UNIVERSITY OF PATRAS

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Division of Telecommunications & Information Technology
Laboratory of Wire Communications and Information Technology

«Development of a graphical user interface for the configuration of 5G base stations»

«Ανάπτυξη γραφικού περιβάλλοντος χρήστη για την διάρθρωση σταθμών βάσης 5G»

D I P L O M A T H E S I S

SPYRIDON PEPPAS

SUPERVISOR: SPYRIDON DENAZIS

PATRAS, JULY 2022

University of Patras, Department of Electrical and Computer Engineering.

Spyridon Peppas

CERTIFICATION

It is certified that the Diploma Thesis titled

«Development of a graphical user interface for the configuration of 5G base stations»

«Ανάπτυξη γραφικού περιβάλλοντος χρήστη για την διάρθρωση σταθμών βάσης 5G»

of the Department of Electrical and Computer Engineering student

SPYRIDON PEPPAS

Registration Number: 1046927

was presented publicly at the Department of Electrical and Computer Engineering at

07/07/2022

and was examined by the following examining committee:

Spyridon Denazis, Professor, ECE (supervisor)

Michael Logothetis, Professor, ECE (committee member)

Ioannis Tomkos, Professor, ECE (committee member)

The Supervisor	The Director of the Division
Spyridon Denazis	Kyriakos Sgarbas
Professor	Associate Professor

Acknowledgements

The thesis was written during a time period confronted by changes on a personal level, due to the fact that it coincided with the starting point of my professional career as a software engineer and having to face the reality of that decision. In that regard, I would like to personally thank Professor Spyros Denazis for trusting me when taking over the supervision of my thesis as well as for his much appreciated patience when it came to delays. I would also like to thank Panagiotis Papaioannou from the Network Architectures and Management group for his patience and above all for the seamless cooperation throughout the development and writeup of my thesis. Last but not least, I would like to thank my family for tirelessly being my main support system throughout the years, my friends that I met throughout my academic journey and Lidia for putting up with me and being my catalyst for positive changes through this last year.

Abstract

Mobile networks have greatly evolved since their introduction in 1979. Today's fifth generation technology standard for broadband cellular networks aims to connect even more devices with each other, reduce transmission latency and improve overall quality of service. 5G deployments reach beyond the traditional public networks into private network deployments with the goal to serve the industrial, automotive, and energy sectors. Often, vertical solutions and tools for private 5G networks are offered by big companies for commercial use. This thesis focused on the development of an open-source client application, with the purpose to contribute, along with other open-source components, in resource management of the private 5G networks deployed by the Network Architectures and Management group of the Wire Communications and Information Technology Laboratory. In that regard, this thesis contains the analysis of the use cases provided to a user, through the use of the client application, as well as explaining the stages of planning and development of the application.

Περίληψη

Τα δίκτυα κινητής τηλεφωνίας έχουν εξελιχθεί σημαντικά από την πρώτη κατασκευή τους το 1979. Τα σημερινά πέμπτης γενιάς δίκτυα κινητής τηλεφωνίας στοχεύουν στη σύνδεση ακόμη περισσότερων συσκευών μεταξύ τους, στη μείωση της καθυστέρησης μετάδοσης και στη βελτίωση της συνολικής ποιότητας υπηρεσίας. Οι εγκαταστάσεις 5G ξεπερνούν τα παραδοσιακά δημόσια δίκτυα και πλέον συναντώνται σε εγκαταστάσεις ιδιωτικών δικτύων με σκοπό να εξυπηρετήσουν τους κλάδους της βιομηχανίας, της αυτοκινητοβιομηχανίας και της ενέργειας. Συχνά, κάθετες λύσεις και εργαλεία για ιδιωτικά δίκτυα 5G προσφέρονται από μεγάλες εταιρείες για εμπορική χρήση. Η παρούσα διπλωματική εστίασε στην ανάπτυξη μίας εφαρμογής πελάτη ανοιχτού κώδικα, με σκοπό να συμβάλει, μαζί με άλλες δομές ανοιχτού κώδικα, στη διαχείριση πόρων των ιδιωτικών δικτύων 5G που χρησιμοποιούνται από την ομάδα αρχιτεκτονικής και διαχείρισης δικτύων του εργαστηρίου ενσύρματων τηλεπικοινωνιών και τεχνολογίας της πληροφορίας. Έτσι, η παρούσα εργασία περιέχει την ανάλυση των περιπτώσεων χρήσης που παρέχονται σε έναν χρήστη, μέσω της εφαρμογής πελάτη, καθώς και εξηγεί τα στάδια σχεδιασμού και ανάπτυξης της εφαρμογής.

Table of Contents

List of tables and figures **9**

1. Introduction **15**
1.1 Research Objective and Structure 15

2. Theoretical framework **17**
2.1 5G Technology 17
2.1.1 5G Core 17
2.1.2 5G RAN (Radio Access Network) 17
2.1.3 Non-standalone and Standalone 5G Architecture 19
2.2 Software Development 20
2.2.1 Front-end development 21
2.2.2 Back-end development 21
2.2.3 API 23

3. State of the art **26**
3.1 Resource management 26
3.1.1 RM in 5G RAN 26
3.1.2 RM in private 5G networks 26
3.2 The system 27
3.3 The idea 30

4. Analysis **31**
4.1 Use cases 31
4.1.1 Authentication-Authorization 32
4.1.2 The Resource 33
4.1.3 Monitor the resources 37
4.1.4 Update a resource 41
4.1.5 Delete a resource 42
4.1.6 Create a resource 42
4.2 Issues 45
4.3 Responsive design 45

5. Planning and development **47**
5.1 Tools and languages used 47
5.2 Solutions adapted 48
5.3 Client Application domain 49
5.3.1 Model 49
5.3.2 View 51
5.3.3 Services 53
5.4 System Architecture 55
5.5 Communication 56
5.5.1 Client – Server (Users data) 56
5.5.2 Client – Server (Resources data) 57
5.5.3 CORS 59
5.6 Deployment 60

6. Conclusion **63**
6.1 Future improvements and development 63

References **64**

List of tables and figures

Figure 2.1: 5G system with 5G Core Service Based Architecture (SBA) + IMS core [6] 18

Figure 2.2: Radio-access network interfaces..... 19

Figure 2.3: LEFT: Representation of 5G NSA architecture deployment RIGHT: Representation of 5G SA architecture deployment..... 20

Figure 2.4: REST API model [20] 24

Figure 3.1 Parts of the RAN, the gNodeB deployed on the edge node, running on a standard PC [19] 27

Figure 4.1: Application use case diagram..... 31

Figure 4.2: Views of the: (A) Login page (B) Sign up page (C) sign up confirmation page..... 32

Figure 4.3: Admin only view of the user management dashboard in the client..... 32

Figure 4.4: Admin authorization level only (A) User management (B), (C), (D) Delete buttons (E) Create a resource page..... 33

Figure 4.5: Fields of the model of the resource based on the TMF639 specification [18]..... 35

Figure 4.6: Fields of the model of the characteristic type field..... 36

Figure 4.7: Views of the Home page. (A) Grid view (B) List view..... 38

Figure 4.8: View of the Map page..... 39

Figure 4.9: Views of the Single Resource Page. (A) resource's Resource Status is 'available' (B) resource's Resource Status has any other possible value..... 40

Figure 4.10: Update a resource's (A) category, description, resource version, (B) action parameters, (C) action, (D) action and action parameters..... 41

Figure 4.11: Create a resource page in the client application..... 44

Figure 4.12: Views of the UI. On the left side from a laptop's viewport. On the right side from a smartphone's viewport..... 46

Figure 5.1: Model of the User object..... 49

Figure 5.2: Class diagram: Angular models..... 50

Figure 5.3: Class diagram: Angular views..... 51

Figure 5.4: Class diagram: Angular services..... 53

Figure 5.5: System architecture..... 55

Figure 5.6: Sequence diagram: authentication..... 56

Figure 5.7: Sequence diagram: example of a request fetching Resources..... 57
Figure 5.8: JSON example of a Resource state..... 59

Figure 5.9: Environment variables used in the applications..... 60

ΕΚΤΕΤΑΜΕΝΗ ΕΛΛΗΝΙΚΗ ΠΕΡΙΛΗΨΗ

«Ανάπτυξη γραφικού περιβάλλοντος χρήστη για την διάρθρωση σταθμών βάσης 5G»

ΟΝΟΜΑΤΕΠΩΝΥΜΟ ΦΟΙΤΗΤΗ:
ΠΕΠΠΑΣ ΣΠΥΡΙΔΩΝ

ΟΝΟΜΑΤΕΠΩΝΥΜΟ ΕΠΙΒΛΕΠΟΝΤΟΣ:
ΔΕΝΑΖΗΣ ΣΠΥΡΙΔΩΝ

Τα τελευταία χρόνια ο κόσμος έχει γνωρίσει ραγδαία ανάπτυξη στην αγορά των κινητών συσκευών και την ψηφιακή επανάσταση που έχει αλλάξει τις ζωές των ανθρώπων και τις καθημερινές τους συνήθειες. Την ίδια στιγμή, όσο αριθμός των συσκευών ανά άνθρωπο αυξάνεται, η ανάγκη για μεγαλύτερο όγκο δεδομένων και ταχύτητες μεταφοράς αυξάνονται σημαντικά. Με σκοπό να εξυπηρετήσουν αυτές τις ανάγκες, τα κινητά δίκτυα έχουν εξελιχθεί σημαντικά από το 1979, όταν το 1ης γενιάς δίκτυο γεννήθηκε, μέχρι σημερινά δίκτυα 5ης γενιάς. Τα 5G δίκτυα, εκτός από αυξημένη ταχύτητα και μικρότερο χρόνο καθυστέρησης, εισάγουν στην αγορά νέες τεχνολογίες όπως το edge computing, το cloud computing και, ανάμεσα σε άλλα, συνεισφέρουν στο διαδίκτυο των πραγμάτων (IoT), συνδέοντας δισεκατομμύρια συσκευές με ελάχιστη καθυστέρηση και μειωμένη κατανάλωση ενέργειας. Ένα 5G δίκτυο το καταφέρνει αυτό κάνοντας χρήση σημαντικά μεγαλύτερου αριθμού σταθμών βάσεων (Base station, BS) εγκατεστημένων σημαντικά πιο κοντά το ένα στο άλλο σε σχέση με το 4G δίκτυο. Οι BS είναι πόροι (resource) του 5G δικτύου, συγκεκριμένα είναι μέρος του δικτύου ραδιοπρόσβασης (Radio Access Network, RAN) και διαχειρίζονται την μαζική κίνηση του δικτύου και τις υπολογιστικές απαιτήσεις του. Έτσι είναι απαραίτητη η αποτελεσματική διαχείριση των πόρων (resource management, RM) αυτών στο δίκτυο μέσω της καλύτερης χρήσης του φάσματος, της εξοικονόμηση κατανάλωσης ενέργειας και της ελαχιστοποίησης καθυστέρησης με σκοπό την βελτίωση της ποιότητας εξυπηρέτησης (QoS).

Ο σκοπός σε αυτή τη διπλωματική ήταν η δημιουργία ενός γραφικού περιβάλλοντος χρήστη (GUI) που παρέχει πρόσβαση στις λειτουργίες μίας υπάρχουσας RM εφαρμογής εξυπηρετητή (RM application server), η οποία έχει αναπτυχθεί από την ομάδα αρχιτεκτονικής και διαχείρισης δικτύων του εργαστηρίου ενσύρματων τηλεπικοινωνιών και τεχνολογίας της πληροφορίας. Η RM server εφαρμογή αποτελεί ένα εργαλείο για την διαχείριση των σταθμών βάσεων, gNodeB, σε ιδιωτικά δίκτυα 5G που χρησιμοποιεί η ομάδα . Έτσι, το GUI δίνει τη δυνατότητα σε ένα χρήστη να διαχειριστεί εύκολα και γρήγορα τα BS των 5G δικτύων. Τα gNodeB έχουν μοντελοποιηθεί, κατά την ανάπτυξη του RM server, με βάση τη προδιαγραφή ανοιχτού κώδικα TMF639 του TM Forum, που προορίζεται για διαχείριση πόρων (resource inventory management).

Παρακάτω θα γίνει αναφορά στην δομή της διπλωματικής με σύντομες και περιεκτικές περιγραφές σχετικά με το περιεχόμενο του κάθε κεφαλαίου, καθώς και παραπομπές στην αντίστοιχη πληροφορία στο κυρίως κείμενο που είναι γραμμένο στα αγγλικά.

Κεφάλαιο 2

Στο κεφάλαιο 2 υπάρχει η εισαγωγή στο θεωρητικό πλαίσιο που σχετίζεται με την διπλωματική. Αρχικά, σχετικά με τα δίκτυα 5ης γενιάς γίνεται αναφορά στα κύρια στοιχεία από τα οποία αποτελείται, τα οποία είναι το δίκτυο ραδιοπρόσβασης (RAN), το δίκτυο πυρήνα (5G core network) και ο εξοπλισμός χρήστη (UE). Σχετικά με το RAN, περιγράφεται το gNodeB, το οποίο αποτέλεσε την περίπτωση χρήσης του GUI, που αναπτύχθηκε σε αυτή τη διπλωματική, ως πόρος διαχείρισης για τις κάθετες λύσεις ιδιωτικών δικτύων 5ης γενιάς. Επιπλέον περιγράφονται οι δύο αρχιτεκτονικές εφαρμογής των 5G δικτύων, οι οποίες είναι η αυτοδύναμη αρχιτεκτονική (standalone, SA) και η μη-αυτοδύναμη αρχιτεκτονική (non-standalone, NSA). Η NSA είναι το μοντέλο ανάπτυξης κατά το οποίο συνδυάζονται μέρη ενός δικτύου 4G κατά την ανάπτυξη ενός δικτύου 5G με την πιο συνηθισμένη τακτική να είναι να συνδυάζεται το δίκτυο πυρήνα του 4G δικτύου, δηλαδή το Evolved Packet Core (EPC), με το δίκτυο πρόσβασης (Access Network) του 5G δικτύου. Από την άλλη, η SA αρχιτεκτονική αξιοποιεί πλήρως το 5G και στο δίκτυο πυρήνα, προσφέροντας όλες τις δυνατότητες της νέας τεχνολογίας. Στη συνέχεια περιγράφονται οι όροι και οι τεχνολογίες που χρησιμοποιήθηκαν για την ανάπτυξη του GUI στο **section 2.2.1**, στο **section 2.2.2** οι τεχνολογίες για την ανάπτυξη μίας εφαρμογής εξυπηρετητή και στο **section 2.2.3** περιγράφεται τι είναι η διεπαφή προγραμματισμού εφαρμογών (API).

Κεφάλαιο 3

Στο κεφάλαιο 3, αρχικά γίνεται αναφορά στην υπάρχουσα τεχνολογία σχετικά με την διαχείριση πόρων στα δίκτυα 5ης γενιάς. Συγκεκριμένα αναφέρονται οι υποψήφιες λύσεις που στοχεύουν στην βελτίωση της απόδοσης του φάσματος και της ενεργειακής απόδοσης στα δημόσια τηλεπικοινωνιακά δίκτυα. Τέτοιες λύσεις συναντώνται στο RAN καθώς και στο core network των δικτύων. Σημαντικότερο, όμως στα πλαίσια της διπλωματικής, είναι οι RM λύσεις για τα ιδιωτικά 5G δίκτυα. Σε αυτή την περίπτωση προσφέρονται εμπορικές κάθετες λύσεις από μεγάλες εταιρίες όπως η Qualcomm και η Cisco που συμπεριλαμβάνουν τα δικά τους ιδιωτικά εργαλεία για RM. Στη συγκεκριμένη διπλωματική στόχος είναι η ανάπτυξη ενός τέτοιου εργαλείου αλλά ανοιχτού κώδικα.

Στη συνέχεια γίνεται μία σύντομη περιγραφή για το κάθε μέρος του συστήματος που αφορά την ανάπτυξη της εφαρμογής. Συγκεκριμένα:

- το **gNodeB** αποτελεί το BS του 5G δικτύου και εμφανίζεται σε μορφή λογισμικού, από την εταιρία Amarisoft, και μπορεί να τρέχει σε ένα standard PC.
- **RM server** αποτελεί την εφαρμογή εξυπηρετητή, η οποία είναι υπεύθυνη για την επικοινωνία με τον πράκτορα (agent) του κάθε gNodeB καθώς και να αποθηκεύει την κατάσταση του καθενός.
- **Client application** είναι η GUI εφαρμογή που αναπτύχθηκε στα πλαίσια αυτής της διπλωματικής και θα αναλυθεί παρακάτω.

- **Authentication application server** είναι μία επιπλέον εφαρμογή εξυπηρετητή που αναπτύχθηκε στα πλαίσια της διπλωματικής για να καλύψει τις ανάγκες της αυθεντικοποίησης και της εξουσιοδότησης που χρειάστηκαν για την επικοινωνία μεταξύ της **client application** και του **RM server**.
- **TMF639 REST OpenAPI specification** αποτελεί την προδιαγραφή για την διεπαφή επικοινωνίας μεταξύ των εφαρμογών του συστήματος και βασίζεται στην μοντελοποίηση της κατάστασης (state) του gNodeB με βάση το ανοιχτό πρότυπο για resource inventory management.

Τέλος, στο **section 3.3** διατυπώνεται η ιδέα πίσω από την ανάπτυξη της παρούσας διπλωματικής, η οποία είναι η κάλυψη της ανάγκης ύπαρξης ενός γραφικού περιβάλλοντος για την διεπαφή για τον RM server, που έχει αναπτυχθεί από την ομάδα αρχιτεκτονικής και διαχείρισης δικτύων. Επιπλέον γίνεται αναφορά στην πλατφόρμα ανάπτυξης της εφαρμογής που είναι το web, καθώς τίθεται εύκολη η πρόσβαση από οποιαδήποτε σύγχρονη συσκευή μέσω ενός περιηγητή (browser).

