Διαδικτυακή εφαρμογή για τον δυναμικό συνεπιβατισμό μεταφορικών μέσων

Εθνικό Μετσόβιο Πολυτεχνείο

Ευστράτιος Γιακουμάκης

1η Σεπτεμβρίου 2024

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

Κεφάλαιο 2

Flutter

Το Flutter είναι ένα framework ανάπτυξης cross-platform εφαρμογών για Android, iOS και Web που δημιουργήθηκε και αναπτύσσεται από την Google. Η Google παρουσίασε για πρώτη φορά το 2015 μία πειραματική έκδοση του Flutter υπό την ονομασία "Sky", αναφέροντας ως πρωταρχικό στόχο του νέου αυτού framework την εύκολη και γρήγορη ανάπτυξη αποδοτικών mobile εφαρμογών. Το Flutter είναι άρρηκτα συνδεδεμένο με την γλώσσα προγραμματισμού Dart, η οποία αναπτύσσεται επίσης από την Google.

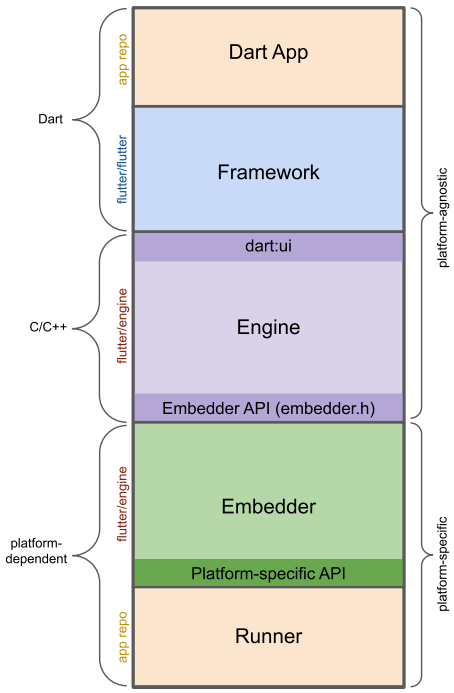
Ως framework ανάπτυξης cross-platform εφαρμογών, το Flutter καθιστά δυνατή την συγγραφή ενιαίου κώδικα για όλες τις υποστηριζόμενες πλατφόρμες. Ο προγραμματιστής αρκεί να αναπτύξει την εφαρμογή μία φορά στη γλώσσα Dart, αγνοώντας σε μεγάλο βαθμό τις λεπτομέρειες υλοποίησης της κάθε πλατφόρμας, και όλες οι διαδικασίες που είναι απαραίτητες για την λειτουργία της εφαρμογής στη εκάστοτε πλατφόρμα γίνονται αυτόματα. Συγκεκριμένα, μέσω του Flutter SDK ο κώδικας Dart που περιλαμβάνει την λογική και το UI της εφαρμογής μεταγλωττίζεται σε native machine code (πχ. ARM για κινητές συσκευές)

2.1 Cross-platform

Η cross-platform λειτουργία επιτυγχάνεται εν μέρει μέσα από την συμπερίληψη στο Flutter ειδικού rendering engine (Skia/Impeller) ο οποίος χειρίζεται εξ ολοκλήρου την σωστή εμφάνιση των στοιχείων του UI ανεξαρτήτως της πλατφόρμας. Έτσι εφαρμογές Flutter δεν χρησιμοποιούν τα native UI components του κάθε λογισμικού συστήματος, αλλά αντιθέτως «ζωγραφίζουν» τα δικά τους components πάνω σε έναν καμβά, pixel ανά pixel. Αυτή η μέθοδος απεικόνισης προσφέρει περισσότερη ομοιομορφία στην εμφάνιση των εφαρμογών σε όλες τις πλατφόρμες, ξεχωρίζοντας το Flutter από άλλα frameworks όπως το React Native (το οποίο αποτελεί τον κύριο ανταγωνιστή του Flutter στον τομέα των cross-platform εφαρμογών).

