Διαδικτυακή εφαρμογή για τον δυναμικό συνεπιβατισμό μεταφορικών μέσων

Εθνικό Μετσόβιο Πολυτεχνείο

Ευστράτιος Γιακουμάκης

1η Σεπτεμβρίου 2024

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Το πρόβλημα

Το πρόβλημα το οποίο καλούμαστε να λύσουμε αφορά τον συνωστισμό που παρατηρείται συχνά στη στάση λεωφορείου του *Στρατιωτικού Νοσοκομείου* (στο εξής ΣΝ) στη λεωφόρου Κατεχάκης κοντά στη σχολή του Ε.Μ.Π. Ένα σημαντικό ποσοστό των φοιτητών προσέρχεται καθημερινά στη σχολή χρησιμοποιώντας τα μέσα μαζικής μεταφοράς, και συγκεκριμένα το λεωφορείο 242 που μεταφέρει άτομα από τη στάση ΣΝ απευθείας στη σχολή. Η στάση έχει κομβική θέση λόγω του μεγάλου αριθμού φοιτητών που τη χρησιμοποιούν, ωστόσο το γεγονός αυτό έχει ως αποτέλεσμα την αδυναμία επαρκούς κάλυψης των αναγκών από τα δρομολόγια των λεωφορείων, και επομένως τη συγκέντρωση πλήθους στην περιοχή. Η συγκέντρωση πλήθους με τη σειρά της σημαίνει ότι οι φοιτητές αναγκάζονται να σταθούν εκτός του πεζοδρομίου και εντός της λεωφόρου, διαταράσσοντας την κυκλοφορία και θέτοντας την ασφάλειά τους σε κίνδυνο. Εκτός αυτού, πολλές φορές ο αριθμός φοιτητών είναι αρκετά μεγάλος ώστε να υπερβαίνει τη χωρητικότητα του κάθε λεωφορείου που φτάνει στη στάση, με αποτέλεσμα οι επιβαίνοντες στο λεωφορείου να συνωστίζονται, και οι φοιτητές που αδυνατούν να επιβιβαστούν να χρειάζεται να αναμένουν το επόμενο λεωφορείο. Η κατάσταση είναι χειρότερη κατά τις ώρες αιχμής, ενώ δυσχεραίνεται περαιτέρω όταν παρατηρείται κυκλοφοριακή συμφόρηση ή καθυστερήσεις στα δρομολόγια, αλλά και κατά τη διάρκεια της εξεταστικής περιόδου όπου η προσέλευση των φοιτητών είναι μέγιστη.

Ταυτόχρονα με τους φοιτητές που μεταβαίνουν στη σχολή μέσω των ΜΜΜ, ένας αριθμός φοιτητών αντιθέτως χρησιμοποιεί οχήματα ΙΧ για τη μεταφορά του. Έχει παρατηρηθεί επομένως το φαινόμενο οι φοιτητές αυτοί να σταματάνε το όχημά τους στη στάση για να επιτρέψουν σε συμφοιτητές τους να εισέλθουν στο όχημα και να μεταβούν μαζί στη σχολή, μειώνοντας κατά ένα μικρό βαθμό τον συνωστισμό.

Η εφαρμογή που αναπτύξαμε έχει στόχο την αξιοποίηση αυτής ακριβώς της συμπεριφοράς προκειμένου να καταπολεμηθεί πιο στοχευμένα και αποτελεσματικά το παραπάνω πρόβλημα. Συγκεκριμένα η εφαρμογή συντονίζει το ταίριασμα των εν κινήσει οδηγών με τους πεζούς που αναμένουν στη στάση, προσφέροντας με αυτόν τον τρόπο ένα πιο οργανωμένο πλαίσιο στο οποίο μπορεί να συμβεί ο συνεπιβατισμός ο

οποίος αναπτύχθηκε οργανικά στις δεδομένες συνθήκες.

1.2 Συνεπιβατισμός

Ο όρος *συνεπιβατισμός* (carpooling στα Αγγλικά) αναφέρεται στη χρήση ενός οχήματος ΙΧ από πολλά άτομα για την άφιξη σε έναν κοινό προορισμό. Στην πιο συνηθισμένη περίπτωση, ο συνεπιβατισμός περιλαμβάνει την εκ των προτέρων συνεννόηση μεταξύ του επιβάτη και του οδηγού πριν την έναρξη της διαδρομής, προκειμένου να αποφασιστεί ο ακριβής προορισμός, η ώρα συνάντησης, και το σημείο επιβίβασης. Επομένως, αυτό το είδος συνεπιβατισμού εν γένει διαθέτει λιγότερη ευελιξία, και περιορίζεται κυρίως σε φίλους, ή συνεργάτες. Μία άλλη μορφή συνεπιβατισμού αναπτύχθηκε στην Washington DC τη δεκαετία του ’70 και έπειτα εξαπλώθηκε και σε άλλες πόλεις της Αμερικής, το λεγόμενο casual carpooling ή ”slugging”. Σε ορισμένους αυτοκινητοδρόμους τα οχήματα ΙΧ με περισσότερους από έναν επιβάτη είχαν πρόσβαση σε λωρίδες ταχείας κυκλοφορίας και μειωμένα τέλη διοδίων, γεγονός που έδινε κίνητρο σε οδηγούς να δέχονται πεζούς στο όχημα τους και να τους μεταφέρουν στον προορισμό τους. Καθώς η δραστηριότητα αυτή ωφελούσε αμφότερους τους πεζούς και τους οδηγούς, το casual carpooling άνθισε στις πόλεις αυτές, με τη γρήγορη καθιέρωση ημιεπίσημων ”στάσεων” όπου οι πεζοί ακόμη και σήμερα μπορούν να αναμένουν τους διερχόμενους οδηγούς. Αυτό το είδος συνεπιβατισμού δίνει τη δυνατότητα σε άτομα άγνωστα μεταξύ τους να μοιραστούν το όχημα άμεσα και χωρίς την ανάγκη προηγούμενης συνεννόησης, ωστόσο είναι περιορισμένο ως προς τα σημεία συνάντησης, καθώς αυτά είναι λιγοστά και καθορισμένα.

Με την έλευση των κινητών εφαρμογών η δημοφιλία του συνεπιβατισμού αυξήθηκε ραγδαία, μέσω των λεγόμενων ride-hailing apps. Μέσω των εφαρμογών αυτών οι χρήστες έχουν πλέον τη δυνατότητα να αναζητήσουν επιτόπου οδηγό, οπουδήποτε και αν βρίσκονται. Οι εφαρμογές παρουσιάζουν όλους τους κοντινούς διαθέσιμους οδηγούς στον χρήστη, ο οποίος αφού επιλέξει εκείνον που προτιμάει διαλέγει ελεύθερα τον προορισμό του. Έτσι αίρονται πρακτικά όλοι οι περιορισμοί που αναφέρθηκαν προηγουμένως, ωστόσο οι ανέσεις που προσφέρονται στον πεζό επιβάλλουν τη χρέωση της υπηρεσίας. Μέσα από την πελατειακή σχέση επιβάτη-οδηγού που διαμορφώνεται με αυτόν τον τρόπο η υπηρεσία απομακρύνεται από τον κλασσικό συνεπιβατισμό, και μετατρέπεται σε ένα εμπορικό προϊόν που ταυτίζεται σχεδόν με τα ταξί. Πράγματι, σχεδόν όλες οι διαθέσιμες εφαρμογές ride-hailing είναι εμπορικές. Η πρώτη εταιρεία που εισήλθε στον τομέα των ride-hailing εφαρμογών ήταν η Uber, η οποία το 2010 δημιούργησε την επώνυμη εφαρμογή της, και η οποία διατηρεί μέχρι και σήμερα το μεγαλύτερο μερίδιο της αγοράς σε παγκόσμιο επίπεδο.