Κεφάλαιο 4

Στο κεφάλαιο 4 ξεκινάει η ανάλυση της εφαρμογής που αναπτύχθηκε στα πλαίσια της διπλωματικής.

Αρχικά στο **section 4.1**, αναλύονται οι περιπτώσεις χρήσεις της εφαρμογής από την πλευρά ενός χρήστη παρουσιάζοντας κάθε φορά τις σελίδες και τα στοιχεία που έχουν αναπτυχθεί στο GUI. Συγκεκριμένα, ξεκινάει με τις σελίδες που συναντάει κάποιος την πρώτη φορά που φτάνει στη SPA (single page application), οι οποίες είναι α) login page, β) sign up page και γ) sign up confirmation page (***Figure 4.2***). Αυτές οι σελίδες υπάρχουν για να ξεχωρίζουν τα δικαιώματα που έχει ο κάθε χρήστης/επισκέπτης στην σελίδα, αφού για το authentication και ουσιαστικά την είσοδο στο κυρίως μέρος της εφαρμογής, πρέπει πρώτα ο χρήστης να συνδεθεί για να επαληθεύσει τα στοιχεία του. Επιπλέον αναλύονται και τα δύο authorization επίπεδα που υπάρχουν στην εφαρμογή και αφορούν α) το διαχειριστικό επίπεδο (admin level) και β) το επίπεδο πρόσβασης (access level). Έπειτα η ανάλυση περνάει στο κεντρικό κομμάτι της εφαρμογής που είναι το μοντέλο του gNodeB ως πόρος, που ορίζεται από το πρότυπο TMF639 (***Figure 3.8***). Αφού δοθεί η επεξήγηση των πεδίων του μοντέλου, περιγράφονται αναλυτικά, σε συνδυασμό με απεικονίσεις από το UI, οι δυνατότητες που έχει ένας χρήστης στην εφαρμογή. Συγκεκριμένα:

- Έλεγχος της κατάστασης των πόρων (**section 4.1.3**)
- Ενημέρωση της κατάστασης ενός πόρου (**section 4.1.4**)
- Αφαίρεση ενός πόρου (**section 4.1.5**)
- Δημιουργία ενός πόρου (**section 4.1.6**)

Στη συνέχεια γίνεται αναφορά στα προβλήματα που αντιμετωπίστηκαν κατά την ανάλυση (**section 4.2**) και τέλος παρουσιάζεται ο λόγος που το responsive design

υπήρξε σημαντικός παράγοντας κατά την ανάπτυξη της εφαρμογής σε συνδυασμό με τις αντίστοιχες απεικονίσεις από το UI (**section 4.3**).

Κεφάλαιο 5

Στο κεφάλαιο 5 περιγράφονται τα εργαλεία και τα βήματα γύρω από τον σχεδιασμό και την ανάπτυξη της εφαρμογής. Στο **section 5.1** γίνεται αναφορά στα εργαλεία και τις γλώσσες προγραμματισμού που χρησιμοποιήθηκαν, όπου για την SPA χρησιμοποιήθηκε το Angular framework και τον auth server το NodeJS runtime για χρήσης της JavaScript γλώσσας σε εξυπηρετητή. Επίσης παρέχεται η τεκμηρίωση του κώδικα των εφαρμογών που δημιουργήθηκαν με σκοπό την πιθανή συνέχιση ανάπτυξης στο μέλλον.

Στη συνέχεια, γίνεται σύντομη αναφορά σε κάποιες λύσεις που υιοθετήθηκαν κατά τον σχεδιασμό (**section 5.2**) και ξεκινάει η περιγραφή των στοιχείων της εφαρμογής πελάτη (client application) που αναπτύχθηκε με χρήση του angular framework. Έτσι, τα στοιχεία που περιγράφονται είναι:

a. τα μοντέλα του Resource (έχει γίνει αναλυτική περιγραφή στο **section 4.1.2**) και του User (**section 5.3.1**)
b. οι σελίδες (views) με τις δομές (components) τους και τις μεθόδους τους (**section 5.3.2**)
c. οι υπηρεσίες (services) με τις μεθόδους και τους τρόπους επικοινωνίας τους (**section 5.3.3**)

Έπειτα γίνεται η αναφορά στην αρχιτεκτονική του συστήματος (**section 5.4**) και αναλύονται οι τρόποι επικοινωνίας του πελάτη (client) με τον εξυπηρετητή (server) (**section 5.5**), καθώς υπάρχουν δύο διαφορετικές περιπτώσεις. Η μία είναι αυτή όπου ο client ζητάει δεδομένα που αφορούν το μοντέλο του User από τον auth server και η άλλη, όταν ο client ζητάει δεδομένα που αφορούν το μοντέλο του Resource και προέρχονται από τον RM server, όμως το αιτήματα και οι απαντήσεις πρέπει να περάσουν μέσα από τον auth server. Πάνω σε αυτό γίνεται και μία σύντομη αναφορά στο CORS (**section 5.5.3**).

Κλείνοντας το κεφάλαιο 5, παρέχεται ένας οδηγός για την εγκατάσταση και το πως να τρέξει η εφαρμογή σε ένα εξυπηρετητή, τοπικό ή απομακρυσμένο (**section 5.6**).

Κεφάλαιο 6

Στο κεφάλαιο 6, παρέχονται τα συμπεράσματα της ανάπτυξης της εφαρμογής στα πλαίσια της διπλωματικής, καθώς και παρατίθενται προτάσεις για μελλοντικές προσθήκες και βελτιώσεις.

1. Introduction

In recent years, the world has witnessed an exponential growth of the mobile market and a digital revolution that has transformed human's lives and daily habits. At the same time, as the number of devices per person increases, the need for more data and faster transfer rates is exponentially increasing as well. In the course to serve such demands, the mobile networks have evolved drastically since the, not too distant, 1979, when the 1st generation network was deployed, to today's 5th generation networks. Each new generation, apart from increased data transfers, introduced significant advances to the world. 2G paved the way from the analog past to the digital future, while also introducing text (SMS) and multimedia (MMS) messages between devices. 3G standardized the way that users are accessing data from any location in the world with 'data packets', which led the rise of new services such as video conferences and voice over IP. 4G, while it required hardware upgrades to both the network's infrastructure and users' devices, it made possible for the consumers to enjoy high-definition video streaming, high-quality video conferences and cloud gaming services among others. The latest generation, 5G, arrived with changes on all parts of the network. The 5GCore, the 5G RAN and the new 5G-compatible user equipment (UE) are all evolved technologies of the network, bringing along new concepts like edge computing and fog computing that put resources closer to the UE with the aim to reduce transmission latency and obtain better QoS. In addition, 5G is designed to serve the internet of things (IoT), by connecting billions of devices with ultra-low latency and reduction in power consumption. 5G achieves these tasks by deploying many more base stations than 4Gs and in closer proximity with each other [29].

Base stations, called gNodeB in 5G, are resources of the network, that handle the massive traffic and computational demand, therefore making resource management (RM) crucial for 5G. RM aims, through resource allocation, to boost the usage of spectrum resources, as well as saving energy consumption and reducing transmission latency to achieve improved quality of service (QoS). RM concerns both the radio access network (RAN) and the 5G core network (5GC), which are the two main components of the 5G mobile network [30].

1.1 Research Objective and Structure

The research objective for this thesis was to build a graphic user interface (GUI) for accessing the functionalities of an existing resource management (RM) server application. The RM server was developed from the network architecture and management group (NAM) of the Wire Communications and Information Technology Laboratory. The RM server is an application, running on a server, for managing the base stations in private 5G networks that are deployed by the group. Hence, the development of the UI was necessary to provide easy and efficient access to the RM

server and make its operations attainable through any modern device's browser. In the context of the thesis, any acquired knowledge in software engineering during the academic years, was utilized to develop a secure client web application.
This thesis document is structured as follows:

In chapter 2, the theoretical framework, which this thesis is based on, is introduced, for the purpose of better understanding the analysis and the development process of the thesis. In particular, the 5G network is described briefly, along with the technology's main components and different architectures of deployment. In addition, a short description for the software development tools and the interface used in both the front-end and the back-end is given.

In chapter 3, an overview on the current state of resource management in the 5G networks, as utilized in public infrastructures as well as in private networks, is provided. Furthermore, the parts built and used for this thesis are specified and the components of the 5G network that these parts are concerned about, are designated. Lastly, the idea and the goals behind this thesis are stated.

In chapter 4, the use cases, served from the components that were developed in the thesis, are analyzed. The interactions of the user are showcased by providing images from the respective views in the UI and the additional actions available to a super user (admin role), compared to a simple user, are specified.

In chapter 5 the planning and the development are referenced. In particular, the tools and languages used, and the solutions adapted, during the design and development of the applications are stated. Moreover, the models, views, and services adopted in the single page application (SPA) development are explained by providing class diagrams and a more detailed look in the system architecture, that this thesis is concerned with, is taken. In the end, the communication between the client and the server applications are explained and a deployment guide, using docker, is provided.

In chapter 6, a general evaluation of the system developed is provided, along with ideas for future improvements and development.

2. Theoretical framework

2.1 5G Technology

5G, in telecommunications, is the 5th generation technology standard for broadband cellular networks. It is the planned successor of the 4G network, which currently accounts for more than 50% of mobile connections globally [3] and are available for 86% of the world population [4]. According to GSMA data, 5G networks have already amassed a 17% global coverage [5] since the first deployment in South Korea in 2019. The driver behind the next generation mobile networks expansion can be credited to the expanding consumer appetite for bandwidth that is needed for applications like 4K streaming, virtual reality and autonomous driving, among other use cases. In order to cover such demands, for the new technology, various techniques have been developed that accomplish high throughput at high speeds, ultra-low latency and extreme high reliability all without sacrificing one for the other [1].

Deployment methods of 5G networks can be divided into two architectures, the non-standalone and the standalone one, with the latter offering the complete technological advantages. Before comparing these two, a short description of the two components found in these architectures is necessary.

2.1.1 5G Core

The 5G core is the core component at the heart of the new 5G specification and enables the increased throughput demand that 5G must support. The new 5G core, as defined by 3GPP in the 3GPP TS 23.501 [2], utilizes cloud-aligned, service-based architecture (SBA) that spans across all 5G functions and interactions including authentication, security, session management and aggregation of traffic from end devices. The 5G core further emphasizes NFV as an integral design concept with virtualized software functions capable of being deployed using the MEC (Multi-Access Edge Computing) infrastructure that is central to 5G architectural principles. Some of the changes, accompanied by the new technology, are migration to millimeter wave, massive MIMO and network slicing. Unlike 4G's core network, EPC (Evolved Packet Core), 5G Core is leveraging virtualization and cloud native software design at unprecedented levels [7].

2.1.2 5G RAN (Radio Access Network)

The radio access network evolved as well following the 5G specification. It is responsible for connecting a user device with the network through an air interface.

In 5G RAN, the antenna and a set of virtualized networking technologies support the 5G new radio, with advanced capabilities when compared to previous RAN standards. The radio spectrum used by 5G networks covers more and higher radio frequencies than 4G LTE networks and can be divided in the following bands [9]:

- **Low band**: 600-700Mhz, 30-250Mbps bandwidth, hundreds of km² coverage
- **Mid band**: 1.7-4.7Ghz, 100-900Mbps bandwidth, several km in radius coverage
- **High band**: (millimeter waves) 24-39Ghz, 1-3Gbps bandwidth, <1km in radius coverage

To achieve these, 5G requires new types of nodes, called gNodeB, an upgrade from the previous generation eNodeB used by 4G technology and the main subject in the present research [8].

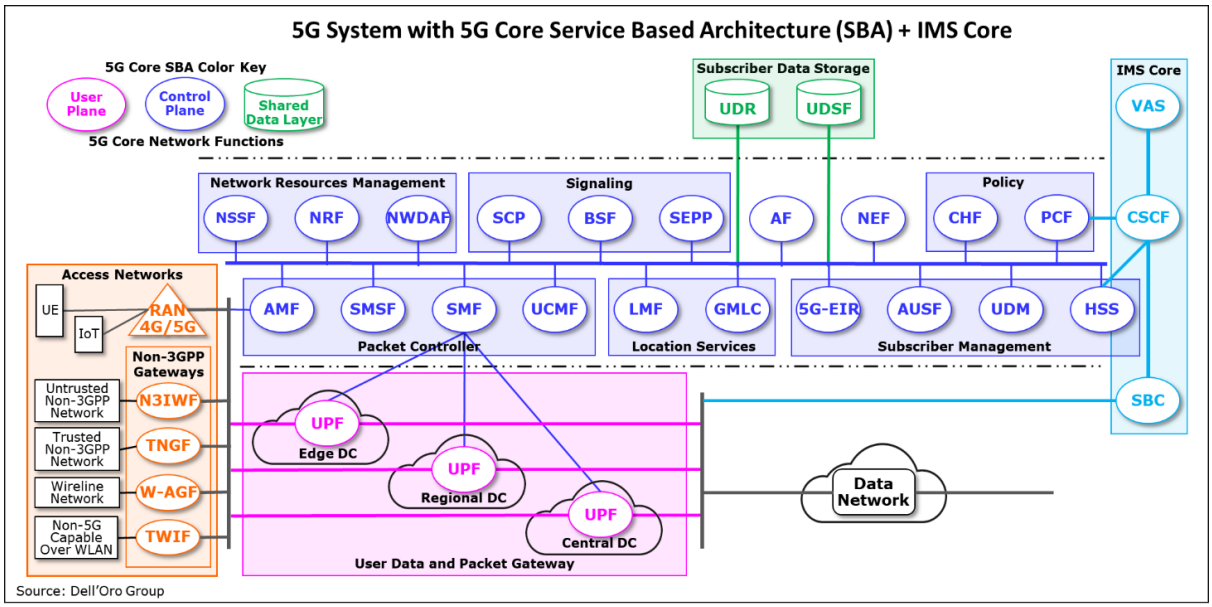

***Figure 2.1**: 5G system with 5G Core Service Based Architecture (SBA) + IMS core [6]*

gNodeB

A gNodeB, or gNB, is a 3GPP 5G Next Generation base station which supports the 5G New Radio. It is responsible for all radio-related functions in one or several cells, for example radio resource management, admission control, connection establishment, routing of user-plane data to the UPF and control-plane information to the AMF, and quality-of-service (QoS) flow management. It is important to note that a gNB is a logical node and not a physical implementation. The gNB nodes are connected to the 5G core network through the NG interface, more specifically to the UPF by means of the NG user-plane part (NG-u) and to the AMF by means of the NG control-plane part (NG-c). One gNB can be connected to multiple UPFs/AMFs for the purpose of load sharing and redundancy.

The Xn interface, connecting gNBs to each other, is mainly used to support dual connectivity and lossless active-state mobility between cells by means of packet forwarding. It may also be used for multi-cell Radio Resource Management (RRM) functions.

There is also a standardized way to split the gNB into two parts, a central unit (gNB-CU) and one or more distributed units (gNB-DU) using the F1 interface. In case of a split gNB, the RRC, PDCP, and SDAP protocols, described in more detail later, reside in the gNB-CU and the remaining protocol entities (RLC, MAC, PHY) in the gNB-DU. The interface between the gNB (or the gNB-DU) and the device is known as the Uu interface. [10]

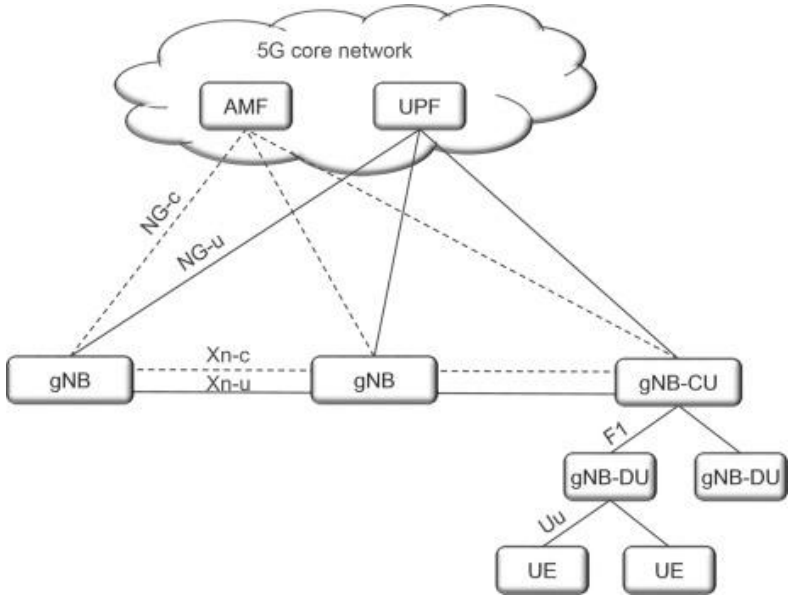

***Figure 2.2**: Radio-access network interfaces*

2.1.3 Non-standalone and Standalone 5G Architecture

Non-standalone (NSA) and standalone (SA) architectures are two deployment options that differ on how they implement their respective core network and the radio access network.

NSA deployment is where 5G service is offered without the end-to-end 5G infrastructure. It is usually based on a pre-existing 4G core network (EPC) with the addition of a gNB base station. This technique has some different options as well, where one is having the gNB communicating directly with the EPC core network and another is having an eNB (4G base station) to act as a master node and a gNB to act as a slave node. Operators are benefiting from an NSA option in the early stages of 5G deployment, by utilizing the existing 4G infrastructure thus reducing the time and the cost required for the network to be deployed and be operational.

