ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

IoT System Development and Implementation for ECG Signal Analysis and Visualization

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Μπακόλας Ιωάννης

Επιβλέπων : Δημήτριος Σούντρης
Αναπληρωτής Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούνιος 2017

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

IoT System Development and Implementation for ECG Signal Analysis and Visualization

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Μπακόλας Ιωάννης

Επιβλέπων : Δημήτριος Σούντρης
Αναπληρωτής Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 19η Ιουνίου 2017.

.....	.....	.....
Δημήτριος Σούντρης	Κιαμάλ Πεκμεντζή	Γεώργιος Ματσόπουλος
Αναπληρωτής Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Αναπληρωτής Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούνιος 2017

.....

Μπακόλας Ιωάννης

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Abstract

The purpose of this diploma thesis is the design, development and implementation of an end-to-end ECG analysis and visualization IoT system using market ready technologies that can be used in real life medical applications and scenarios. For the development of the system we chose technologies and frameworks by taking account factors like cost, support, platform portability, backwards compatibility with older versions of software and hardware, power consumption, computational and networking load. The system consists of three main parts, the first part is the embedded Linux based application which reads the ECG signal and transmits it via classic or low energy Bluetooth to a smartphone or tablet. The second part is the android application that receives the ECG signal, stores it on the device, visualizes it (real-time visualization support on BLE connections) and sends it as a http packet through internet to the server for database storing. The last part is the webservice that runs on the server, its task is to receive the packet from the android app, evaluate that the packet has the right format and connect to the database to insert or update the record. The performance of every part of the system was validated by experimental evaluation of trade-offs that affect the run-time operation of all sub-systems, thus leading to the optimum configuration of the complete system.

Keywords: ECG analysis, Internet of Things (IoT), Android, Bluetooth, Databases, Embedded Systems, Smartphone, Tablet, Wearable Devices

Περίληψη

Ο σκοπός αυτής της διπλωματικής είναι ο σχεδιασμός, η ανάπτυξη και η υλοποίηση ενός IoT συστήματος για την ανάλυση και οπτικοποίηση ηλεκτροκαρδιακού σήματος χρησιμοποιώντας εμπορικές τεχνολογίες το οποίο μπορεί να χρησιμοποιηθεί σε πραγματικές ιατρικές εφαρμογές. Για την ανάπτυξη του συστήματος επιλέξαμε τεχνολογίες και δομές λαμβάνοντας υπόψιν παράγοντες όπως κόστος, υποστήριξη, συμβατότητα με παλιότερες εκδόσεις λογισμικού και υλικού, κατανάλωση ενέργειας, υπολογιστικό και επικοινωνιακό φορτίο. Το σύστημα αποτελείτε από τρία κυρίως τμήματα, το πρώτο τμήμα είναι η ενσωματωμένη εφαρμογή σε Linux η οποία διαβάζει το ηλεκτροκαρδιακό σήμα και το μεταδίδει μέσω κλασσικού ή χαμηλής ενέργειας Bluetooth σε smartphone ή tablet. Το δεύτερο τμήμα είναι η εφαρμογή σε android η οποία δέχεται το ηλεκτροκαρδιακό σήμα, το αποθηκεύει στην συσκευή, το οπτικοποιεί (η οπτικοποίηση σε πραγματικό χρόνο υποστηρίζετε σε συνδέσεις BLE), και το στέλνει σαν πακέτο http μέσω Internet στον server για αποθήκευση στην βάση δεδομένων. Το τελευταίο τμήμα είναι το web service το οποίο τρέχει στον server, η δουλειά του οποίου είναι να λάβει το πακέτο από την εφαρμογή σε android, να ελέγξει αν το πακέτο έχει την σωστή μορφοποίηση και να συνδεθεί με την βάση δεδομένων για να εισάγει ή να ανανεώσει την εγγραφή. Η επίδοση κάθε τμήματος του συστήματος ελέγχθηκε παίρνοντας πειραματικές μετρήσεις των διαφόρων παραγόντων που επηρεάζουν την λειτουργία των υποσυστημάτων κατά την διάρκεια λειτουργείας ώστε να έχουμε μια πλήρη και συνολική εκτίμηση της λειτουργίας του συστήματος.

Λέξεις-Κλειδιά: Ανάλυση ηλεκτροκαρδιακού σήματος, Internet of Things (IoT), Android, Bluetooth, Βάσεις δεδομένων, Ενσωματωμένα συστήματα, Smartphone, Tablet, Φορετές συσκευές

Acknowledgments

I would like to express my sincere thanks to Professor Dimitrios Soudris for giving me his trust and the opportunity to carry out my diploma thesis under his supervision. His guidance and teachings provided me with the skills and the motivation to accomplish this work. I am also grateful to my supervisor Vasileios Tsoutsouras for his guidance and support. I would also like to thank him for the continuous productive and smooth cooperation throughout the conduction of this thesis. Lastly, I would like to give my most sincere thanks to my family for their unexhausted encouragement and enormous support they showed me in achieving my goals. Without their assistance and love the challenges in my life would seem unbearable.

Περιεχόμενα

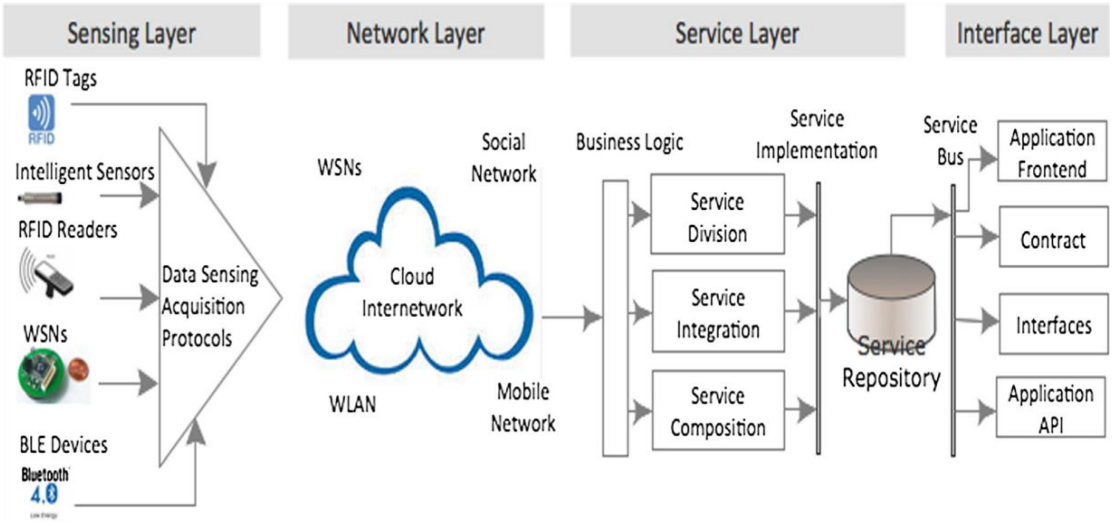
Εκτεταμένη Περίληψη

Ο σκοπός αυτής της διπλωματικής είναι ο σχεδιασμός, η ανάπτυξη και η υλοποίηση ενός συστήματος για την ανάλυση και οπτικοποίηση ηλεκτροκαρδιακού σήματος με δυνατότητες IoT. Χρησιμοποιήσαμε τεχνολογίες οι οποίες μπορούν να χρησιμοποιηθούν σε πραγματικές ιατρικές εφαρμογές και έχουν υιοθετηθεί από πολλές εμπορικές εταιρείες και οργανισμούς για την ανάπτυξη συστημάτων ευρείας κλίμακας. Παρακάτω παρουσιάζονται οι βασικές ιδέες και μεθοδολογίες που χρησιμοποιήθηκαν για τον σχεδιασμό του συστήματος, τα επιμέρους υποσυστήματα από τα οποία αποτελείται καθώς και τα αποτελέσματα των πειραματικών μετρήσεων που τα αφορούν ώστε να έχουμε μια πλήρη εικόνα για την λειτουργία και την συμπεριφορά του συστήματός μας.

Μία από τις μεγαλύτερες προκλήσεις που πρέπει να ξεπεράσουμε σήμερα και στο εγγύς μέλλον είναι η δραματική αύξηση του κόστους της υγειονομικής περίθαλψης σε παγκόσμια κλίμακα. Ένας από τους βασικούς παράγοντες που προκαλούν την αύξηση του κόστους της υγειονομικής περίθαλψης είναι η ανάγκη παρακολούθησης των ασθενών τόσο κατά τη διάρκεια της θεραπείας τους όσο και μετά. Με την πρόοδο της τεχνολογίας μπορούμε να αναπτύξουμε και να υλοποιήσουμε συσκευές που μπορούν να παρακολουθούν με επιτυχία και με μεγάλη ακρίβεια τα περισσότερα ανθρώπινα βιολογικά σήματα χρησιμοποιώντας ενσωματωμένους αισθητήρες, έχουν δυνατότητες διασύνδεσης στο διαδίκτυο, μπορούν να λειτουργούν για μήνες λόγω της χαμηλής κατανάλωσης ενέργειας, είναι μικρές και αρκετά ελαφριές ώστε να τοποθετούνται στο ανθρώπινο σώμα εύκολα χωρίς να προκαλούν ενόχληση και έχουν χαμηλό κόστος παραγωγής. Αυτές οι φορετές ιατρικές συσκευές φαίνεται να είναι η λύση για να σταματήσει ή να περιοριστεί η άνοδος του κόστους της υγειονομικής περίθαλψης.

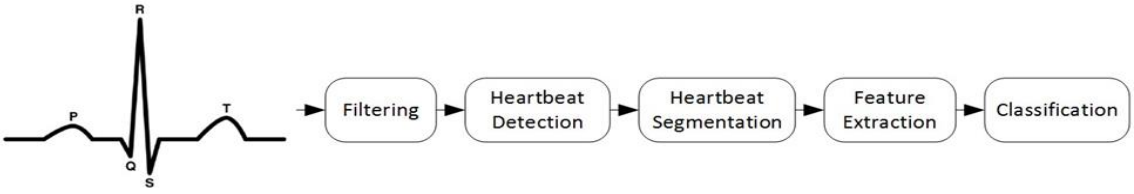
Τα παραπάνω μπορούν να εφαρμοστούν σε ευρεία κλίμακα χάρη στην ανάπτυξη και ταχύτατη διάδοση του Internet of Things (IoT). Γενικά, το IoT είναι μια έννοια που αναφέρεται στη διασύνδεση καθημερινών αντικειμένων όπως οικιακές συσκευές, οχήματα, κτίρια και άλλα αντικείμενα που διαθέτουν ενσωματωμένα ηλεκτρονικά κυκλώματα, τρέχουν κάποιο λογισμικό, αισθητήρες ή την δυνατότητα να συνδέονται σε δίκτυα και επιτρέπει σε αυτά τα αντικείμενα να συλλέγουν και να ανταλλάσσουν δεδομένα . Οι αρχές του IoT μπορούν να εφαρμοστούν σε εφαρμογές μικρής κλίμακας, όπως τον αυτοματισμό ενός σπιτιού, ή σε μεγάλα και πολύπλοκα έργα όπως η παρακολούθηση και η διαχείριση ολόκληρου του συστήματος μεταφοράς ή ενεργειακής διανομής μιας πόλης. Ένα από τα μεγαλύτερα μειονεκτήματα που πλήττει την ανταγωνιστικότητα των προϊόντων IoT είναι η έλλειψη προτύπων, πρωτοκόλλων και πλαισίων που βοηθούν και επιταχύνουν το σχεδιασμό και την ανάπτυξη τέτοιων εφαρμογών, ο λόγος για τον οποίο συμβαίνει αυτό είναι ότι τέτοιες εφαρμογές αποτελούνται από πολλές ετερογενείς συσκευές που παράγονται από διαφορετικούς κατασκευαστές, όπου ο κάθε ένας χρησιμοποιεί το δικό του λογισμικό και εφαρμόζει τα δικά του πρότυπα, έτσι οι σχεδιαστές και οι προγραμματιστές αυτών

των συστημάτων πρέπει να βρουν αποτελεσματικούς τρόπους και λύσεις για να τις κάνουν να επικοινωνούν μεταξύ τους σε ένα κοινό δίκτυο. Όπως έχουμε προαναφέρει το IoT χαρακτηρίζεται από τα ετερογενή χαρακτηριστικά του, αυτό κάνει τη δημιουργία μιας αρχιτεκτονικής ως βάσης για την ανάπτυξη εφαρμογών πραγματικά δύσκολη. Η προσέγγιση που φαίνεται πιο ελπιδοφόρα και προτείνεται είναι η αρχιτεκτονική προσανατολισμένη στις υπηρεσίες (SOA). Η αρχιτεκτονική αυτή ορίζει τέσσερα επίπεδα τα οποία φαίνονται στην εικόνα 1.



Εικόνα 1: Προσανατολισμένη στις υπηρεσίες αρχιτεκτονική για την ανάπτυξη IoT συστημάτων.

Όπως αναφέραμε το σύστημα που σχεδιάσαμε εκμεταλλεύεται και αξιοποιεί την ανάλυση του ηλεκτροκαρδιακού σήματος. Ένα ηλεκτροκαρδιογράφημα ή ΗΚΓ καταγράφει την ηλεκτρική δραστηριότητα στην καρδιά. Το σήμα ΗΚΓ παράγεται καθώς ο καρδιακός μυς συσπάτε λόγω της αντίδρασής του στην ηλεκτρική αποπόλωσης των μυϊκών κυττάρων και καταγράφεται από ηλεκτρόδια τοποθετημένα στο σώμα του ασθενούς. Το ΗΚΓ είναι ένα από τα πιο χρήσιμα και φθηνά τεστ που χρησιμοποιούνται για την πρωταρχική διάγνωση της πάθησης του ασθενή. Ωστόσο, η σωστή ερμηνεία και διάγνωση του ΗΚΓ απαιτεί χρόνο και εμπειρία από τον θεράποντα γιατρό. Μια λύση για την αντιμετώπιση αυτού του προβλήματος είναι η αυτοματοποίηση της ανάλυσης ΗΚΓ με τη βοήθεια υπολογιστικών συσκευών. Πολλοί ερευνητές έχουν αναπτύξει αλγόριθμους μηχανικής μάθησης οι οποίοι με την κατάλληλη εκπαίδευση μπορούν πολύ γρήγορα και με μεγάλη ακρίβεια να εντοπίσουν ανωμαλίες και προβλήματα στο ΗΚΓ με αποτέλεσμα να έχουν την δυνατότητα ακόμα και να διαγνώσουν τον ασθενή χωρίς να απαιτείται φυσική παρουσία του γιατρού.



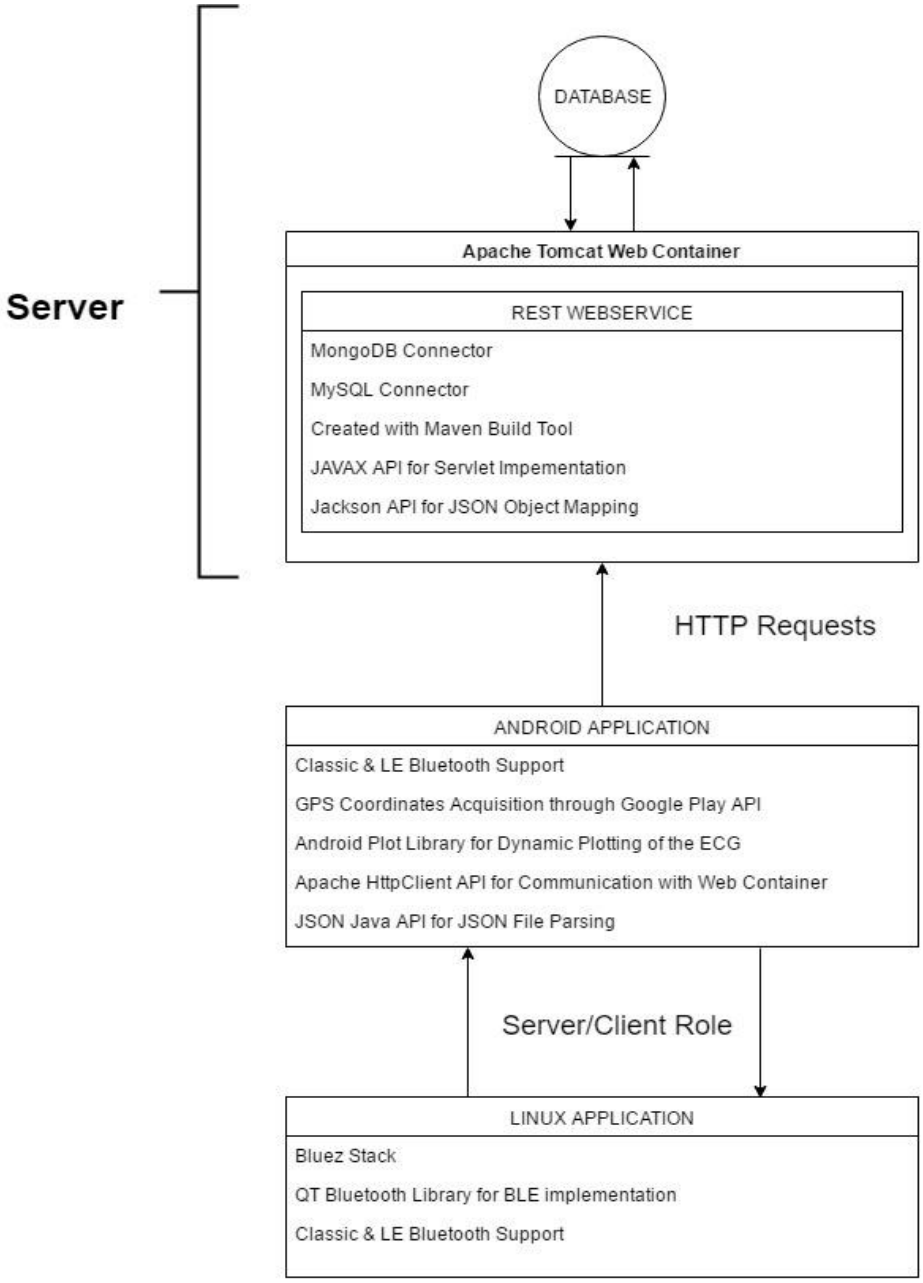
Εικόνα 2: Διάγραμμα ανάλυσης ΗΚΓ.

Bluetooth

Η ασύρματη τεχνολογία Bluetooth είναι ένα σύστημα επικοινωνίας μικρής εμβέλειας που προορίζεται να αντικαταστήσει τα καλώδια που συνδέουν φορητές ή σταθερές ηλεκτρονικές συσκευές, αναπτύχθηκε αρχικά ως ασύρματη εναλλακτική λύση στα καλώδια δεδομένων των θυρών RS-232. Η προώθηση, η ανάπτυξη και η διαχείριση του Bluetooth γίνεται από την Ομάδα Ειδικού Ενδιαφέροντος Bluetooth (SIG). Η τεχνολογία μπορεί να υλοποιηθεί σε δύο μορφές: Basic Rate (BR) με την προαιρετική δυνατότητα EDR (Enhanced Data Rate) και χαμηλής ενέργειας (LE). Σύμφωνα με τις προδιαγραφές του SIG, οι κατασκευαστές μπορούν να ενσωματώσουν την μία ή και τις δύο μορφές στις συσκευές τους, αλλά μόνο μία μπορεί να είναι ενεργή και διαθέσιμη για σύνδεση ανά πάσα στιγμή. Το σύστημα BR / EDR επικεντρώνεται περισσότερο στην ταχύτητα και τη διακίνηση των δεδομένων ενώ το σύστημα LE επικεντρώνεται στη χαμηλότερη κατανάλωση κατά την μεταφορά δεδομένων αλλά και όσο το σύστημα βρίσκεται σε αναμονή, τη μικρότερη πολυπλοκότητα και το χαμηλότερο κόστος κατασκευής από το BR/EDR. Η τεχνολογία Bluetooth χρησιμοποιείται ήδη ευρέως ως τεχνολογία κατασκευής δικτύων για την επικοινωνία μεταξύ συσκευών σε εφαρμογές IoT, καθώς έχει ορισμένα βασικά πλεονεκτήματα σε σύγκριση με άλλες τεχνολογίες. Το Bluetooth βοηθά την ανάπτυξη εφαρμογών διαχωρίζοντας τα χαμηλότερα επίπεδα του πρωτοκόλλου από τον προγραμματιστή, δημιουργεί αυτό το επίπεδο αφαίρεσης χρησιμοποιώντας τα αποκαλούμενα προφίλ. Τα προφίλ του Bluetooth καθορίζουν τις απαιτούμενες λειτουργίες και τις ρυθμίσεις κάθε στρώματος του συστήματος Bluetooth από το φυσικό (ραδιοφωνικό) επίπεδο μέχρι το L2CAP, καθορίζουν επίσης τη συμπεριφορά των εφαρμογών και τη μορφή δεδομένων. Η δομή που είναι προσανατολισμένη στις υπηρεσίες των υψηλότερων επιπέδων της τεχνολογίας Bluetooth την καθιστά ιδανική για την ενσωμάτωσή της στην (SOA) αρχιτεκτονική για την ανάπτυξη συστημάτων IoT.