1.3 Η λειτουργία της εφαρμογής

Η εφαρμογή που αναπτύξαμε πλησιάζει περισσότερο στη φιλοσοφία της το casual carpooling, αλλά ενσωματώνει και χαρακτηριστικά από τις εφαρμογές ride-hailing. Το σημείο συνάντησης καθώς και ο τελικός προορισμός είναι καθορισμένα: ο πεζός πρέπει να μεταβεί στη στάση του ΣΝ για να μπορέσει να αναζητήσει οδηγούς μέσω της

εφαρμογής, ενώ ο προορισμός είναι προφανώς η σχολή του πολυτεχνείου. Μόλις ένας οδηγός ενεργοποιήσει την εφαρμογή και εισέλθει εντός μίας ακτίνας μερικών χιλιομέτρων από τη στάση, ενημερώνεται για την ύπαρξη πεζών που περιμένουν, και δηλώνει την πρόθεση του να τους δεχτεί στο όχημά του. Οι πεζοί με τη σειρά τους ενημερώνονται άμεσα μόλις κάποιος οδηγός δεχτεί να τους παραλάβει, και "κλειδώνουν" τη θέση τους στο όχημα. Το μήνυμα για τον διαθέσιμο οδηγό στέλνεται ταυτόχρονα σε πολλαπλούς πεζούς, και η αντίστοιχη θέση παραδίδεται σε εκείνον που απάντησε πιο γρήγορα στην ειδοποίηση. Ύστερα από την επιτυχή αντιστοίχιση πεζού και οδηγού, τα στοιχεία του ενός γίνονται διαθέσιμα στον άλλον ώστε να είναι δυνατή η ταυτοποίηση (όνομα, φωτογραφίες, αριθμός κυκλοφορίας), ενώ παράλληλα παρουσιάζεται το στίγμα τους στο χάρτη για τον προσδιορισμό της θέσης και του χρόνου άφιξης. Μόλις ο οδηγός παραλάβει τους πεζούς από τη στάση, κατευθύνονται μαζί προς τη σχολή, και φτάνοντας σε αυτήν, η εφαρμογή αυτόματα τερματίζει τη συνεδρία. Τέλος, παρέχεται η ευκαιρία αξιολόγησης τόσο από πλευράς του οδηγού όσο και από πλευράς του επιβάτη.

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 τη γλώσσα προγραμματισμού Dart, η οποία αναπτύσσεται επίσης από την Google.

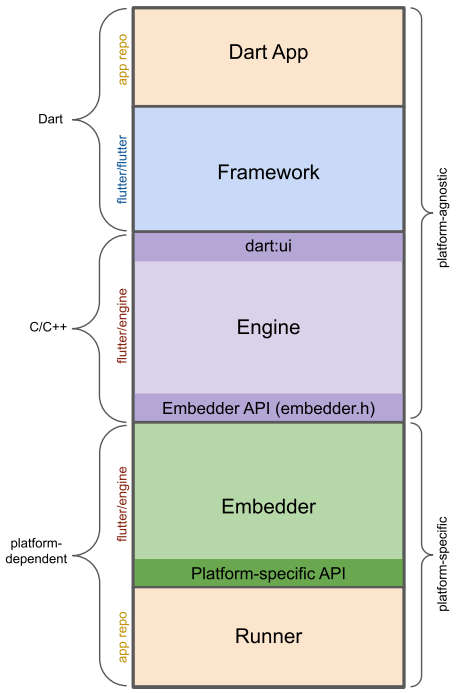
Ως framework ανάπτυξης cross-platform εφαρμογών, το Flutter καθιστά δυνατή τη συγγραφή ενιαίου κώδικα για όλες τις υποστηριζόμενες πλατφόρμες. Ο προγραμματιστής αρκεί να αναπτύξει την εφαρμογή μία φορά στη γλώσσα Dart, αγνοώντας σε μεγάλο βαθμό τις λεπτομέρειες υλοποίησης της κάθε πλατφόρμας, και όλες οι διαδικασίες που είναι απαραίτητες για τη λειτουργία της εφαρμογής στην εκάστοτε πλατφόρμα γίνονται αυτόματα. Συγκεκριμένα, μέσω του Flutter SDK ο κώδικας Dart που περιλαμβάνει τη 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 τη συμπερίληψη στο Flutter ειδικού rendering engine (Skia/Impeller) ο οποίος χειρίζεται εξ ολοκλήρου τη σωστή εμφάνιση των στοιχείων του UI ανεξαρτήτως της πλατφόρμας. Έτσι εφαρμογές Flutter δε 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 τη 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 τη 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 τη 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). Ο κεντρικός αυτός ρόλος των widget δικαιολογεί το σλόγκαν που έχει υιοθετήσει το Flutter, *"everything is a widget"*.

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

Το Node.js είναι ένα open-source runtime environment για την JavaScript. Δημιουργήθηκε το 2009 από τον Ryan Dahl με στόχο τη δυνατότητα εκτέλεσης αποδοτικών προγραμμάτων υψηλού ταυτοχρονισμού (high concurrency) βασιζόμενη σε μία event-driven και non-blocking αρχιτεκτονική. Η κύρια χρήση του Node.js είναι ως περιβάλλον για την εκτέλεση backend εφαρμογών, δηλαδή για τη λειτουργία ενός server.

Μέσω του Node.js ο Ryan Dahl θέλησε να δώσει λύση σε ένα πρόβλημα που ήταν μέχρι τότε διαδεδομένο στον τομέα των εφαρμογών server, αυτό της αδυναμίας διαχείρισης μεγάλου όγκου ταυτόχρονων συνδέσεων. Η δομή των περισσότερων εφαρμογών server βασιζόταν στην αυστηρά σειριακή εκτέλεση εντολών και στη χρήση πολλών ξεχωριστών threads για την εξυπηρέτηση όλων των συνδέσεων, επομένως η απαιτήσεις πόρων και το latency ήταν ιδιαίτερα αυξημένα για εφαρμογές μεγάλης κλίμακας. Η χρήση διαφορετικών threads θεωρούταν απαραίτητη για την αποφυγή παύσης εκτέλεσης της εφαρμογής στην περίπτωση blocking κλήσεων I/O, όπως πρόσβαση στον δίσκο, σε μία βάση δεδομένων, ή στο διαδίκτυο. Το Node.js κατάφερε να ξεπεράσει αυτό το εμπόδιο και να λειτουργεί με ένα μόνο thread, αξιοποιώντας *non-blocking* κλήσεις I/O και το λεγόμενο *event loop*.