NSA 5G is not coming without a cost though. Opting to use the old generation EPC for that type of deployment has its limitations by not allowing the network to deliver its peak performance in high speed, low latency, etc. [1]

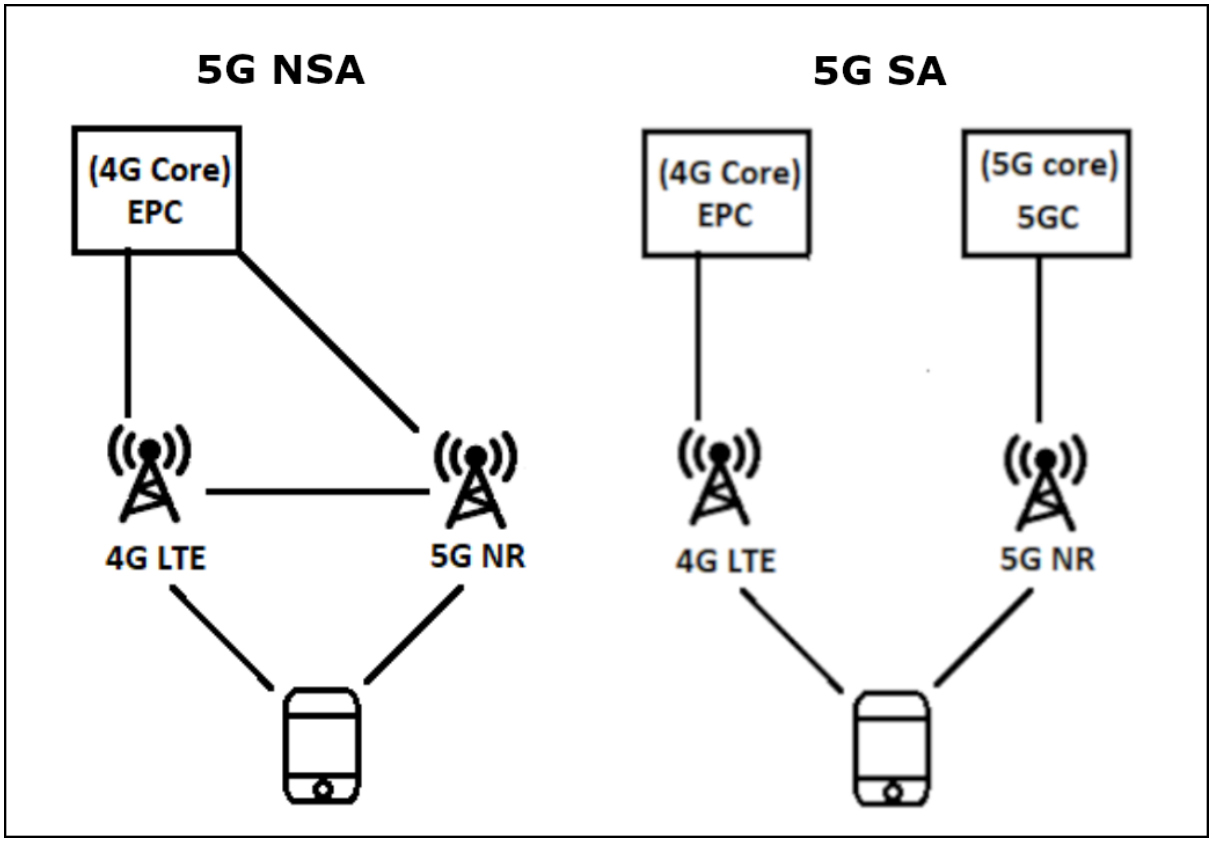

***Figure 2.3**: LEFT: Representation of 5G NSA architecture deployment. RIGHT: Representation of 5G SA architecture deployment.*

SA architecture of 5G takes advantage of both the next generation infrastructures. A gNB of the radio access network is directly connected to the 5G core network, offering all the great benefits of the 5G specification. Based on a totally new, cloud- based, virtualized, microservices-based core infrastructure, some of the anticipated benefits of introducing 5G standalone technologies include faster connection times (lower latency), support for massive numbers of devices, programmable systems enabling faster and more agile creation of services and network slices, with improved support for SLA management within those slices, and the advent of voice-over new radio. Introduction of 5G standalone is expected to facilitate simplification of architectures, improve security and reduce costs. [1]

2.2 Software Development

The software developed as part of this thesis was built for the web. This was chosen as web applications can be easily accessed by any modern device (smartphones, PCs, etc.) through their browsers (Chrome, Mozilla, etc.). A user can simply type in the domain name or IP (internet protocol) address, which the application is served from, on any browser and access it, providing that both the user's device and the server, serving the application, are on the same network (the internet, a local network, etc.).

What follows is a short description of the tools and specifications used in the development of the project.

2.2.1 Front-end development

Front-end web development involves the UI/UX designing and development of the graphical user interface of a website or web application using HTML, CSS, and JavaScript or any front-end framework.

Single Page Application (SPA)

A single page application is a web application that runs on a client's browser and utilizes JavaScript to dynamically rewriting the current web page with new data from the web server, instead of the default method of a web browser loading entire new pages. The goal is faster transitions that make the website feel more like a native app. In a SPA, a page refresh never occurs; instead, all necessary HTML, JavaScript, and CSS code is either retrieved by the browser with a single page load,[1] or the appropriate resources are dynamically loaded and added to the page as necessary, usually in response to user actions [16].

Angular

Angular is a component-based framework for building scalable front-end web applications. It is built on Typescript, a JavaScript superset, and consists of a collection of well-integrated libraries that cover a wide variety of features, including routing, forms management, client-server communication, and more. It extends the HTML vocabulary of applications with special Angular syntax in templates. For example, Angular helps get and set DOM (Document Object Model) values dynamically with features such as built-in template functions, variables, event listening, and data binding [12].

2.2.2 Back-end development

Back-end development refers to the server side of an application. It focuses on databases, scripting, website architecture. Code written for back-end development helps browsers to communicate with database information.

Microservices

Microservices, or also known as microservice architecture, is an architectural style that structures an application as a collection of services that are independently deployable and highly maintainable and testable amongst other benefits. The

microservice architecture enables the rapid, frequent and reliable delivery of large, complex applications.

Node.js

Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser. Node.js has an event-driven architecture capable of asynchronous I/O. These design choices aim to optimize throughput and scalability in web applications with many input/output operations, as well as for real-time Web applications [13].

MySQL

MySQL is an open-source relational database management system (RDBMS). SQL stands for Structured Query Language. It is a domain-specific language used in programming and designed for managing data held in a RDBMS. SQL statements are used to perform tasks such as update data on a database or retrieve data from a database.

JSON

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. JSON is built on two structures:

- A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

These are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable with programming languages also be based on these structures [21].

HTTP request

An HTTP request, according to IBM's documentation [22], is made by a client, to a named host, which is located on a server. The aim of the request is to access a resource on the server.

To make the request, the client uses components of a URL (Uniform Resource Locator), which includes the information needed to access the resource.

A correctly composed HTTP request contains the following elements:

- A request line.
- A series of HTTP headers, or header fields.
- A message body, if needed.

Docker

Docker is a software platform that allows someone to build, test, and deploy applications quickly. Docker packages software into standardized units called containers that have everything the software needs to run including libraries, system tools, code, and runtime. Using Docker, someone can quickly deploy and scale applications into any environment and know their code will run [28].

2.2.3 API

An application programming interface (API), according to Red Hat [14], is a connection between computers or between computer programs. It is a type of software interface, offering a service to other pieces of software. A document or standard that describes how to build or use such a connection or interface is called an API specification. A computer system that meets this standard is said to implement or expose an API. The term API may refer either to the specification or to the implementation.

One main advantage of the APIs is that they enable a product or a service to communicate with other products and services without having to know how they're implemented. In this way an API can simplify app development. Moreover, when new tools and products are designed, or old ones are managed APIs can provide:

- flexibility,
- simplified design, administration and use
- opportunities for innovation.

Cloud-native application development, like in the case of 5G Core, relies on connecting a microservices application architecture through APIs as they are a simplified way to connect a private infrastructure through cloud-native app development, but they also to share data with customers and other external users.

There are three API release policies:

1. **Private**: The API is only for use internally. This gives companies the most control over their API.
2. **Partner**: The API is shared with specific business partners. This can provide additional revenue streams without compromising quality.

3. **Public**: The API is available to everyone. This allows third parties to develop apps that interact with your API and can be a source for innovation.

Regarding on how somebody decides to manage his API, it can help him share his resources while not giving up security and control. The applications that consume information given by the API can be used in every kind of system from legacy systems to distributed architectures like the IoT.

Representational State Transfer (REST) Architecture

REST is a set of architectural constraints, not a protocol or a standard. API developers can implement REST in a variety of ways.

When a client request is made via a RESTful API, it transfers a representation of the state of the resource to the requester or endpoint. This information, or representation, is delivered in one of several formats via HTTP: JSON (JavaScript Object Notation), HTML, XLT, Python, PHP, or plain text. JSON is the most generally popular file format to use because, despite its name, it's language-agnostic, as well as readable by both humans and machines.

Something else to keep in mind: Headers and parameters are also important in the HTTP methods of a RESTful API HTTP request, as they contain important identifier information as to the request's metadata, authorization, uniform resource identifier (URI), caching, cookies, and more. There are request headers and response headers, each with their own HTTP connection information and status codes.

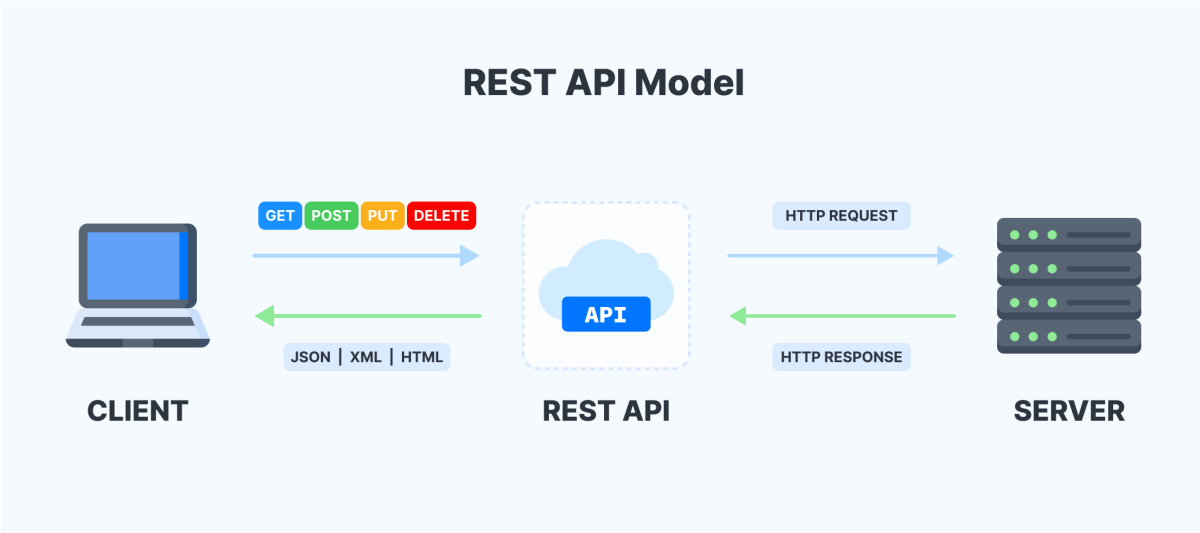

***Figure 2.4**: REST API model [20]*

In order an API to be considered RESTful, it has to conform to these criteria:

- A client-server architecture made up of clients, servers, and resources, with requests managed through HTTP.
- Stateless client-server communication, meaning no client information is stored between get requests and each request is separate and unconnected.

- Cacheable data that streamlines client-server interactions.
- A uniform interface between components so that information is transferred in a standard form. This requires that:

    - resources requested are identifiable and separate from the representations sent to the client.
    - resources can be manipulated by the client via the representation they receive because the representation contains enough information to do so.
    - self-descriptive messages returned to the client have enough information to describe how the client should process it.
    - hypertext/hypermedia is available, meaning that after accessing a resource the client should be able to use hyperlinks to find all other currently available actions they can take.

- A layered system that organizes each type of server (those responsible for security, load-balancing, etc.) involved the retrieval of requested information into hierarchies, invisible to the client.
- Code-on-demand (optional): the ability to send executable code from the server to the client when requested, extending client functionality.

Open API

According to its official web page [15], the OpenAPI Specification (OAS) defines a standard, language-agnostic interface to RESTful APIs which allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection. When properly defined, a consumer can understand and interact with the remote service with a minimal amount of implementation logic. An OpenAPI definition can then be used by documentation generation tools to display the API, code generation tools to generate servers and clients in various programming languages, testing tools, and many other use cases.

An OpenAPI document is a document (or set of documents) that defines or describes an API. An OpenAPI definition uses and conforms to the OpenAPI Specification. For the document to conform to the OAS it is itself a JSON object, which may be represented either in JSON or YAML format. An OpenAPI document MAY be made up of a single document or be divided into multiple, connected parts at the discretion of the user. In the latter case, $ref fields MUST be used in the specification to reference those parts as follows from the JSON Schema definitions.
The latest version of the OAS is the 3.0.3 version, released in 2020-02-20. Its first official release was the 1.0 version released at 2011-08-10 under the name Swagger Specification which was later donated to the OpenAPI initiative when it reached the 2.0 version. After that Smartbear, the original creator of Swagger, remained a member of the OpenAPI initiative and has since freely released the Swagger Codegen which is a tool to generate a REST API from an OpenAPI document.

3. State of the art

3.1 Resource management

The growing adoption of the 5G networks alongside the rapid development of IoT (internet of things), connecting large number of devices, and the increasing consumer demand for highest quality of multimedia is putting more and more pressure on the cellular network. Thus, concepts like edge computing, fog computing and network slicing have been suggested and, at some point, introduced to aid the network in reducing latency and improving quality of service (QoS). Resource management (RM) is crucial for 5G given the enormous network traffic and computational resource requirements. Through efficient resource allocation, RM can reduce transmission latency, save energy, and efficiently optimize the utilization of spectrum resources. RM, in 5G, concerns both the RAN and the core network, but the focal point of the thesis is about the radio access network. [30]

3.1.1 RM in 5G RAN

Improving spectrum efficiency (SE) and energy efficiency (EE) are the main goals of RM in the RAN. Regarding spectrum, different solutions have been introduced, each one with its own challenge to overcome. Cognitive radio network (CRN) divides the spectrum into licensed and unlicensed bands to enhance efficiency, the interference for the primary user (PU) and secondary user (SU) cannot be ignored. Cloud radio access network (C-RAN) in combination with device-to-device (D2D) communication may be one of the solutions to increase the SE and reduce the latency but still may struggle with multi-users. Fog radio access network (F-RAN) is another concept that joins the communication infrastructure into fog computing, where computing components are closer to UEs, but struggles on the limited resources available.

Regarding energy efficiency, RM is needed to face this as an emergency issue and find a balance in the tradeoff problem between QoS and EE. As mobile devices increase constantly, energy consumption is becoming an important issue for telecom operators as well, due to increasing costs. Therefore, concepts like the F-RAN, with power allocation algorithms and C-RAN with learning techniques are proposed. [30]

3.1.2 RM in private 5G networks

Concepts like C-RAN, F-RAN and CRN are mostly related to public 5G networks and infrastructures. This thesis, however, is concerned about the concept of private 5G networks, since the existing RM server developed from the NAM group

is intended for use in such vertical infrastructure. There are certain commercial end-to-end solutions offered by big telecommunication companies like Qualcomm and Cisco for 5G private networks, which also include their own private tools for resource management. In the case of this thesis, though, the intention was to build open-source components while utilizing Open APIs to ensure easy access for everyone concerned.

3.2 The system

The 5G mobile network consists of the following components, the 5G core network (5GC), the radio access network (RAN) and the user equipment (UE). The gNodeBs are part of the edge nodes that belong in the RAN component of 5G. Although a general knowledge of the 5G components is helpful, it is not mandatory for the development. The RM server is responsible for the communication with the base station's server. The main concern for the development of the UI is the interface of communication between the client and the RM server.

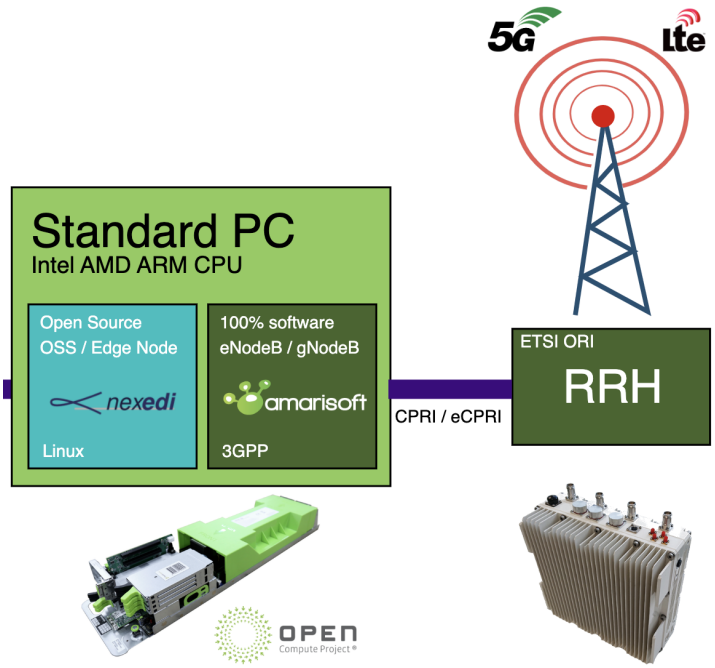

***Figure 3.1**: Parts of the RAN, the gNodeB deployed on the edge node, running on a standard PC [19]*

The components that are part of the development in this thesis are namely, the gNodeB instances, the Open API specification that is used for modeling the state of the gNodeBs, a resource management server application, a single page web application client, a web server application for providing authentication and the interface that these applications use to communicate through the network.