Παρουσίαση του Συστήματος

Το σύστημα που αναπτύξαμε και εφαρμόσαμε σε αυτή τη διπλωματική αποτελείται από τρία διακριτά μέρη. Το πρώτο μέρος είναι η εφαρμογή που βασίζεται στην BlueZ stack του Linux και χρησιμεύει ως σημείο εισόδου του συστήματος. Σκοπός του είναι να διαβάζει την είσοδο από τα ηλεκτρόδια που τοποθετούνται στο σώμα του ασθενούς και να μεταδίδει τις μετρήσεις μέσω Bluetooth στην εφαρμογή Android. Η εφαρμογή Android αποτελεί το δεύτερο κομμάτι του συστήματος. Μπορεί να λειτουργεί σε οποιαδήποτε συσκευή (smartphone, tablet κ.λπ.) που χρησιμοποιεί το Android ως λειτουργικό σύστημα και διαθέτει μια μονάδα Bluetooth. Η εφαρμογή εκτελεί μια σειρά λειτουργιών. Αρχικά, λαμβάνει το σήμα ΗΚΓ που έχει καταγραφεί και το αποθηκεύει σε ένα αρχείο κειμένου για μεταγενέστερη χρήση, καθώς μεταδίδεται το σήμα, ο χρήστης μπορεί να δει τον παλμό σε πραγματικό χρόνο χάρη στις δυνατότητες δυναμικής σχεδίασης της εφαρμογής. Μετά την ολοκλήρωση της μετάδοσης και αφού αποθηκεύσει τα δεδομένα στη συσκευή, μπορούμε να επιλέξουμε να ανοίξουμε το αρχείο κειμένου για να εξετάσουμε τις μετρήσεις, να σχεδιάσουμε ολόκληρο το σήμα ΗΚΓ ή να το ανεβάσουμε στη βάση δεδομένων μέσω του Διαδικτύου σε μορφή JSON. Για να επικοινωνήσουμε με την απομακρυσμένη βάση δεδομένων, αναπτύξαμε το τρίτο μέρος του συστήματός μας που είναι ένα web service. Το web service υλοποιείται ως servlet που τρέχει σε οποιοδήποτε web container. Έχουμε αναπτύξει δύο κλάσεις για την σύνδεση με τις βάσεις δεδομένων, μια για την MongoDB και μία για τη βάση δεδομένων MySQL. Το web service λαμβάνει αιτήσεις HTTP που περιέχουν αρχεία JSON, μετά την ανάλυση και την αξιολόγηση τους συνδέεται με τη βάση δεδομένων για την αποθήκευση των δεδομένων στην επιθυμητή μορφή. Ένα σχήμα που περιγράφει ολόκληρο το σύστημα με τις αλληλεπιδράσεις μεταξύ των τμημάτων καθώς και μερικές βασικές δομές, βιβλιοθήκες και εργαλεία που χρησιμοποιήθηκαν για την ανάπτυξη κάθε μεμονωμένου τμήματος δίνετε στην εικόνα 3. Για τον πρωταρχικό σχεδιασμό του συστήματος και κατά την ανάπτυξή του, λάβαμε σοβαρά υπόψη τα πλεονεκτήματα ή τα μειονεκτήματα κάθε τεχνολογίας και τις βασικές αρχές σχεδιασμού που περιγράψαμε παραπάνω για την παραγωγή ενός εύρωστου συστήματος που μπορεί να ικανοποιήσει τις ανάγκες και να ξεπεράσει τις δυσκολίες στην υλοποίηση και λειτουργία σε πραγματικά ιατρικά σενάρια χρήσης.



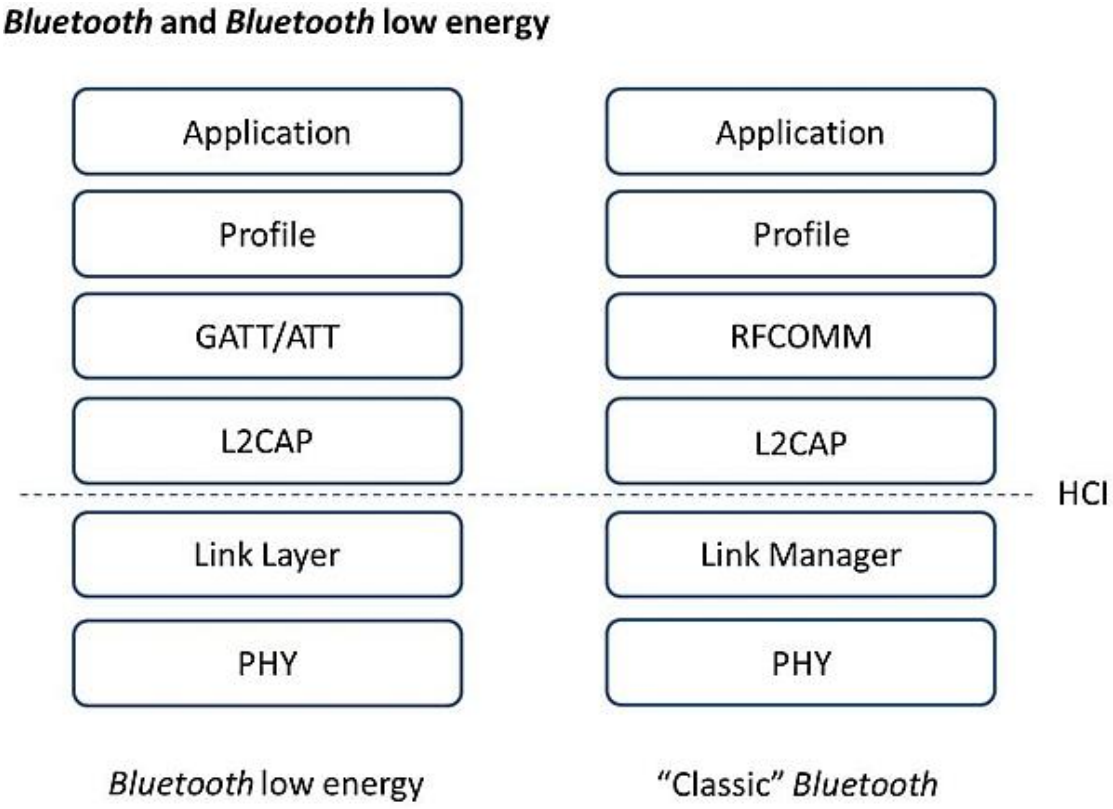
Εικόνα 3: Διάγραμμα του συνολικού συστήματος.

Bluetooth Εφαρμογή παρακολούθησης ΗΚΓ και μεταφοράς δεδομένων βασισμένη στο Linux

Η εφαρμογή αυτή αποτελεί το πρώτο κομμάτι του συστήματος και έχει την δυνατότητα να συνδέεται με Classic ή LE Bluetooth. Για την μεταφορά με το κλασσικό Bluetooth χρησιμοποιήσαμε την BlueZ stack του Linux για την ανάπτυξη της εφαρμογής σε γλώσσα C. Η BlueZ είναι η επίσημη στοίβα του Bluetooth πρωτοκόλλου για το λειτουργικό σύστημα Linux και είναι ένα από τα βασικά υποσυστήματα του πυρήνα κάθε διανομής. Για την ανάπτυξη της εφαρμογής χρησιμοποιήσαμε την έκδοση 5.37

της στοίβας. Η στοίβα BlueZ υλοποιεί υψηλότερα πρωτόκολλα μεταφοράς δεδομένων του Bluetooth (RFCOMM, OBEX, A2DP κ.λπ.), εγκαθιστά και διατηρεί τους διακομιστές SDP ή GATT όπου αποθηκεύονται πληροφορίες σχετικά με τις υπηρεσίες και τα προφίλ που υποστηρίζονται από τη συσκευή. Κατά τον σχεδιασμό της εφαρμογής αποφασίσαμε να χρησιμοποιήσουμε το προφίλ SPP έτσι ώστε οι δύο συσκευές να επικοινωνούν μέσω του πρωτοκόλλου RFCOMM, δηλαδή τα πακέτα δεδομένων να μεταδίδονται με τη σειρά που αποστέλλονται στον ελεγκτή του Bluetooth, ακόμη και αν απαιτείται αναμετάδοση. Η επικοινωνία μέσω του κλασσικού Bluetooth γίνεται με sockets. Αρχικά η εφαρμογή μας ψάχνει για άλλες συσκευές και συνδέεται με τον SDP server κάθε συσκευής για να δει αν υποστηρίζει το SPP προφίλ. Όταν βρει μια κατάλληλη συσκευή ανοίγει ένα socket και στέλνει τα δεδομένα κάνοντας χρήση της συνάρτησης write() της στοίβας. Αφού τελειώσει η μετάδοση, η σύνδεση κλείνει καλώντας την συνάρτηση close() με όρισμα το socket που είχε ανοίξει με την συσκευή.

Για την σύνδεση μέσω Bluetooth Low Energy χρησιμοποιήσαμε την βιβλιοθήκη Bluetooth της Qt η οποία είναι γραμμένη σε C++. Η ανάπτυξη εφαρμογών Bluetooth LE ακολουθεί ένα διαφορετικό μοντέλο προγραμματισμού από το Classic Bluetooth. Στο BLE, υπάρχει μόνο ένα πρωτόκολλο επικοινωνίας που ονομάζεται πρωτόκολλο Attribute (ATT), έτσι οι συσκευές επικοινωνούν με βάση τα αποκαλούμενα attributes. Τα attributes είναι εγγραφές σε έναν μεγάλο πίνακα που δημιουργείται και αποθηκεύεται σε κάθε συσκευή χαμηλής ενέργειας Bluetooth, μπορούν να αναγνωριστούν και να προσπελαθούν με μοναδικά για το καθένα UUIDs. Για να γίνει πιο εύκολο σε μια εφαρμογή να χρησιμοποιεί το Bluetooth Low Energy η στοίβα προσθέτει ένα επιπλέον επίπεδο πάνω από το πρωτόκολλο ATT, το επίπεδο αυτό είναι το προφίλ Generic Attribute (GATT). Το προφίλ αυτό χρησιμοποιεί τα αποκαλούμενα characteristics ως αντικείμενα που υλοποιούν την βασική δομή δεδομένων του πρωτοκόλλου και περιλαμβάνουν όχι μόνο τα δεδομένα που πρέπει να ανταλλάσσονται αλλά και πληροφορίες για αυτά τα δεδομένα. Τα characteristics είναι αντικείμενα υψηλότερου επιπέδου που μεταφράζονται σε ομάδες πολλαπλών attribute χαμηλότερου επιπέδου. Η εφαρμογή μας διαφημίζει τις υπηρεσίες και τα προφίλ που υποστηρίζει με την μέθοδο startAdvertising(). Όταν μια συσκευή η οποία ψάχνει την υπηρεσία που προσφέρει η εφαρμογή μας την εντοπίσει δημιουργεί αυτόματα μια σύνδεση και τα δεδομένα του ΗΚΓ στέλνονται με την μέθοδο writeCharacteristic(). Όταν η άλλη συσκευή λάβει τα δεδομένα που χρειάζεται αποσυνδέεται και η εφαρμογή μας συνεχίζει να διαφημίζει τις υπηρεσίες της ώστε να μπορούν άλλες συσκευές να συνδεθούν.



Εικόνα 4: Σύγκριση της ιεραρχίας της στοίβας των δύο Bluetooth.

Εφαρμογή Android

Το λειτουργικό σύστημα Android έχει την δικιά του στοίβα για το Bluetooth που ονομάζεται Bluedroid. Στην εφαρμογή που αναπτύξαμε ο χρήστης επιλέγει αν θα συνδεθεί με κλασσικό ή με χαμηλής κατανάλωσης Bluetooth. Για να συνδεθούμε με το κλασσικό Bluetooth χρησιμοποιούμε την μέθοδο που μας παρέχεται από την στοίβα listenUsingInsecureRfcommWithServiceRecord() για να εκκινήσουμε τον server ώστε να αρχίσουμε να δεχόμαστε συνδέσεις. Και εδώ η επικοινωνία γίνεται μέσω socket οπότε καλούμε τις μεθόδους read() και close() για να διαβάσουμε δεδομένα ή να κλείσουμε την σύνδεση αντίστοιχα. Ο προγραμματισμός της σύνδεσης LE είναι λίγο πιο περίπλοκος. Κάθε λειτουργία BLE προκαλεί μια επανάκληση, η οποία εκτελείται σε ένα ξεχωριστό νήμα που τρέχει στο υπόβαθρο. Οι επανακλήσεις μπορούν να συμβούν οποτεδήποτε καθώς η εφαρμογή εκτελείται και αυτό έπρεπε να το λάβουμε υπόψη κατά την ανάπτυξη της εφαρμογής. Για να εκκινήσουμε μια σύνδεση με την άλλη συσκευή δημιουργούμε ένα αντικείμενο της κλάσης BluetoothGatt το οποίο αναλαμβάνει την επικοινωνία με τον GATT server της άλλης συσκευής. Για να διαβάσουμε τα δεδομένα χρησιμοποιούμε την μέθοδο readCharacteristic() η οποία αν είναι επιτυχείς προκαλεί μια επανάκληση την οποία πρέπει να διαχειριστούμε. Όταν ολοκληρώσουμε την μεταφορά των δεδομένων κλείνουμε την σύνδεση καλώντας την μέθοδο close() στο αντικείμενο BluetoothGatt που είχαμε δημιουργήσει. Επίσης, η εφαρμογή μας έχει δυνατότητες για στατική αλλά και για δυναμική σχεδίαση του ΗΚΓ

σήματος. Για να το πετύχουμε αυτό κάναμε χρήση της βιβλιοθήκης Android Plot που έχει αναπτυχθεί για αυτό τον σκοπό.

Τέλος, η εφαρμογή συνθέτει HTTP πακέτα τα οποία περιέχουν τις πληροφορίες για τον ασθενή μαζί με τα δεδομένα του ΗΚΓ σε μορφή JSON αρχείου και τα αποστέλλει μέσω διαδικτύου στο web service που έχουμε αναπτύξει.

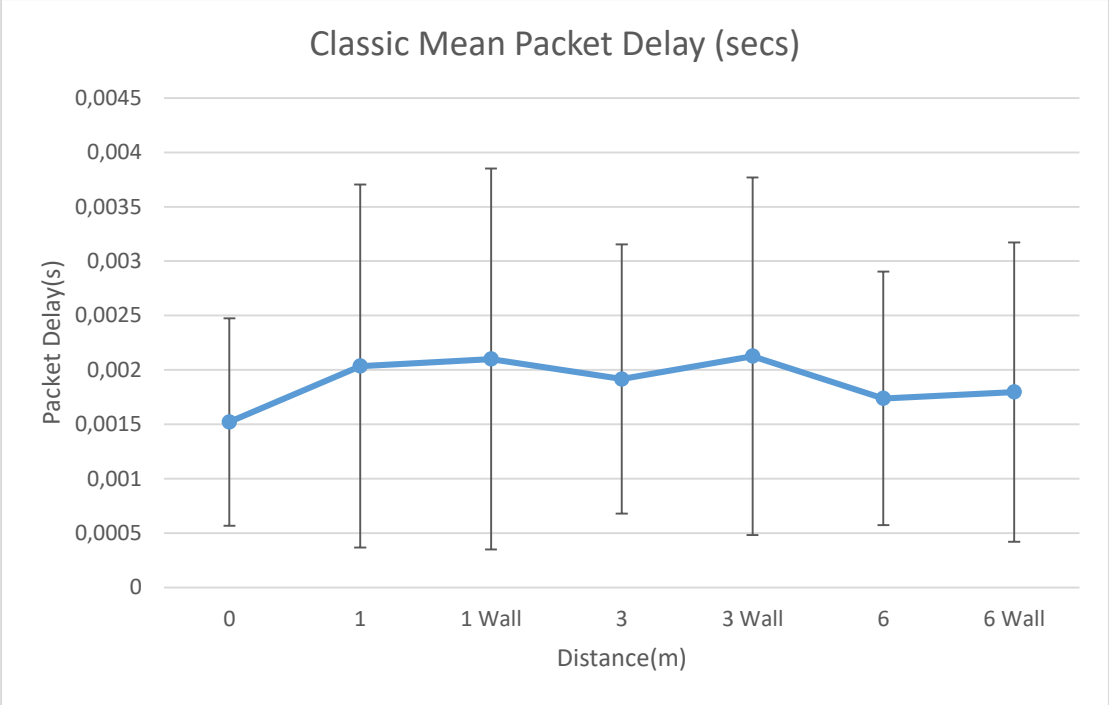
RESTful Web Service

Ένα web service είναι μια υπηρεσία που προσφέρεται από μια ηλεκτρονική συσκευή σε άλλες συσκευές που επικοινωνούν μεταξύ τους μέσω του World Wide Web. Η συνηθέστερη χρήση του είναι η παροχή ενός αντικειμενοστραφούς διαδικτυακού περιβάλλοντος σε ένα διακομιστή βάσης δεδομένων και συχνά συνδυάζεται με μια εφαρμογή για κινητά που παρέχει μια γραφική διεπαφή στον χρήστη. Η RESTful είναι μια αρχιτεκτονική που καθορίζει τρόπους για τη δημιουργία web service που μπορούν να χειριστούν δυναμικά και ετερογενή δεδομένα χωρίς αυστηρούς και καλά καθορισμένους κανόνες επικοινωνίας ή πρωτόκολλα. Τα συστήματα REST αποσκοπούν στη γρήγορη απόδοση, την αξιοπιστία και την ικανότητα επαναχρησιμοποίησης στοιχείων τα οποία μπορούν να μεταβάλλονται και να ενημερώνονται χωρίς να επηρεάζουν το σύστημα στο σύνολό του.

Δημιουργήσαμε το web service του συστήματος χρησιμοποιώντας το Java Servlet API. Ένα Java servlet είναι ένα πρόγραμμα Java που επεκτείνει τις δυνατότητες ενός διακομιστή. Μπορούμε να κάνουμε override τις μεθόδους doGet (), doPost (), doPut (), doDelete () που παρέχονται από το Servlet API για να πιάσουμε τα HTTP αιτήματα και να τα διαχειριστούμε. Αυτές οι μέθοδοι εκτελούνται κατά τη διάρκεια λειτουργείας της υπηρεσίας, οπότε όταν το web service λάβει ένα αίτημα, δημιουργείται ένα νέο νήμα για την εκτέλεση της αντίστοιχης μεθόδου. Τα web service διανέμονται και εγκαθίστανται σε μορφή συμπιεσμένων πακέτων WAR. Για την αλληλεπίδραση και την σύνδεση με τις βάσεις δεδομένων χρησιμοποιήσαμε το πρόγραμμα οδήγησης της MongoDB και τα API του προγράμματος οδήγησης JDBC για την MySQL. Για κάθε παραληφθείσα αίτηση, το web service πρέπει να αποσπάσει τα δεδομένα σε JSON μορφή και να τα χειριστεί προγραμματιστικά. Για να μπορέσουμε να το κάνουμε αυτό, καθορίσαμε μια κλάση JsonObject για να χαρτογραφήσουμε τα δεδομένα σε αυτήν. Για την ανάλυση και χαρτογράφηση του αρχείου χρησιμοποιήσαμε την βιβλιοθήκη Jackson.

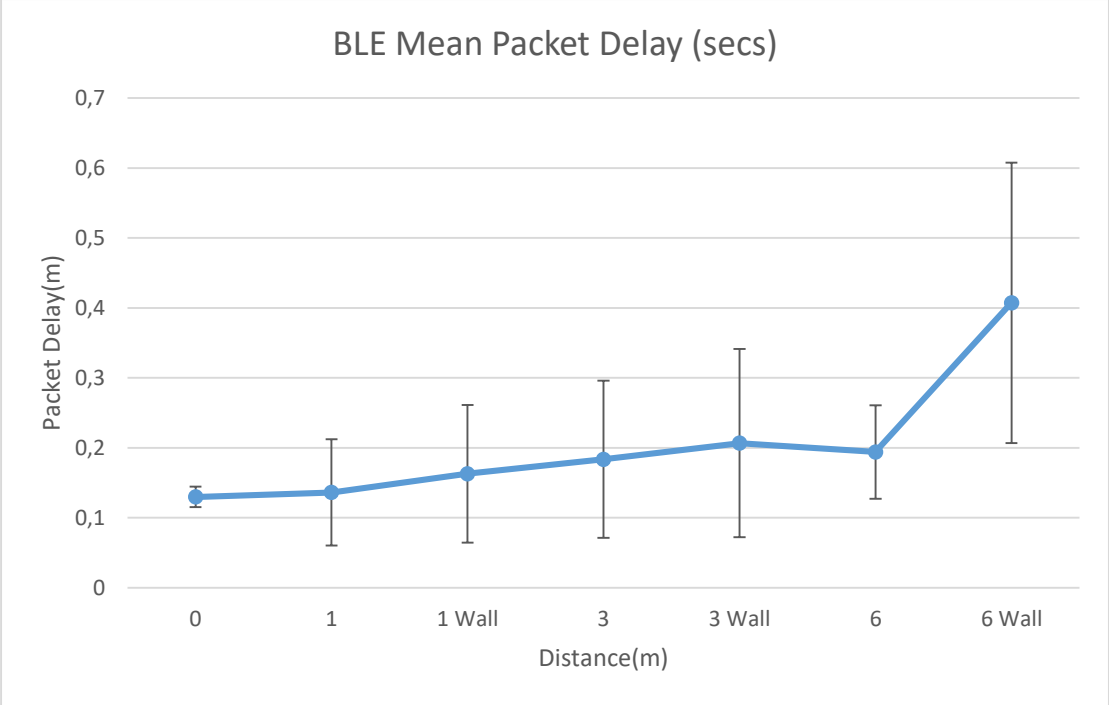
Πειραματικά Αποτελέσματα

Αρχικά εξετάσαμε τον κλασικό σύνδεσμο Bluetooth. Μετρήσαμε την καθυστέρηση του πακέτου υπολογίζοντας τη χρονική διαφορά μεταξύ δύο διαδοχικών ληφθέντων πακέτων.