Ο κεντρικός άξονας του Node.js είναι το event loop. Το event loop είναι ο βρόχος ο οποίος εκτελείται από το πρόγραμμα καθ' όλη τη διάρκειά ζωής του, και ο οποίος δέχεται όλα τα events που εμφανίζονται (πχ. requests ή asynchronous operations), τα μεταβιβάζει στο λειτουργικό σύστημα για επεξεργασία, και έπειτα εκτελεί τα αντίστοιχα callbacks μόλις ολοκληρωθούν. Το γεγονός ότι το Node.js αναθέτει τις μακροχρόνιες εργασίες στο kernel του συστήματος σημαίνει ότι το ίδιο είναι ελεύθερο να συνεχίσει να επεξεργάζεται τα εισερχόμενα events και να εκτελεί τις σύγχρονες διεργασίες, χωρίς να χρειάζεται να περιμένει την ολοκλήρωση των blocking διεργασιών. Αυτό, σε συνδυασμό με τη single-threaded φύση του event loop, επιτρέπει στο Node.js να χειρίζεται πολύ μεγαλύτερο αριθμό ταυτόχρονων συνδέσεων από άλλες υλοποιήσεις (όπως έναν κλασσικό PHP server), με λιγότερες απαιτήσεις σε μνήμη.

Προκειμένου να εκτελέσει τον κώδικα JavaScript εκτός του περιβάλλοντος ενός browser, το Node.js χρησιμοποιεί τον V8 engine της Google.

Ο κώδικας που εκτελεί το Node.js είναι κώδικας JavaScript. Η JavaScript επιλέχθηκε ως η γλώσσα εκτέλεσης όχι μόνο γιατί ήταν ήδη διαδεδομένη στον χώρο των

διαδικτυακών εφαρμογών, αλλά κυρίως γιατί είναι κατάλληλα σχεδιασμένη για να υποστηρίξει τις απαιτήσεις τους Node.js. Από την απαρχή της η JavaScript υποστήριζε τον event-driven και ασύγχρονο προγραμματισμό, καθώς αυτός ήταν απαραίτητος για την ομαλή non-blocking λειτουργία των ιστοσελίδων, ενώ οι ανώνυμες συναρτήσεις και τα function closures ευνοούσαν περαιτέρω τη χρήση των callbacks. Η χρήση της JavaScript στο Node.js δίνει επίσης τη δυνατότητα για full-stack JavaScript development, δηλαδή ομοιόμορφη χρήση της JavaScript για την ανάπτυξη τόσο του frontend όσο και του backend μέρους μίας εφαρμογής.

Το Node.js άρχισε να γίνεται διάσημο σύντομα μετά από την παρουσίασή του, και πλέον η πλειοψηφία όλων των εταιριών παγκοσμίως χρησιμοποιούν το Node.js σε κάποιο τουλάχιστον μέρος του backend τους. Κάποιες εταιρίες που χρησιμοποιούν σε μεγάλο βαθμό το Node.js είναι η Netflix[1], η PayPal[9], η LinkedIn[3], η NASA[6], και η Bloomberg[4]. Στο παρελθόν η Uber επίσης χρησιμοποιούσε το Node.js για το μεγαλύτερο μέρος του backend της, και μάλιστα ήταν από τις πρώτες μεγάλες εταιρίες που το υιοθέτησαν[5].

Κεφάλαιο 5

Websocket

Κεφάλαιο 6

Frontend

Η εφαρμογή κινητού δημιουργήθηκε χρησιμοποιώντας το framework του Flutter. Το Flutter επιλέχθηκε ως framework κυρίως λόγω της δυνατότητας που προσφέρει για εύκολη και γρήγορη ανάπτυξη εφαρμογών, και λόγω της απαίτησής μας για cross-platform λειτουργία. Η ανάπτυξη της εφαρμογής στη native γλώσσα κάθε πλατφόρμας θα απαιτούσε σημαντικά περισσότερο χρόνο, καθώς θα χρειαζόταν να δημιουργήσουμε δύο ανεξάρτητες εφαρμογές (μία για την πλατφόρμα Android, και μία για την πλατφόρμα iOS) γράφοντας πρακτικά τη διπλάσια ποσότητα κώδικα σε δύο διαφορετικές γλώσσες. Εκτός από τον διπλασιασμό του χρόνου συγγραφής του κώδικα, η ύπαρξη δύο codebase αυξάνει τον αριθμό των ενδεχόμενων σφαλμάτων που μπορεί να προκύψουν, καθιστά σε βάθος χρόνου πιο δύσκολη τη συντήρηση, και απαιτεί επιπλέον προσοχή για τη διασφάλιση της ισότητας των δύο εκδόσεων της εφαρμογής (feature parity). Αξιοποιώντας το Flutter γράψαμε έναν ενιαίο κώδικα ο οποίος λειτουργεί αμέσως και στις δύο πλατφόρμες, ενώ είμαστε σίγουροι ότι η συμπεριφορά της εφαρμογής θα είναι ίδια για όλους τους χρήστες ανεξαρτήτως της συσκευής που χρησιμοποιούν.

Ως οδηγός, ο χρήστης έχει τη δυνατότητα να προσθέσει στο προφίλ του τα οχήματα που χρησιμοποιεί, προσδιορίζοντας τα χαρακτηριστικά που επιτρέπουν την αναγνώρισή τους από τους πεζούς. Συγκεκριμένα, για το κάθε όχημα ο χρήστης προσδιορίζει το μοντέλο (επιλέγοντας από μία περιεκτική λίστα) και τον αριθμό πινακίδας. Προκειμένου να διευκολυνθεί η άμεση αναγνώριση του οχήματος από τους επιβάτες ο χρήστης μπορεί προαιρετικά να προσθέσει το χρώμα αλλά και μία φωτογραφία του.

Πέρα των στοιχείων του οχήματος, ο κάθε χρήστης (πεζός ή οδηγός) μπορεί επίσης προαιρετικά να ανεβάσει μία φωτογραφία προφίλ με την οποία θα παρουσιάζεται στους υπόλοιπους χρήστες. Τέλος, ο χρήστης έχει μία βαθμολογία η οποία βασίζεται σε όλες τις προηγούμενες αλληλεπιδράσεις τους με άλλους χρήστες της εφαρμογής, αναπαριστώντας την ποιότητα του ως οδηγός ή επιβάτης.