2.2 Αρχιτεκτονική

Μια εφαρμογή Flutter αποτελείται από τα παρακάτω αρχιτεκτονικά επίπεδα:



- **Runner και Embedder**: Αποτελούν τη βάση της εφαρμογής και τον δίαυλο επικοινωνίας με το λογισμικό σύστημα. Είναι υπεύθυνα για την εκκίνηση της εφαρμογής, την ενημέρωσή της σχετικά με όλα τα system events, και την δημιουργία του βάθρου πάνω στο οποίο ο engine θα εμφανίσει τα περιεχόμενα της εφαρμογής. Εφόσον τα δύο αυτά επίπεδα είναι άμεσα συνυφασμένα με το λογισμικό, ο σχετικός κώδικας είναι προσαρμοσμένος ειδικά για την εκάστοτε πλατφόρμα (πχ. χρησιμοποιείται Java/C++ για Android, και Objective-C για iOS)

- **Engine**: Αποτελεί τον πυρήνα μίας εφαρμογής Flutter. Συνιστάται από ένα runtime το οποίο εκτελεί τον κώδικα της εφαρμογής που είναι γραμμένος σε Dart, και χειρίζεται παράλληλα την εμφάνιση όλων των γραφικών στην οθόνη. Εφόσον επικοινωνεί με το λογισμικό διαμέσου του embedder, ο engine είναι ανεξάρτητος της πλατφόρμας (platform-agnostic)

- **Framework και Dart App**: Αποτελούν το υψηλού επιπέδου τμήμα της εφαρμογής. Το Flutter framework περιέχει όλα τα components με τα οποία ο προγραμματιστής θα συνθέσει την εφαρμογή, καθώς και εργαλεία για τον χειρισμό υπηρεσιών όπως τα animations και το gesture detection. Το τελευταίο επίπεδο είναι το μέρος της εφαρμογής που σχεδιάζεται εξ ολοκλήρου από τον developer, και περιέχει όλη την λογική μαζί με το UI, όπως αυτά έχουν οριστεί από τον προγραμματιστή. Τόσο το Flutter framework όσο και ο κώδικας της εφαρμογής είναι γραμμένα στη γλώσσα Dart.

2.3 Widgets

Η βασική μονάδα όλων των στοιχείων του UI στο Flutter είναι το widget. Ένα widget περιγράφει ένα μέρος του UI της εφαρμογής, όπως ένα κουμπί, κείμενο, ή εικονίδιο. Πέρα από τα εμφανή λειτουργικά στοιχεία, στα widgets συμπεριλαμβάνονται και στοιχεία που ελέγχουν την διάταξη/layout της εφαρμογής, όπως τα widget στοίχισης, padding, και δημιουργίας στηλών/γραμμών. Η εφαρμογή έχει ως ρίζα ένα βασικό widget το οποίο αποτελεί το υπόβαθρο του συνόλου της διεπαφής, και όλα τα στοιχεία του UI δομούνται μέσα από την σύνθεση και την εμφώλευση απλών widget. Έτσι όλες οι εφαρμογές Flutter έχουν την δομή ενός δέντρου, με το κάθε widget να διαθέτει έναν γονιό/parent (το widget μέσα στο οποίο είναι εμφωλευμένο) και ενδεχόμενα παιδιά/children (τα widget τα οποία περιέχει).

Κεφάλαιο 3

JavaScript

Η JavaScript είναι γλώσσα προγραμματισμού η οποία κατέχει κεντρική θέση στον τομέα των διαδικτυακών εφαρμογών. Η JavaScript χρησιμοποιείται πρωτίστως στο client-side/frontend μέρος των εφαρμογών, όπου εκτελείται από τον browser του χρήστη με σκοπό να προσφέρει διαδραστικότητα σε ιστοσελίδες HTML/CSS, αλλά η χρήση της εντοπίζεται και στο backend κομμάτι, δηλαδή σε servers.