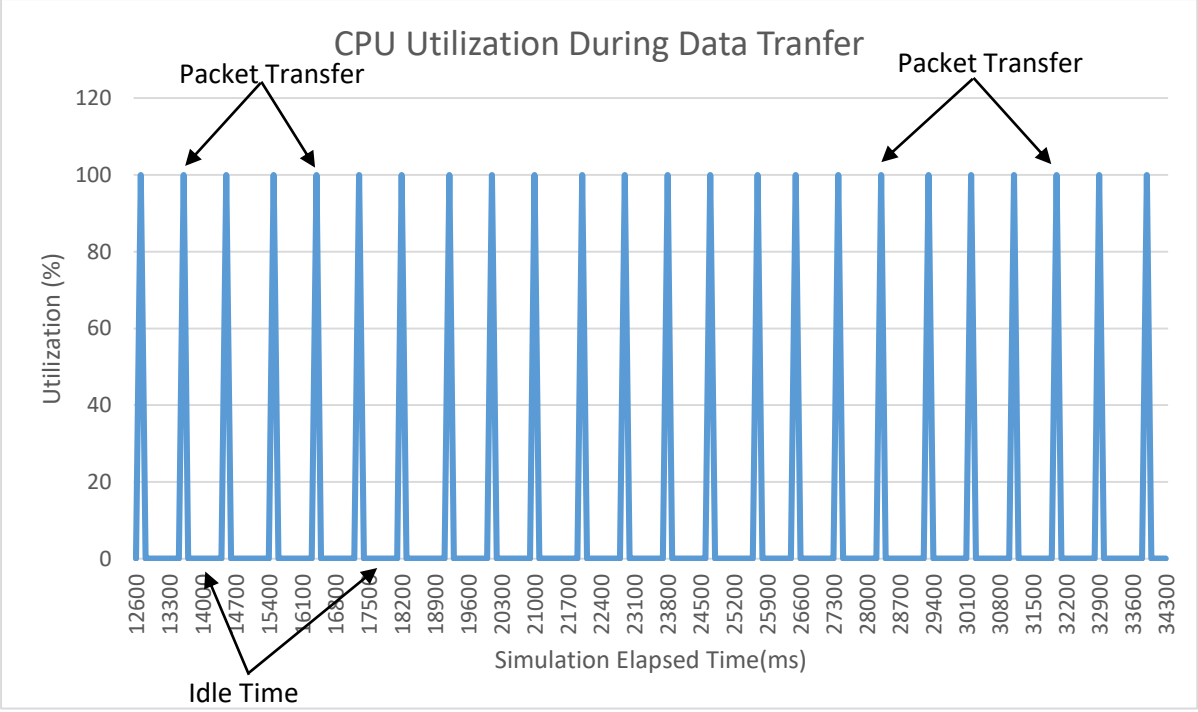
Εικόνα 5: Μέση καθυστέρηση των πακέτων με σύνδεση Classic Bluetooth.

Ακολουθήσαμε μια ελαφρώς διαφορετική προσέγγιση για τον υπολογισμό της καθυστέρησης του πακέτου για την επικοινωνία χαμηλής ενέργειας Bluetooth. Χρησιμοποιήσαμε τις προδιαγραφές του προφίλ GATT, έτσι μετρήσαμε την καθυστέρηση μεταξύ των αιτήσεων ανάγνωσης GATT του πελάτη και των απαντήσεων ανάγνωσης GATT του διακομιστή για τον υπολογισμό της καθυστέρησης.

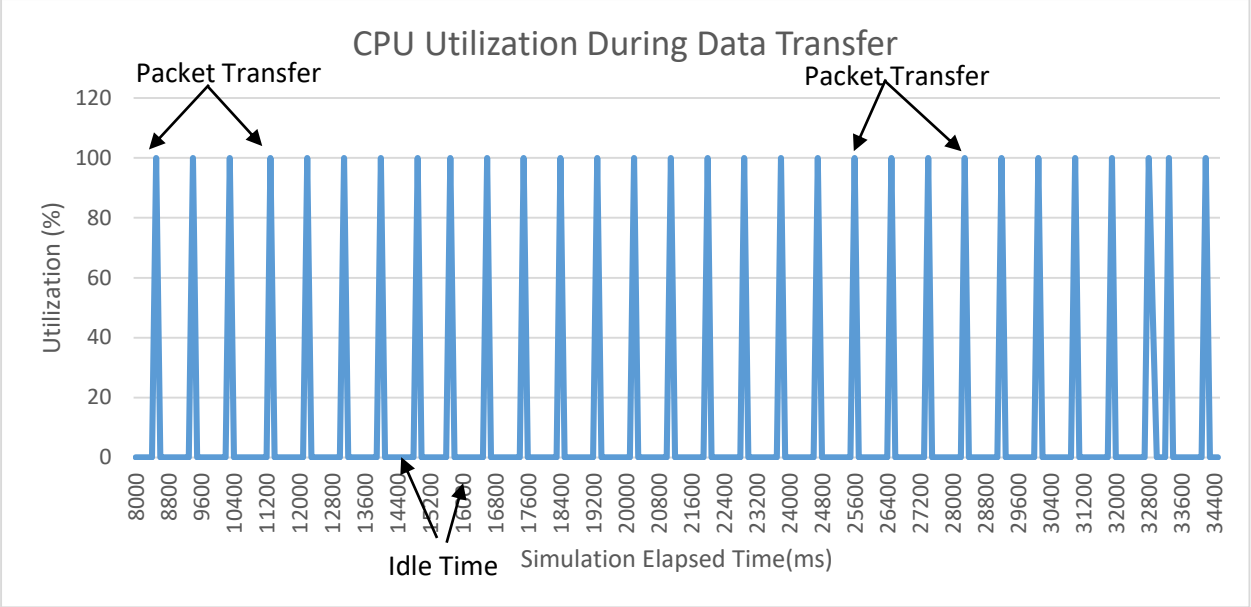


Εικόνα 6: Μέση καθυστέρηση των πακέτων με σύνδεση LE Bluetooth.

Θέλαμε να υλοποιήσουμε την Bluetooth εφαρμογή παρακολούθησης ΗΚΓ σε μια πραγματική IoT πλατφόρμα ώστε να βγάλουμε συμπεράσματα για τις λειτουργικές της απαιτήσεις σε επεξεργαστική ισχύ. Στο πλαίσιο αυτό επιλέξαμε να υλοποιήσουμε την εφαρμογή στην πλατφόρμα Raspberry Pi 3 model B.

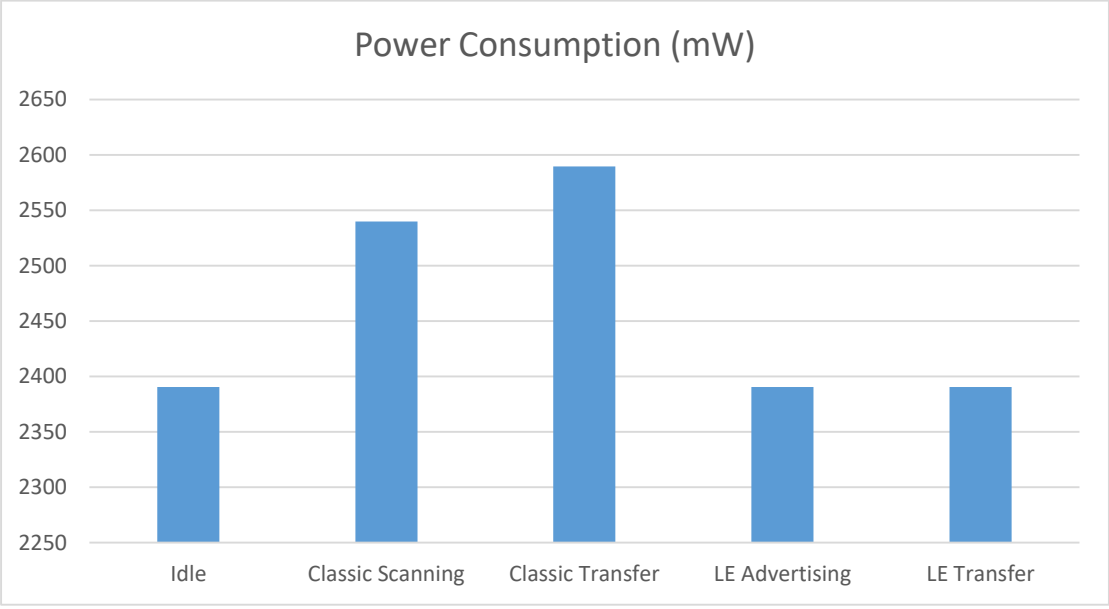


Εικόνα 6: Αξιοποίηση CPU κατά την μεταφορά δεδομένων LE Bluetooth.



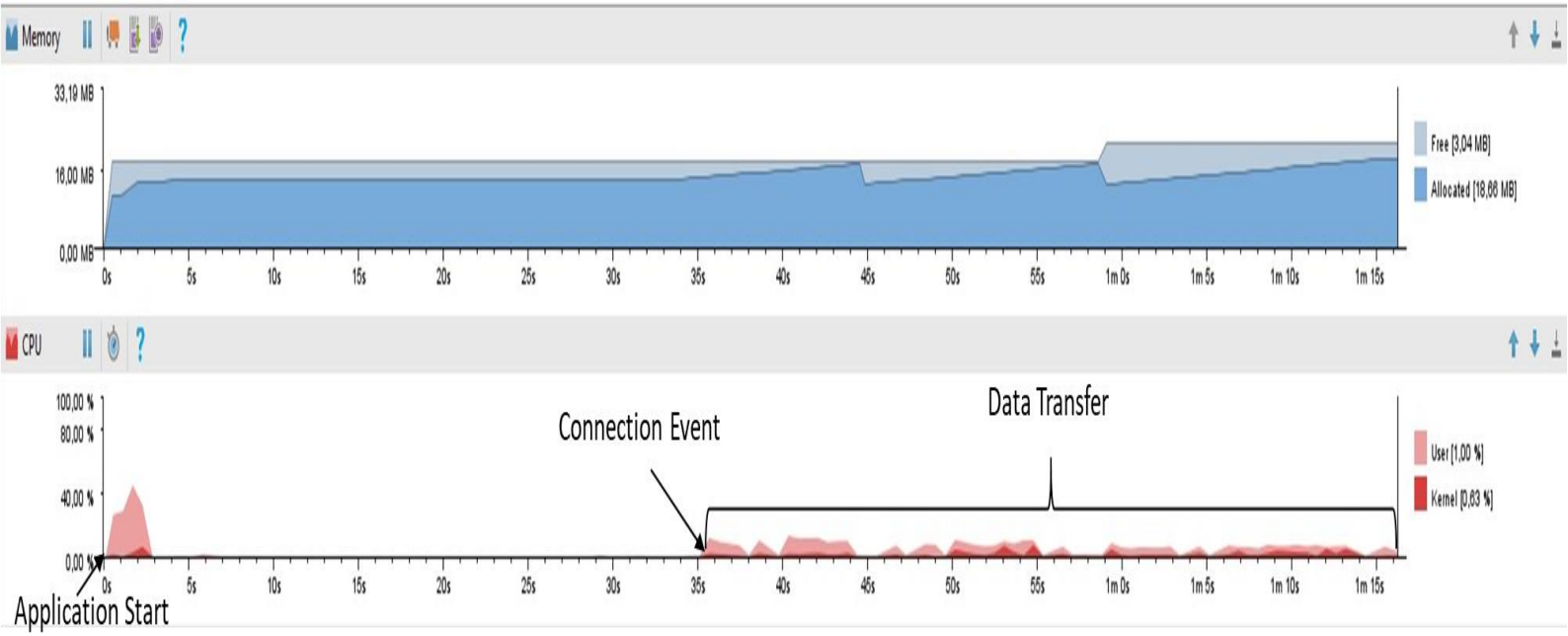
Εικόνα 7: Αξιοποίηση CPU κατά την μεταφορά δεδομένων Classic Bluetooth.

Μετρήσαμε επίσης το ρεύμα που καταναλώνεται από την πλατφόρμα μέσω της θύρας USB χρησιμοποιώντας έναν κατάλληλο προσαρμογέα USB για να λάβουμε μια εκτίμηση για την κατανάλωση της εφαρμογής μας. Η κατανάλωση ρεύματος μπορεί να υπολογιστεί με τον τύπο P = V * I.

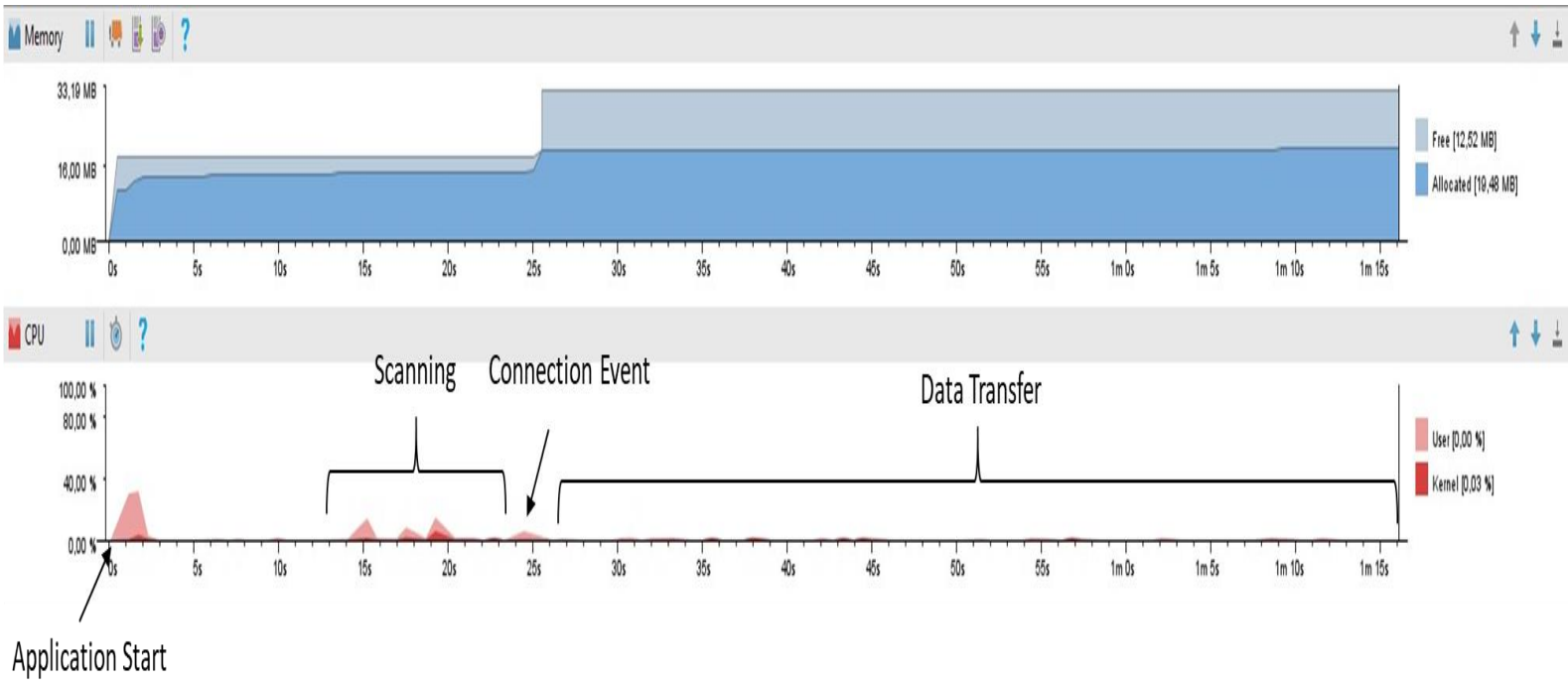


Εικόνα 8: Κατανάλωση ισχύος.

Το Android Studio προσφέρει μια σειρά από εργαλεία για την παρακολούθηση των συνδεδεμένων συσκευών, τη χρήση της CPU και της GPU, τη διαχείριση μνήμης και τη χρήση του δικτύου ενώ εκτελείται μια εφαρμογή. Χρησιμοποιήσαμε αυτά τα εργαλεία για να παρακολουθήσουμε το ποσό των πόρων που απαιτούνται από την εφαρμογή μας, ώστε να μπορούμε να εξάγουμε χρήσιμα συμπεράσματα σχετικά με τη λειτουργία της.

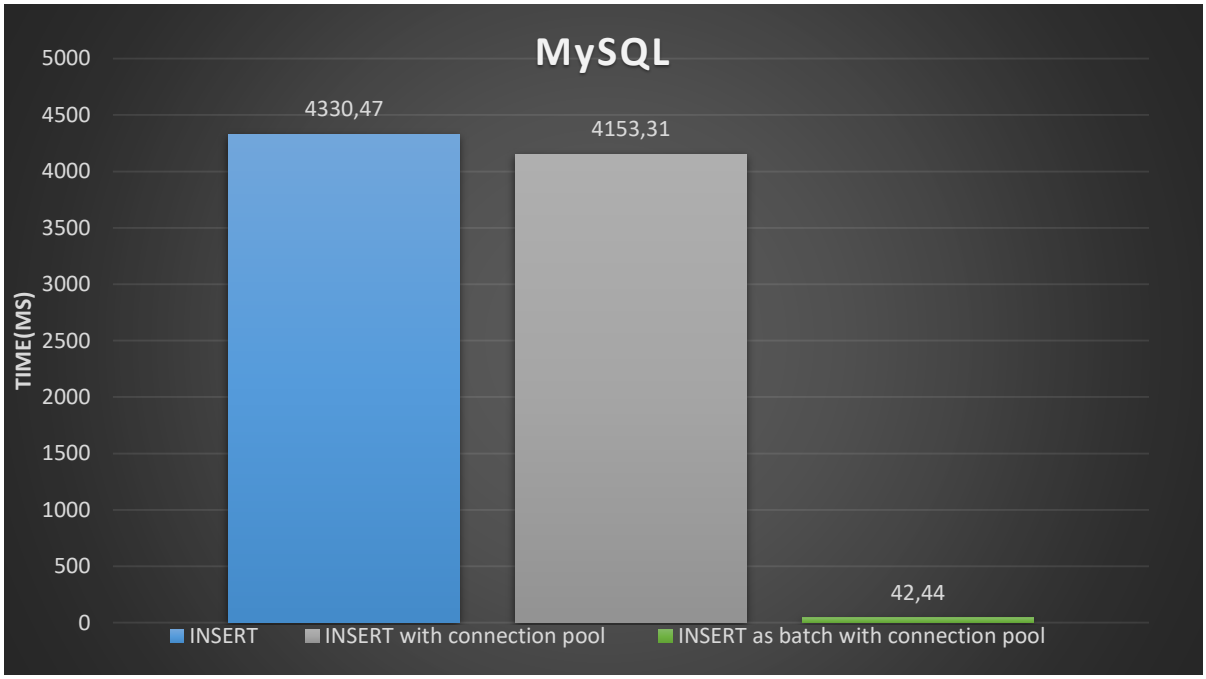


Εικόνα 9: Χρήση της CPU και Memory για σύνδεση Classic Bluetooth.