Κατά την έναρξη της εφαρμογής, ο χρήστης παραπέμπεται στην υπηρεσία ταυτοποίησης της σχολής προκειμένου να επαληθευτεί η ταυτότητά του ως φοιτητής. Κατά την επιτυχή ταυτοποίηση η εφαρμογή ανοίγει μία σύνδεση WebSocket με τον server, στέλνοντας ταυτόχρονα το ΑΜ και το όνομα του φοιτητή. Σημαντικό είναι το γεγονός ότι η σύνδεση με τον server επιτυγχάνεται μόνο εφόσον έχει επαληθευτεί η ταυτότητα του χρήστη. Με αυτόν τον τρόπο όχι μόνο απαγορεύεται η χρήση της εφαρμογής από

τρίτους, αλλά και ο server προστατεύεται από κακόβουλες ενέργειες. Ακόμα και αν κάποιος χρήστης επιχειρήσει να κάνει κατάχρηση της εφαρμογής ή να ”επιτεθεί” στον server θα πρέπει να έχει ήδη συνδεθεί με τα προσωπικά του στοιχεία, στην οποία περίπτωση η ταυτότητά του μπορεί να προσδιοριστεί και να ληφθούν τα απαραίτητα μέτρα όπως ο αποκλεισμός του συγκεκριμένου λογαριασμού.

Για την ταυτοποίηση χρησιμοποιούμε το AppAuth, έναν OAuth/OpenID Connect client ο οποίος είναι σχεδιασμένος για την επαλήθευση ταυτότητας χρηστών μέσω κινητών εφαρμογών. Η ταυτοποίηση πραγματοποιείται με ασφαλή τρόπο στη συσκευή και το αποτέλεσμα της είναι ένα token (JWT) που περιέχει τα προσωπικά στοιχεία του χρήστη μαζί με μία ”υπογραφή” που πιστοποιεί την αυθεντικότητά του. Το token αυτό στέλνεται στον server κατά την έναρξη της συνεδρίας πάντα μέσω HTTPS σύνδεσης, και εφόσον επαληθευτεί ως έγκυρο ο server γνωρίζει πια την ταυτότητα του χρήστη πίσω από τη συγκεκριμένη σύνδεση. Η διαδικασία γίνεται με τέτοιο τρόπο ώστε σε καμία περίπτωση να μη φανερώνεται ο κωδικός λογαριασμού του χρήστη στην εφαρμογή, ενώ η πιθανότητα κλοπής της ταυτότητάς του είναι κατεσταλμένη. Τα προσωπικά δεδομένα του χρήστη προστατεύονται περαιτέρω καθώς μόνο τα απολύτως απαραίτητα στοιχεία αιτούνται από την εφαρμογή (ΑΜ και όνομα), ο χρήστης πληροφορείται ρητά για την κοινοποίηση των δεδομένων αυτών, και απαιτείται η τελική συναίνεση του για την αποστολή τους στον server.

Μετά από την επιτυχή ταυτοποίηση του, ο χρήστης έχει τη δυνατότητα να επιλέξει το ρόλο του ως οδηγός ή ως επιβάτης.

Η διαδικασία ξεκινάει όταν ο οδηγός πατήσει το κουμπί εκκίνησης, έχοντας προηγουμένως έχει προσδιορίσει το όχημα με το οποίο μετακινείται. Η εφαρμογή αρχικά παραμένει ανενεργή, παρατηρώντας μόνο το στίγμα του οδηγού έως ότου αυτός πλησιάσει αρκετά στη στάση όπου περιμένουν οι πεζοί. Σε αυτό το σημείο η εφαρμογή επικοινωνεί με τον server, ρωτώντας για την ύπαρξη πεζών στη στάση, και αν υπάρχουν όντως πεζοί ειδοποιεί τον οδηγό. Αν ο οδηγός αποδεχτεί, τότε κάποιος από τους πεζούς ορίζεται ως ο επιβάτης του.

Όσον αφορά τον πεζό, η εφαρμογή επίσης παραμένει αρχικά ανενεργή μέχρι κάποιος οδηγός να γίνει διαθέσιμος. Τότε ένας μικρός αριθμός από πεζούς (το πολύ 5) ειδοποιείται για την ύπαρξη του οδηγού, και ο πρώτος από αυτούς που αποδεχτεί την πρόσκληση ορίζεται ως ο επιβάτης του οδηγού.

Από αυτό το σημείο και πέρα, οδηγός και επιβάτης παρουσιάζονται με τα στοιχεία του αλλουνού (όνομα, αριθμός πινακίδας, φωτογραφίες κτλ.) καθώς και με έναν χάρτη της περιοχής πάνω στον οποίο βρίσκονται τα στίγματά τους. Ο οδηγός κατευθύνεται προς τον πεζό, και ο πεζός παρατηρεί στον χάρτη τη διαδρομή του οδηγού του. Μόλις ο οδηγός πλησιάσει στη στάση, και οι δύο τους ειδοποιούνται από την εφαρμογή ώστε ο πεζός να επιβιβαστεί στο όχημα. Σε περίπτωση που ο οδηγός αποχωρήσει μην έχοντας λάβει τον πεζό, η συνεδρία τους ακυρώνεται και η εφαρμογή του πεζού συνεχίζει να αναζητά άλλους οδηγούς. Η εφαρμογή παρακολουθεί την πορεία του οχήματος προς τη σχολή, και μόλις αυτό φτάσει στον προορισμό του, τερματίζει τη διαδρομή και δίνει την επιλογή σε οδηγό και πεζό να αξιολογήσουν ο ένας τον άλλο.

```
class Car {
    String id;
    String model;
```

```
    int seats;
    String license;
    String? picture;
    int? color;
}

class User with ChangeNotifier {
    String id;
    String fullName;
    String givenName;
    String? picture;
    int ratingsSum;
    int ratingsCount;
    Map<String, Car> cars;
}
```

Κεφάλαιο 7

Backend

Η φύση της εφαρμογής επιβάλλει την ύπαρξη ενός κεντρικού server ο οποίος συντονίζει τις διάφορες λειτουργίες. Οι πεζοί και οι οδηγοί χρειάζεται να επικοινωνούν μεταξύ τους σε πραγματικό χρόνο, και στην επικοινωνία αυτή μεσολαβεί ο server: ο κάθε χρήστης στέλνει όλα τα μηνύματα του στον server, ο server με τη σειρά του τα επεξεργάζεται αλλάζοντας αναλόγως την εσωτερική του κατάσταση, και έπειτα προωθεί περαιτέρω μηνύματα στους απαραίτητους χρήστες.

Για την εφαρμογή του server χρησιμοποιήσαμε το Node.js. Η εφαρμογή του server αποτελείται από δύο υπηρεσίες:

- την κύρια υπηρεσία API που χειρίζεται όλη την εσωτερική λογική και με την οποία οι χρήστες επικοινωνούν
- την υπηρεσία που χειρίζεται τα αρχεία πολυμέσων, και συγκεκριμένα τις εικόνες που ανεβάζουν οι χρήστες

Ο διαχωρισμός αυτός έγινε καθώς διαμερίζοντας τον server σε ειδικευμένα μέρη μπορούμε να πετύχουμε καλύτερη επίδοση. Αποθηκεύοντας ξεχωριστά όλα τα αρχεία πολυμέσων (αρχεία σχετικά μεγάλου μεγέθους) και μεταφέροντας την αρμοδιότητα διαχείρισής τους σε άλλο server, ελαφραίνουμε τον φόρτο από την κύρια υπηρεσία μας και πετυχαίνουμε μικρότερο latency. Ακόμα και στην περίπτωση όπου οι δύο υπηρεσίες βρίσκονται στον ίδιο server, έχουμε καλύτερη απόδοση καθώς εκτελώντας δύο διεργασίες Node.js αποφεύγεται το blocking του event loop· οι δύο διεργασίες είναι ανεξάρτητες, επομένως μπορούν να εκτελούνται σε διαφορετικούς πυρήνες CPU, με την κάθε διεργασία να έχει το δικό της event loop.