Δημιουργήθηκε από τον Brendan Eich το 1995 υπό την ηγεσία της Netscape ως πρόσθετο στον τότε δημοφιλή browser Netscape Navigator, προκειμένου να καταστήσει δυνατή την εκτέλεση δυναμικής συμπεριφοράς στις έως τότε στατικές ιστοσελίδες. Σταδιακά και άλλες ανταγωνίστριες εταιρείες ενσωμάτωσαν την JavaScript στους browser τους, αναπτύσσοντας η καθεμία τον δικό της engine για την εκτέλεση κώδικα JavaScript (V8 από την Google, JavaScriptCore από την Apple, SpiderMonkey από τη Mozilla). Η εξάπλωση της JavaScript οδήγησε στην ανάγκη τυποποίησης της γλώσσας για χρήση σε διαδικτυακές εφαρμογές, ώστε να διευκολυνθεί η συγγραφή κώδικα συμβατού με όλους τους browser της αγοράς. Έτσι το 2009, μετά από τη σύσταση επιτροπής αποτελούμενη από εκπροσώπους των κύριων εταιρειών (Google, Microsoft, Mozilla, και άλλες) δημιουργήθηκε το πρότυπο ECMAScript 5, το οποίο έκτοτε ακολουθείται από όλους τους συμβατικούς browsers.

Η JavaScript είναι τυπικά interpreted γλώσσα, ωστόσο σε όλες πλέον τις συμβατικές υλοποιήσεις της (συμπεριλαμβανομένων όλων των σύγχρονων browser) ο κώδικας γίνεται και just-in-time compiled (JIT). Με αυτόν τον τρόπο επιτυγχάνεται η γρήγορη ταχύτητα εκτέλεσης η οποία είναι απαραίτητη στις διαδικτυακές εφαρμογές, καθώς συνδυάζεται ο μικρός χρόνος εκκίνησης ενός interpreter μαζί με τις υψηλές επιδόσεις που προσφέρει η μεταγλώττιση κομβικών τμημάτων του κώδικα σε κώδικα μηχανής.

Όσον αφορά τα χαρακτηριστικά της, η JavaScript είναι γλώσσα που προσφέρεται τόσο για αντικειμενοστραφή όσο και για συναρτησιακό προγραμματισμό.

Οι αντικειμενοστραφείς λειτουργίες της γλώσσας υλοποιούνται μέσα από τα πρότυπα (prototypes). Κάθε αντικείμενο που δημιουργείται κατά την εκτέλεση του κώδικα μέσω ενός constructor διαθέτει ένα property που αναφέρεται στο πρότυπο στο οποίο ανήκει. Το πρότυπο είναι και αυτό με τη σειρά του ένα αντικείμενο το οποίο περιέχει ένα σύνολο από properties. Αυτό σημαίνει ότι κάθε αντικείμενο έχει αυτόματα πρόσβαση σε όλες τις ιδιότητες του προτύπου στο οποίο ανήκει, και επομένως *όλα* τα αντικείμενα