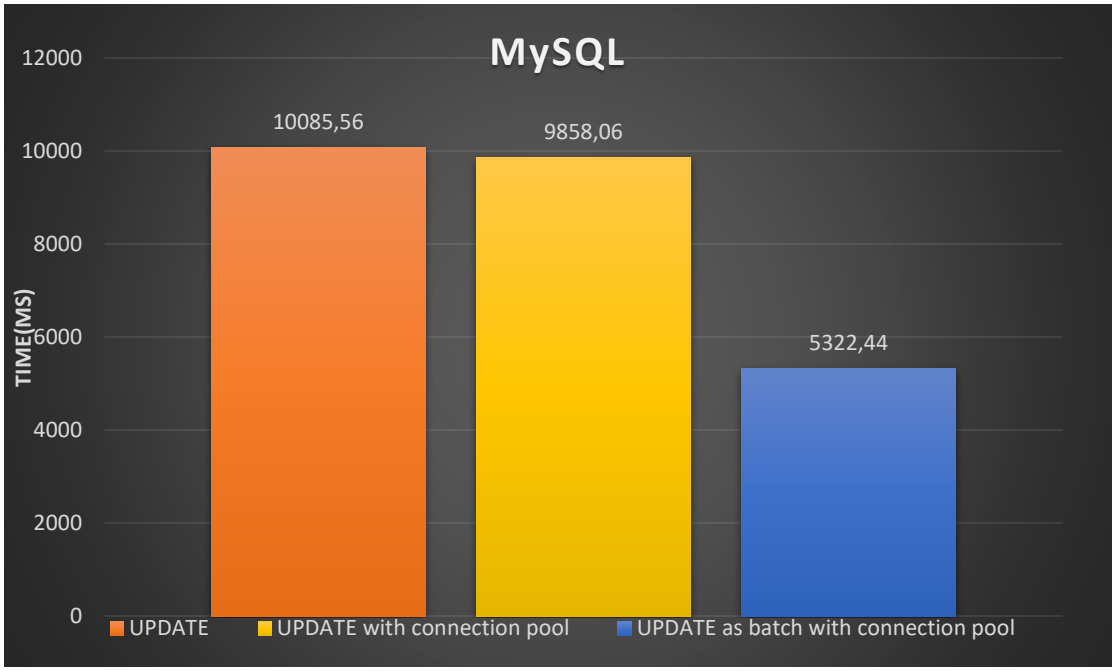


Εικόνα 10: Χρήση της CPU και Memory για σύνδεση LE Bluetooth.

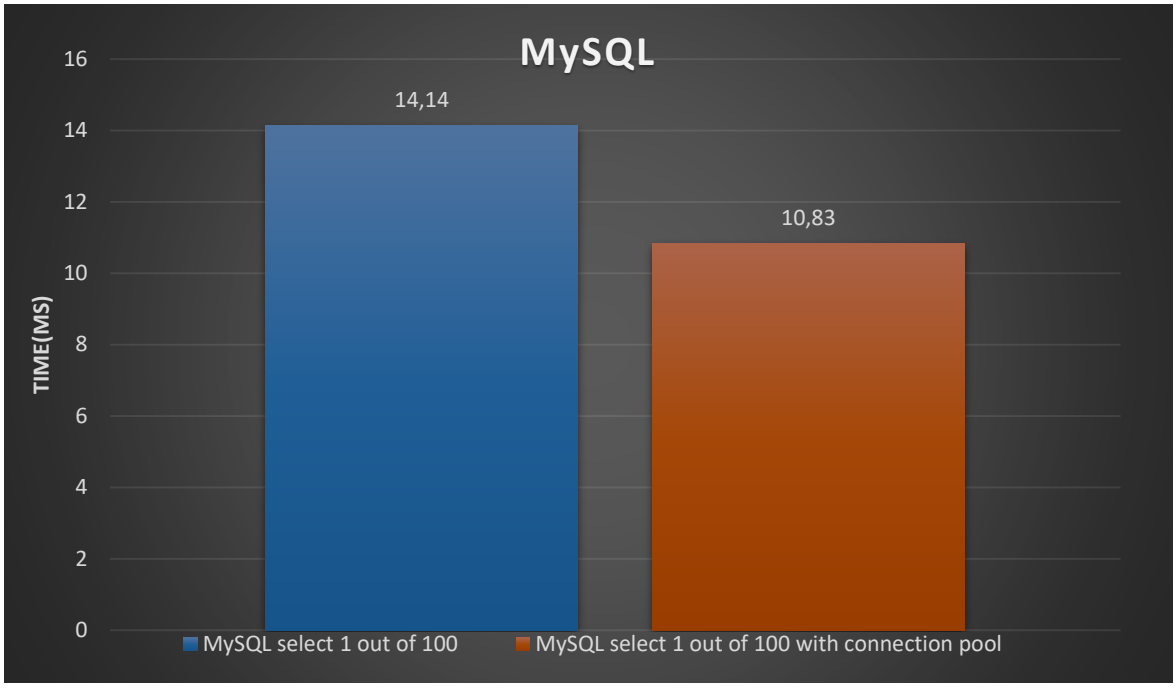
Τέλος τεστάραμε το web service του συστήματος τόσο με την MySQL όσο και με την MongoDB για να συγκρίνουμε την απόδοσή τους και πόσο καλά συνεργάζεται η κάθε μια με το υπόλοιπο σύστημα. Στα πειράματα μετρήσαμε το χρονικό διάστημα σε χιλιοστά του δευτερολέπτου που είναι απαραίτητο για την εισαγωγή ή την ενημέρωση μιας εγγραφής ενός ασθενούς στη βάση δεδομένων.



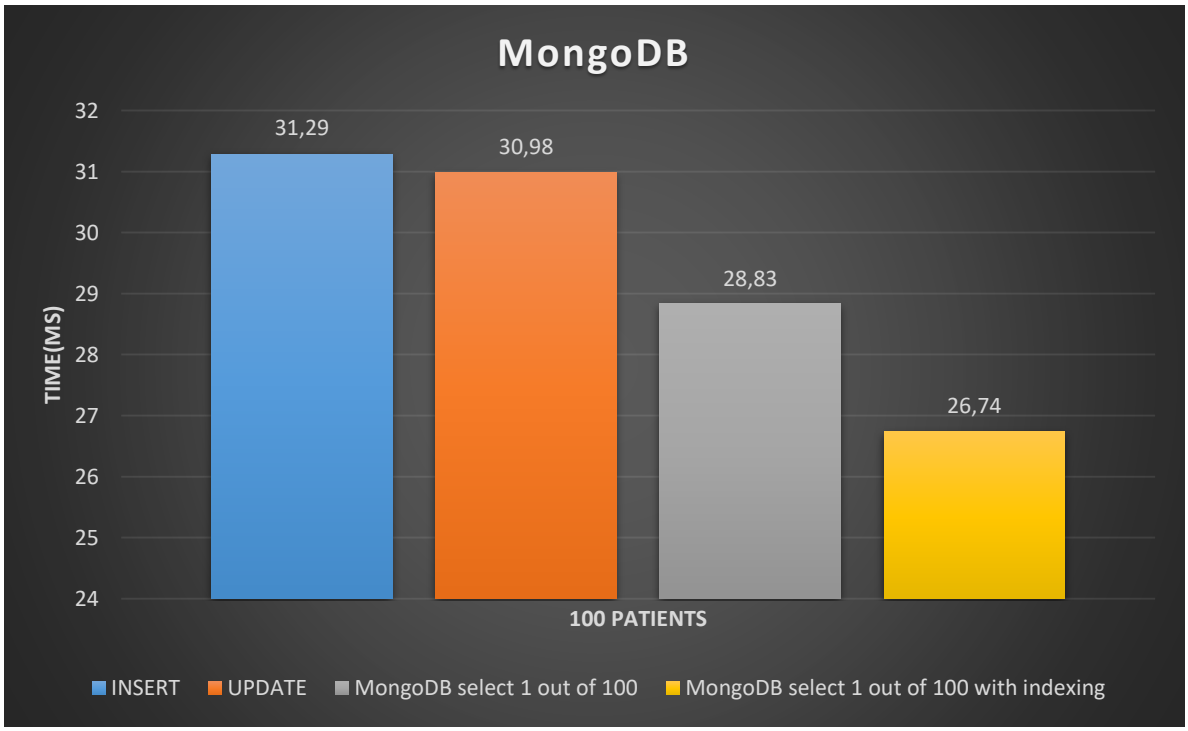
Εικόνα 11: Πειραματικά αποτελέσματα για Insert στην MySQL.



Εικόνα 12: Πειραματικά αποτελέσματα για Update στην MySQL.



Εικόνα 13: Πειραματικά αποτελέσματα για Select στην MySQL.



Εικόνα 14: Πειραματικά αποτελέσματα για την MongoDB.

Συμπεράσματα και προτάσεις για μελλοντική έρευνα

Σε αυτή τη διπλωματική σχεδιάσαμε, αναπτύξαμε και υλοποιήσαμε σε μια πλατφόρμα IoT ένα πλήρως λειτουργικό σύστημα παρακολούθησης ΗΚΓ. Με τη διεξαγωγή δοκιμών και την απόκτηση πειραματικών δεδομένων μπορέσαμε να αξιολογήσουμε πλήρως το σύστημα. Το Classic Bluetooth συνιστάται για σταθερές συσκευές με συνεχή παροχή ρεύματος, αφού προκάλεσε αύξηση της κατανάλωσης ενέργειας κατά 10% περίπου στο Raspberry Pi ή για περιπτώσεις χρήσης που συνεπάγονται συνεχή παρακολούθηση πολλαπλών ασθενών ταυτόχρονα σε μεγαλύτερες αποστάσεις. Το Bluetooth Low Energy είναι πιο κατάλληλο για μια φορητή συσκευή με περιορισμένους ενεργειακούς πόρους που λειτουργεί πολύ κοντά στη συσκευή Android. Συγκρίνοντας τις βάσεις δεδομένων επιλέξαμε την MongoDB ως καλύτερη λύση επειδή ήταν πολύ γρηγορότερη στην αποθήκευση αρχείων χωρίς περαιτέρω βελτιώσεις στον κώδικα. Αντιθέτως, έπρεπε να αφιερώσουμε πολύ χρόνο κατά την διάρκεια της ανάπτυξης του web service για να βελτιώσουμε την απόδοση της MySQL και ακόμη και τότε οι χρόνοι εισαγωγής των δεδομένων ήταν πιο αργοί σε σύγκριση με αυτούς της MongoDB.

Παρόλο που οι NoSQL βάσεις δεδομένων κλιμακώνουν πολύ εύκολα και μπορούν να χειριστούν δυναμικά δεδομένα, η αρχιτεκτονική τους δεν επιτρέπει την παρουσίαση των σχέσεων μεταξύ των δεδομένων. Αυτές οι βάσεις δεδομένων είναι κατάλληλες για να λειτουργούν παράλληλα με web services για γρήγορη εξυπηρέτηση πελατών, καθώς έχουν σχεδιαστεί για να λειτουργούν αποτελεσματικά σε clusters χωρίς την ανάγκη ακριβού υλικού. Προτείνουμε μια επέκταση στο σύστημα που καλύπτει την ανάγκη δημιουργίας σχέσεων μεταξύ των δεδομένων. Το εκτεταμένο σύστημα θα έχει βάσεις δεδομένων NoSQL που θα χρησιμεύουν ως caches για δεδομένα που προσπελάζονται πολύ συχνά και που μπορούν να βρίσκονται και να συντηρούνται από μεμονωμένα νοσοκομεία. Στο ανώτατο επίπεδο του συστήματος θα υπάρχει μια βάση δεδομένων SQL με ένα κατάλληλο σχήμα που θα μπορεί να συσχετίζει τα δεδομένα που έχουν αποκτηθεί από τα μεμονωμένα νοσοκομεία. Τα δύο επίπεδα θα συνδέονται με ένα σύστημα διαχείρισης που θα μεταφέρει αποτελεσματικά τα δεδομένα μεταξύ τους. Το προτεινόμενο σύστημα μπορεί να λειτουργεί παράλληλα με το σύστημα που παρουσιάσαμε σε αυτή τη διπλωματική χωρίς να χρειάζεται αλλαγές ή επανασχεδιασμό των τμημάτων του.

Το λειτουργικό σύστημα Android δεν έχει τρόπο διαχείρισης και χρονοπρογραμματισμού εντολών που αποστέλλονται στον ελεγκτή BLE από μια εφαρμογή. Η ανάπτυξη ενός τέτοιου διαχειριστή-χρονοπρογραμματιστή μπορεί να ενισχύσει τις IoT δυνατότητες της εφαρμογής μας. Επίσης, θα είναι πολύ χρήσιμο για κάθε άλλη εφαρμογή που χρησιμοποιεί το BLE. Επιπλέον, το UI της εφαρμογής μπορεί να βελτιωθεί περαιτέρω για να γίνει πιο φιλικό προς το χρήστη.

Contents

List of Figures

CHAPTER 1

Introduction

The purpose of this diploma thesis is the design, development and implementation of an ECG analysis and visualization system with IoT capabilities that can be used in real life medical applications and scenarios. In this scope, technologies and frameworks that are used widely in real life applications and are adopted by the industry were chosen. We present the system we design and evaluate its individual parts in the next chapters.

In chapter 1, an introduction is made, covering the three broader and more general technological and research fields that are connected with the development of our system. Chapter 2 is an overview and summary of the technologies, architectures and practices that were used during the development and implementation. Its purpose is to make the reader familiar with some basic concepts so he can understand with greater ease the design choices we made. In Chapter 3 we demonstrate the complete system that we aim to develop along with the designing of its individual components. Chapter 4 is about system evaluation and profiling, furthermore we analyze the reasons behind every design choice we made using data we acquired from testing and experimentation with the system. In the final chapter we summarize our conclusions and we propose directions and ideas for further development and improvement of the system we presented in this thesis.

1.1 Wearable Medical Devices

One of the greatest challenges we have to overcome nowadays and in the near future is the dramatically increase of healthcare cost in a global scale. Many countries, companies, researchers and engineers are working in finding ways to successfully address the problem. One of the key factors that cause the rise of the healthcare cost is the need for monitoring the patients both during their treatment and after. Another way to further reduce the cost is to shift public health policies away from reactive models of

healthcare to preventative ones, that means we need to find ways of monitoring individuals before they get sick and have the need to visit their doctor or get admitted to the hospital. Fortunately, with the advancement of technology we are able to develop and implement devices that can successfully and accurately monitor most of human bio signals by using embedded sensors, connect to the Internet through Wi-Fi or mobile communication networks (3G/4G), can operate for months because of their low energy consumption, are small and light enough to be attached to a person without causing discomfort and they have low production cost. These wearable medical devices seem to be the solution to stop healthcare cost from rising. They still need more improvement and more time to overcome the design challenges that arise but with proper funding and research from public sector, private companies and universities they can change the way we practice medicine in the future and also create new markets and great investment opportunities.

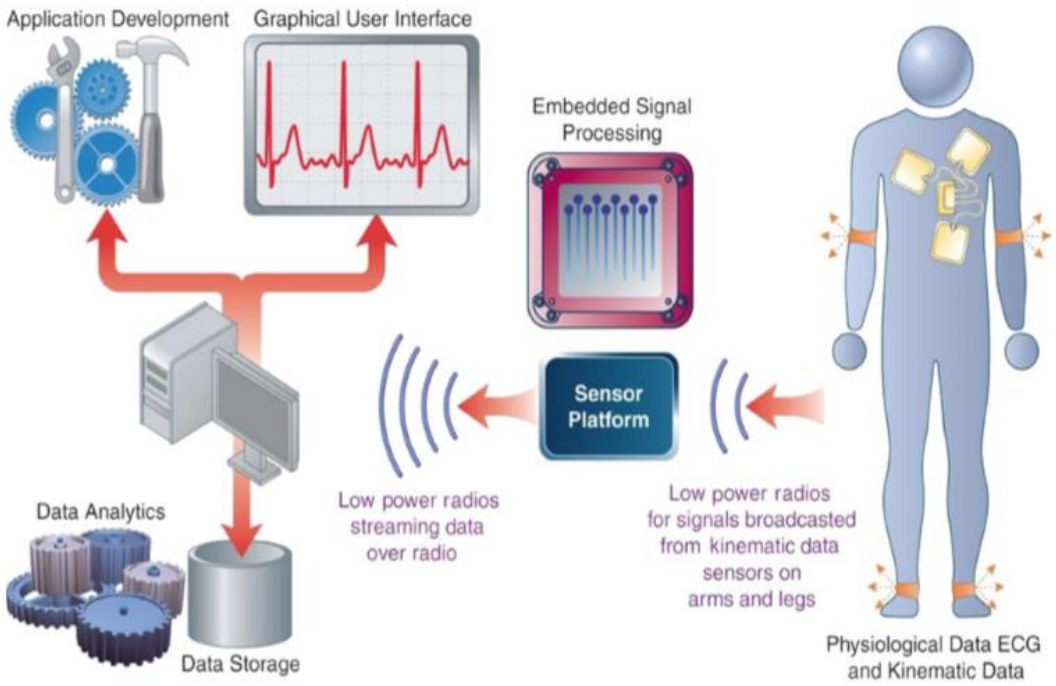


Figure 1.1: Example of a wearable medical device functionality [5].

1.2 Internet of Things

Generally speaking, IoT is a concept that refers to the networked interconnection of everyday objects such as physical devices, vehicles, buildings, and other items embedded with electronics, software, sensors, actuators, and network connectivity that enable these objects to collect and exchange data. IoT principles can be applied from small scale applications like home automation to large and complex projects like monitoring and management of a city's whole transportation or energy distribution system. Advances in IoT development and implementation have created a tremendous number of novel applications and many companies have rushed to get involved by designing and distributing their own IoT devices and platforms or embedding them into existing products. In addition, many companies have integrated IoT systems into their manufacturing process because of their ability to enable rapid manufacturing of new products, dynamic response to product demands, and real-time optimization of manufacturing production and supply chain networks, by networking machinery, sensors and control systems together. Also, the need to process great amounts of data in such fast paced environments has caused great investments and advancements in Big Data analytics field.

Despite its rapid growth and expansion there are some difficulties and challenges that slow its adaptation, by causing numerus practical problems to IoT application development and implementation, that need to be addressed. One of the greatest drawbacks that decrease the competiveness of IoT products is the lack of standards, protocols and frameworks that assist and accelerate the design and development of IoT applications, the reason for this is that such applications consists of many heterogeneous devices which are produced by different vendors, may run unique software and implement their own standards so developers have to find efficient ways and solutions to make them communicate with each other in a common network, also many countries have their own regulations for wireless communications which makes the defining of a network standard even more difficult.

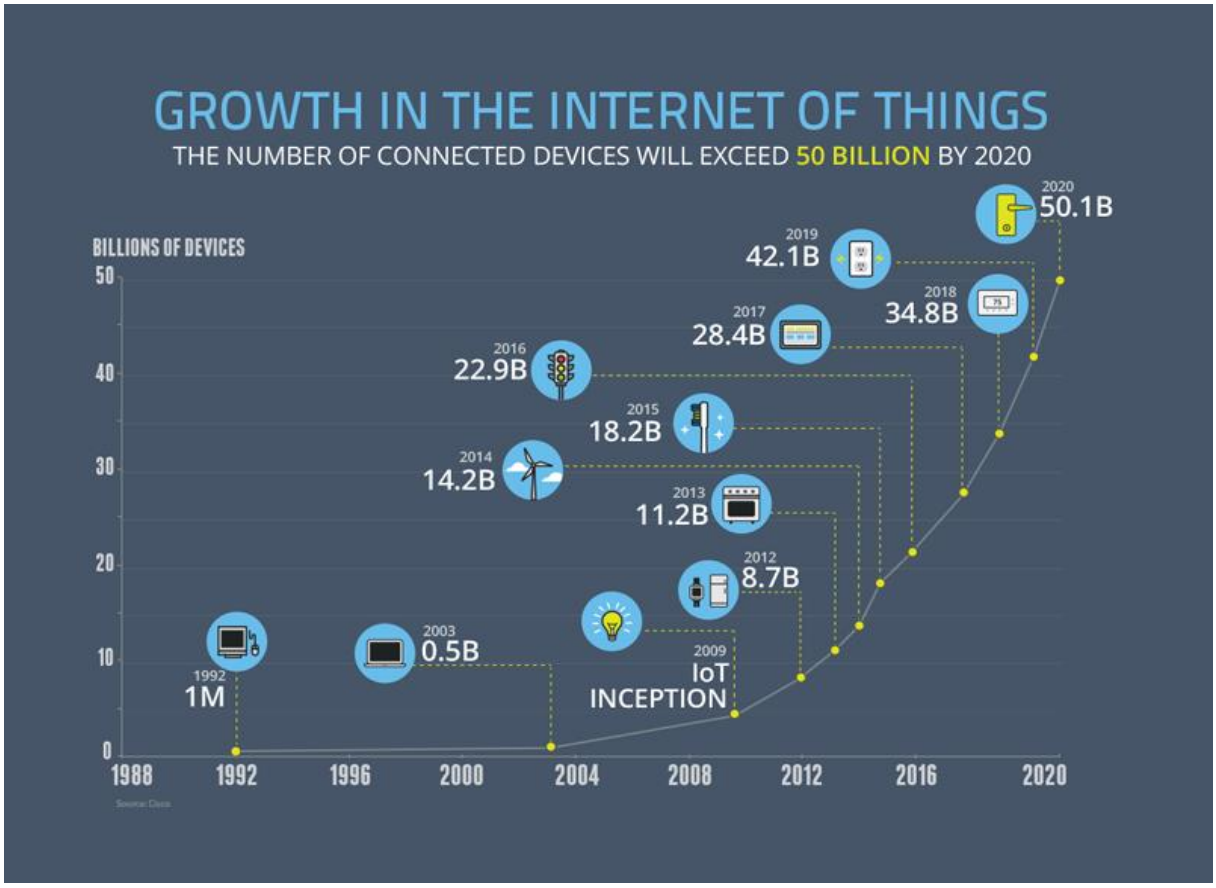


Figure 1.2: IoT growth through time [16].