Για τον web server χρησιμοποιήσαμε το Nginx, κυρίως ως reverse proxy. Το Nginx βρίσκεται ”μπροστά” από το Node.js και δέχεται πρώτο όλα τα requests που φτάνουν στον server. Μέσω ενός reverse proxy, ανακατευθύνουμε τα requests στη θύρα της υπηρεσίας API ή της υπηρεσίας πολυμέσων ανάλογα με το URL, καταφέρνοντας με αυτόν τον τρόπο να έχουμε το ίδιο domain και για τους δύο server. Ο server Nginx είναι επίσης υπεύθυνος για την αποστολή των στατικών αρχείων στον χρήστη (στην προκειμένη περίπτωση αρχεία εικόνων), μία λειτουργία στην οποία υπερέχει του Node.js ως προς την ταχύτητα. Τέλος, μέσω του Nginx ενσωματώνουμε την κρυπτογράφηση

SSL/TLS για την ασφαλή επικοινωνία των χρηστών με τον server μέσω HTTPS, με πιστοποιητικό που λαμβάνουμε από τη γνωστή αρχή πιστοποίησης *Let's Encrypt*.

Τα δεδομένα των χρηστών πρέπει να συγχρονίζονται μεταξύ των συσκευών στις οποίες συνδέονται, επομένως είναι απαραίτητη η αποθήκευσή τους στον server. Για αυτόν τον σκοπό χρησιμοποιούμε μία σχεσιακή βάση δεδομένων MySQL. Η βάση αποτελείται από δύο tables, ένα για τις πληροφορίες των χρηστών και ένα για τα οχήματα. Συγκεκριμένα οι πληροφορίες που συγκρατούνται στη βάση για τον κάθε χρήστη αποτελούνται από τον αριθμό μητρώου του (ο οποίος λειτουργεί ως το στοιχείο ταυτοποίησης), τη βαθμολογία του, και προαιρετικά μία φωτογραφία. Όσον αφορά τα οχήματα, αποθηκεύεται ο ΑΜ του χρήστη στον οποίο αυτό ανήκει, το μοντέλο, και ο αριθμός πινακίδας του, ενώ προαιρετικά μπορεί να συμπεριληφθεί το χρώμα και μία φωτογραφία. Έχοντας αποθηκεύσει τα δεδομένα στη βάση, κάθε φορά που ένας χρήστης συνδέεται στην εφαρμογή τα υπάρχοντα στοιχεία του στέλνονται από τον server στη συσκευή του, ενώ ό,τι νέα στοιχεία προκύψουν στέλνονται στον server για την ανανέωση της βάσης.

Η επικοινωνία client-server επιτυγχάνεται μέσα από την τεχνολογία WebSocket. Ο χρήστης της εφαρμογής, αφού ταυτοποιηθεί ως φοιτητής από την υπηρεσία της σχολής, ανοίγει σύνδεση WebSocket με τον server. Η σύνδεση αυτή διαρκεί για όλη τη διάρκεια της συνεδρίας, και τερματίζεται μόλις ο χρήστης αποσυνδεθεί. Το πρωτόκολλο WebSocket χρησιμοποιήθηκε καθώς αξιολογήθηκε ως το καταλληλότερο για τις απαιτήσεις της εφαρμογής ως προς τη real-time επικοινωνία. Τα WebSockets υπερέχουν τόσο της κλασσικής τεχνολογίας του long polling, καθώς παρέχουν άμεση και αμφίδρομη επικοινωνία με μικρές καθυστερήσεις, αλλά και της τεχνολογίας WebRTC, καθώς προσφέρουν εγγυήσεις για την επιτυχή αποστολή των πακέτων χωρίς σφάλματα.

Όπως αναφέραμε προηγουμένως, το backend αποτελείται από δύο ανεξάρτητα μέρη, έναν κλασσικό HTTP server για τα αρχεία πολυμέσων, και έναν WebSocket server για το API. Ο media server ανταποκρίνεται απλά σε GET/POST/DELETE requests με τον ευνόητο τρόπο, επιστρέφοντας/ανεβάζοντας/διαγράφοντας αρχεία εικόνων αντιστοίχως. Το μόνο στοιχείο που αξίζει να σημειωθεί είναι η ύπαρξη ελέγχων για την προστασία έναντι κακόβουλων ενεργειών, συγκεκριμένα η ύπαρξη ανώτατου ορίου των 2MB για τα αρχεία που ανεβάζονται[1], η απαγόρευση αρχείων που δεν έχουν τύπο εικόνας PNG/JPEG, και φυσικά η απαγόρευση ανεβάσματος αρχείων από χρήστες που δεν είναι πιστοποιημένοι. Η λειτουργία του API server είναι πιο ενδιαφέρουσα. Για τη σύνδεση μέσω WebSocket απαιτείται πρώτα η αποστολή ενός HTTP upgrade request από τον client προς τον server, ακολουθούμενη από την αποδοχή της αίτησης από τον server (opening handshake). Σε αυτή την εναρκτήρια αίτηση, ο χρήστης στέλνει με τη μορφή JWT (JSON Web Token) τα στοιχεία της ταυτότητάς του όπως αυτά παραχωρήθηκαν από τον Identity Provider της σχολής, και ο server ελέγχει την εγκυρότητά τους. Αν για οποιονδήποτε λόγο ο server κρίνει την ταυτότητα του χρήστη ως μη έγκυρη[2], τότε η σύνδεση τερματίζεται αμέσως. Αν η ταυτότητα του χρήστη είναι έγκυρη, τότε η σύνδεση επιτυγχάνεται και ο χρήστης έχει πλέον τη δυνατότητα να επικοινωνήσει

[1] Στην πραγματικότητα το όριο των 2MB επιβάλλεται από το Nginx για όλα τα αιτήματα που φτάνουν στον server

[2] Η ταυτότητα ενός χρήστη κρίνεται ως μη έγκυρη αν το JWT έχει λήξει, αν δεν περιέχει όλα τα ταυτοποιητικά στοιχεία που είναι απαραίτητα, αν η υπογραφή του δεν αντιστοιχεί στον Identity Provider της σχολής, ή αν το JWT απουσιάζει εντελώς