του ίδιου προτύπου μοιράζονται όλα τα properties του. Κάθε φορά που προσπελάζεται μία ιδιότητα ενός αντικειμένου, αρχικά ελέγχεται η ύπαρξη property του αντικειμένου με το ίδιο όνομα (own property). Σε περίπτωση που η ιδιότητα δε βρεθεί ανάμεσα στα own properties του αντικειμένου, η ιδιότητα αναζητείται ανάμεσα στα properties του προτύπου στο οποίο ανήκει. Η διαδικασία αυτή συνεχίζεται αναζητώντας αναδρομικά στο πρότυπου του προτύπου κ.ο.κ, τερματίζοντας είτε μόλις η ιδιότητα βρεθεί, είτε μόλις η αλυσίδα των προτύπων φτάσει στο βασικό πρότυπο που ονομάζεται 'Object'. Συνεπώς, τα πρότυπα υλοποιούν ακριβώς την έννοια της *κληρονομικότητας*, η οποία αποτελεί τον βασικό πυλώνα του αντικειμενοστραφούς προγραμματισμού. Εξάλλου, τα πρότυπα προσφέρουν μεγάλα περιθώρια βελτιστοποίησης, καθώς ιδιότητες οι οποίες είναι ίδιες για μία οικογένεια αντικειμένων μπορούν να τοποθετηθούν σε ένα κοινό πρότυπο, με αποτέλεσμα τη μείωση της απαιτούμενης μνήμης και την αύξηση της επίδοσης. Η έκδοση ECMAScript 5 η οποία κυκλοφόρησε το 2015 πρόσθεσε κλάσεις στην JavaScript ως βασική πια μονάδα της γλώσσας. Η υλοποίηση των κλάσεων γίνεται στην πραγματικότητα και αυτή μέσα από τα πρότυπα, αλλά όλες οι ιδιότητες που αναμένονται από μία ολοκληρωμένη αντικειμενοστραφή γλώσσα (στατικές μέθοδοι, access modifiers, getters/setters, και φυσικά κληρονομικότητα) είναι πια άμεσα διαθέσιμες.

Από την άλλη, η JavaScript χαρακτηρίζεται και ως γλώσσα συναρτησιακού προγραμματισμού. Οι συναρτήσεις στην JavaScript αντιμετωπίζονται ως απλά αντικείμενα, γεγονός που σημαίνει ότι με απόλυτα φυσικό τρόπο μπορούν να χρησιμοποιηθούν ως μεταβλητές, να οριστούν ως literals (anonymous functions), και να αποτελέσουν παραμέτρους άλλων συναρτήσεων. Οι συναρτήσεις στην JavaScript είναι δηλαδή first-class citizens. Έτσι η γλώσσα προσφέρεται για προγραμματισμό που βασίζεται στη σύνθεση συναρτήσεων, ενώ η ύπαρξη περαιτέρω λειτουργιών όπως τα function closures επιτρέπουν πιο προχωρημένες μεθόδους συναρτησιακού προγραμματισμού.

Ένα ακόμη σημαντικό χαρακτηριστικό της JavaScript είναι το weak και dynamic typing. Το dynamic typing αντιδιαστέλλεται με το static typing, και αναφέρεται στο γεγονός ότι η γλώσσα δεν πραγματοποιεί στατικό έλεγχο των τύπων κατά τη μεταγλώττιση του κώδικα, αλλά αντιθέτως καθορίζει τους τύπους των μεταβλητών μόνο κατά τη στιγμή της εκτέλεσης, βάσει της τιμής τους. Αυτό σημαίνει ότι ο τύπος μίας μεταβλητής δεν καθορίζεται όταν αυτή δηλώνεται, και μπορεί να αλλάξει ανά πάσα στιγμή κατά την εκτέλεση του προγράμματος (σε αντίθεση για παράδειγμα με τη C/C++ όπου ο τύπος δηλώνεται πάντα όταν ορίζεται μια μεταβλητή και παραμένει αμετάβλητος καθ' όλη τη διάρκεια του προγράμματος).

Ο όρος weakly typed από την άλλη δεν είναι τόσο καθορισμένος. Σε γενικές γραμμές αναφέρεται στο γεγονός ότι η γλώσσα επιτρέπει την αυτόματη και υπόρρητη μετατροπή τιμών από ένα τύπο σε άλλον. Η JavaScript χαρακτηρίζεται ως weakly typed επειδή πραγματοποιεί implicit type conversions πρακτικά σε κάθε περίπτωση πράξης μεταξύ διαφορετικών τύπων, αντί να εγείρει εξαίρεση τύπου όπως θα έκανε μία strongly typed γλώσσα. Για παράδειγμα, στην JavaScript μία "πρόσθεση" ενός string με έναν αριθμό όπως `"example" + 2024` μετατρέπει τον αριθμό σε string και έπειτα πραγματοποιεί concatenation, με τελικό αποτέλεσμα το string `"example2024"`.