A short description of the parts follows:

The gNodeB instance

The gNodeBs are deployed in software form with the use of Amarisoft's LTEENB server software that runs on a standard PC system hardware configuration. A gNodeB's agent communicates with the resource management server with API requests [11].

The RM application server

The web server application is responsible for storing the data that describes the state of the gNodeBs. It is also responsible for monitoring the instances, as well as communicating any operations with the BS's agent.

The client application

The client application offers a graphic user interface for the user to access the available operations of the RM server. The UI is developed for the web, using the Angular framework. A front-end development framework like Angular helps building a high-quality UI, offers fast development process and guarantees effective long-term support for the app by the developers familiar with this well-known and widely used framework.

The authentication application server

An additional web server application exists between the SPA client and the RM server serving two important purposes. Firstly, it is responsible for authenticating a client and authorizing access from the UI and secondly, it is responsible for reverse-proxying all HTTP requests between the SPA client and the RM server, so that the unprotected RM server stays as hidden as possible from a client.

The interface – TMF639 API REST Specification

The interface used for communication between the client and the RM server follows a REST API architecture. It uses HTTP requests to transfer JSON data between the client and the server. The REST API interface in use, is described by the TMF639 specification of the TM Forum.

The TMF639 specification defines an open-source model for resource inventory management. It is an Open API defined by the TM Forum based on REST to provide a consistent and standardized mechanism to query and manipulate resource inventory. Resource is an abstract entity that describes the common set of attributes shared by all concrete resources in the inventory.

In the present development, the “Resource” model described in the TMF639 specification refers to a gNodeB’s state.

At this stage, it is important to acknowledge that neither the Amarisoft’s software, nor the RM server application were used in the development and testing of the client application. The RM server was replaced by a JSON Server found here: https://github.com/typicode/json-server, that offers a full fake REST API. The gNodeB instances are represented by their state, which are stored in a JSON file, that is accessed by the JSON Server, in an array of “Resource” objects as defined by TMF639 specification. This development environment setup offers identical interactions with the indented production deployment, with the major difference being the lack of feedback from an actual gNodeB instance running on the Amarisoft’s software.

Moving forward, while describing the development of the applications the following parts of the system are referred to accordingly:

a) The **gNodeBs**, as the **resources**

b) the **radio management application server,** as the **RM server**

c) The **client application**, as the **client UI** or the **SPA** (single page application)

d) The **web application server** for authentication, authorization and proxying requests, as the **auth server**

3.3 The idea

The main idea for this thesis is to develop a graphic user interface (GUI) for a accessing the available operations of an existing RM server, which has been developed by the NAM group of the Wired Communications laboratory. The RM server is responsible for managing the gNodeBs, the 5G network's base stations, in vertical deployments of private 5G networks. A gNodeB can be deployed in any standard PC running the appropriate software and the RM server communicates and controls it through the base station's agent.

Theoretically, a UI for this purpose could be developed for any modern platform (iOS, Android, web), but in the present thesis it has been opted to go with a universal approach, that of a single page web application. Therefore, as a web application, it can be accessed by any modern device with a web browser and JavaScript support.

A UI is useful for server applications like the RM server because it enables a user to interact with the application's operations through a human friendly visual environment. If a UI is not available, such service could only be accessed by a command line or other dedicated software, like Postman, by manually fillisng the necessary requests.

The development of the client application was accomplished in coordination and constant communication with the developer of the RM server, in order to define and adjust properly the communication interface between the two applications according to the required needs and use cases.

The development of this application was to benefit a user by giving access through a graphical UI to:

- Monitor the status of the gNodeBs
- Operate on the gNodeBs:
    - Create/deploy new instances
    - Delete/remove existing ones
    - Update them with available commands and data
- Authorize access to selected operations according to the user's role
- Use it through any modern device's browser

Lastly, it is important to note that, due to the use of the open-source TMF639 specification for resource inventory management, the client application, with some adjustments, could be used to manage other resources like, for example, espresso machines or air condition units. But in the context of this thesis, the indented use case, as it is analyzed in the next chapter, are the gNodeBs of the 5G network.

4. Analysis

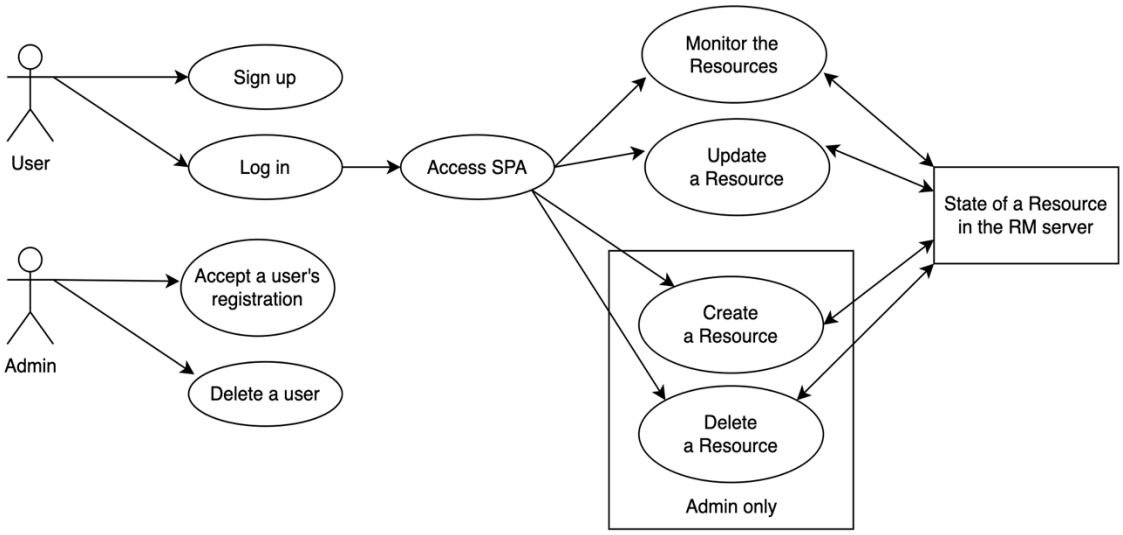

***Figure 4.1**: Application use case diagram*

4.1 Use cases

The use cases of the application are shown in ***Figure 4.1*** above. During the planning stage of the development, it was decided for the application to offer an authentication system, as well as having some authorization level for a user to access certain features.

In addition, a user can interact with the state of the resources through the client UI. Examples of resources' states and the different options that a user has to operate on them are described in this chapter.

Note that instead of the RM server, a JSON server offering a full fake REST API is used, thus the application only interacts with the state of the resources stored in it and there is no real time interaction with deployed gNodeB instances.

The use cases of the application are analyzed below, along with showcasing the respective UI views from the angular application.

Note that most values of data displayed in the following images of the UI, are not real values, and they are used for demonstration purpose only.

4.1.1 Authentication-Authorization

The authentication and authorization strategies are taking place between the angular application and the auth server. A user is authenticated by typing in the SPA login page ***(Figure 4.2, A)*** the email and the password and sending an HTTP request with that data to the auth server. The auth server checks if that combination exists in the user database and respond by creating an authenticated session between the client and the auth server or respond with an error if the credentials do not match.

A user, visiting the application for the first time, has the option to sign up, through the client UI, by entering a name, an email and a password ***(Figure 4.2, B)***. Upon successful registration, the user lands on a confirmation page and is faced with the first type of authorization of the system ***(Figure 4.2, C)***.

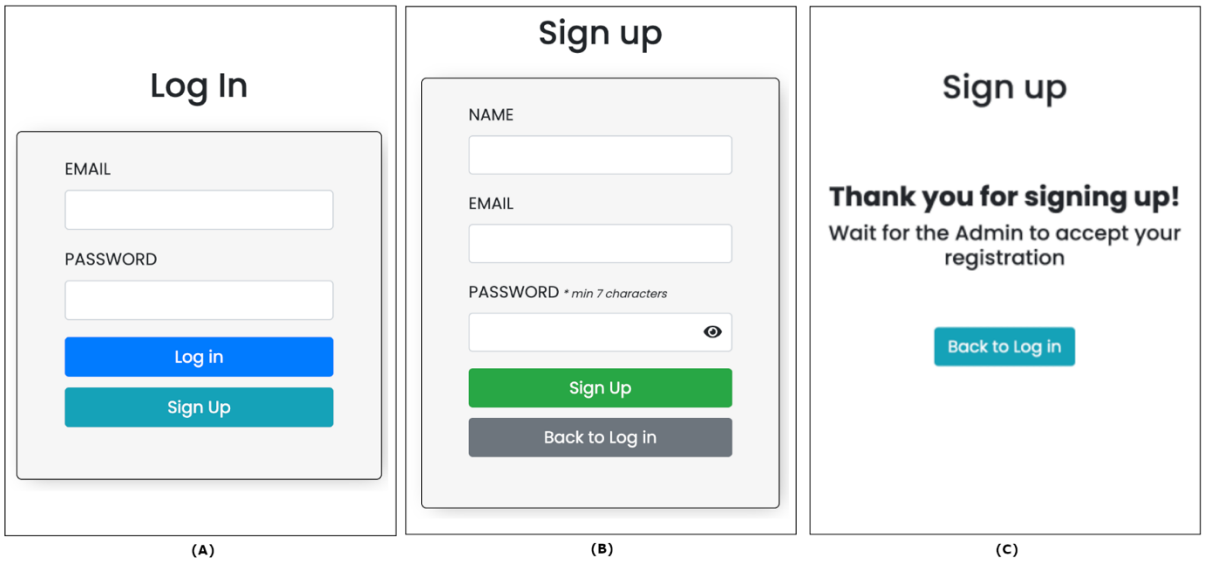

Figure 4.2*: Views of the: (A) Login page (B) Sign up page (C) sign up confirmation page*

The authorization in the system has two levels:

- **Access level**: for accessing the protected views of the application
- **Admin level**: for accessing the admin-level views and function in the SPA

Therefore, for the newly signed up user to access the inside of the application, an admin needs to verify the registration and toggle the user's access status. The options to toggle a user's access, as well as deleting a registered user, are present in the client application, only for an admin, and can be done with the use of the buttons as shown in ***Figure 4.3***.

Figure 4.3*: Admin only view of the user management dashboard in the client*

Admin authorization level

An admin authorization level has been implemented in the system to keep a simple user from accessing certain operations through the same client application.

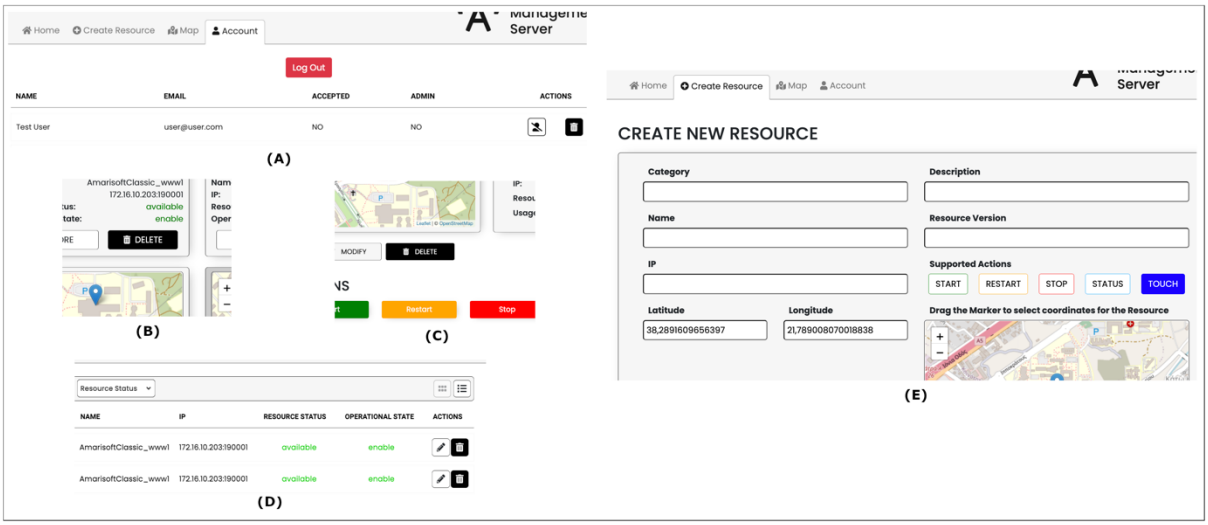

***Figure 4.4**: Admin authorization level only (A) User management (B), (C), (D) Delete buttons (E) Create a resource page*

In the case of the present UI development, an admin has the following additional controls:

- Access in the user management dashboard of the client, ***Figure 4.4 (A)***
- Access to delete a resource from the server, ***Figure 4.4 (B), (C), (D)***
- Access in the create resource page, ***Figure 4.4 (E)***

Note that the option to toggle a user's admin role has not been implemented in the client. This action is only possible by accessing the MySQL database directly with the use of software like phpMyAdmin. This way ensures the admin role in the application is attainable only by someone that can go through the extra security step of knowing the credentials needed to connect to the database.

4.1.2 The Resource

The "Resource" described in the TMF639 API REST specification, as already mentioned, refers to a gNodeB instance. The JSON representation of the resource is stored, in production environment, in the RM server. There is never a direct exchange of requests between the client SPA and the gNodeB software, that is why a JSON server is used in place of the actual RM server code, as it is perfect to offer the same REST API behavior. Therefore, any data shown representing the state of a resource is for demonstration purpose and does not indicate data from a real gNodeB instance.

Note that, moving forward, the name of a field belonging to a resource state is highlighted in bold, while the value of the field is highlighted in bold with single quote notation, as a way to assist with the better understanding of the analysis.

Below, the available options, built in the angular application that a user can use to manage the resources through the UI, are analyzed.

The state of the resource

The state of the resource is the JSON representation that consists of the selected fields from the TMF369 specification and is stored in the RM server.
In the table below, the fields are described as found the specification [18].

KEYS	**VALUES**
Id	A string. Identifier of an instance of the resource. Required to be unique within the resource type. Used in URIs as the identifier for specific instances of a type.
href	A string. The URI for the object itself.
Category	A string. Category of the concrete resource. e.g Gold, Silver for MSISDN concrete resource.
Description	A string. free-text description of the resource.
Name	A string. A string used to give a name to the resource.
Resource version	A string. A field that identifies the specific version of an instance of a resource.
Administrative state	A resource administrative state type (ResourceAdministrativeStateType). Tracks the administrative state of the resource, such as locked, unlocked, shutdown and so on.
Operational state	A resource operational state type (ResourceOperationalStateType). Tracks the operational state of the resource, such as enable, disable and so on.
Activation feature	A list of features (Feature [*]). Applicable configuration features of a resource for activation.

Resource characteristic	A list of characteristics (Characteristic [*]). Describes a given characteristic of an object or entity through a name/value pair.
Resource status	A resource status type (ResourceStatusType). Tracks the resource status of the resource, such as standby, alarm, available, reserved, suspended and so on.
Usage state	A resource usage state type (ResourceUsageStateType). Tracks the usage state of the resource, such as idle, active, busy and so on.

Figure 4.5*: Fields of the model of the resource based on the TMF639 specification [18]*

Note that in the specification there are more fields describing a resource but for the development of our application only the ones mentioned where selected to be used.

In addition, the **resource characteristics** field is used to provide additional custom information, provided with the use of the characteristics type of the specification under that field, where each one is used for a specific purpose in the state of the resource. Thus, the below table do not represent a key-value pair, but the “name”: value and “value”: value pairs.

“name”: value	**“value”: value**
IP	IP stands for internet protocol address, but in the case of this thesis a WebSocket is a more accurate description, since both an IP and port is used to target the unique gNodeB instance on the network
Location	A list of two numbers. Tracks the latitude and longitude specified for the resource.
Supported Actions	A list of strings. Tracks the actions that a distinct resource supports.
Action	A string. Tracks the last action that was called on the resource.
Action parameters	Object with key-value pair the name of the parameter and its value.

Figure 4.6*: Fields of the model of the characteristic type field.*

It is obvious that some of the fields can have any random value that is appropriate to their context, but there are also certain fields that have a pre-determined selection of available values that they can accept. Below these specific fields with their values are analyzed and it is explained how certain fields' values affect the behavior of the client UI.

◊ The **administrative state** field can have one of the following values:

- **'locked'**
- **'unlocked'**
- **'shutdown'**

◊ The **operational state** field can have on the following values:

- **'enable'**
- **'disable'**

When a resource's state has the value **'disable'** in its respective **operational state** field, then the user has no access to its single resource page (***Figure 4.9***), the resource is displayed with grey background in the home page's view as shown in ***Figure 4.7***, it is showing up as a grey pin in the map page's map as shown in ***Figure 4.8*** and there is no update request available for it.

◊ The **resource status** field can have one of the following values:

- **'standby'**
- **'alarm'**
- **'available'**
- **'reserved'**
- **'unknown'**
- **'suspended'**

When a resource's state has any other value except **'available'** in its respective **resource status** field, then the user cannot operate any update requests on the resource, thus, the modify buttons and the set of action buttons are not displayed in the client UI. The difference is visible between the views **(A)** and **(B)** in ***Figure 4.9***.