με το API του server μέσω της εφαρμογής. Κατά την επιτυχή σύνδεση του χρήστη, ο server αρχικά αναζητά τον χρήστη στη βάση δεδομένων. Αν ο χρήστης συνδέεται στην εφαρμογή για πρώτη φορά και επομένως δεν υπάρχει ο αντίστοιχος λογαριασμός στη βάση, τότε δημιουργείται ένας νέος άδειος λογαριασμός· διαφορετικά ο server διαβάζει τα αποθηκευμένα στοιχεία από τη βάση και τα στέλνει πίσω στον χρήστη. Από το σημείο αυτό και πέρα, ο server αναμένει την αποστολή μηνυμάτων από την εφαρμογή, εκτελώντας τις ανάλογες λειτουργίες κατά τη λήψη τους, και στέλνοντας ενδεχομένως άλλα μηνύματα πίσω στον χρήστη. Το κάθε μήνυμα που ανταλλάσσεται μεταξύ εφαρμογής και server είναι ένα αντικείμενο JSON το οποίο περιέχει δύο πεδία: τον τύπο και τα δεδομένα. Ο τύπος του κάθε μηνύματος υποδεικνύει την ενέργεια που επιθυμεί να λάβει ο χρήστης, και ανήκει σε μία προκαθορισμένη λίστα που γνωρίζει ο server και η εφαρμογή. Στα δεδομένα του μηνύματος συμπεριλαμβάνονται ό,τι άλλες πληροφορίες είναι απαραίτητες για τον πλήρη προσδιορισμό της ενέργειας που πρέπει να εκτελεστεί. Παρακάτω περιγράφουμε όλους τους τύπους των μηνυμάτων μαζί με τη λειτουργία τους.

Νέος πεζός/οδηγός: Στέλνεται από τον χρήστη μόλις αυτός συνδέεται στην εφαρμογή ως πεζός ή οδηγός. Μαζί με το μήνυμα στέλνονται και οι συντεταγμένες του χρήστη, καθώς και το όχημα του αν πρόκειται για οδηγό. Ο server προσθέτει τον χρήστη στη λίστα των πεζών ή στη λίστα των οδηγών αντίστοιχα, και ειδοποιεί τον χρήστη για την επιτυχή προσθήκη του στην υπηρεσία.

Παύση πεζού/οδηγού: Στέλνεται από τον χρήστη όταν επιθυμεί να διακόψει την υπηρεσία. Ο πεζός ή οδηγός αφαιρείται από την αντίστοιχη λίστα. Σε περίπτωση που ο χρήστης βρίσκεται εν μέσω διαδρομής τότε αυτή ακυρώνεται, και ειδοποιούνται οι χρήστες που πρέπει· αν πρόκειται για παύση οδηγού τότε ενημερώνονται όλοι οι επιβάτες του, ενώ αν πρόκειται για παύση επιβάτη τότε ενημερώνεται ο οδηγός του.

Ενημέρωση πεζού/οδηγού: Στέλνεται κάθε φορά που ένας ενεργός χρήστης μετακινείται. Ταυτόχρονα στέλνεται το στίγμα του χρήστη. Ο server ενημερώνει τις συντεταγμένες που έχει αποθηκεύσει για τον χρήστη, και ειδοποιεί τους άλλους συμμετέχοντες χρήστες για τη νέα θέση του.

Ερώτηση για ύπαρξη πεζών: Στέλνεται από έναν νέο οδηγό όταν αυτός επιθυμεί να μάθει αν υπάρχουν διαθέσιμοι πεζοί που αναμένουν στη στάση. Ο server απαντάει καταφατικά αν υπάρχουν όντως πεζοί.

Αναζήτηση επιβατών: Στέλνεται από έναν οδηγό όταν αυτός δηλώνει την επιθυμία του να δεχτεί επιβάτες για μεταφορά. Ο server επιλέγει έναν αριθμό από πεζούς που βρίσκονται στη στάση και τους στέλνει αίτημα για την αποδοχή του οδηγού.

Αποδοχή οδηγού: Στέλνεται από έναν πεζό ο οποίος αποδέχεται τον οδηγό του. Ο server επιβεβαιώνει την ανάθεση του οδηγού στον πεζό, και ενημερώνει τον οδηγό για τον νέο επιβάτη του.

Εγκατάλειψή πεζού: Στέλνεται από έναν πεζό στον οποίο έχει ανατεθεί οδηγός, αλλά του οποίου ο οδηγός προσπέρασε τη στάση χωρίς να τον παραλάβει. Ο server

αφαιρεί τον πεζό από τους επιβάτες του οδηγού και αρχίζει αμέσως να αναζητεί νέο οδηγό για εκείνον.

Άφιξη στον προορισμό: Στέλνεται από τον οδηγό όταν αυτός φτάσει στη σχολή με τους επιβάτες του. Ο server τερματίζει τη διαδρομή αφαιρώντας τον οδηγό και τους επιβάτες από τις αντίστοιχες λίστες.

Αποστολή αξιολόγησης: Στέλνεται από χρήστη που επιθυμεί να βαθμολογήσει την εμπειρία του με έναν οδηγό ή πεζό. Τα δεδομένα περιλαμβάνουν το ΑΜ του υπό αξιολόγηση χρήστη μαζί με την τιμή της βαθμολογίας. Ο server ελέγχει την εγκυρότητα των στοιχείων[3] και ενημερώνει την τιμή της βαθμολογίας του αντίστοιχου χρήστη στη βάση δεδομένων.

Προσθήκη/ενημέρωση/αφαίρεση οχήματος: Στέλνεται από χρήστη που επιθυμεί να αλλάξει ένα από τα διαθέσιμα οχήματά του. Τα δεδομένα αποτελούν όλα τα στοιχεία του εν λόγω οχήματος, τα οποία ο server ελέγχει πριν προσθέσει το όχημα στη βάση δεδομένων.

Προσθήκη/ενημέρωση/αφαίρεση φωτογραφίας: Στέλνεται από χρήστη που επιθυμεί να αλλάξει τη φωτογραφία στο προφίλ του. Ο server κατεβάζει τη φωτογραφία του χρήστη στο δίσκο και αφού ελέγξει την καταλληλότητά της μέσω ενός φίλτρου/neural network, τη θέτει ως τη φωτογραφία του χρήστη στη βάση δεδομένων.

Αποσύνδεση: Στέλνεται από τον χρήστη όταν αυτός κλείνει την εφαρμογή ή αποσυνδέεται από τον λογαριασμό του. Ο server διακόπτει την τρέχουσα διαδρομή του χρήστη και τερματίζει τη σύνδεση του.

[3]Συγκεκριμένα ελέγχεται αν η τιμή της βαθμολογίας κυμαίνεται μεταξύ του 1 και του 5, και το αν έχει ως στόχο τον οδηγό/επιβάτη με τον οποίο ο χρήστης μόλις ολοκλήρωσε τη διαδρομή του.