Το dynamic/weak typing της JavaScript μπορεί να θεωρηθεί πιο πρακτικό από το static/strong typing άλλων γλωσσών, καθώς απαλλάσσει τον προγραμματιστή από την ανάγκη ορισμού και διαχείρισης όλων των τύπων, ενώ μειώνει τον νοητικό φόρτο που

οφείλεται στα casting και τις ρητές μετατροπές τύπων. Ωστόσο, αυτό το είδος ”χαλαρού” συστήματος τύπων μπορεί να δημιουργήσει προβλήματα. Οι δυναμικοί τύποι δεν προφυλάσσουν το πρόγραμμα από κάποια βασικά σφάλματα τα οποία ένα στατικό σύστημα τύπων θα εντόπιζε άμεσα κατά τη μεταγλώττιση του προγράμματος, ενώ οι ”κρυφές” μετατροπές τύπων μπορούν πολύ εύκολα να έχουν απρόσμενες συνέπειες οι οποίες δεν είναι άμεσα εμφανείς στον προγραμματιστή. Παράλληλα, η χρήση τύπων μπορεί στην πραγματικότητα να μειώσει τον νοητικό φόρτο του προγραμματιστή· εφόσον ο τύπος κάθε μεταβλητής είναι γνωστός και αμετάβλητος, η λειτουργία του προγράμματος είναι πιο εύκολα ελέγξιμη και συντηρήσιμη.

3.1 TypeScript

Η TypeScript είναι γλώσσα που χτίζει πάνω στην JavaScript, προσφέροντας στατικό έλεγχο τύπων κατά τη μεταγλώττιση του κώδικα προκειμένου να αντιμετωπίσει τις αδυναμίες του dynamic typing. Οποιοδήποτε πρόγραμμα JavaScript μπορεί να προαχθεί σε TypeScript προσθέτοντας προαιρετικούς τύπους στις μεταβλητές που χρησιμοποιεί. Έτσι, σε αντίθεση με τη γνήσια JavaScript όπου οι τύποι όλων των μεταβλητών καθορίζονται μόνο τη στιγμή της εκτέλεσης, η TypeScript συνάγει τους τύπους των μεταβλητών, και ελέγχει για πιθανά σφάλματα τύπων *πριν* την εκτέλεση του κώδικα, αποφεύγοντας έτσι ένα σημαντικό μέρος των runtime errors και bugs. Η TypeScript δίνει επίσης τη δυνατότητα στον προγραμματιστή να προσθέσει type annotations σε παραμέτρους συναρτήσεων, ενώ είναι επίσης δυνατός ο ορισμός interfaces για τον χειρισμό δομημένων δεδομένων (structured data), όπως αυτά που στέλνει και δέχεται ένα API. Με αυτόν τον τρόπο, εκτός από την αυξημένη προστασία έναντι σφαλμάτων, λανθασμένης χρήσης μεταβλητών και ανεπιθύμητων συμπεριφορών, ο ίδιος ο κώδικας γίνεται συγχρόνως πιο κατανοητός και εύκολα διαχειρίσιμος από τον προγραμματιστή. Ένα αρχείο TypeScript μεταφράζεται πάντα σε καθαρή JavaScript (transpilation) πριν εκτελεστεί. Κατά τη διάρκεια της μετάφρασης αυτής όλες οι πληροφορίες που αφορούν τους τύπους αφαιρούνται από το πρόγραμμα εφόσον έχει επαληθευτεί η ορθότητά τους, με αποτέλεσμα ο τελικός κώδικας να είναι εκτελέσιμος από οποιονδήποτε JavaScript engine χωρίς την ανάγκη ειδικής μεταχείρισης των χαρακτηριστικών της TypeScript.

Κεφάλαιο 4

Node.js

Κεφάλαιο 5

Websocket

Κεφάλαιο 6

Frontend

Κεφάλαιο 7

Backend

Παράρτημα Α΄

Συμπληρωματικό υλικό