◊ The **usage state** field can have one of the following values:

- **'idle'**
- **'active'**
- **'busy'**

◊ The **supported actions** field is a list of values and the **action** field a single value from the following options:

- **'start'**
- **'restart'**
- **'stop'**
- **'status'**
- **'touch'**

Although an action seems to potentially cause a certain event to a resource, this behavior was not observed using the present client application due to the reason that that communication with a real gNodeB deployment was not possible.

The above information sums up the structure of a resource, which is used in the present development, and the behavior of the UI that some of the fields' values cause. The exact JSON representation of the resource object is shown in **chapter 5**, where the steps of the development, including parts of the code, are explored.

4.1.3 Monitor the resources

The option to monitor the resources is provided through the client UI by having the angular application to retrieve the requested data from the RM server that stores their states. There are three different pages in the angular application for monitoring the status of the resources.

Home page

The first one is the home page of the UI, where a user has access to a list of the existing resources retrieved from the RM server. There, a few of the fields of the resource's state are displayed for each item.

In the home page, apart from the fields, there are a) a map, displaying the location of the resource according to the given latitude and longitude for each one, and b) a button that leads to a dedicated page with a single resource's details.

The views of the home page of the SPA are shown in ***Figure 4.7***. The client application has two options built in for displaying the resources that a user can toggle between, a grid view ***(Figure 4.7, A)*** and a list view ***(Figure 4.7, B)***. Furthermore, in order to assist the user to search for a certain resource, a few filter options were developed and added to the UI.

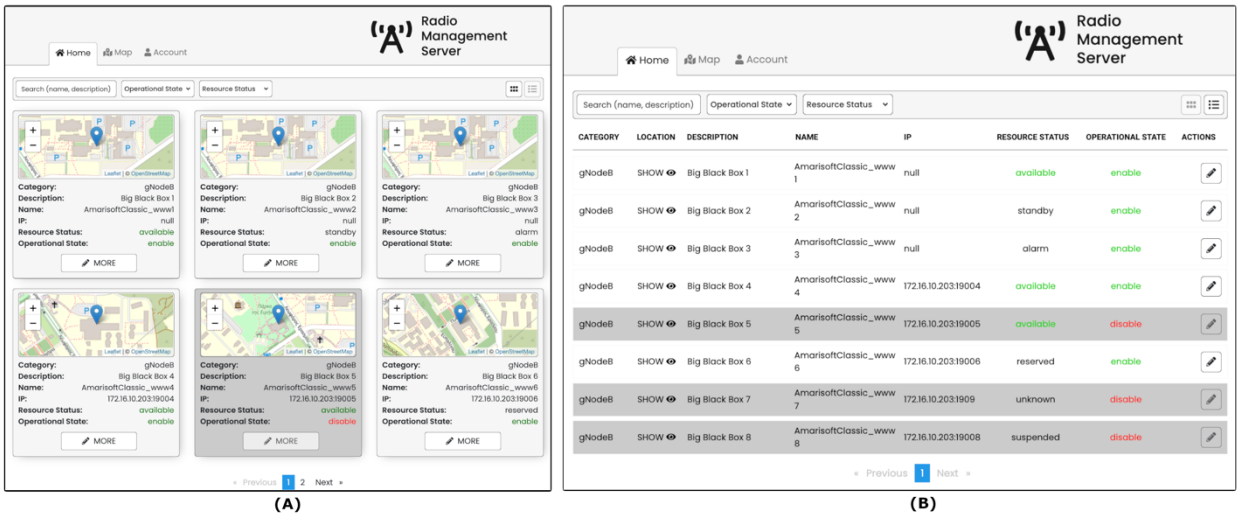

***Figure 4.7**: Views of the Home page. (A) Grid view (B) List view*

These filters include:

- A search by text: matching the input text with the values in the **Name** field or **Description** field of a resource
- A drop-down list: matching the selected option with the value of the **Operational State** field of a resource
- A drop-down list: matching the selected option with the value of the **Resource Status** field of a resource

In addition, due to a resource representing the state of a gNodeB instance that is live on the network and because a gNodeB's state may change frequently, a polling technique has been utilized. Which means that a request to retrieve the latest state of the resources is sent, every 10 seconds, to the RM server, after the initial loading of the page on the client.

Map page

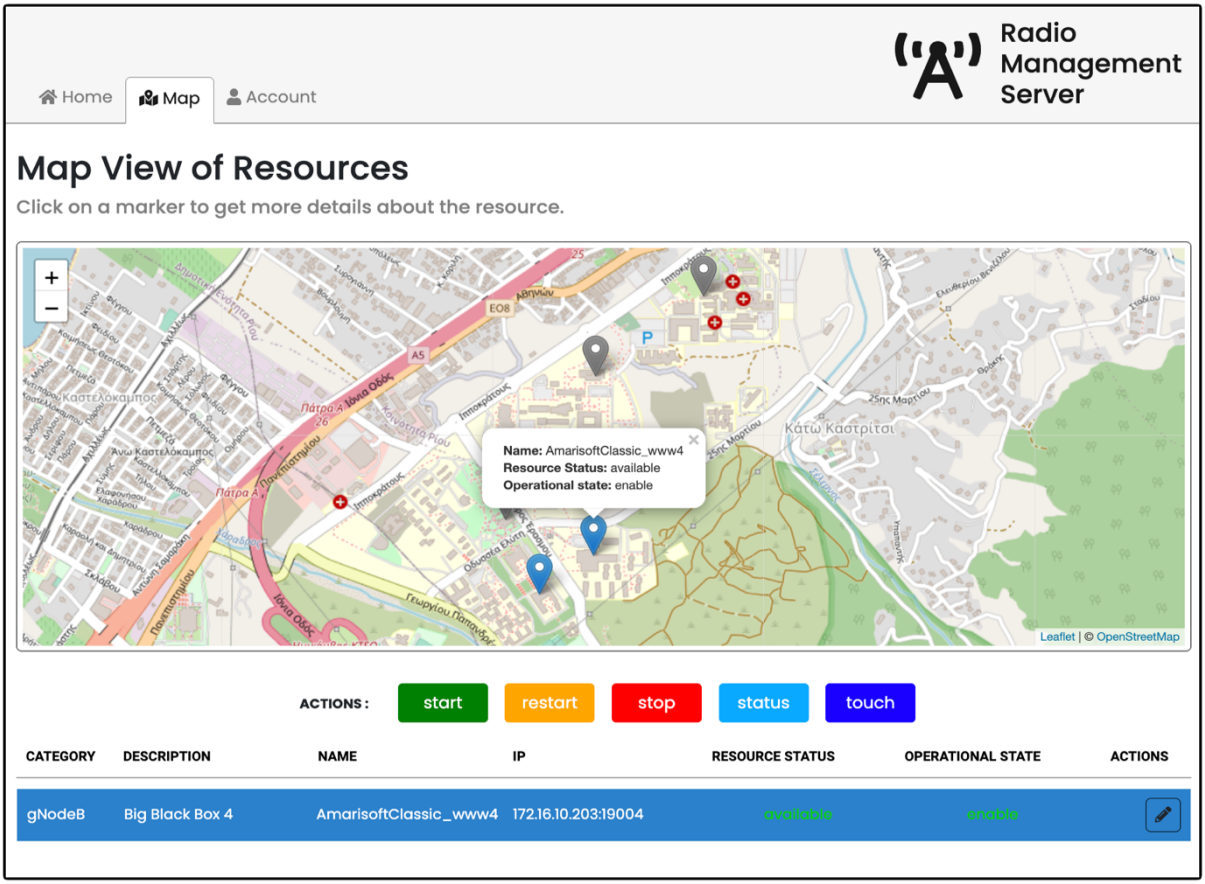

***Figure 4.8**: View of the Map page*

The map page is another interface that a user can monitor the resources from. In this page the main area is covered by the map, in which all the resources are included and displayed as pins. These pins differ in color, where the grey ones represent resources with the value '**disable**' in their field **Operational state**, while the blue ones stand for the rest of the remaining values of the same field.

A user, by interacting with a pin on the map, gets access to some additional information for the resource, as shown in ***Figure 4.8***:

a) a tooltip box, appearing above the selected pin containing information about the **Name**, the **Resource Status** and the **Operational State**
b) a selection (in blue background), appearing below the map, similar to the way an item is displayed in the home screen in list view, from which the user can access the single resource page.
c) a set of buttons, appearing below the map, displaying the **Supported Actions** available for the highlighted resource

When the page loads, the map renders all the retrieved resources and sets the zoom and pan of the map in a way to contain them all in the view.

Single Resource page

The Single Resource page contains all the information stored in the state of a resource, that the SPA retrieves from the RM server.

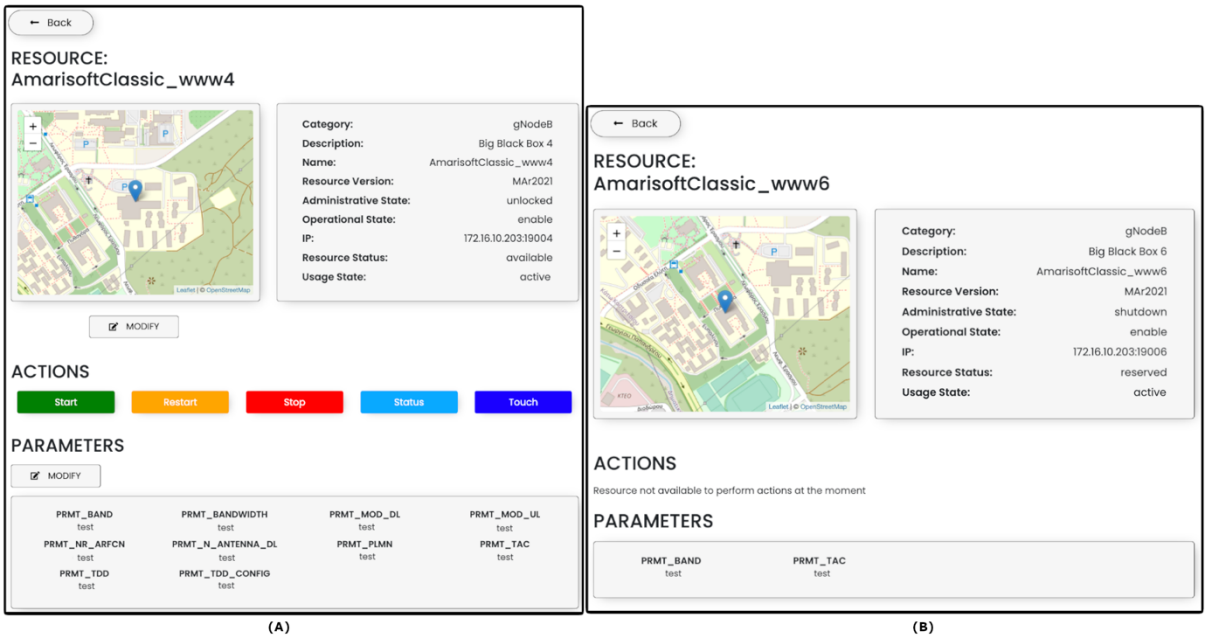

***Figure 4.9**: Views of the Single Resource Page. (A) resource's **Resource Status** is '**available**' (B) resource's **Resource Status** has any other possible value*

The view of the page (***Figure 4.9***) contains four important parts:

a) a map, where the location of the resource is displayed with a blue pin
b) a block of information about the resource
c) a set of action buttons
d) a block displaying the resource's parameters

It is obvious that in this page is where the complete functionality of a resource exists. There, a user has the option to monitor, update or delete[1] the state of the resource by using the available displayed buttons or inputs. More on the update actions are explained on **section 4.1.4** and on **section 4.1.5** for the delete action.

In addition, the two views shown in ***Figure 4.9*** represent two distinct resources that have a different value in a particular field of their state, hence, the reason that the modify and the action buttons are shown in view **(A),** while they are hidden in view **(B)**.

[1] Delete button is visible in **Figure 4.4 (C)**, as the **Figure 4.9** displays a non-admin user's view.

4.1.4 Update a resource

Updating a resource is available in the client application through certain actions, that in the background are sending an update request with specified data to the RM server to change the state of a resource stored there.
The single resource page (***Figure 4.9***) and the map page (***Figure 4.8***) are the views that a user has access to update operations (***Figure 4.10***) for a resource. As mentioned before, if a resource's **resource status** field has any value other than **'available'** or its **operational state** field has the value **'disable'**, then no update action is available through the client UI.

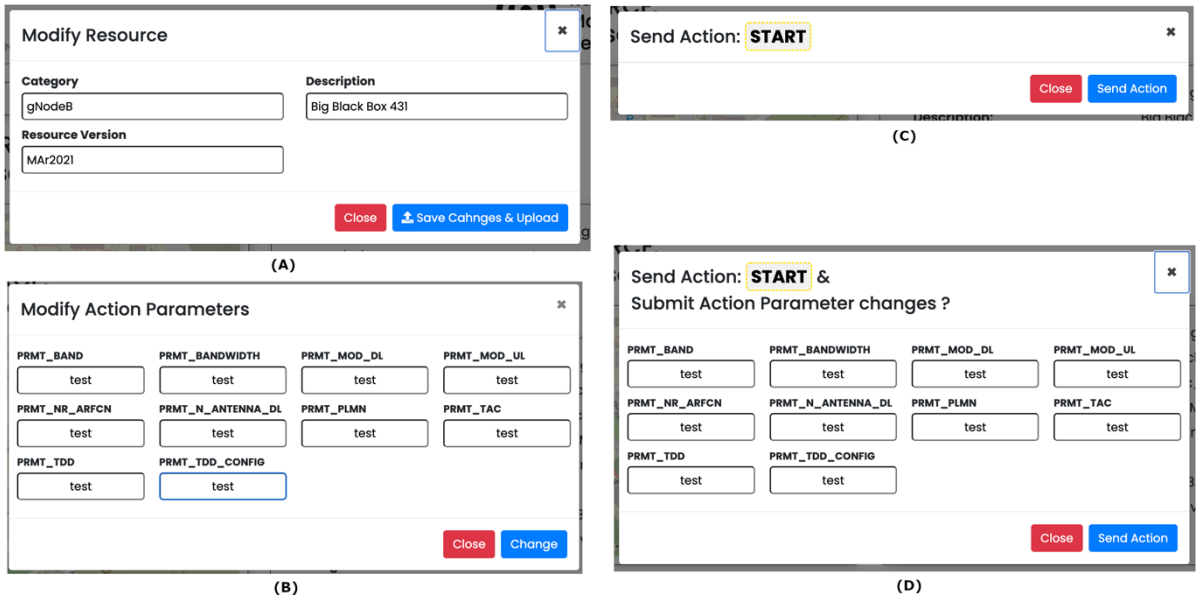

Figure 4.10**: Update a resource's (A) **category**, **description**, **resource version**, (B) **action parameters**, (C) **action**, (D) **action** and **action parameters

These operations are explained here:

◊ Update a resource's information. A user can edit the **category**, **description** and **resource version** fields by using the modify button in the single resource page.

◊ Send a specific **action** to the RM server for a resource to follow. A user can choose one from the set of action buttons that appear either on the single resource page or the map page.

◊ Update a resource's parameters. A user can edit the **action parameters** through the single resource page by using the modify button in that section. For the update request to be sent to the RM server, the user must click on the "Save changes" button, after having changed any of the parameters.

◊ Send an action and update the parameters. Instead of clicking the "Save changes" button, after changing any of the parameters, the user can select any of the

available action buttons and get a modal like the one shown in ***Figure 4.10 (D)***. That way the update request contains both operations at the same time.

4.1.5 Delete a resource

To delete a resource is another option that a user has available in the client UI to manage the resources. As already mentioned in **3.1.1** though, the delete actions are only possible for admin level authorized user accounts.

Deleting a resource in the client application means that a user is selecting the respective action from the UI (***Figure 4.4, B – C – D***) on the selected instance's place. After the user confirms the action, the angular application is sending the request with the appropriate data to the RM server to start the process of removing the instance from its storage.

Since in the present development a JSON server is used in place of the RM server, the action is as simple as the server just removing the element from its data file. In the case of having a connection with the real RM server, this action could be a bit more complicated, as the RM server, before just removing the data, it would need to communicate with the gNodeB instance and instruct it accordingly for the action. This occasion is explained here because in a real setup situation the action from the client UI may take longer to complete or even have unexpected behavior (e.g., an untested error).
The delete button for each resource is present in the following views of the UI:

- In the home page in each list/block item (***Figure 4.4, B – D***)
- In the single resource page (***Figure 4.4, C***)

4.1.6 Create a resource

Creating a resource is the last option that is available in the system for managing the resources. This action can be applied in the client application by accessing the Create Resource page (***Figure 4.11***) using the navigation tabs. As already mentioned in **section 4.1.1** though, the create resource page is only accessible for admin level authorized users.

A user utilizing the UI to create a resource has the following data inputs to fill in before sending the request to complete the action:

- **Category**: a string input.
- **Description**: a string input.
- **Name**: a string input.

Note that the **name** field's value is not available to update as analyzed in the **section 4.1.4**. This was a coordinated requirement during the planning of the development

from the developer of the RM server, as that field needs to be unique for each resource in the database of the RM server with the purpose of being used as the main field (instead of the **id** or the **href**) for calling a GET resource/**name** request for retrieving a single resource.

- **IP**: a string input**.**

The **IP** field input needs to be in a WebSocket (e.g., IP:PORT) form that is associated with the instance of the gNodeB created.

- **Supported actions**: a list of actions.

The user can toggle on (filled background) or off (white background) the buttons with the desired action that will be included in the **supported actions** array of the soon to be created resource.

- **Location** (latitude and longitude): a set of two decimal numbers.

