναρξης του `fn server` σε λειτουργία αποσφαλμάτωσης (`Debug mode`). Με τον τρόπο αυτό παρέχεται μια πληρέστερη εικόνα των διαδικασιών που εκτελούνται παρασκηνιακά κατά την εκτέλεση μιας συνάρτησης. Ως εκ τούτου ο προγραμματιστής είναι σε θέση να κατανοήσει σε μεγαλύτερο βαθμό το πώς λειτουργεί το εργαλείο, να εντοπίσει τυχόν σφάλματα που προκύπτουν κατά την εκτέλεση και να τα διορθώσει.

Κατά την διάρκεια εκπόνησης της εργασίας αυτής πολλές φορές βρεθήκαμε αντιμέτωποι με σφάλματα κατά την εκτέλεση των συναρτήσεων. Η ύπαρξη του `Debug mode` μας βοήθησε σε όλες αυτές τις περιπτώσεις να τα ανακαλύψουμε και να απαλείψουμε.

Τέλος η ύπαρξη του γραφικού περιβάλλοντος του εργαλείου δίνει στον χρήστη ακόμη μεγαλύτερη ευελιξία όσον αφορά την εκτέλεση και την παρατήρηση των αποτελεσμάτων των συναρτήσεων. Επιπρόσθετα παρέχει μια συνολική εικόνα των εφαρμογών και των συναρτήσεων που περιλαμβάνει η κάθε μια από αυτές κάτι το οποίο είναι πολύ βοηθητικό για τον προγραμματιστή που χρησιμοποιεί την πλατφόρμα αυτή.

ΠΑΡΑΠΟΜΠΕΣ

- 1 <https://fnproject.io/tutorials/>
- 2 <https://github.com/fnproject/fn#quickstart>
- 3 <https://github.com/fnproject/docs>
- 4 <https://github.com/fnproject/ui>