Another challenge is security and privacy of the data that is being transmitted. Many applications have to exchange information that is sensitive to individuals, companies, professionals and concern personal, medical, financial, investment and industrial data that is crucial to the smooth, safe, and successful operation of industrial production process, transportation, product and service distribution, financial or trade deals and medical treatment. In addition, the need for new state legislations and rules that guarantee privacy and free access to the flow of information for the public, arises. Furthermore, concerns have been raised that the Internet of things is being developed rapidly without appropriate consideration of the profound security challenges involved and the regulatory changes that might be necessary. In particular, as the Internet of things spreads widely, cyber-attacks are likely to become an increasingly physical rather than simply virtual threat. There are three main problems that need to be addressed by developers and manufacturers, in order to ensure privacy and security, that are being highlighted by the report [7].

- **User consent:** It is about acquiring the required level of permission from the users who are affected by the devices or services. New technologies, that request consent from users in an efficient and effective manner, must be developed due to the fact that users have limited time and technical knowledge to engage in the process.
- **Control, Customization, and Freedom of Choice:** The data owner must have full control on his data, allowing the users to delete or move data from one service provider to another at any time and be able to choose their own hardware devices and software components from different vendors to build their smart environment. Also, users must retain their right to revoke or change any consent they have given in the past at any time.
- **Anonymity:** Network communication devices typically have MAC addresses that can be used to trace back the data communication paths so user location can be easily tracked as a consequent. As a result, new technologies such as Tor(torproject.org) must be developed in order to secure privacy.

The need to uniquely address the huge number of "things" in an IoT application is a great design challenge and is causing lots of problems to the developer. Technologies and protocols like IPv4 and Auto-ID center doesn't seem to solve the problem adequately. Fortunately, the combination of URI framework and the IPv6 protocol that many new devices support is proving to be the final solution to the problem. IPv6 is more suitable than IPv4 because it uses a 128-bit address naming instead of 32-bit, so it can identify 2^{128} unique devices. IPv6 provides other technical benefits in addition to a larger addressing space. In particular, it permits hierarchical address allocation methods that facilitate route aggregation across the Internet, and thus limit the expansion of routing tables so it improves networking performance between devices.

As it has been stated before IoT is defined by its heterogenic nature, that makes creating an architecture as base for developing applications really challenging. The approach that seems more promising and is proposed is that of a service-oriented architecture(SOA) [4]. The implementation of such a SOA is based on four discrete and inter-connected layers which can be summarized as follows:

- **Sensing layer:** All sensors that are integrated into the devices hardware which are used to measure real world quantities and status changes that are important for the application's function.
- **Network layer:** This layer contains all the necessary network interfaces and protocols that are needed for supporting connections among the devices, both wired or wireless and manages network traffic.
- **Service layer:** Its purpose is to create, sustain and support services that are needed by higher level applications and users.
- **Interface layer:** In order to use an object oriented model for the development of higher level applications a separation between these applications and the rest lower level architecture is needed. In that scope we create this layer that contains methods, APIs and frameworks that are used for the interaction with lower level layers.

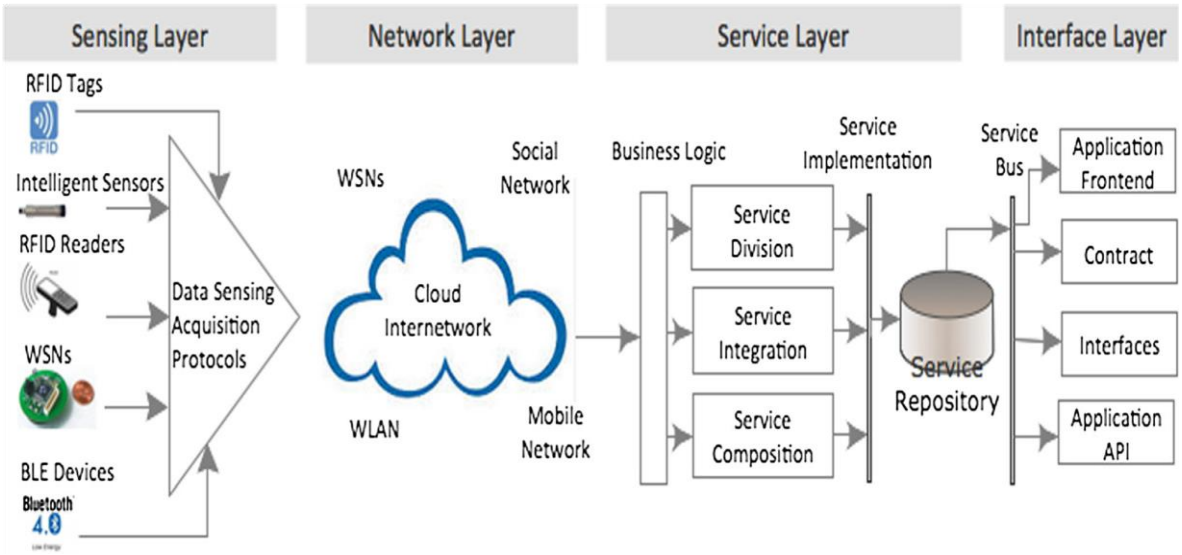


Figure 1.3: Service oriented architecture for IoT [4].

To successfully implement the proposed architecture, we must have the capability to quickly track and identify every object and service in the IoT system. The technique of assigned unique identity to an object is called a universal unique identifier (UUID). Fortunately, many wireless technologies, like Bluetooth, and commercial devices that are sold in the market are already making use of it. Furthermore, a universally accepted

and well defined service layer, that contains APIs and protocols to support higher application functions or features, is crucial for the successful implementation of such a system.

1.3 ECG Analysis

An electrocardiogram or ECG, records electrical activity in the heart. The ECG signal is produced as cardiac muscle contracts in response to electrical depolarization of the muscle cells and is recorded by electrodes placed on a patient's body. ECG is one of the most useful and inexpensive test in emergency medicine, also is crucial for making the diagnosis of cardiac ischemia. An ECG waveform consists of five waves and six intervals and segments, all of them are needed for an accurate analysis and interpretation. The waves, intervals and segments are presented in the figures below.

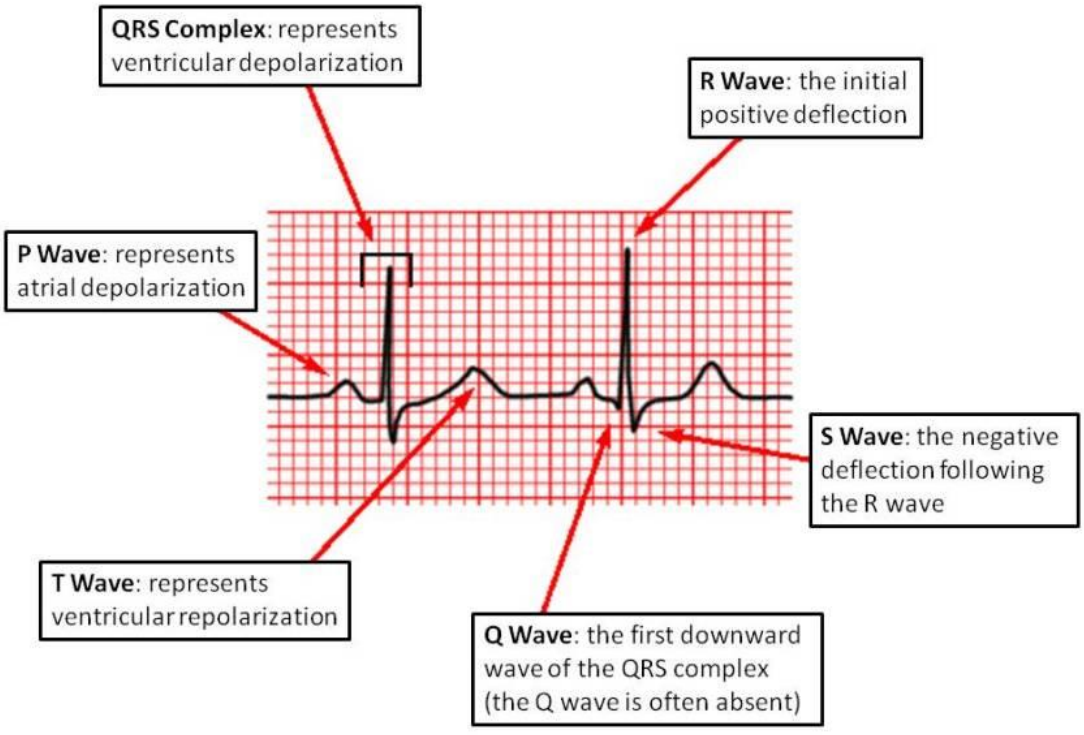


Figure 1.4: ECG waves [14].

- **PR Interval:** From the start of the P wave to the start of the QRS complex.
- **PR Segment:** From the end of the P wave to the start of the QRS complex
- **J Point:** The junction between the QRS complex and the ST segment
- **QT Interval:** From the start of the QRS complex to the end of the T wave
- **QRS Interval:** From the start to the end of the QRS complex
- **ST Segment:** From the end of the QRS complex (J point) to the start of the T wave

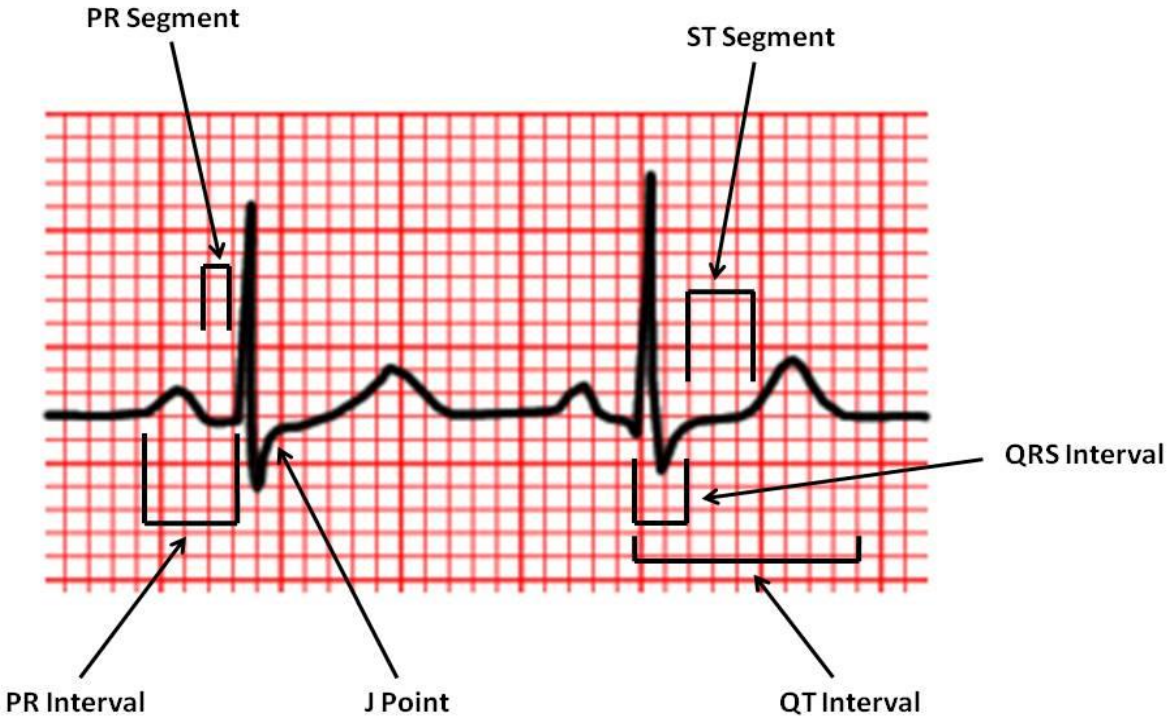


Figure 1.5: ECG intervals and segments [14].

It is obvious that to analyze an ECG signal even an experienced doctor needs a considerable amount of time, just to monitor and diagnose a single patient. That means hospitals have to hire more personnel to be able to operate emergency rooms or ICUs, as a result the operating costs increase, if the hospital is understaffed, the delay for a doctor to diagnose or monitor can be fatal for patients receiving treatment. A solution to address this problem is to automate ECG analysis with the aid of computational devices. Many researchers have developed machine learning algorithms that implement support vector machines (SVM) to analyze data and recognize patterns. By giving an

SVM sets of training examples it can build statistical models to classify newer inputs into categories, with enough training we can achieve almost 100% accuracy in diagnosing ECG anomalies or faults. Also an important advantage of SVMs is that they can be trained prior to real life operation which means they can be deployed in a short amount of time and analyze ECG with maximum accuracy from the moment they are assigned to a patient. Combining the SVM and IoT technologies together we can introduce new automated ways of patient monitoring and diagnosing in a very large scale. We have the capability to deploy many SVM devices designed as modules of IoT systems in ICUs and emergency rooms in order to eliminate the need for continuous human supervision of every individual patient.

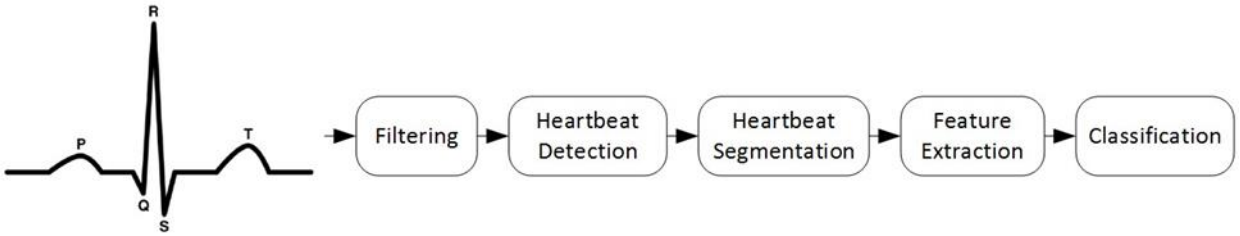


Figure 1.6: ECG analysis flow [13].

Furthermore, wearable medical devices have the features and capabilities, as we have mention before, to implement such SVMs and connect to networks. In addition, because of their low cost supplying every person with such a device is possible, in that way medical services can monitor, diagnose or administer treatment to patients without the need for hospital admission, it is also a very efficient and automated way to collect a huge amount of medical data. As a result, healthcare services reduce operating costs and can statistical analyze their data with greater accuracy because of the significant larger sample. The greatest challenge would be to develop a secure and robust network for these devices to communicate with each other and with medical servers.

CHAPTER 2

Background and related work

2.1 Bluetooth for IoT network implementation

Bluetooth wireless technology is a short-range communications system intended to replace the cables connecting portable or fixed electronic devices, it was originally developed as a wireless alternative to RS-232 data cables and point-to-point audio. Bluetooth is managed and directed by the Bluetooth Special Interest Group (SIG), its key features are robustness, low power consumption, and low cost, there are also many optional features that can be omitted or implemented according to the design goals of different vendors allowing product differentiation. The technology can be implemented into two forms: Basic Rate (BR) with the optional Enhanced Data Rate (EDR) feature and Low Energy (LE). According to SIG specifications vendors can integrate one of the forms or both in their devices but only one can be active and available for connection at any time. The BR/EDR system is focused more on speed and data throughput as it

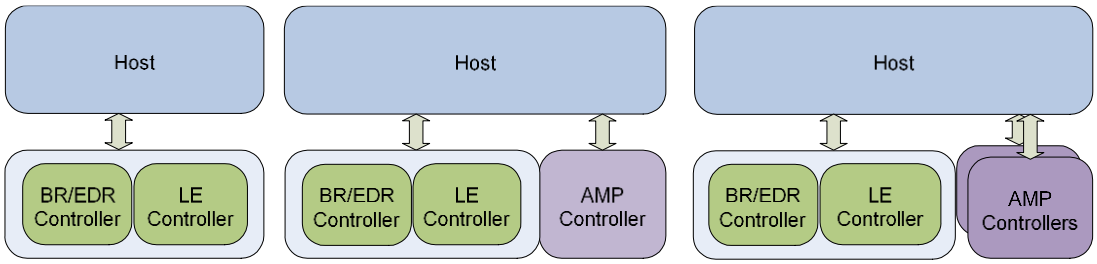


Figure 2.1: Bluetooth host and controller combinations [11].

theoretically achieves 2.1 Mb/s transmission rate (with EDR enabled) whereas LE system is focused on lower current consumption, lower complexity and lower cost than BR/EDR. The core system morphology of both BR/EDR and LE is the same, it consists of a Host and one or more Controllers. A Host is a logical entity defined as all of the layers below the non-core profiles and above the Host Controller Interface (HCI). Bluetooth is already widely utilized as the network building technology for communication between devices in IoT applications, as it has some key advantages compared to other technologies like ZigBee or Z-Wave:

- **Adaptation and support:** Almost all vendors worldwide are producing devices and applications that use Bluetooth for short-range connectivity and have been doing so for many years. Also, due to the wide use of the technology developers have accumulate a lot of experience working with the protocols and features so the process of designing and implementing new devices or applications is faster and easier. Furthermore, producers can have the support and the technical expertise of a big consortium (SIG), whose sole purpose is to promote the technology, during the development of their products and have access to resources that accelerate that development.
- **Low cost:** Because of the wide adaptation, chips implementing the technology are very cheap as they are massively produced. In addition, techniques like firmware and software reuse accelerate the development process and in result reduce the cost of new product creation.
- **High level architecture:** We have highlighted before that the proposed way to develop an IoT system is to use a service oriented architecture because of the heterogenic sensors and devices that need to communicate. Bluetooth assists application development by separating the lower levels of the protocol from the programmer, it creates this abstraction level by utilizing the so called profiles. Bluetooth profiles define the required functions and features of each layer in the Bluetooth system from the physical (radio) layer to L2CAP, they also define application behavior and data format. Collections of profiles are called services. If two devices comply with all the requirements of a profile, then application interoperability is enabled between them. In addition, Bluetooth modules (both BR/EDR and LE) implement and operate service discovery (SDP) and general

attribute (GATT) servers. Other devices connect to these servers to search for available services or exchange and set connection parameters to comply with supported profiles. As a result, every application that runs on a device can connect through Bluetooth just by searching for compatible services without the need to have additional information about the other device's hardware or software structure, it only has to comply with the service's profiles. The service oriented structure of Bluetooth's higher levels makes it an ideal technology for integration into the service oriented architecture of IoT systems [4].

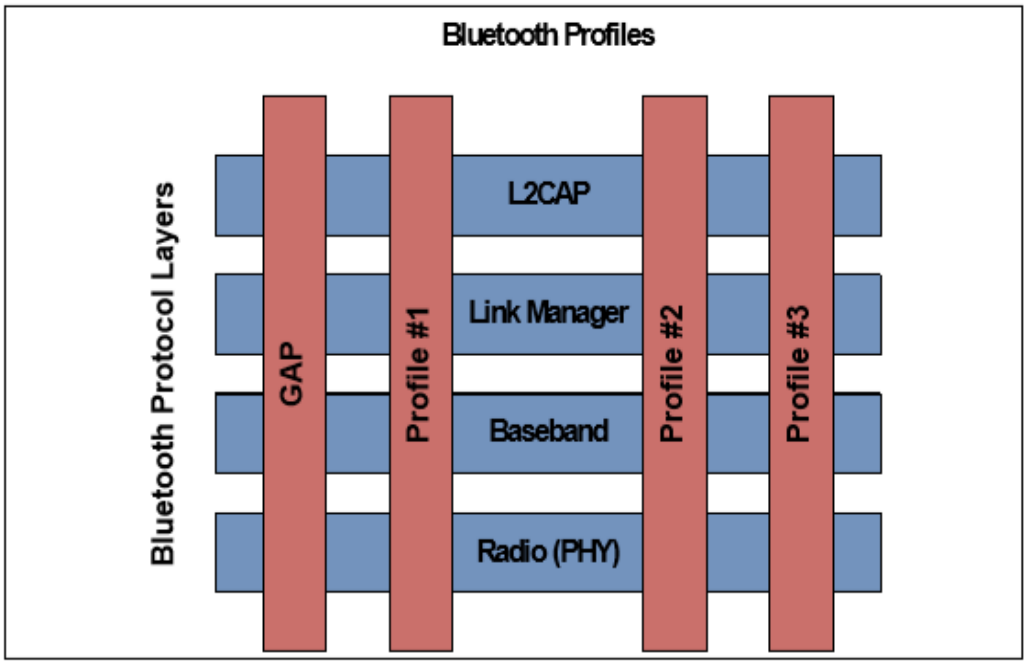


Figure 2.2: Bluetooth profiles [11].