The **location** field of a resource is filled with two decimal numbers that stand for its exact latitude and longitude coordinates on the world map. This set of inputs can be filled using two ways available in the UI. One way is to write or edit the numbers by typing them in their respective boxes and the other way is by dragging the pin on the displayed map at the exact position of the resource, as soon as the pin is dropped in place, those inputs are updated automatically.

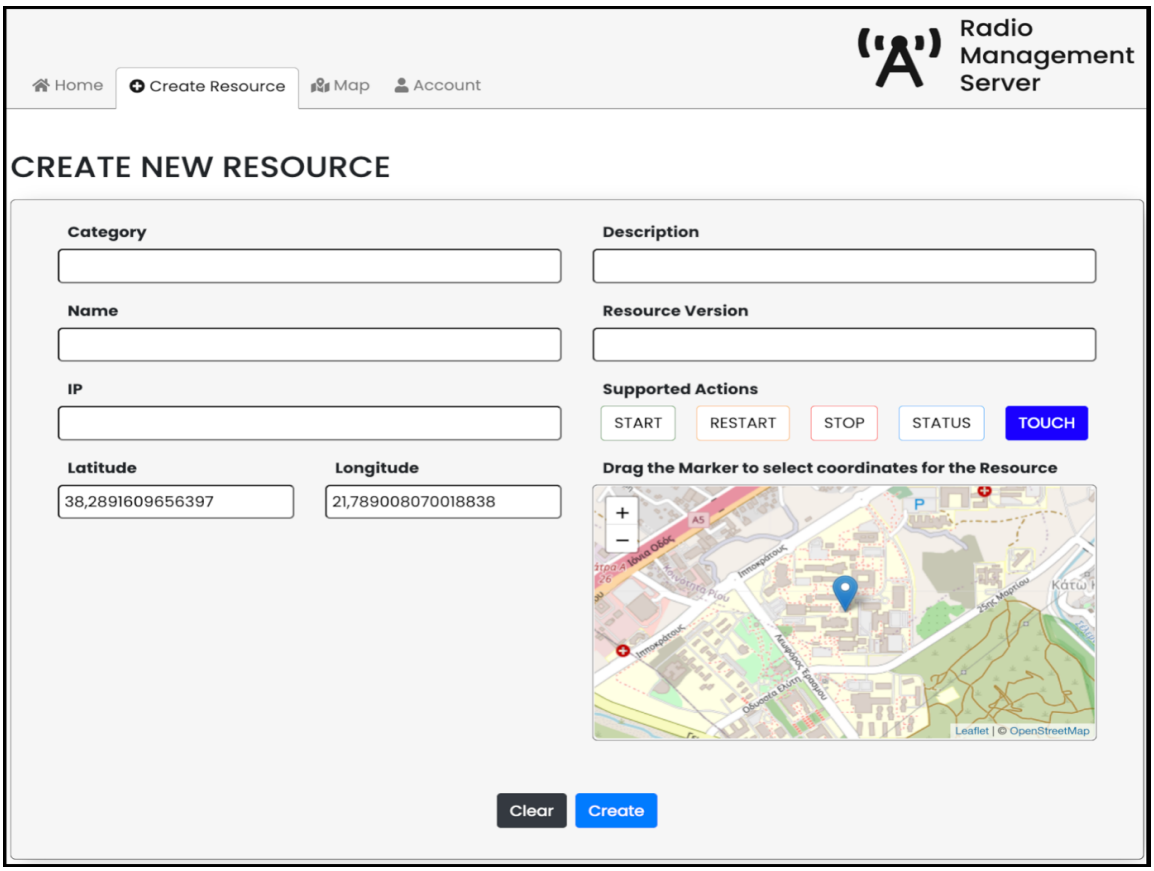

Figure 4.11*: Create a resource page in the client application*

When creating a resource, the SPA sends the desired request with the data that the user defined to the JSON server and the server in turn will just register whatever data received. In the case of an actual setup with the RM server, that server needs to communicate with the Amarisoft software and appropriately fire up a gNodeB instance in the network with the data received. This is mentioned because the behavior in the client most likely will vary in a similar request by either having a slower response time or even untested errors.

To sum up with the analysis, the use cases of this application focus on two authorization levels of user, one with the admin role and a simple user. Both levels can access the client UI and manage the resources, but a non-admin has less available actions to operate on the resources. Thus, the interface provides an easy tool to achieve managing the resources and at the same time protects the system from accidental user mismanagement.

4.2 Issues

The main problems faced in the analysis phase of the system have been the following:

◊ Providing an interface that can be used from different devices. Since a web application can be accessed by most modern devices, it has been vital for the UI during the development to take that into account.

◊ Security and authorization. Since this kind of application may be used by many different people in a group of some project, it was a necessary requirement to limit some actions. Thus, the admin authorization level was implemented, keeping some operations out of the reach of a simple user.

◊ Keeping track of the resource's state in real time and limiting access to certain operations depending on it. As it was shown previously, resources are represented by several fields that, in production setup, display the actual values of a gNodeB instance's state which is online and is often operating on the RAN side of the 5G network. Therefore, it was necessary for the UI to display the updated information without the interaction of a user. For those reasons, auto refreshing techniques of the data have been implemented, as well as limiting actions on non-available instances.

4.3 Responsive design

Developing an application for the web has some benefits, as they have already been mentioned, with the main one being that it can be accessed by a browser found in any modern device. But, getting the content of an application to fit in all the different screen sizes is the hard part. Nowadays, devices which anyone can use to browse the web vary in sizes from as small as a smartwatch (the present application has not been tested on one) to a bit bigger like the smartphones to a lot bigger ones like a laptop and even a smart TV. Therefore, the development of this application needed to consider those needs, by building the UI as easy, to view and navigate by any device that has a screen equal or bigger than an iPhone 7's (the smallest screen tested), as possible.

In ***Figure 4.12*** some samples of the UI are included, displaying the same page on two different viewports, one from a laptop (width: 1440px) and the other from a smartphone (width: 375px).

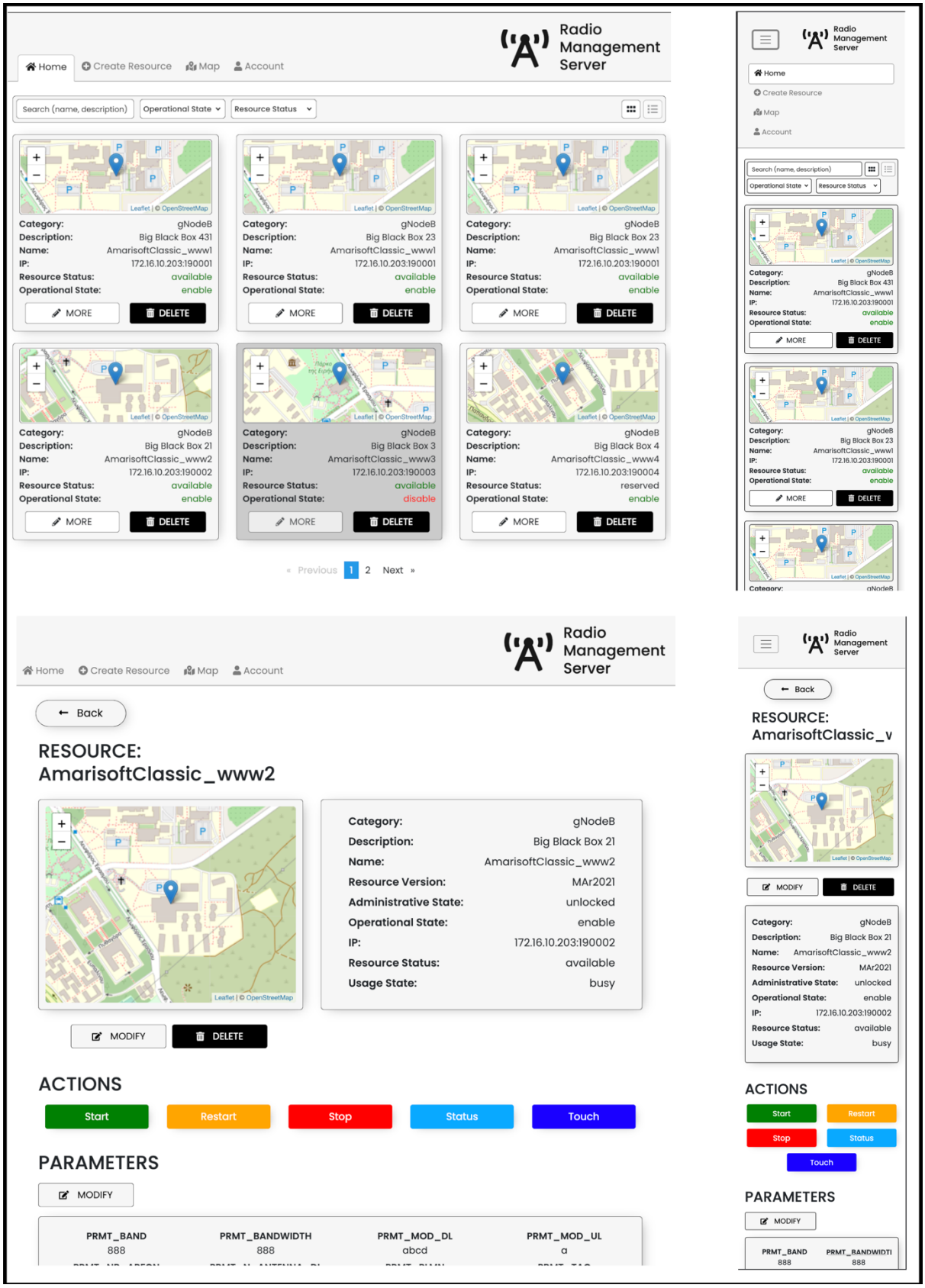

***Figure 4.12**: Views of the UI. On the left side from a laptop's viewport. On the right side from a smartphone's viewport.*

5. Planning and development

The development of the project can be divided into two applications. The first one is the development of the angular client application and the other one is the development of the auth server. Both applications aim to serve a practical UI for the RM server, that is developed by a different party, which party has been in constant communication, during the development of this thesis, in order to define the needs and requirements for the present application, as well as offer their feedback for any adjustments needed.

5.1 Tools and languages used

The different tools and languages used for the development of the two applications are mentioned below, followed by a small description on the way they were utilized.

Git and GitHub

Intensive use was made, throughout the development process, of both Git, as a code versioning tool, and GitHub, as a hosting provider for the software. The repository of the project can be found here:

https://github.com/spiros3p/Radio-Management-Server-thesis

In addition, the mentioned repository consists of two separate repositories, one for the angular application and one for the auth server application, that are added as submodules to the parent repository of the thesis' project. The intention behind that setup, is that both repositories are public and can be pulled by third party sources to be used in different projects as independent applications.

Documentation

Documentation of the software for both applications has been created. Since the client application is indented to be used alongside the RM server by the NAM group, having documentation as part of this thesis was a necessary process, in order to better streamline the information among the parties that intend to use and, maybe, further develop new features for the project.

The documentation for the angular SPA was created with the use of Compodoc [23], an open-source tool that generates static documentation from the JsDoc [24] comments in the code, and is deployed through GitHub pages and can be found here:

https://spiros3p.github.io/Radio-Management-Server-thesis-Angular-Documentation/

Documentation for the auth server was created using different tools, the Swagger UI Express and the swagger-jsdoc packages, that when combined with each other, they utilize the JsDoc comments in the code to produce a swagger UI that displays information about the endpoints (routes) of the server application. There are two types of routes in the auth server, those that their entire logic is handled in the server and those that their requests are only reverse-proxied between the RM-server and the client through the auth server. Hence, the documentation for the proxied routes was produced from the TMF639 OpenAPI specification, while the other one for the rest of the routes was manually added in the code in separate instances and can be found by running the software locally and visiting the following addresses:

- **localhost:{port}/rms-api-docs**, for the Resources endpoints according to TMF639 specification
- **localhost:{port}/resource-api-docs**, for the endpoints handled by the auth server

Development environment

Visual studio code from Microsoft has been the code editor of choice during the development, running on a MacBook Pro 13" with macOS Catalina 10.15.7. For the applications to run on the local machine, apart from the use of a local server, Docker was used as well. The containerization of the applications on the local machine aimed to simulate a production environment with all parts of the system running and communicating with each other, through Docker containers.

Programming languages

The main language used throughout the applications has been JavaScript. The Angular framework utilizes Typescript, which is a super set of JavaScript with syntax for types, in combination with HTML and CSS, through SCSS in this case, to build a single page application for the client UI. On the backend, code was written in JavaScript through the ExpressJS framework and the use of the NodeJS runtime environment. For the user database, MySQL was used, which is a RDBMS extending the capabilities of the structured query language.

5.2 Solutions adapted

When a new user signs up from the client, the new entry in the database table is, by default, filled with a false value for the two columns that control the access in the application and the admin level of the user. Thus, after the first setup and deployment of the application, the first user that registers form the client and try to login, will be unable to access it. An administrator that manages the database must access it directly and manually edit those values.

In addition, a user, admin or not, has no function available to change the admin authorization level of any user. The only way that this is possible is again through accessing the database directly and manually changing the respective value.

During the planning of the development, it was decided for the RM server to not host the user authentication service and, instead, that service's development to be part of the present thesis. For that to work and the RM server to be as protected as possible, the auth server, which is part of this development, needed to be the one sending the requests for the resources and not the client SPA directly. Hence, the auth server, in addition to the authentication service, it was developed to offer a reverse-proxy service and act as a middleman between the client and the RM server for the resources' requests.

5.3 Client Application domain

The client application was developed with the use of the Angular framework. Several components were built to make up the views, while services were developed for both the communication between the components and the communication between the client and the server. Further below the models, views and services developed in angular are explained.

5.3.1 Model

The models used in the application are ultimately two. First, is the Resource model which is generated with the use of swagger editor and the Open API standard from the TMF639 REST API specification. Second, is the User model that enables the authentication feature between the client and server. The Resource model partly consists of other models, such as the Feature and the Characteristic models. The Resource model was explained in **section 4.1.2**. The user model is explained below.

User

Id	the id of the entry in the table
Name	The full name of the user
Email	The email of the user
Password	The hashed password of the user
Accepted	Binary value that indicates a user's access to the main views
Admin	Binary value that indicates if a user has an admin role

Figure 5.1*: Model of the User object*

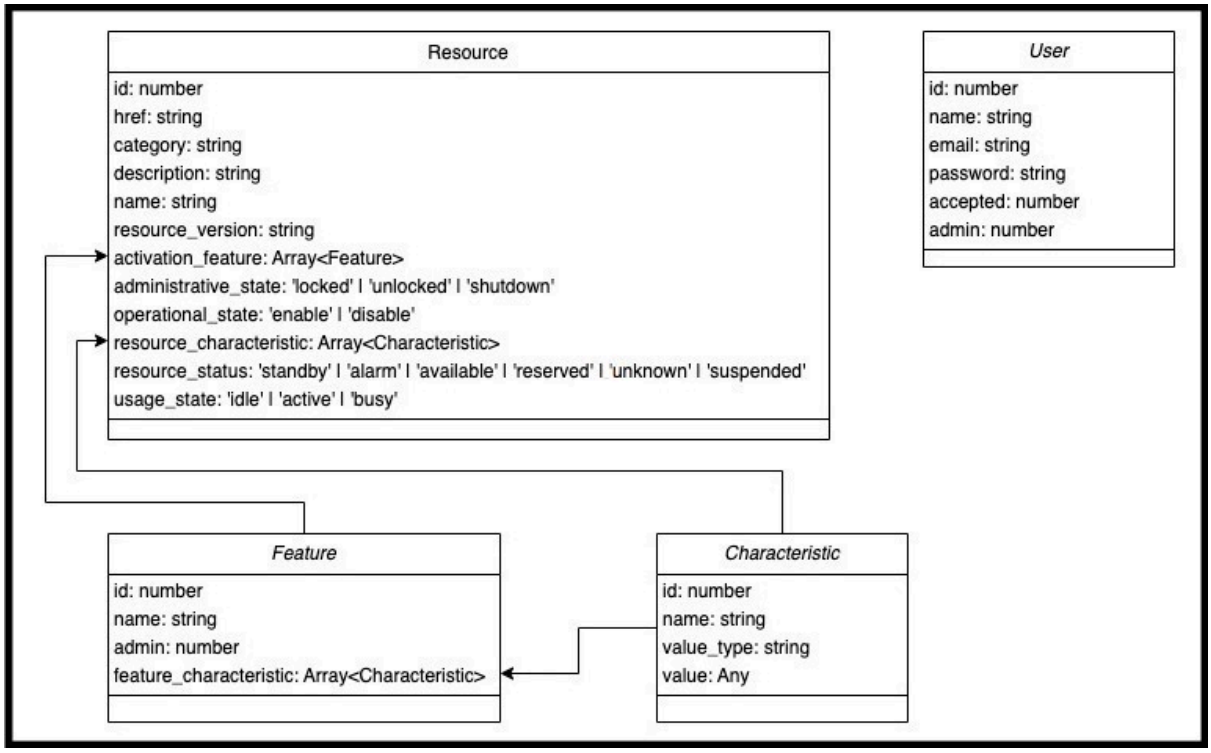

***Figure 5.2**: Class diagram: Angular models*

The password of the user is not stored in its original value given by the user. The auth server generates a hash value from the password input string and stores it in the database. Additionally, since SQL has no true or false values available to use, both Accepted and Admin entries use the *tinyint* type and are both stored as 0 or 1 in the database table, representing a false or true value respectively, and also used as such in the angular application.