Παράρτημα Α΄

Συμπληρωματικό υλικό

Εντοπισμός ακατάλληλου περιεχομένου μέσω μηχανικής όρασης

Όπως αναφέραμε, ένα μέρος της εφαρμογής αφορά στο ανέβασμα εικόνων από τους χρήστες για εμφάνιση στο δημόσιο προφίλ τους. Ήταν σημαντικό να προσφέρουμε τη δυνατότητα αυτή στους χρήστες, καθώς θεωρήσαμε ότι οι φωτογραφίες καθιστούν την εφαρμογή πιο εύχρηστη, επιτρέποντας στους χρήστες να αναγνωρίσουν εύκολα ο ένας τον άλλον, ενώ ενισχύουν ταυτόχρονα την κοινωνική διάσταση της εφαρμογής. Ωστόσο, αυτή η επιλογή που δίνεται στους χρήστες ενδέχεται να παρουσιάσει κινδύνους. Ο κάθε χρήστης μπορεί να ανεβάσει φωτογραφίες ακατάλληλου περιεχομένου, και οι φωτογραφίες αυτές μπορεί να εμφανιστούν στις συσκευές άλλων χρηστών, γεγονός που πρέπει σε κάθε περίπτωση να αποφευχθεί. Η επώνυμη φύση της εφαρμογής που επιβάλλεται από την απαίτηση ταυτοποίησης μειώνει σημαντικά την πιθανότητα τέτοιων περιστατικών, όμως πρέπει να είμαστε σε θέση να εντοπίσουμε άμεσα αυτό το υλικό αν αυτό εμφανιστεί.

Για τον σκοπό αυτό χρησιμοποιήσαμε ένα μοντέλο μηχανικής μάθησης. Αφού δοκιμάσαμε αρκετά μοντέλα, αποφασίσαμε να επιλέξουμε για την περίπτωσή μας το OpenNSFW2[10], το οποίο αποτελεί σύγχρονη υλοποίηση του διάσημου μοντέλου OpenNSFW[7] της Yahoo. Το μοντέλο είναι τύπου Residual Network, και βασίζεται στο ResNet50 της Microsoft, με υλοποίηση στη βιβλιοθήκη Tensorflow. Επιλέξαμε το συγκεκριμένο μοντέλο καθώς έπειτα από δοκιμές συμπεράναμε ότι προσφέρει τη μεγαλύτερη ακρίβεια με μικρό κόστος τόσο όσον αφορά το μέγεθος όσο και τον υπολογιστικό φόρτο. Για τη λειτουργία του μοντέλου δημιουργήσαμε μέσω του framework Flask έναν απλό τοπικό server Python[1] ο οποίος λειτουργεί παράλληλα με τον κύριο server. Ο server αυτός φορτώνει το μοντέλο μηχανικής όρασης και αναμένει αιτήματα HTTP για την αξιολόγηση του περιεχομένου ενός δεδομένου αρχείου. Μόλις λάβει ένα αίτημα, ο server φορτώνει το αρχείο εικόνας στο μοντέλο, και λαμβάνει ως αποτέλεσμα

[1] Αρχικά επιχειρήσαμε να χρησιμοποιήσουμε το Node.js και για αυτόν τον σκοπό ώστε να έχουμε ομοιομορφία όσον αφορά τη γλώσσα, όμως κάποιες ασυνέπειες στην υλοποίηση των διεπαφών του Tensorflow έκαναν αδύνατη τη φόρτωση του μοντέλου στη JavaScript

την πιθανότητα εμφάνισης ακατάλληλου περιεχομένου, την οποία στέλνει ως απόκριση πίσω στον κύριο server. Είναι σημαντικό να σημειωθεί ότι εφόσον οι δύο servers ανήκουν σε διαφορετικές διεργασίες η λειτουργία τους είναι παράλληλη, και καθώς η επικοινωνία επιτυγχάνεται μέσω ασύγχρονων non-blocking HTTP calls ο κύριος server σε καμία περίπτωση δεν παύει να επεξεργάζεται αιτήματα περιμένοντας την απόκριση του server μηχανικής όρασης. Έτσι μπορούμε να έχουμε γρήγορη αναγνώριση εικόνων και επομένως άμεση εντόπιση πιθανών παραβάσεων, χωρίς όμως αυτό να είναι εις βάρος της ταχύτητας του server και επομένως της εμπειρίας του χρήστη.

Authentication

Για την ταυτοποίηση των χρηστών είναι απαραίτητη η επικοινωνία της εφαρμογής με το σύστημα ταυτοποίησης του Πολυτεχνείου, το οποίο βασίζεται στο πρωτόκολλο SAML[2]. Αντί να επικοινωνούμε απευθείας με τον Identity Provider της σχολής, εκτελούμε την ταυτοποίηση μέσω ενός server Keycloak[3] ο οποίος είναι ρυθμισμένος έτσι ώστε να λειτουργεί ως Identity Broker εκ μέρους του. Χρησιμοποιώντας το Keycloak η ταυτοποίηση μπορεί να γίνει μέσω του πρωτοκόλλου OpenID Connect το οποίο είναι πιο εύκολα υλοποιήσιμο από το SAML σε μοντέρνες εφαρμογές.

Find out who to credit for Keycloak server

OpenID Connect

Το OpenID Connect (OIDC) είναι ένα πρωτόκολλο authentication που βασίζεται πάνω στο πρωτόκολλο authorization OAuth 2.0.[4]

Write about how OAuth/OIDC works

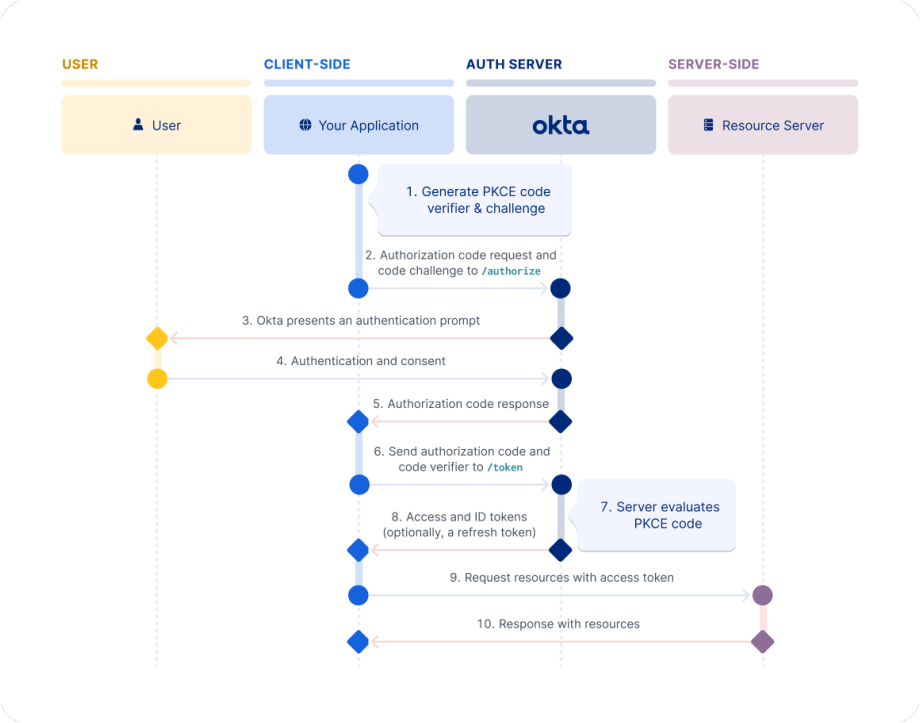
Θα αναλύσουμε αρχικά τη διαδικασία ταυτοποίησης που γίνεται για την πρόσβαση των στοιχείων του χρήστη από την ιστοσελίδα της εφαρμογής μέσω του browser. Η ροή ταυτοποίησης ξεκινάει όταν ο χρήστης ζητάει πρόσβαση στον πόρο που απαιτεί δικαιώματα πρόσβασης, δηλαδή στη σελίδα του προφίλ του. Ο web server δέχεται το αίτημα και ελέγχει αν ο συγκεκριμένος χρήστης είναι ήδη συνδεδεμένος. Αν όχι, τότε ξεκινάει το λεγόμενο authorization code flow για την ταυτοποίηση του χρήστη. Ο server ανακατευθύνει τον browser του χρήστη στην ιστοσελίδα του Keycloak με τις κατάλληλες παραμέτρους που προσδιορίζουν το αίτημα. Αφού ο χρήστης εισάγει τα στοιχεία του, το Keycloak ανακατευθύνει με τη σειρά του τον browser πίσω στην ιστοσελίδα του server, περιλαμβάνοντας στις παραμέτρους του συνδέσμου έναν κώδικα που δίνει πρόσβαση στα στοιχεία του χρήστη. Μόλις ο server λάβει την αίτηση αυτή από τον χρήστη, παρουσιάζει τον κώδικα στο Keycloak, το οποίο τελικά του προσφέρει τα στοιχεία του χρήστη με τη μορφή JWT. Σε αυτό το σημείο ο server επαληθεύει την εγκυρότητα του token, και αφού εξάγει τις πληροφορίες που χρειάζεται (στην περίπτωσή μας το ΑΜ του χρήστη) τις αποθηκεύει στο session του χρήστη. Από το σημείο αυτό και πέρα, κάθε αίτημα του χρήστη προς τον server συνοδεύεται από ένα session cookie το οποίο ο server μπορεί να χρησιμοποιήσει για να αντιστοιχίσει το αίτημα του χρήστη με το κατάλληλο session, και επομένως με την ταυτότητα του.