- **Security and privacy:** For real life IoT applications security of the data that is being exchanged and privacy of the users is a really important aspect [7]. It is imperative for developers to take them into account when designing an application. Bluetooth utilizes built-in features and techniques to ensure that, so it assist developers to reduce design time. The Bluetooth security model includes five distinct security features: pairing, bonding, device authentication, encryption and message integrity. The model keeps updating and improving in every new version of Bluetooth core specifications, from version 4.0 onwards a

whole new security model for BLE that is based on previous model but with many extra features that also ensure user privacy.

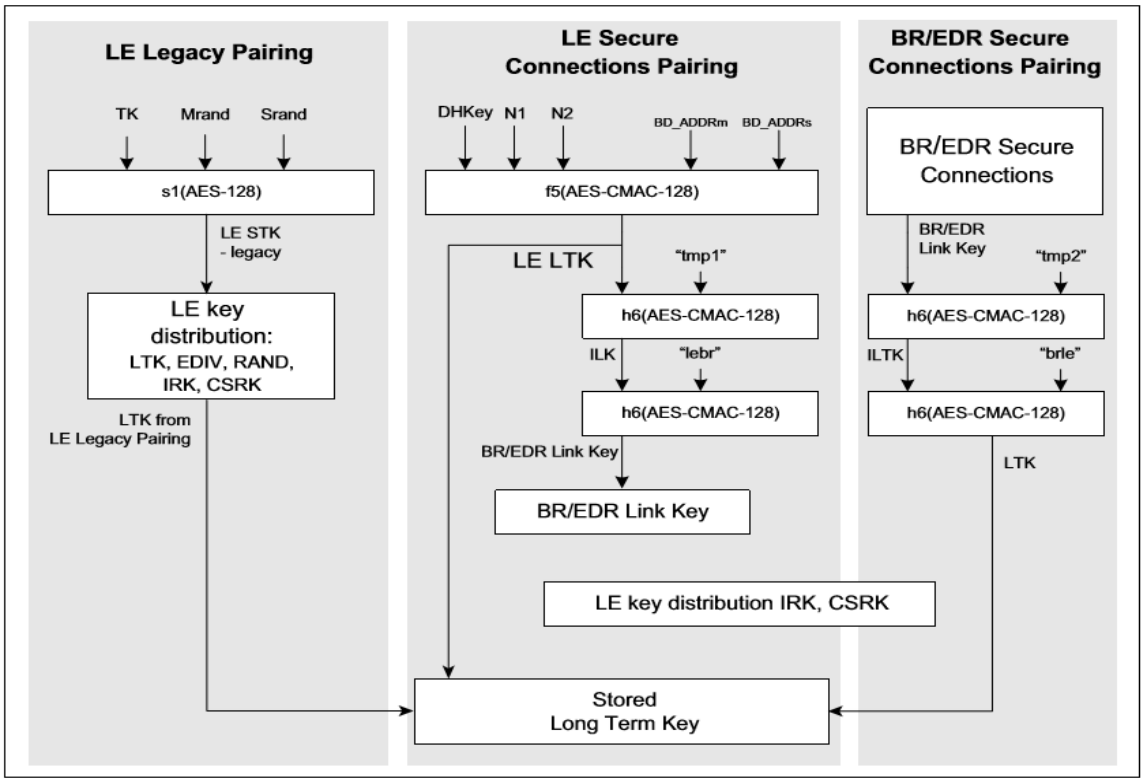


Figure 2.3: Key generation hierarchy [11].

Bluetooth LE supports a feature that reduces the ability to track a LE device over a period of time by changing the Bluetooth device address on a frequent basis. This feature is not used in discovery mode but in connection mode or during connection procedures. Private addresses can be resolved by the controller or by the host, when resolution is performed exclusively in the host, a device experience increased power consumption because device filtering must be disabled, so in general it is not a good practice to exclude controller from address resolving.

2.2 Medical Databases

Adoption of Electronic Medical Records (EMR) by hospitals and medical practitioners can have a huge impact and greatly reduce healthcare cost, minimize medical errors, and generally improve health [10]. EMR systems have computer databases as their core and are designed or expanded by taking account the features, specifications and limitations of them. Also, with EMR healthcare administrators are given the capability to collect geospatial data about patients so they can conduct deeper statistical analysis on patient demographics and the geographic characteristics of deceases.

Item	2006 (0)	2007 (1)	2008 (2)	2009 (3)	2010 (4)	2011 (5)	2012 (6)	2013 (7)	Total
Cost									
System costs									
System infrastructure	1,241	-	-	-	98	93	88	84	1,604 (10.0)
System application	1,006	1,274	315	192	188	184	179	174	3,512 (21.9)
Office supply	306	286	-	105	102	98	95	91	1,084 (6.7)
Sub-total	2,554	1,560	315	296	388	375	363	348	6,199 (38.6)
Induced costs									
Paper-charts scan	-	-	724	519	-	-	-	-	1,243 (7.7)
MTs support	166	1,186	1,107	1,118	1,185	1,313	1,255	1,281	8,612 (53.6)
Sub-total	166	1,186	1,831	1,636	1,185	1,313	1,255	1,281	9,854 (61.4)
Total PV of annual costs	2,720	2,746	2,146	1,934	1,573	1,687	1,618	1,630	16,054 (100.0)
Benefit									
Cost reductions									
Supplies for paper-charts	11	52	258	100	248	91	231	3	1,076 (5.5)
Chart storage space	-	-	14	145	142	139	135	129	703 (3.6)
Chart management FTE	-	-	180	782	799	847	816	807	4,231 (21.5)
Clerks decreased	-	-	165	165	165	165	165	164	990 (5.0)
Supplies for MDIS	7	67	78	415	168	165	161	155	944 (4.8)
Sub-total	18	120	695	1,335	1,522	1,408	1,507	1,339	7,945 (40.4)
Additional revenues									
From remodeling storage	-	-	-	17	26	35	25	21	125 (0.6)
From temporary storage	-	-	261	300	280	275	269	262	1,646 (8.4)
From MT support	-	-	551	1,411	1,421	2,747	1,928	1,899	9,956 (50.6)
Sub-total	-	-	811	1,728	1,727	3,056	2,223	2,182	11,727 (59.6)
Total PV of annual benefits	18	120	1,506	3,063	3,249	4,465	3,731	3,521	19,672 (100.0)
Accumulated NPV	(2,702)	(5,329)	(5,969)	(4,839)	(3,163)	(385)	1,726	3,617	
Benefit-cost ratio									1.23
Discounted payback period									6.18

Figure 2.4: Cost-benefit analysis in 1000US$ [10].

In order to assist adoption and development of EMR systems the computer database (Clinical repository) at their core must fulfill certain design requirements and have

certain capabilities. These databases need to have a data model to define its functional requirements and to produce a formal description, a schema of all the data generated in the enterprise and how it was all related, and a database structural design to define its technical requirements [1]. Furthermore, for a design to be considered successful the database must be able to provide rapid retrieval of data for individual patients, and to have the capability to adapt to changing information needs of growth and new technology. Object-oriented and no-SQL databases seem to have an advantage in fulfilling these requirements as they are far more scalable and support dynamic data schemas, allowing the storing of heterogeneous medical information [1]. These databases are often integrated into already existing or new systems that have parts which need to communicate with them, so they must be able to cooperate with the various subsystems and also be operationally and structurally independent of all the other subsystems and application programs. With this flexibility, different parts can be upgraded or replaced at any time and simultaneously keep the whole system operational. The unbreakable continuous operation of a system is imperative for a medical application who has to serve a large number of patients. Privacy of patient data is also important and the database together with the system must be secure enough to ensure there will be no leak of information at any point no matter what technical difficulties arise or who will try to gain access to the stored data [7].

2.3 Medical Application Development with Android

App development starts with a clear understanding of what the medical app needs to do, for whom, and the environment it needs to operate in. Hardware and device specifications must also be considered while developing, for example smartphone users need faster acquisition of data and more compact design of user interface than tablet or desktop users [12]. Android is a mobile OS that was designed to assist application development by giving access to many APIs, architectures and frameworks to programmers and testers. In addition, Google support and services like google maps, gmail, google plus and chrome integration with the operating system are really beneficial for the developer and especially useful to medical applications. The ability

to configure every subsystem or configuration of the device can be beneficial to the programmer.

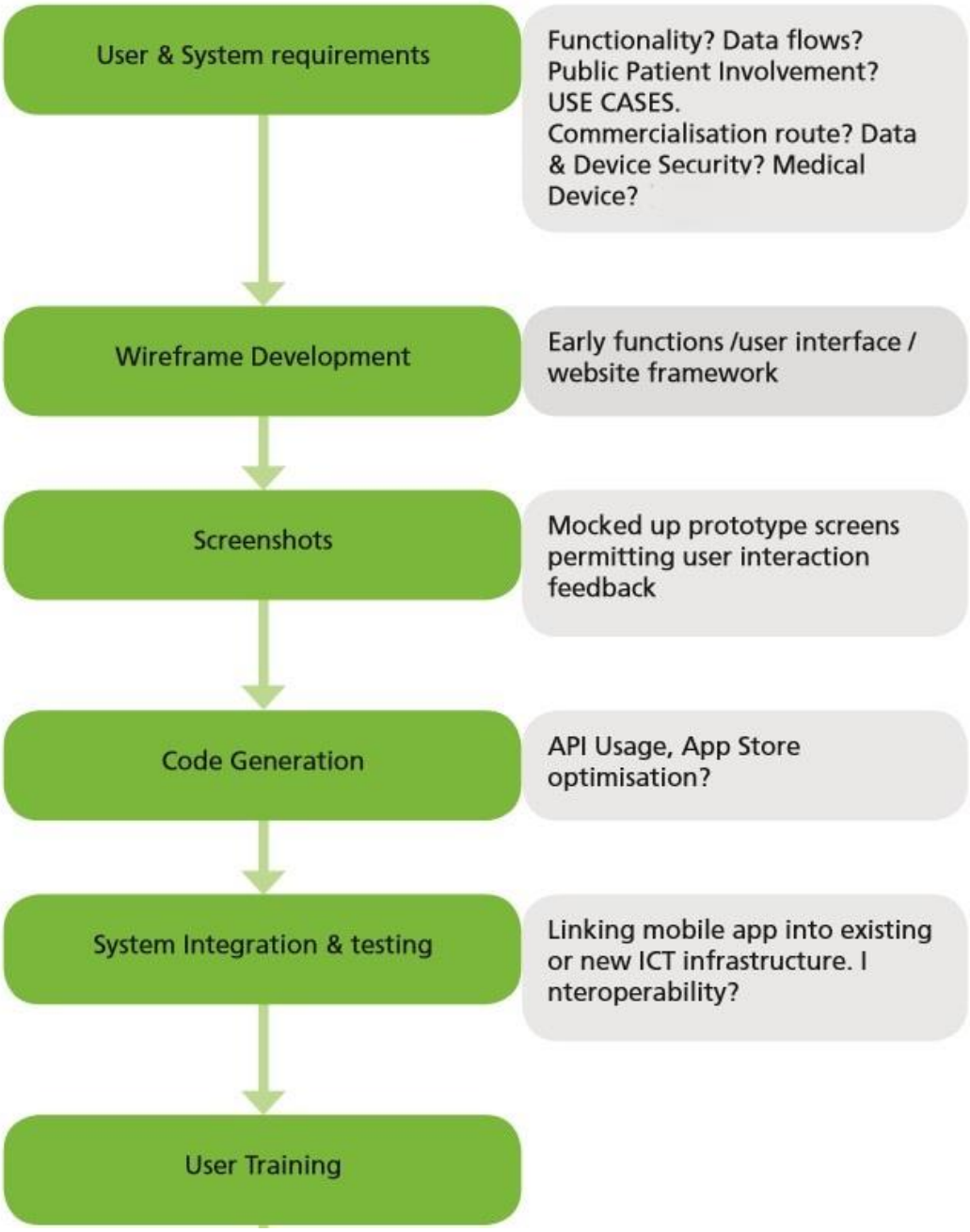


Figure 2.5: Application developing process flow [12].

This flexibility leads to product differentiation cause each vendor customize the operating system for his own devices, so to be sure that an application operates and behaves as it was designed on every device, developers need to conduct more extensive testing by running it on at least one device from each vendor. When reliability is

concerned, Android is not the best solution as the system freeze and crush when multiple applications or heavy computational loads are running. Developers need to optimize and test their code in order to minimize that risk [2].

CHAPTER 3

System Overview

The system we developed and implemented in this thesis consists of three discrete parts. The first part is the Linux based application and it serves as the system's entry point. Its task is to read the input from the electrodes (Voltage in millivolt scale) that are placed onto the patient's body and transmit the measurements through Bluetooth to the Android application. This application is the second part of the system. It can run on any device (smartphone, tablet etc.) that utilize Android as operating system and has a Bluetooth module. The application performs a number of tasks. Firstly, it receives the ECG signal that has been captured and stores it in a text file for later use, as the signal is being transmitted the user can see the pulse in real time thanks to dynamic plotting capabilities of the app. After the transmission is completed and the data stored on the device, we can choose to open the text file to review the measurements, plot the whole ECG signal or upload it on the database through the Internet in JSON format. In order to communicate with the remote database, we developed the third part of our system which is a web service. The web service is implemented as a servlet running in any web container with connectors to the database. We have developed two connectors, one for the MongoDB and one for the MySQL database, the connector for MySQL works with any SQL database with a matching schema like the one we used simply by loading the appropriate driver for the target database. The web service receives HTTP requests that contain JSON files, after parsing and evaluating them it connects to the database for storing the data in the desired format. A graph that describes the whole system with the interactions between the parts as well as some key frameworks, libraries and tools used for the development of each individual part follows:

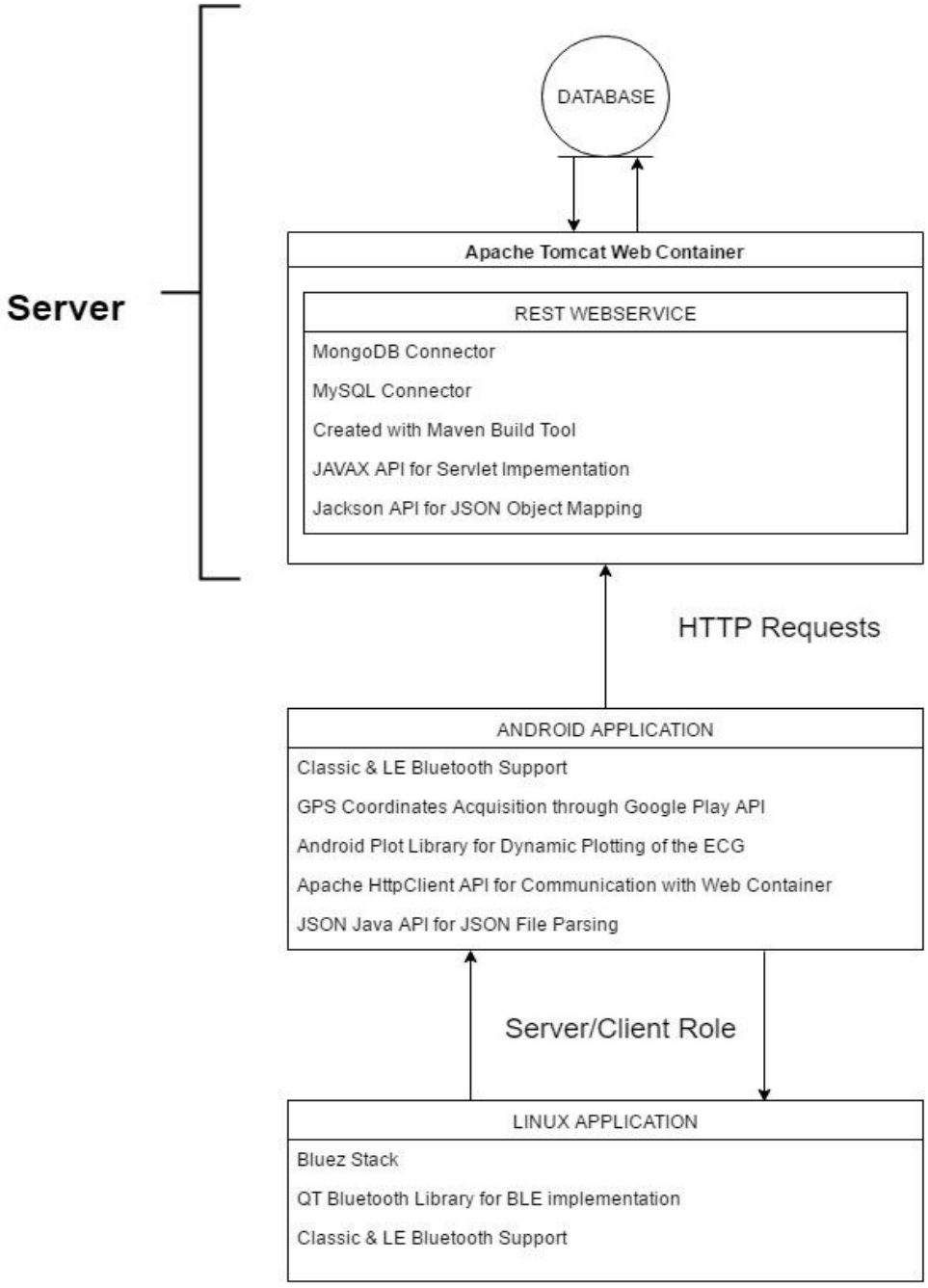


Figure 3.1: System Overview.

For the primary design of the system and during its development we took under serious consideration the advantages or the disadvantages of each technology and the key design principles that were described in the previous two chapters to produce a robust system that can satisfy the needs and overcome the implementation difficulties of the deployment and operation in real life medical scenarios.

3.1 Linux Based ECG Monitor and Data Transfer Bluetooth Application

3.1.1 BlueZ Stack - Classic Bluetooth Development

BlueZ is the official Bluetooth protocol stack for Linux operating system and is one of the basic sub-systems of every distribution's kernel, so it is installed as part of it by default. For the development we used BlueZ version 5.37. BlueZ stack includes software implementations of higher Bluetooth transfer protocols (RFCOMM, OBEX, A2DP etc.), sets up and maintains the SDP or GATT servers where information about services and profiles supported by the device is stored, creates two interfaces between user space programs and applications and the Bluetooth hardware controller's firmware that communicates with the operating system in kernel space, also the stack includes C libraries containing functions that are needed for the development of Bluetooth programs or applications. The first interface is L2CAP socket, it operates as a multiplexer for higher level protocols.

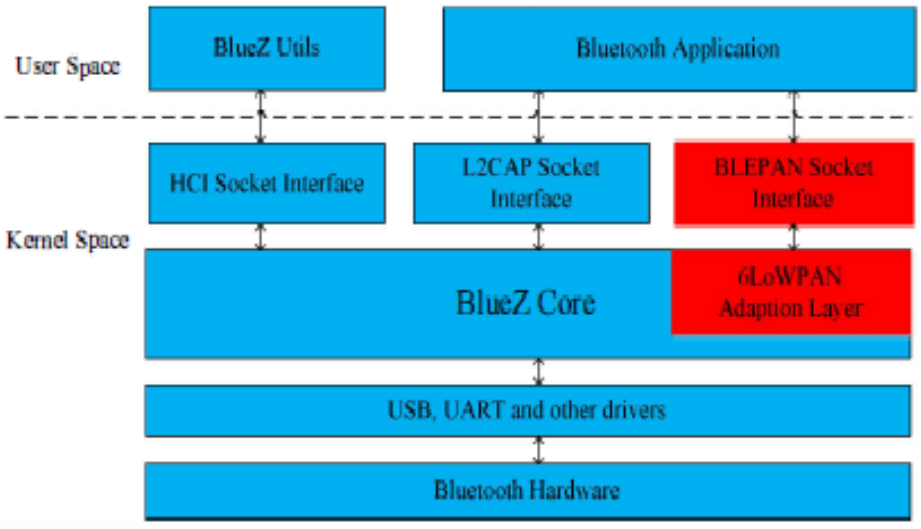


Figure 3.2: BlueZ stack architecture [26].

Every high level protocol has predetermined parameter values and options (enable or disable packet retransmission, MTU size, order of packets to be transferred etc.), in order to use such a protocol, the developer can open an L2CAP socket and pass the desired protocol as a parameter, then the interface automatically sets up the proper command packets, that contain values and enable or disable features according to protocol specifications and sends them to the hardware controller for execution. The second interface is the HCI socket, by opening that socket the developer can directly send commands to the hardware controller in order to directly program it. This interface gives the ability to the programmer to create his own transfer protocols or change the parameters of the existing ones according to the application's needs.