5.3.2 View

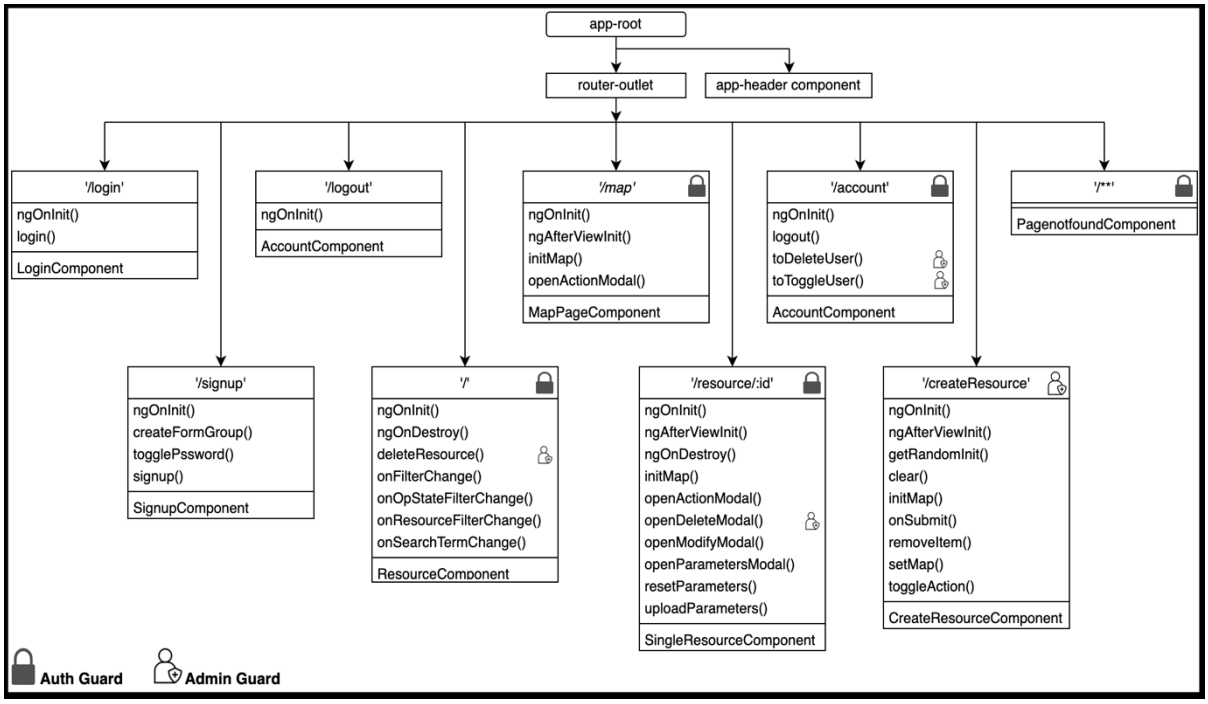

***Figure 5.3**: Class diagram: Angular views*

A client application developed with the angular framework is a single page application. This description is justified since what the client is usually served, upon its request to the server, is a single HTML file, with an almost empty <body> element and a few CSS and JS files. Thus, the JavaScript code in the angular framework takes over the task to dynamically render or remove the HTML content, which is presented by the angular views built with components.

The main component for the views of the SPA is the **app component**, which consists of two child components, the **header component** and the **router component**. These act as a layout for the views of the application. The header (displayed in ***Figure 4.7***) is present in all the routes, displaying the logo of the application and, depending on the authentication status of the user, the navigation bar.

The route component is responsible for switching between the views of the angular application, that consist of one or more components and ultimately represent the available routes a user can navigate to, through the UI. Most of these routes require from the user to be authenticated, other routes require the extra authorization level of the admin role from the user and lastly some routes are public and accessible by anyone.

As shown in ***Figure 4.2*** the login, signup and logout views are the only accessible views by a non-authenticated user. The rest of the views, appear only after the successful login from the user.

The views are the following:

◊ **'/login', the login view**

Renders the **LoginComponent** (***Figure 4.2, a***). This is where the user's credentials are sent to the auth server for authentication. Upon a successful request, a session cookie is returned with the response headers, the user's information is returned and stored in the local storage and the client navigates to the home view, '/'.

◊ **'/logout', the logout view**

Renders the **LogoutComponent**. This component, practically, has no HTML to display. The reason it was added is, in case a user manually hits the '/logout' route from the address bar, hence on initialization, it sends the logout HTTP request to the auth server to end the user's session and then navigates to the client to the 'login' route.

◊ **'/signup', the signup view**

Renders the **SignupComponent** (***Figure 4.2, b***). This is where the user's details for registering are sent to the auth server, in order for the record to be added into the user's table in the database.

◊ **'/', the home view** (requires authentication)

Renders the **ResourcesComponent** (***Figure 4.7***) which, during initialization, is responsible for fetching the resources data from the RM server through an HTTP request. In addition, it is composed of three child components, a) the filter component, which displays and handles the logic for the existing filters, b) the resources view block component, which displays the resources in block view and c) the resources view list component, which displays them in the list view. The two later components are displayed one at a time by utilizing a method in the **ui.service** injectable.

◊ **'/map', the map view** (requires authentication)

Renders the **MapPageComponent** (***Figure 4.8***). Upon view initialization, the map is rendered, the resources are fetched and added as markers on it. The component also handles the PATCH HTTP requests to update a resource, when the action is available for it.

◊ **'/resource/:id', the single resource view** (requires authentication)

Renders the **SingleResourceComponent** (***Figure 4.9***). The component, on initialization, fetches the respective Resource's data from the server. The DELETE and PATCH HTTP requests for the resource are also present in the component, utilizing methods from the **resource.service** to delete or update one. The component relies heavily on the use of modals for the user to verify an action to be requested.

◊ **'/account', the account view** (requires authentication + admin authorization)
Renders the **AccountComponent** (***Figure 4.4, a***). There, a non-admin user can only see the log out button. If the user has an admin level authorization, the client fetches all the users' data from the auth server, allowing to the admin to delete a user and toggle a user's access in the application.

◊ **'/createResource', the create resource view** (requires admin authorization)
Renders the **CreateResourceComponent** (***Figure 4.11***), where an admin level user has access to create a resource by filling the appropriate information and sending the POST request from the **resource.service**.

◊ **'/*', the page not found view** (requires authentication)
Renders the **PageNotFoundComponent**, where a user with an active client-server session will land by trying to access an invalid path. A non-authenticated user trying to access an invalid path will simply be "redirected" to the /login view from the router, due to the **authguard.service**.

5.3.3 Services

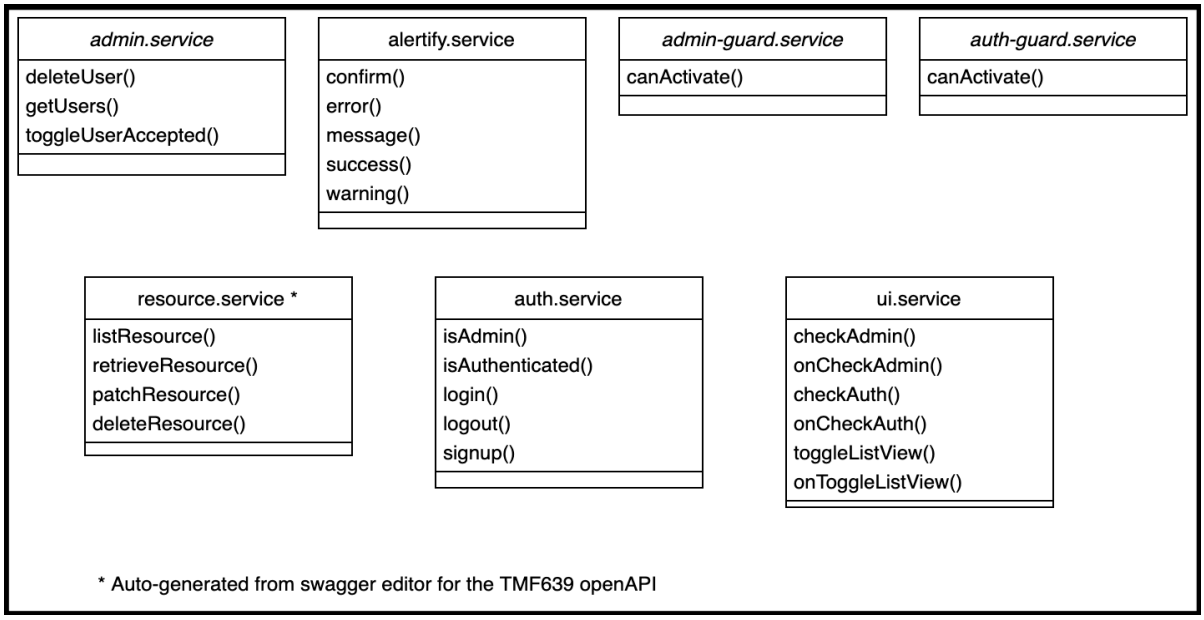

Figure 5.4*: Class diagram: Angular services*

The angular services used in the project contain mostly the methods calling the HTTP requests for the resources and the users. Only the **alertify.service** and the **ui.service** hold methods that handle changes in the client UI without any HTTP request.

In addition, the **admin-guard.service** and **auth-guard.service**, are only used in the router module, defining the appropriate access for the authenticated and the admin authorized level users.

All the services are injected into the components they are needed by being passed in the constructor parameters.

The services used in the project are the following:

◊ **admin.service**

Contains the requests that are available for an admin level user, through the **'/account'** route. Through that service, an admin is able to fetch all the users from the database, delete a user and toggle a user's access in the application from the client UI.

◊ **auth.service**

Contains the requests regarding the authentication and authorization of a user. Apart from the login(), logout() and signup() methods that are used in their respective views, the isAdmin() and isAuthenticated() methods are called during the navigation from one view to another, to determine if the user has access to that view and what kind of access that is.

◊ **resource.service**

The methods in this service are auto generated from the swagger editor with the use of the TMF639 OpenAPI specification. It contains all the requests available for the resources that are sent to the auth server and, from there, are proxied to the RM server.

◊ **alertify.service**

This service imports the alertify package [25] and provides the appropriate methods in the components that is injected, for the banner notifications that appear temporarily in the client UI and provide the user with feedback after some action.

◊ **ui.service**

Contains the methods that are responsible for the UI changes that happen on some components. For example, the header component, when it is rendered, calls both the checkAuth () and the checkAdmin() methods to decide whether to render the navigation bar and to render the Create Resource tab respectively. In addition, the toggleListView() method is called to handle the transition from the grid view to the list view of the resources and vice versa.

◊ **admin-guard.service**

It is a guard service used by the router in certain defined routes. Its role is to call the isAdmin() method of the **auth.service**, during the navigation to the defined route/s and before the client renders the desired view. If the response is not positive, the SPA navigates to the home view ('/').

◊ **auth-guard.service**

It is a guard service used by the router in certain defined routes. Its role is to call the isAuthenticated() method of the **auth.service**, during the navigation to the defined route/s and before the SPA renders the desired view. If the response is not positive, the client navigates to the login view ('/login').

5.4 System Architecture

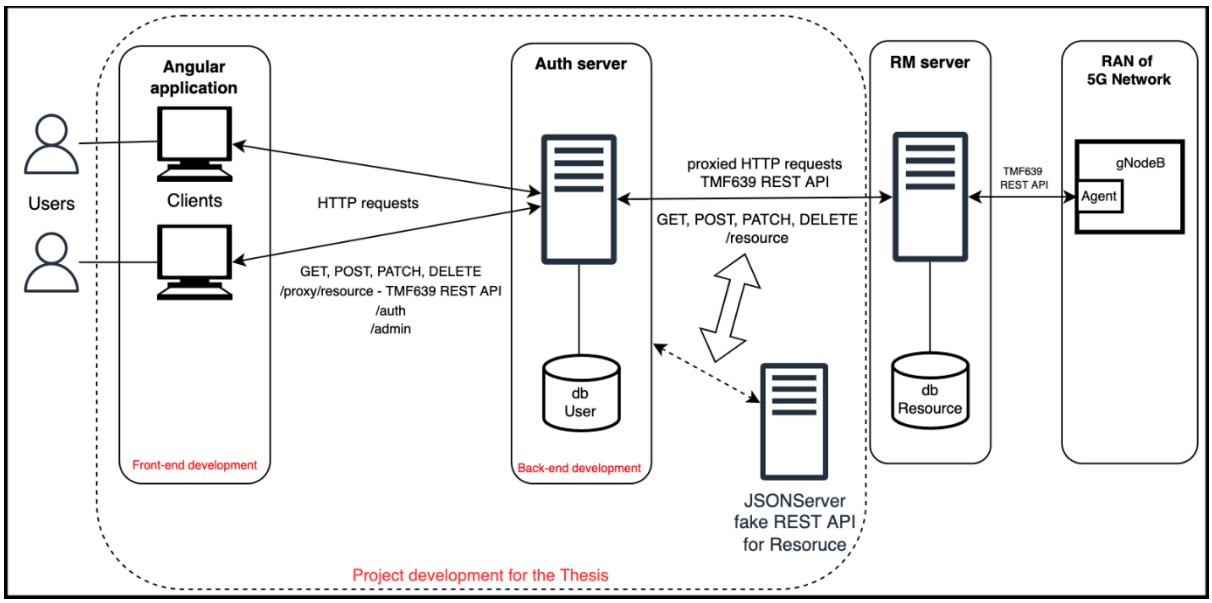

***Figure 5.5**: System architecture*

The general architecture of the system related to this project is shown in ***Figure 5.5***. At the user's end, we find the client UI which is the single page application built with the angular framework. The client is then communicating, through HTTP requests, with the auth server, which is part of this development. As previously mentioned, the auth server, handles the requests, that are related to the users table and the authentication, itself and proxies the requests related to the Resources to the RM server. More detailed information about the HTTP requests and how they are proxied are provided in **section 5.5** below. In addition, as previously explained, during the development the actual RM server was not used. Instead, it was replaced by a fake JSONServer, that fully implements the REST API interface, to store and operate on dummy data for the state of the resources.

5.5 Communication

5.5.1 Client – Server (Users data)

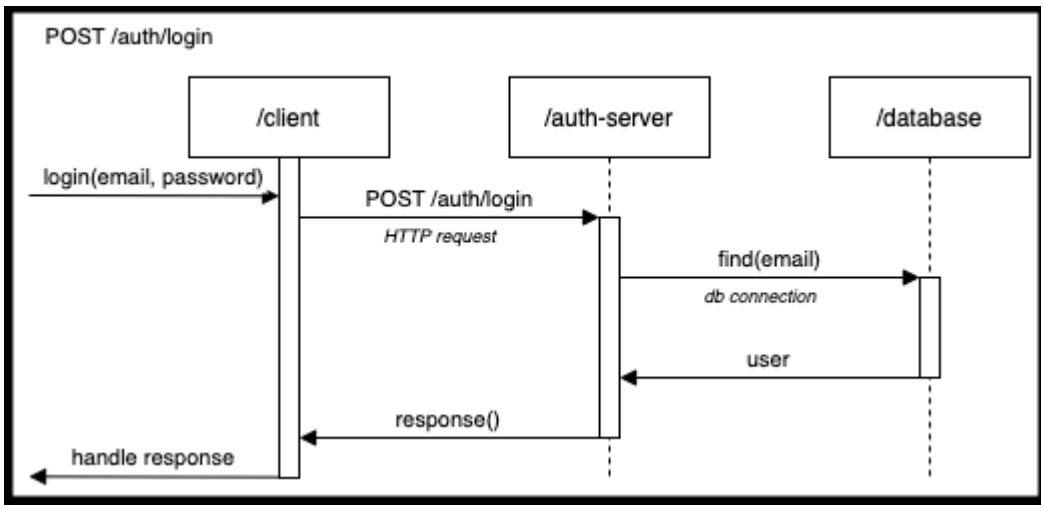

***Figure 5.6**: Sequence diagram: authentication*

The sequence diagram in ***Figure 5.6*** shows how the user authentication is happening between the client and the server. This sequence example is identical for every other action the client offers, related to the user's table.

In general, from the SPA, a HTTP request, with the appropriate data, is sent to the auth server's defined endpoint. On the auth server, the router calls the appropriate controller to connect on the database and operate the proper action. After the database operation ends, the controller sends the appropriate HTTP response back to the client. Note that most routes in the auth server are protected, that means that a middleware method is used, to check if the client sending the request has been authenticated and has an active session with the server. Thus, before accessing the protected requests, the client needs to pass through the login page.

In this example, the authentication is happening in the following way. A user inputs the email and the password and proceeds to click the log in button. Then the SPA sends the HTTP POST request with the body data to the *'/auth/login'* endpoint of the auth server. The auth server receives the request and calls the defined controller to handle it. The controller connects to the database and searches for the user entry with the provided email. If it is successful, it returns the entry and hashes the password input to compare it with the hashed password value stored in the table. If the comparison is successful, the next middleware function is called, which is responsible to check if the now authenticated user can access the main UI, by checking the *'accepted'* column of the user entry from the table. If this is successful as well, the auth server finally returns a response to the client with the user data and a session cookie in the header of the response. That session cookie is carried

with every following request between the client and the auth server and it is responsible to maintain the authentication between the two applications.