Explain sessions

[2]Το SAML (Security Assertion Markup Language) είναι ένα σύστημα σχεδιασμένο για την ασφαλή ανταλλαγή ισχυρισμών (assertions) που αφορούν την ταυτότητα και τις εξουσιοδοτήσεις ενός χρήστη

[3]Ο server Keycloak δημιουργήθηκε από τους φοιτητές της σχολής Άγγελο Κολαΐτη και

[4]Το authorization αφορά την



Σχήμα Α΄.1: Το authorization code flow σε περισσότερη λεπτομέρεια. Στην περίπτωσή μας ο *client* είναι ο server, και ο *auth server* είναι το Keycloak. Η διαδικασία σταματάει στο βήμα 8 καθώς μας ενδιαφέρει μόνο το ID Token το οποίο ταυτοποιεί τον χρήστη, και δε χρειαζόμαστε το access token για πρόσβαση σε εξωτερικό API/*resource server*.[8]

Το authentication του χρήστη στην εφαρμογή κινητού λειτουργεί με παρόμοιο τρόπο, με τη διαφορά ότι ο *client* είναι πια εφαρμογή κινητού και όχι browser, γεγονός που σημαίνει ότι απαιτείται ειδική μέριμνα για την προστασία των προσωπικών στοιχείων του χρήστη. Για τον σκοπό αυτό συστήνεται η χρήση του AppAuth, μίας βιβλιοθήκης που δημιουργήθηκε από την OpenID Foundation για authentication εντός κινητών εφαρμογών. Το βασικό σημείο που χρήζει προσοχής είναι η επικινδυνότητα της χρήσης *embedded user-agent* και συγκεκριμένα των in-app webview στη διαδικασία ταυτοποίησης. Τα webviews χρησιμοποιούνται εκτεταμένα σε κινητές εφαρμογές καθώς επιτρέπουν τη φόρτωση ιστοσελίδων εντός της εφαρμογής, παρέχοντας αυξημένο έλεγχο στην εμφάνιση και στη λειτουργία του ”browser”. Αυτές όμως οι δυνατότητες που διαθέτει η εφαρμογή υπέρ των webviews τα καθιστούν πλέον ακατάλληλα για authentication, καθώς η εφαρμογή έχει πλήρη πρόσβαση στα στοιχεία σύνδεσης που ο χρήστης εισάγει στη φόρμα ταυτοποίησης.[2, §8.12] Για αυτόν τον λόγο το AppAuth χρησιμοποιεί εξ ολοκλήρου τον αξιόπιστο default system browser του χρήστη[5] για το

[5]Safari στις συσκευές iOS, και Google Chrome στις συσκευές Android

authentication flow, επιστρέφοντας στο τέλος της διαδικασίας μόνο το authorization code στην εφαρμογή. Έτσι προστατευόμαστε από πιθανούς κινδύνους ασφαλείας που εγκυμονούν τα webviews, και διατηρείται η εμπιστοσύνη του χρήστη στην ασφάλεια της εφαρμογής. Τέλος, ο εξωτερικός browser διατηρεί όλους τους αποθηκευμένους κωδικούς και τα sessions του χρήστη, διευκολύνοντας τη σύνδεση του στην εφαρμογή και επομένως βελτιώνοντας την εμπειρία του.

Βιβλιογραφία

[1] Kristofer Baxter. *Making Netflix.com Faster*. https://netflixtechblog.com/making-netflix-com-faster-f95d15f2e972.

[2] William Denniss και John Bradley. *OAuth 2.0 for Native Apps*. RFC 8252. Οκτ. 2017. DOI: 10.17487/RFC8252. URL: https://www.rfc-editor.org/info/rfc8252.

[3] Octav Druta. *Building With Node.js At LinkedIn*. https://medium.com/building-with-x/building-with-node-js-at-linkedin-ae4ea6af12f2.

[4] Daniel Ehrenberg. *Bloomberg Invests in Node.js. Shouldn't You?* https://www.nearform.com/insights/bloomberg-invests-in-node-js-shouldnt-you/.

[5] OpenJS Foundation. *How Uber Uses Node.js to Scale Their Business*. https://web.archive.org/web/20170225065423/https://nodejs.org/static/documents/casestudies/Nodejs-at-Uber.pdf.

[6] OpenJS Foundation. *Node.js Helps NASA Keep Astronauts Safe and Data Accessible*. https://web.archive.org/web/20231127154528/https://openjsf.org/wp-content/uploads/sites/84/2020/02/Case_Study-Node.js-NASA.pdf.

[7] Jay Mahadeokar και Gerry Pesavento. *Open Sourcing a Deep Learning Solution for Detecting NSFW Images — yahoo! Engineering*. https://yahooeng.tumblr.com/post/151148689421/open-sourcing-a-deep-learning-solution-for. Σεπτ. 2016.

[8] Okta. *OAuth 2.0 and OpenID Connect overview*. https://developer.okta.com/docs/concepts/oauth-openid/#authorization-code-flow-with-pkce-flow.

[9] PayPal Tech Blog Team. *Node.js at PayPal*. https://medium.com/paypal-tech/node-js-at-paypal-4e2d1d08ce4f.

[10] Bosco Yung. *Open-NSFW 2*. https://github.com/bhky/opennsfw2. Έκδ. 0.13.7. Δεκ. 2023.