In this thesis we developed the Classic Bluetooth client by utilizing only the stack's libraries and functions. In this operation mode the ECG device operates as a client and the Android device as a server. We designed the application running on the ECG device to be able to connect with any device so instead of hard coding a known MAC address into the application we make use of the SDP server feature. Furthermore, we decided to use the SPP profile so the two devices communicate over the RFCOMM protocol, that means data packets are transmitted in the order they were send to the Bluetooth controller even if retransmission is needed. This adds a small delay in data exchange because every packet has to wait for the previous packets to be delivered first but it ensures data consistency and eliminates the need for packet ordering in server side so makes the development of the server application less complex. The server device opens an RFCOMM socket and adds a record containing the SPP profile into its SDP server. The client application calls the `hci_get_route()` function to get a route to the Bluetooth hardware then it opens an HCI socket by calling the `hci_open_dev()` function and passing the route to the hardware as argument, with the socket open the client can send commands directly to the Bluetooth controller. Now that it can communicate with the Bluetooth hardware the client scans for available devices and every time it finds one sends a request to receive the SDP server's records hosted on the device, then it searches them for the desired profile. The code for device scanning and record searching is shown in figures 3.3, 3.4, 3.5. As we have mentioned before every profile has a unique UUID, so the client searches for a matching UUID in the records.

```
printf("Scanning ...\n");
info = NULL;
num_rsp = 0;
flags = 0;
length = 8; /* ~10 seconds */
num_rsp = hci_inquiry(dev_id, length, num_rsp, NULL, &info, flags);
if (num_rsp < 0) {
    perror("Inquiry failed");
    exit(1);
}
```

Figure 3.3: Code Segment for Device Scanning.

```
session = 0; retries = 0;
while(!session) {
    session = sdp_connect( BDADDR_ANY, &(info+i)->bdaddr, SDP_RETRY_IF_BUSY );
    if(session) break;
    if(errno == EALREADY && retries < 5) {
        perror("Retrying");
        retries++;
        sleep(1);
        continue;
    }
    break;
}
if ( session == NULL ) {
    perror("Can't open session with the device");
    free(info);
    continue;
}
```

Figure 3.4: Code Segment for Connecting to SDP Server.

The UUIDs are 128-bit numbers but if they describe services or profiles that are defined by the official Bluetooth specifications we can refer to them by using only 16 or 32-bits, so we have added these two cases, without any further actions, in our code where we parse the UUIDs from the records for future use.

```
foundit = 0;
responses = 0;
for (; r; r = r->next ) {
        responses++;
        rec = (sdp_record_t*) r->data;
        sdp_list_t *proto_list;

        // get a list of the protocol sequences
        if( sdp_get_access_protos( rec, &proto_list ) == 0 ) {
        sdp_list_t *p = proto_list;

            // go through each protocol sequence
            for( ; p ; p = p->next ) {
                    sdp_list_t *pds = (sdp_list_t*)p->data;

                    // go through each protocol list of the protocol sequence
                    for( ; pds ; pds = pds->next ) {

                            // check the protocol attributes
                            sdp_data_t *d = (sdp_data_t*)pds->data;
                            int proto = 0;
                            for( ; d; d = d->next ) {
                                    switch( d->dtd ) {
                                            case SDP_UUID16:
                                            case SDP_UUID32:
                                            case SDP_UUID128:
                                                    proto = sdp_uuid_to_proto( &d->val.uuid );
                                                    break;
                                            case SDP_UINT8:
                                                    if( proto == RFCOMM_UUID ) {
                                                            printf("rfcomm channel: %d\n",d->val.int8);
                                                            loco_channel = d->val.int8;
                                                            foundit = 1;
                                                    }
                                                    break;
                                    }
                            }
                    }
                    sdp_list_free( (sdp_list_t*)p->data, 0 );
            }
            sdp_list_free( proto_list, 0 );

        }
        if (loco_channel > 0)
            break;

}
```

Figure 3.5: Code Segment for SDP Record Searching.

After the client finds a suitable device it opens a blocking socket by passing the server's address and the communication protocol as arguments and then use the `write()` function to transfer the data. This function returns 0 if it was executed successfully or -1 if it failed for some reason. When the transfer is completed we close the socket and end the connection by calling the `close()` function. It is safe to conclude that classic Bluetooth communication is very similar to simple TCP networking from the programmer's view. The code in figure 3.6 demonstrates the above.

```
status = connect(s, (struct sockaddr *)&loc_addr, sizeof(loc_addr));
if( status < 0 ) {
    perror("uh oh");
}
    //send file name
    printf("Sending file:%s\n",argv[1]);
    status = write(s, argv[1], strlen(argv[1]));
    status = write(s, "\n", 2);
    int j = 0;
do {
    j++;
    if (j<=999)
    {   //nanosleep(&tim,&tim2);
        status = write(s, strings[j], strlen(strings[j]));
        printf ("Wrote line: %d , %d bytes\n", j, status);
        printf("data: %s \n",strings[j]);
    }else
    {
        status = -1;
    }
} while (status > 0);
close(s);
sdp_record_free( rec );
```

Figure 3.6: Code Segment for Sending Data to the Bluetooth Server.

When the connection with the server is terminated the client has the option to stop running or initiate a new search to find and connect with a different device, so we can use the same monitoring device as client to send ECG data to multiple Android devices.

3.1.2 Qt Bluetooth API - BLE Development

Bluetooth low energy application development follows a different programming model than Classic Bluetooth. In BLE, there is only one communication protocol called Attribute (ATT) protocol, so devices communicate with the so called attributes. Attributes are records in a big table that is created and stored in every Bluetooth Low energy device, they can be identified with unique UUIDs and hold discrete values that represent data and are associated with various attribute types defined by SIG specifications, properties, read or write permissions and a handle, they can be accessed by other devices with read or write requests. The protocol is based on the server-client

model, the communication involves the exchange of read or write responses and requests, also the protocol defines four operations that can be performed by both roles:

- **READ:** A read request is send from the client to read the value of an attribute. The server sends a read response containing the value if the client has the appropriate permissions.
- **WRITE:** The client sends a write request containing the new value, if he has sufficient permissions and the value has proper format the server updates the attribute and sends a write response that the operation was successful.
- **NOTIFY:** One of the two communication partners sends a notification that an attribute has changed. The other partner can send a read request to get the new attribute value or can ignore the notification. Notifications are a really simply way to give the developer more control over the power consumption of his application's code.
- **INDICATE:** This action is used when a serious change has happened to an attribute that is crucial for the link to be sustained between the two devices, so the other device has to take action. Indications can't be ignored so the device that sends them stops any further communication until a read request, for that attribute, is received. Indications are very helpful when a patch to the code has been applied to either device and the other needs to conform with new specifications in order to communicate properly.

In order to make it easier for an application to use Bluetooth Low Energy the stack adds one more level above the ATT protocol, this level is the Generic Attribute (GATT) profile. The Generic Attribute Profile (GATT) defines a service framework using the Attribute Protocol. This framework defines procedures and formats of services and their characteristics. The procedures defined include discovering, reading, writing, notifying and indicating characteristics, as well as configuring the broadcast of characteristics. The profile uses the so called characteristics as its basic data structure objects that hold not only the data that has to be exchanged but also information about that data. The entities holding information about the data are called descriptors and they have their own UUIDs for referencing. Characteristics are higher level objects that are translated into lower level groups of multiple attributes. Every service or profile created by the

developer is an extension of the GATT profile and it is advisable to conform with it, as it would make development much more complex and applications from other developers who want to connect will need to have the exact specifications to establish a link. The Bluetooth stack creates and operates a GATT server, just like the SDP server, that stores supported services and profiles, other devices connect to this server to search its records for UUIDs that match with profiles they want to use. It is obvious that because of the programming model followed by Low Energy Bluetooth an object oriented programming language can make development simpler and the generated code will have better flow.

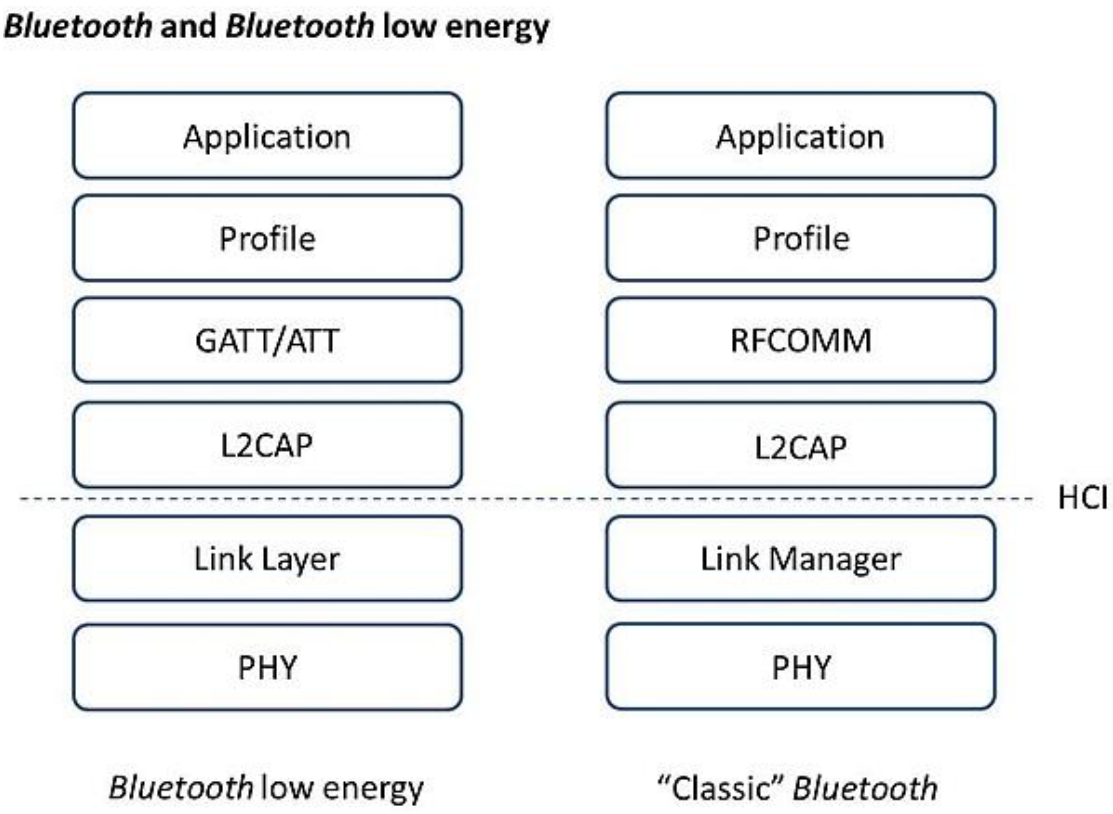


Figure 3.7: Comparison between Classic and Low Energy Bluetooth hierarchy [25].

Furthermore, the BlueZ stack version that was used during the development of the application does not fully support BLE's features like GATT server or certain profiles. These features are characterized as experimental so in order to use them we had to edit the Bluetooth daemon configuration file and add the -E switch. So every time the operating system boots and executes the configuration file to initialize the daemon these

features are enabled automatically. Also, there was almost no documentation for these features and very limited functions or tools to add and edit profiles or server operations to assist the developer. To address all these problems, we decided to use the Qt Bluetooth API version 5.7 written in C++. Qt library and the Qt Creator IDE is distributed by Qt company under GPL, LGPL or even commercial licensing if the need arises for future closed source code application development. In addition, the library is cross-platform and can run on Linux, Windows, MAC operating systems and even on embedded Linux distributions, that makes our application more versatile as it can be ported to almost every device very easily. The library also contains classes, features and even an editor for Graphical User Interface (GUI) creation that synergize with the backend logic without the need for complex interfaces to be developed in order to exchange data and edit its format for display. This is useful for future expansion of the application if it is running on devices with monitors to make interaction with the user easier and more direct.

Now the ECG monitor device takes the role of the server and the Android device is the client. We made this change to take advantage of BLE's low power consumption in idle and advertising modes. The server device exposes the heart rate characteristic that contains the ECG measurement data to scanning devices that are interested to read it. When a client device makes a connection, the server sends them notifications whenever a new measurement or some change to the characteristic has occurred. We decided to send notifications instead of read or write requests because the client has the option to ignore them so the two devices do not engage in any communication, that exchanges even a small amount of data, to further conserve power on both devices. The application first instantiates and passes the desired values to the characteristic objects, these values set up the characteristic type, read-write permissions, data format and size, then it sets up the advertising object for that characteristic and commands the Bluetooth controller to start advertising it, the code is demonstrated in figure 3.8. The characteristic we created has the SIG defined "Heart Rate Measurement" UUID, has one descriptor that enables notifications and is given the Notify and Read properties, that means it can cause notifications and the data it holds can be read from other devices. These values are passed to the characteristic object with `setUuid()` and `setProperties()` methods. The data type and size is defined with the `setValue()` method.

```
cout<<"Setting up characteristic data\n";
QFlags<QLowEnergyCharacteristic::PropertyType> Properties;
Properties.setFlag(QLowEnergyCharacteristic::Notify);
Properties.setFlag(QLowEnergyCharacteristic::Read);
QLowEnergyCharacteristicData charData;
charData.setUuid(QBluetoothUuid::HeartRateMeasurement);
charData.setValue(QByteArray(2, 0));
charData.setProperties(Properties);
//Adding descriptor to characteristic
QLowEnergyDescriptorData clientConfig(QBluetoothUuid::ClientCharacteristicConfiguration,
QByteArray(2, 0));

charData.addDescriptor(clientConfig);

cout<<"Setting up Service data"<<"\n";
QLowEnergyServiceData serviceData;
serviceData.setType(QLowEnergyServiceData::ServiceTypePrimary);
serviceData.setUuid(QBluetoothUuid::HeartRate);
serviceData.addCharacteristic(charData);

QLowEnergyAdvertisingData advertisingData;
advertisingData.setDiscoverability(QLowEnergyAdvertisingData::DiscoverabilityGeneral);
advertisingData.setIncludePowerLevel(true);
advertisingData.setLocalName("HeartRateServer");
advertisingData.setServices(QList<QBluetoothUuid>() << QBluetoothUuid::HeartRate);

cout<<"Advertising..."<<"\n";
const QScopedPointer<QLowEnergyController> leController(QLowEnergyController::createPeripheral());
const QScopedPointer<QLowEnergyService> service(leController->addService(serviceData));
leController->startAdvertising(QLowEnergyAdvertisingParameters(), advertisingData,
                               advertisingData);
```

Figure 3.8: Code Segment for Characteristic Creation and Advertising.

For the advertising we set up the advertising data object by making it discoverable with the `setDiscoverability()` method and adding our service to the service list with the `setServices()` method, then we create one pointer to the Bluetooth controller and a pointer to the service we want to advertise in order to send it to the controller after that we simply send the command to the controller to start advertise our service with the `startAdvertising()` method. To notify the other device that a new measurement has occurred in order for it to read it we firstly get a reference to the characteristic object from the service list and then we call the `writeCharacteristic()` method to update the data, writing the characteristic automatically causes notification to the other device.

```
QLowEnergyCharacteristic characteristic = service->characteristic(QBluetoothUuid::HeartRateMeasurement);
Q_ASSERT(characteristic.isValid());
service->writeCharacteristic(characteristic, value,
QLowEnergyService::WriteWithoutResponse); //Causes notification.
```

Figure 3.9: Code Segment for Updating Characteristic Data.

3.2 Android Application

Every Android application can be built by using four main components that are defined and implemented by the android SDK. These components are:

- **Activities:** Activities represent the application's screens, they manage and support the user interface and provide basic means for user interaction. They run always on the main application thread (UI thread) and are the basic blocks for application development. The entry point of an application is always an activity. Every Activity has a life cycle that is shown in figure 3.10.
- **Services:** Services are used to handle long running operations associated with an activity. They run on the background on separate threads (not on UI thread) and they continue to operate even when the activity is not on the screen.
- **Broadcast Receivers:** They handle the communication between the operating system and the application or between applications. They intercept signals and messages called Intents and initiate actions associated with them.
- **Content Providers:** This component supplies data from one application to others on request. Such requests are handled by the methods of the ContentResolver class. They are used mostly when applications need to communicate with a database.

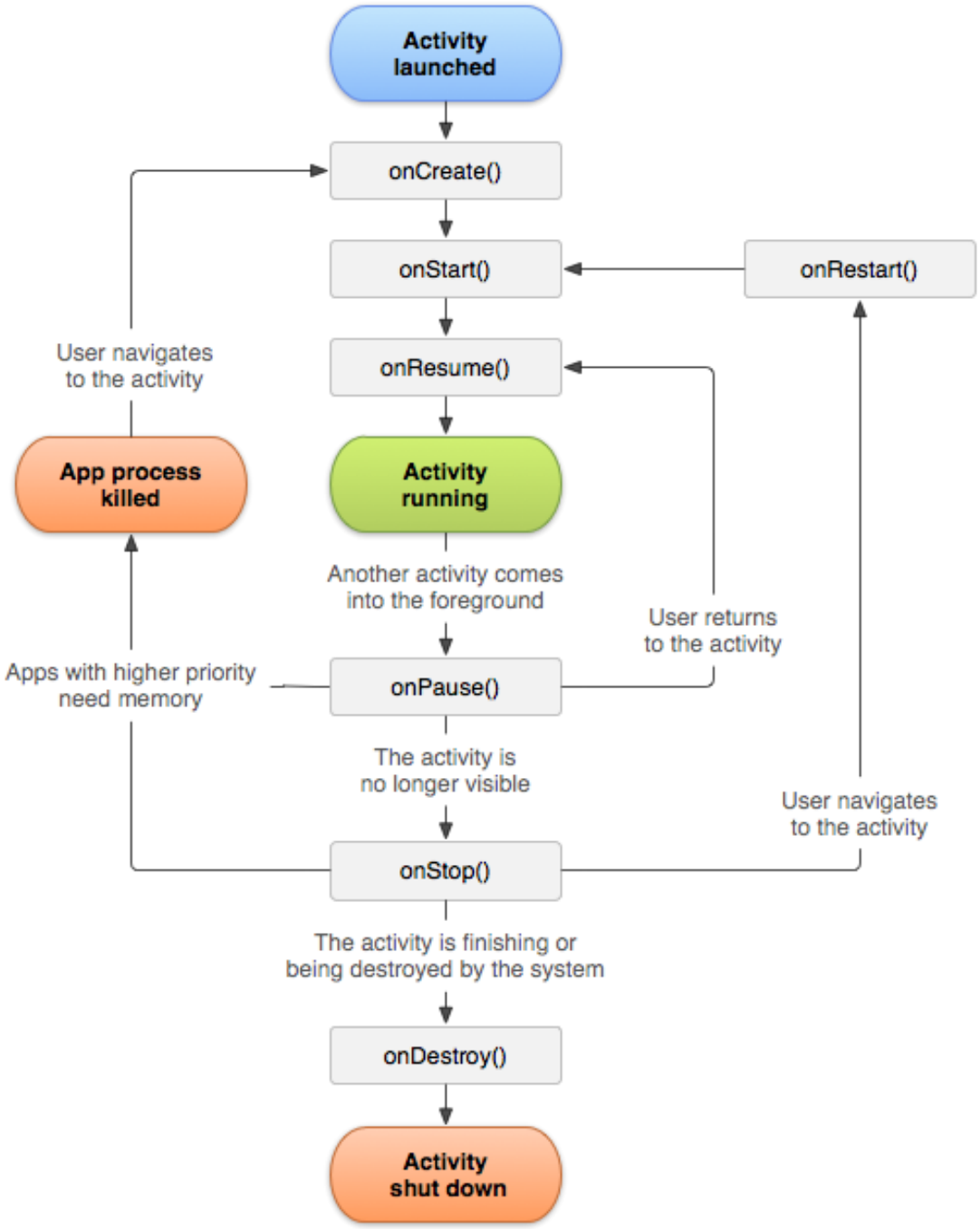


Figure 3.10: Activity Lifecycle [15].