5.5.2 Client – Server (Resources data)

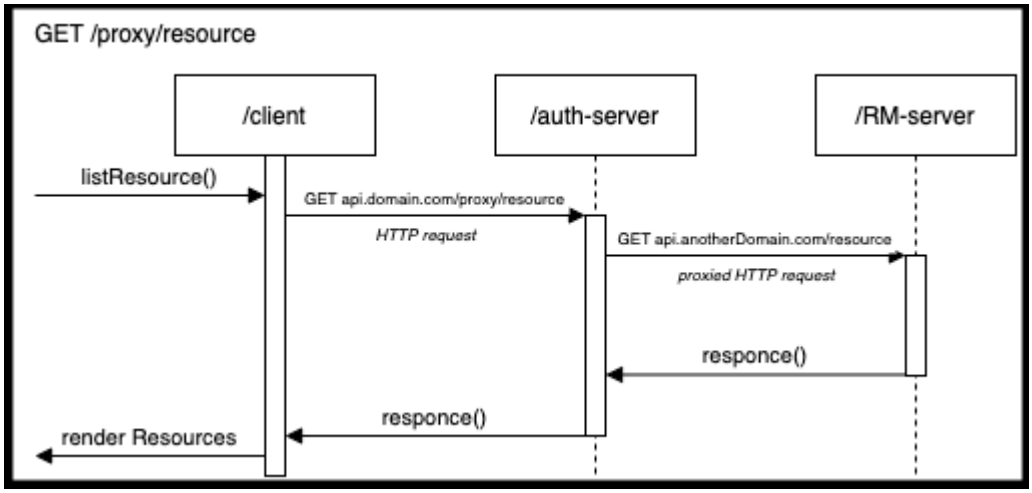

***Figure 5.7**: Sequence diagram: example of a request fetching Resources*

The sequence diagram in ***Figure 5.7*** shows the path that a request, related to the Resources, follows, after it is sent from the client. The path of such request is different from the one analyzed in **section 5.5.1**. That is because, the actual RM server, or even the JSONServer used in this development, has no authentication built in to secure its endpoints. Hence, anyone knowing the address and the endpoints of the RM server can interact and mess around with the Resources state stored in the server. In order to minimize the probability of that scenario to happen, the client should not exchange requests with the RM server directly, or else the information about the RM server would be visible from any user through the browser and most importantly the authentication process would serve little practical use. The way to achieve such task, is made possible with the use of a reverse-proxy service coded in the auth server. Before moving to analyze the sequence diagram, it would be helpful to explain briefly what a reverse-proxy is.

Reverse-proxy

A reverse proxy is a server that sits in front of one or more web servers, intercepting requests from clients and ensures that no client ever communicates directly with that origin server [26]. In the present development that server is offered as a microservice through the auth server. The JavaScript code running on the NodeJS backend ensures that any Resources requests received from the client, are forwarded to the RM server and any response from the RM server is forwarded to the client. Thus, eliminating any direct communication between the client and the RM server.

Moving on to the ***Figure 5.7*** example diagram. The client, in order to fetch the Resources data, calls the **listResource()** method, located in the angular **resources.service** service. This method sends a HTTP GET request to the auth server's *'/proxy/resource'* endpoint. The auth server, in turn, when receiving any request with the *'/proxy'* in its path, it removes the *'/proxy'* from the path and forwards it to the RM server. Then, during the reverse procedure, the response from the RM server will, once again, reach the auth server first. There, a modification on the *'access-control-allow-origin' header* is necessary to overwrite the existing one, referring to the RM server, with that of the auth server. That is required, for the client's browser to identify the response as it is coming from the auth server's origin, which is the initial origin that the client's request was sent. This a safety measure based on the CORS mechanism implemented in most browsers, that is briefly explained in **section 5.5.3**.

The response received at the client is an array of JSON objects, or a single JSON object in case of a GET HTTP requests at the *'/proxy/resource/:id'* endpoint. The JSON object, as already mentioned, follows the TMF639 OpenAPI specification. An example of a resource object is shown in ***Figure 5.8***. The properties of the object are explained in **section 4.1.2**. But at this point, a thorough explanation on some Update operations of a Resource is needed while looking at the example.

Updating a resource

In the example, it is noticeable that nested objects with key-value pairs of "name": "action" and "name": "action_parameters**"** appear under both the **activation_feature** and the **resource_characteristic** properties of the Resource object. The **activation_feature** property, containing the new data, is the one sent in the PATCH request's body to the RM server, while the **resource_characteristic** property's values are the ones used from the client application to be displayed in the UI and this property is not included in the object of an update request. The reason that it works like this, is due to the RM server's application development. Hence, an update operation related to the **action parameters** has the following approximate sequence: 1) the client application sends a PATCH request, with the body data containing the updated **activation_feature** property. 2) The RM server receives the request, reads the new data in the object's **activation_feature** property from the request's body and updates the information in the **resource_characteristic** property of the specific Resource object stored in its database. 3) Finally, the RM server, unless there was an error, returns the object response with the new data in the **resource_characteristic** property.

It is important to mention, that this behavior is not materialized in the present development, since the RM server is replaced by a test JSONServer. Which, by default, during a PATCH request, it updates the information in the corresponding properties of an object.

```
{
  "id": "1171",
  "href": "5741d8dc-479a-11ec-85e9-0242ac150004",
  "category": "gNodeB",
  "description": "Big Black Box",
  "name": "AmarisoftClassic_www",
  "resource_version": "MAr2021",
  "administrative_state": "locked",
  "operational_state": "enable",
  "activation_feature": [
    {
      "name": "gNodeB_service",
      "feature_characteristic": [
        {
          "name": "action_parameters",
          "value": {
            "value": {
              "PRMT_TAC": "111",
            }
          }
        },
        {
          "name": "action",
          "value": {
            "value": "stop"
          }
        }
      ]
    }
  ],
  "resource_characteristic": [
    {
      "id": "5741e548-479a-11ec-85e9-0242ac150004",
      "name": "location",
      "value_type": "array",
      "value": {
        "value": [
          38.2880923964281,
          21.789761781692505
        ]
      }
    },
    {
      "id": "5741e23c-479a-11ec-85e9-0242ac150004",
      "name": "IP",
      "value_type": "string",
      "value": {
        "value": "172.16.10.203:19001"
      }
    },
    {
      "id": "string",
      "name": "action_parameters",
      "value_type": "object",
      "value": {
        "value": {
          "PRMT_TAC": "123",
          "PRMT_BAND": "123",
        }
      }
    },
    {
      "id": "5741e750-479a-11ec-85e9-0242ac150004",
      "name": "supported_actions",
      "value_type": "list",
      "value": {
        "value": [
          "start",
          "restart",
          "stop",
          "status",
          "touch"
        ]
      }
    },
    {
      "id": "string",
      "name": "action",
      "value_type": "string",
      "value": {
        "value": "restart"
      }
    }
  ],
  "resource_status": "available",
  "usage_state": "active"
}
```


***Figure 5.8**: JSON example of a Resource state*

5.5.3 CORS

Cross-Origin Resource Sharing (CORS) is an HTTP-header based mechanism that allows a server to indicate any origins (domain, scheme, or port) other than its own from which a browser should permit loading resources [27].

In this project, the origins, where the client application and the auth server are served from, are different. In a development environment, for example, the client is served on http://localhost:4200 and the server on http://localhost:3000. Hence, the auth server needs to set the *'access-control-allow-origin'* header to the origin of the client in the response.

5.6 Deployment

The applications that have been developed and used in the thesis are deployed online for demonstration purposes. The URLs of the live demo deployment can be obtained from the GitHub repository of the project (see **section 5.1**).

The applications deployed for this project are:

a. Angular SPA (client UI)
b. NodeJS server application (auth server)
c. MySQL (database)
d. phpMyAdmin (to access the database)
e. JSONServer fake REST API (replacement for the RM server)

There have been two options that were used for the deployment of the application in a production environment. The first one is, to manually set them up and deploy them natively in machine, which is the preferable way in a development environment. But for a production environment, which is the case for the live demo deployment, the second option of using Docker and containers to deploy the applications, is the preferable one. Hence, the following steps[1] will guide the reader on how to install and deploy the system with the use of the Docker platform.

Prerequisites

- NodeJS (tested v. 14.17.6) and npm (tested v. 7.24.1)
- Angular CLI (v. 12.2.7)
- Docker
- docker-compose (the one written in python)

Installation

To clone the repository and its submodules to the desired local machine:

$ git clone –recursive https://github.com/spiros3p/Radio-Management-Server-thesis.git

Setup

1. (optional) In the ***./docker-compose.prod.yml*** configuration file, one or more of the five applications defined can be edited out, so that it will not start along with the rest. For example, the table ***tbl_user*** can be added to an existing database and be used for the auth server to connect to. Hence, making the MySQL and phpMyAdmin containers unnecessary in the configuration file.

[1] These steps can also be found in the readme.md file of the GitHub repository.

2. In the ***./docker-compose.prod.yml*** configuration file, it is apparent that some of the values are defined as variables. These variables are read from the ***./.env*** configuration file. This file needs to be created in the root of the repository, either by copying the data from the existing ***./.env-example*** file to a newly created ***./.env*** file or simply renaming the it.

3. In the ***./.env*** file, the environment variables need to be defined to provide the following information:

DB_HOST	Defines the host database, can be either the container database or an existing database elsewhere. e.g., thesis-db
DB_USER	Defines the username of the credentials for the host database. If the container database is used, the user must be set to **root**. e.g., root
DB_PASSWORD	Defines the password of the credentials for the host database. This string can be set freely. e.g., password
DB_DATABASE	Defines the database to be created in the database host and for the auth server to connect to. It can be set freely. e.g., thesisDB
SESSION_SECRET	Defines a key for the Express session to use for signing and/or encrypt/decrypt cookies. It can be set freely. e.g., secret
FRONT_END_IP	Defines the origin that the client application is served from. It is used by the auth server for CORS. e.g., http://localhost:4200
PROXY_IP	Defines the IP address of the RM server (in this case the JSONServer's). It is used by the auth server to proxy the Resources requests. e.g., http://localhost:5000
FRONT_PORT	Defines the PORT that the container of the client application server will listen to. e.g., 4200
AUTH_PORT	Defines the PORT that the container of the auth server application will listen to. e.g., 3000
API_PORT	Defines the PORT that the container of the JSONServer application will listen to. e.g., 5000

Figure 5.9*: Environment variables used in the applications*

Note that in most cases, to target a container, depending on the Docker network type used, 'localhost' could be replaced with '127.0.0.1' or even the IP of the machine in the local network (e.g., http://192.168.1.7:4200).

4. Since the client UI is a single page application, which means that it is a simple HTML file, a server, in this case the NGINX server, is needed to serve the SPA. The NGINX server is deployed in the second part of the docker image for the Angular application. The ***./frontend/nginx/nginx.conf*** configuration file defines the port that the server listens to. This port needs to be the same as the one defined at the FRONT_PORT environment variable of the ***./.env*** file in **step 3**.

5. (optional) In the ***./frontend/src/app/router/app-routing.module.ts*** file, in case of trouble on setting up the auth server for authentication, the angular routes in the file without the AuthGuard can be utilized instead by commenting the appropriate block of code.

6. In the ***./frontend/src/environments/environment.prod.ts*** file, the ***authUrl*** variable must be defined with the IP address that the auth server is served from, while the ***apiUrl*** variable must be set to an empty string for the proper reverse-proxy service to work.

7. To start the containers through the docker-compose configuration file, the following command can be run in the root directory of the repository:

$ docker-compose -f docker-compose.prod.yml up --build -d

8. The database table is not seeded automatically through the code, hence, the ***./backend/db/tbl_users.sql*** file, which has some initial dummy user entries[2], can be imported in the database by navigating to ***http://localhost:5001*** where the phpMyAdmin can be accessed to manage the database.

9. If there are no errors, the client application can now be accessed by navigating to: ***http://localhost:4200***.

[2] username: admin@admin.com password: password, username: test@test.com password: pasword

6. Conclusion

The present thesis was concluded with the successful attempt to take advantage of the software engineering skills, acquired during the academic years. The client application was developed according to the needs and use cases of the RM server. Thus, a user can access the available operations of the RM server through an easy and friendly visual environment from any computer or smartphone connected on the same network.

More importantly, the completion of the thesis's development is adding a tool in the arsenal of the open-source software for 5G vertical deployments. Hence, contributing to the expanding needs for resource management of the mobile networks.

6.1 Future improvements and development

Regarding possible improvements in the development of the applications in this thesis, the following suggestions could be proposed:

◊ Tracking a user's activity/changes in the application. This could be done by adding a new table in the database and having the auth server to log the user events of any UPDATE, DELETE or POST requests for a gNodeB state there.

◊ Improving the overall performance of the SPA by better utilizing the angular framework. There was a low experience level with the angular framework at the beginning, thus, there is surely room for optimizing the code there.

◊ Adding user authentication using a Facebook and/or Google account. It is a widely used concept of allowing a user to access an application using an existing Facebook or Gmail account. Thus, making access easier and faster for the user.

References

[1] D. Chandramouli, R. Liebhart and J. Pirskanen, 5G for the connected world. Hoboken: John Wiley & Sons Ltd., 2019.

[2] 3rd Generation Partnership Project (3GPP), 3GPP TS 23.501, 16.7.0 ed., 3rd Generation Partnership Project, 2020.

[3] K. Bahia and A. Delaporte, "The State of Mobile Internet Connectivity 2020", GSMA, 2020, p. 16.

[4] "Statistics", GSMA, 2022. [Online]. Available: https://www.gsma.com/futurenetworks/all-ip/statistics/. [Accessed: 25- Apr- 2022].

[5] A. Delaporte and K. Bahia, "The State of Mobile Internet Connectivity 2021", GSMA, 2022, p. 48.

[6] D. Bolan, "5G Core – Are We Ready? - Dell'Oro Group", Dell'Oro Group, 2020. [Online]. Available: https://www.delloro.com/5g-core-are-we-ready/. [Accessed: 25- Apr- 2022].

[7] "5G Network Architecture. Core, RAN, & Security Architecture For 5G", Viavisolutions.com. [Online]. Available: https://www.viavisolutions.com/en-us/5g-architecture. [Accessed: 25- Apr- 2022].

[8] C. Craven, "What Is 5G RAN?", sdxcentral, 2021. [Online]. Available: https://www.sdxcentral.com/5g/ran/definitions/what-is-5g-ran/. [Accessed: 26- Apr- 2022].

[9] J. Horwitz, "The definitive guide to 5G low, mid, and high band speeds", VentureBeat, 2019. [Online]. Available: https://venturebeat.com/2019/12/10/the-definitive-guide-to-5g-low-mid-and-high-band-speeds/. [Accessed: 26- Apr- 2022].

[10] E. Dahlman, S. Parkvall and J. Sköld, 5G NR, 2nd ed. Academic Press, 2021.

[11] “LTE Software eNodeB and NR Software gNB”, ver. 2.17, Amarisoft, 2021.

[12] "Angular", Angular.io, 2022. [Online]. Available: https://angular.io/. [Accessed: 08- May- 2022].

[13] "Node.js", Node.js, 2022. [Online]. Available: https://nodejs.org/. [Accessed: 08- May- 2022].

[14] Red Hat, "What is a REST API?," 2020.

[15] OpenAPI Initiative, "OpenAPI Specification".

[16] "SPA (Single-page application) - MDN Web Docs Glossary: Definitions of Web-related terms | MDN", Developer.mozilla.org, 2021. [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/SPA. [Accessed: 13- May- 2022].

[17] "What are microservices?", microservices.io. [Online]. Available: https://microservices.io/. [Accessed: 13- May- 2022].

[18] TMF639 specification, Resource Inventory Management API User Guide, 4th ed. TM Forum, 2020.

[19] "Simple RAN for 5G Sovereignty", 2022. [Online]. Available: https://handbook.rapid.space/evangelist/RS-Presentation.For.5G.Sovereignty. [Accessed: 15- May- 2022].

[20] "How to Create API with no-code | AppMaster", Appmaster.io, 2022. [Online]. Available: https://appmaster.io/blog/how-create-api-no-code. [Accessed: 15- May- 2022].

[21] "JSON", Json.org, 2022. [Online]. Available: https://www.json.org/json-en.html. [Accessed: 15- May- 2022].

[22] ["IBM Docs", Ibm.com, 2022. [Online]. Available: https://www.ibm.com/docs/en/cics-ts/5.3?topic=protocol-http-requests. [Accessed: 15- May- 2022].

[23] "Compodoc - The missing documentation tool for your Angular application", Compodoc.app. [Online]. Available: https://compodoc.app/. [Accessed: 01- Jun- 2022].

[24] "Use JSDoc: Index", Jsdoc.app. [Online]. Available: https://jsdoc.app/. [Accessed: 01- Jun- 2022].

[25] "AlertifyJS", Alertifyjs.com. [Online]. Available: https://alertifyjs.com/. [Accessed: 09- Jun- 2022].

[26] "What is a reverse proxy? | Proxy servers explained", Cloudflare.com. [Online]. Available: https://www.cloudflare.com/learning/cdn/glossary/reverse-proxy/. [Accessed: 11- Jun- 2022].

[27] "Cross-Origin Resource Sharing (CORS) - HTTP | MDN", Developer.mozilla.org. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS. [Accessed: 11- Jun- 2022].

[28] "Home - Docker", Docker. [Online]. Available: https://www.docker.com/. [Accessed: 13- Jun- 2022].

[29] R. Haverans, "From 1G to 5G: A Brief History of the Evolution of Mobile Standards", Brainbridge.be, 2021. [Online]. Available: https://www.brainbridge.be/en/blog/1g-5g-brief-history-evolution-mobile-standards#:~:text=On%20December%201%2C%202018%2C%20South,was%20made%20back%20in%201973. [Accessed: 18- Jun- 2022].

[30] W. Chie, S. Huang, C. Lai and H. Chao, "Resource Management in 5G Mobile Networks: Survey and Challenges," Journal of Information Processing Systems, vol. 16, no. 4, pp. 896-914, 2020. DOI: 10.3745/JIPS.03.0143.