The application we developed has seven activities for interaction with the user. We used the official Android Studio IDE for the development and we have set the minimum SDK version to 18 and the target SDK is version 24, this means that our application is compatible with every device running Android 4.3 (Jelly Bean) up to 7.1.1 (Nougat). Version 18 is the minimum Android version that supports BLE. Android was originally using BlueZ stack but since version 4.2 it changed to Bluedroid, a stack developed by

Broadcom. With this change android dropped the support of all Bluetooth transmission protocols except RFCOMM. Every application project has a manifest xml file in which permissions for access to device's hardware systems and application's activities must be stated and defined. Without the manifest file the project cannot be built. Before we can start developing we have to compose the application's manifest file with all the activities and the permissions we need. We declare permissions with the <user-permission> tag and activities with the <activity> tag.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.diploma.john.medicalapp">

    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.INTERNET" />
    <!-- The app can run on classic BT devices too -->
    <uses-feature
        android:name="android.hardware.bluetooth_le"
        android:required="false" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="MedicalApp"
        android:supportsRtl="true"
        android:theme="@style/AppTheme.NoActionBar">

        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity android:name=".Server_Activity" />
        <activity android:name=".Files" />
        <activity android:name=".options" />
        <activity android:name=".Display_Data" />
        <activity android:name=".Plotting" />
        <activity android:name=".LEConnectionInit" />
        <activity android:name=".BLEConnection" />
    </application>

</manifest>
```

Figure 3.11: Application's Manifest file.

The first screen of our application is defined by the MainActivity class which extends the android AppCompatActivity class. Every activity class has seven methods, as shown in figure 3.10, that we need to override in order to write our own code that implements the application logic. The recommended practice is to put all the code associated with the user interface inside the `onCreate()` method because it is the first method to run when the activity is created, so adding graphical elements into the interface at a later time can disorient the user and cause the application to crush. We have follow this practice in every activity we created. From the start screen the user can enable the device's Bluetooth and choose if he wants to connect with another device or browse the ECG signal files stored in the device. To pass from one activity to the other we send intents that contain the class that defines the activity and the operating system creates it. To connect with other devices, the user must press the user button and then choose if he wants to start a Classic or Low Energy Bluetooth connection.

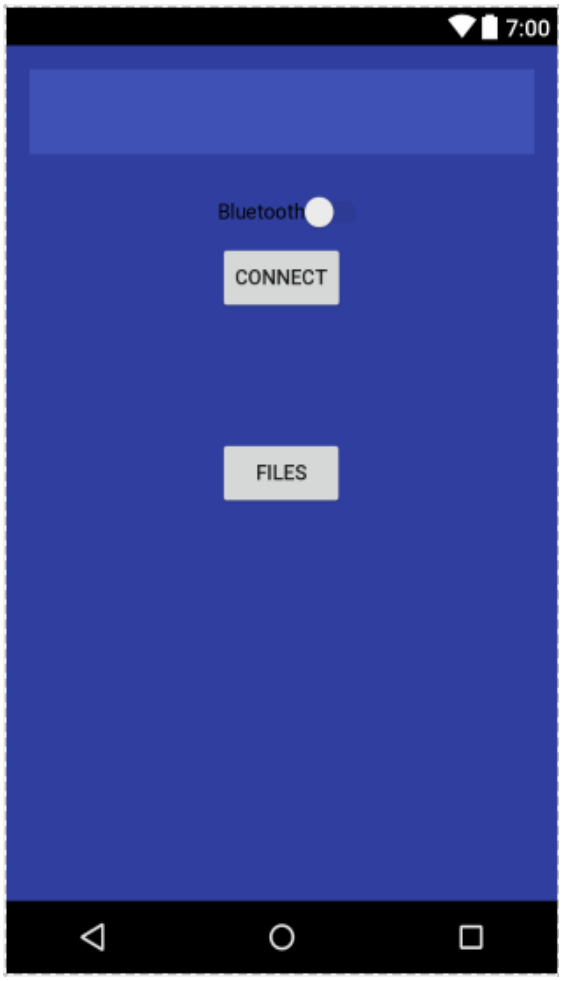


Figure 3.12: Application's Start Screen.

In order to speed up the Bluetooth development and keep the code as less complex as possible we created the BTSupportMethods class to act as a library containing methods for some basic operations that we use repeatedly during the development. First we need to create an object reference to the device's Bluetooth adapter, this is possible with the `getDefaultAdapter()` method provided by the android API. To create a Classic Bluetooth server using the RFCOMM protocol and register our service's UUID to the SDP records to expose it to other devices scanning for our service, we use the `listenUsingInsecureRfcommWithServiceRecord()` method of the newly created adapter object, this method returns a BluetoothServerSocket object. With this method there is no need for a four-digit PIN code to achieve pairing between the devices. Although it is more secure, we chose to not use a PIN code because the client device may not have a screen or an input method for the user to enter the code. By calling the `accept()` method of the BluetoothServerSocket object we get a BluetoothSocket object, with this object we can communicate with the other device to exchange data. As long as we keep the BluetoothServerSocket open and we accept connections we can communicate with multiple devices simultaneously. Every time we accept a connection we start a new thread to handle the communication.

```
try {
    // MY_UUID is the app's UUID string, also used by the client code
    Log.d("SERVERCONNECT", "Waiting for Connection");
    tmp = myAdapter.listenUsingInsecureRfcommWithServiceRecord("myService",
            UUID.fromString("00001101-0000-1000-8000-00805f9b34fb"));//Standard serial port UUID
    String Channel = tmp.toString();
    Log.d("SERVERCONNECT", Channel);
} catch (IOException e) {
    Log.d("SERVERCONNECT", "Could not get a BluetoothServerSocket:" + e.toString());
}
myServerSocket = tmp;

BluetoothSocket socket = null;
try {
    socket = myServerSocket.accept();
    Log.d("SERVERCONNECT", "Connection Accepted:");
} catch (IOException e) {
    Log.d("SERVERCONNECT", "Could not accept connection:" + e.toString());
}

Log.d("SERVERCONNECT", "Now managing Connection:");

ManageConnection(socket);
```

Figure 3.13: Classic Bluetooth Server Code Segment.

The `ManageConnection()` is a method we created to read the data from the Bluetooth socket and store it in a text file in the device.

The Low Energy connection programming is a little more complex. As we mentioned before in the Low Energy communication our application operates as a client so it has to scan for the available devices first and then initiate a connection with them. The available devices are presented in a list to the user. When a list item containing a device is pressed a connection with that device initiates. Every BLE operation causes a callback, that runs in a separated background thread, if it is completed successfully, the callbacks can happen anytime as the application is running so we had to take that into consideration during the development. The methods we had to override in order to handle the callbacks are shown in figure 3.14.

```
// Various callback methods defined by the BLE API.
private final BluetoothGattCallback mGattCallback = new BluetoothGattCallback() {

    @Override
    public void onConnectionStateChange(BluetoothGatt gatt, int status, int newState) {...}

    @Override
    public void onServicesDiscovered(BluetoothGatt gatt, int status) {...}

    @Override
    public void onCharacteristicWrite(BluetoothGatt gatt,
                                      BluetoothGattCharacteristic characteristic, int status)
    {...}


    @Override
    public void onCharacteristicChanged(BluetoothGatt gatt,
                                        BluetoothGattCharacteristic characteristic) {...}

    @Override
    public void onDescriptorWrite(BluetoothGatt gatt,
                                  BluetoothGattDescriptor descriptor, int status) {...}

    @Override
    public void onCharacteristicRead(BluetoothGatt gatt,
                                     BluetoothGattCharacteristic characteristic, int status) {...}
        };
```

Figure 3.14: BLE Callback Methods.

The Android operating system has a major drawback, there is no implementation of a scheduler to store and manage the application's requests to the Bluetooth Low Energy

adapter. This means that if two operations that need access to the adapter are executed in a sequence the second will almost always fail. One solution to this problem is to insert a long enough delay between those two operations to be sure that both will be executed. The difficult thing is to determine a proper delay time that will be long enough for every operation to be completed before another initiates and will not waste system resources for too long. We decided to implement another solution. We used boolean variables as locks to create a state machine that does not allow an operation to initiate while another is still accessing the adapter. The above solutions work and can manage just one connection in an efficient and simple manner, unfortunately the problem persists when we have multiple connections simultaneously. When we establish a connection with the ECG monitor device we send a request to receive its GATT records to find the desired service and retrieve the characteristic associated with the measurement data. After finding the characteristic we write to the descriptor with 00002902-0000-1000-8000-00805f9b34fb UUID code the value 1 in order to enable notifications. This descriptor is defined by the Bluetooth specifications and it has the same UUID in every device. The monitor device sends notifications that the characteristic data has changed. The application read the new ECG value and stores it in a text file. When transmission is over we call the `close()` method of the BluetoothGatt object which represents the GATT server connection.

The application also supports dynamic and static plotting of the ECG signal. To graphically present the ECG data we used the Androidplot library version 0.9.8. The static plotting is handled by the plotting activity we created. We add to the activity's layout.xml file a XYPlot element defined by the library and we present the data on it by using the appropriate methods provided by the library. The dynamic plotting is handled by the BLEConnection activity in a similar way, the key difference is that we represent our data with an ArrayList object that change in size, we use the HISTORY_SIZE variable to set a maximum size for the object and we call the `redraw()` method to re-render the plot with the new data every time we read a new value from the characteristic.

The last feature of our application that connects and feeds the input of the third part of our system is the data uploading to a remote server through the internet. The user has the option to upload the measurement file to a database on demand. We communicate with the server by using the HTTP protocol. The protocol complies to

request-response model and defines four methods for manipulating and accesing data on the server. These methods are:

- **GET:** This method is used usually from the client to retrieve a presentation of the data stored in the server. A GET request should have no other effects on the stored data, there are other methods used to manipulate it.
- **POST:** The POST method requests that the server accept the entity enclosed in the request. This method should be used when we want to store data or other resources to the server.
- **PUT:** The PUT method requests that the enclosed entity be stored under the supplied URI, if the URI already exists it should be updated with the new entity.
- **DELETE:** The DELETE method is used to delete the specified resource or data from the server.

We used the Apache HttpClient API version 4.5.2 to implement the lower level http client needed to send the requests and receive the responses to and from the server. We decided to send the patient and ECG data in a JavaScript Object Notation (JSON) file format. A JSON file stores the data in a key: value format, also it is basically a text file so it can be sent from the client to server easily, consuming very little bandwidth compared to an XML file and can be parsed by any programming language without complex procedures. Another advantage is that it can be manipulated as a JavaScript object directly without any need for conversion, that is really helpful in the rapid development of data presentation forms and websites that the end user can navigate to manipulate and interact with the data. In our system this format synergizes very well with the database we chose because MongoDB stores the data in BSON format that is almost identical to JSON. Our application creates a JSON file with the id, longitude, latitude and points keys for data mapping.

```
{"employees":[
    { "firstName":"John", "lastName":"Doe" },
    { "firstName":"Anna", "lastName":"Smith" },
    { "firstName":"Peter", "lastName":"Jones" }
]}
```

Figure 3.15: JSON file format example [19].

We also used the Google play-services API version 9.6.1 to expose our application to Google's Location API for patient's coordinates acquisition in order to store them in the database. By having geospatial information, we can conduct deeper statistical analysis on the collected data from multiple patients or hospitals and connect deceases to geographic locations and monitor their evolution through time. In addition, by creating and connecting to a Google API client we can take advantage of many APIs that can be useful for future feature development to make the application more versatile. To initiate a connection with our remote server we first create a HttpClient object with the `DefaultHttpClient()` constructor. In order to store data to the database we have to send a POST request to the server, so we create a post object passing the server's URL string to its constructor and adding our JSON file to the request's body with the `setEntity()` method, finally we make the client to execute the request with the `execute()` method and then wait for a response from the server. If the server sends the code 200 the transaction was successful.

```
String URLString = "http://nickey432.microlab.ntua.gr:8080/MedicalDBWebservice/update";
                   HttpClient client = new DefaultHttpClient();
                   HttpPost post = new HttpPost(URLString);
                   post.setEntity(se);
                   response = client.execute(post);
```

Figure 3.16: Code Segment for POST Request Execution.

3.3 RESTful web service and MongoDB

A web service is a service offered by an electronic device to other devices that communicate with each other via the World Wide Web. The most common use for a web service is to provide an object-oriented web-based interface to a database server and is often combined with a mobile application that provides a graphical interface to the user. We developed a RESTful web service to receive and manage the database transactions of our system. RESTful is an architecture that defines ways to create web services that can handle dynamic and heterogeneous data formats without strict and well specified communication rules or protocols. REST systems aim for fast performance, reliability, and the ability to grow, by re-using components that can be managed and updated without affecting the system as a whole, even while it is running. We created the system's web service using the Java Servlet API. A Java servlet is a Java program that extends the capabilities of a server. The servlet technology specifications are defined by Oracle. Servlets have a well-defined lifecycle from the moment they are loaded till they are destroyed. This cycle has three stages, the initialization stage which contains operations that need to be done before the servlet starts accepting requests (e.g. database connector initialization), the service stage this is where the main servlet logic and the database transactions are taking place and lastly the destroy stage, at this stage the servlet should free any resources it had bind, destroy unused objects and close connections with the databases. We can override the methods doGet(), doPost(), doPut(), doDelete() provided by the Servlet API to intercept HTTP requests and manage them.

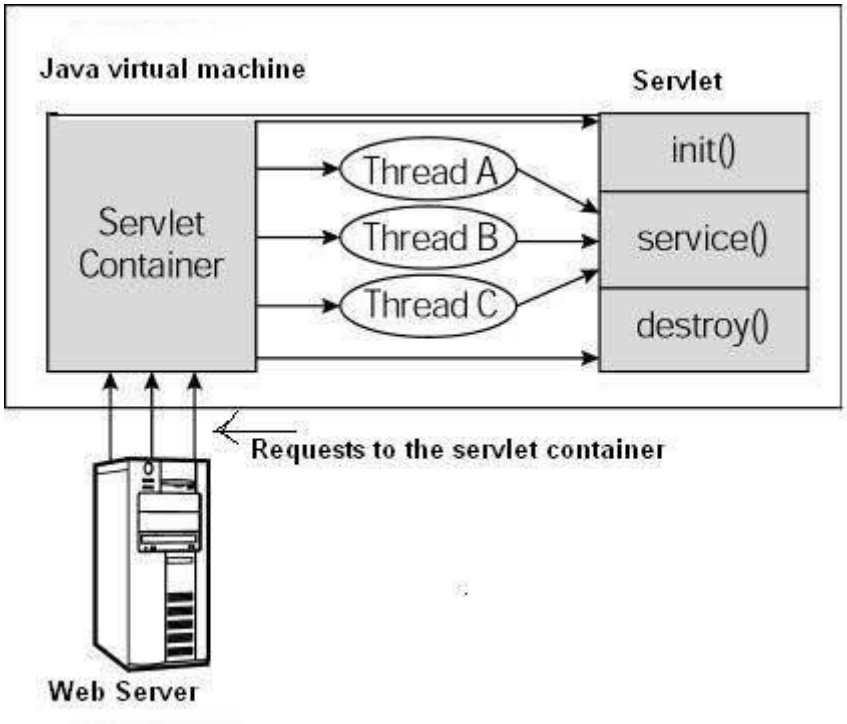


Figure 3.17: Servlet Lifecycle [23].

These methods are executed during the service stage, so when our web service receives a request a new thread is spawned to run the respective method. All the spawned threads end their lifecycle after a response to the client has been sent. Every servlet application must be deployed to a web container. A web container specifies a runtime environment in order to load and create servlet instances, it is also responsible for managing the lifecycle of servlets, mapping a URL to a particular servlet, manage request and response objects. For the deployment of our web service we used the very well documented and popular open source Tomcat Server 8, provided by the Apache Foundation, as a web container. Web services are distributed and deployed in compressed WAR packages. These packages are basically ZIP files that contain Java classes, a mandatory web.xml file and various resources. The web file is needed by the web container in order to load the servlets and map URLs to them, it can also contain commands for the container to use certain resources, interfere with the servlet lifecycle or establish routes between the servlets for communication inside the container.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
             http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
         version="3.1">

    <display-name>Medical Application</display-name>

    <resource-ref>
        <description>Connection Pool</description>
        <res-ref-name>jdbc/ecg_database</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <res-auth>Container</res-auth>
    </resource-ref>

    <servlet>
        <servlet-name>MedicalServlet</servlet-name>
        <servlet-class>WebService.MainServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>MedicalServlet</servlet-name>
        <url-pattern>/greeting</url-pattern>
        <url-pattern>/update</url-pattern>
    </servlet-mapping>
</web-app>
```

Figure 3.18: Web.xml File.

The resources inside the WAR package can be anything that the web service needs to operate, images, css, html or other files generated by other individual programs, URLs and even whole libraries and APIs.

Our web service creates and initiates the database connectors in the `init()` method when it is loaded to Tomcat. We created two connector classes one for the MongoDB and one for the MySQL database, in order to compare them and to make the system compatible with SQL databases too. For the connector and database interaction development we used the MongoDB Driver and the MySQL JDBC Driver APIs. For every received request the web service has to extract the JSON data and handle it programmatically. To be able to do this we defined a JsonObject class in order to map the data to it. For the parsing and mapping of the file we used the Jackson API version 2.7.4.

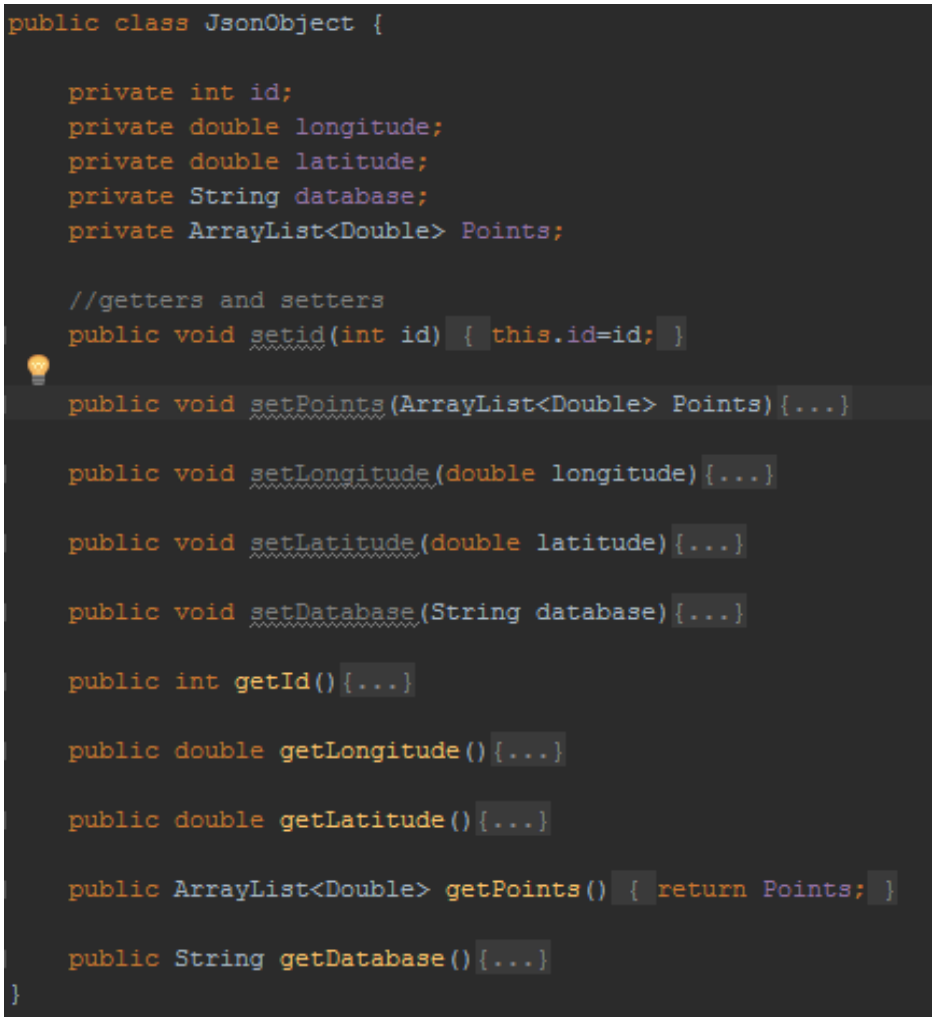


Figure 3.19: Json Object Class.

```
ObjectMapper JsonMapper = new ObjectMapper().
        configure(DeserializationFeature.ACCEPT_SINGLE_VALUE_AS_ARRAY, true);
JsonObject jsonObj = JsonMapper.readValue(json, JsonObject.class);
```

Figure 3.20: JsonObject Mapping Code Segment.

Another part of the web service is the module DBActions, this module contains two classes that define methods we developed in order to implement database transactions. The module can be extended in the future by adding classes that define methods for other databases. When we extract the data from a POST request we call the `InsertDoctoMongoDB()` or the `InserttoMySQL()` method. These methods check if the patient's id already exists in the database, if it does then we simply update the patient's record with the new data, if the patient does not exist in the database we

create a new record to insert him. All these happen inside the overridden `doPost()` servlet method. We have also overridden the `doGet()` method to intercept GET requests in order to execute simple queries on the database and then display the result records on a browser screen.

CHAPTER 4

System Evaluation

4.1 Bluetooth Data Transfer Testing Results

In order to produce safe and useful conclusions about Bluetooth performance we decided to evaluate two major aspects that critically affect device communications, power consumption and packet delay. All the tests were conducted by sending 1000 values or around 11KB of ECG data from the monitor device, running our Linux application, to the Android device. Since Android version 4.4 the operating system has a packet sniffing feature in order to help developers with application debugging, to enable this feature we selected Settings→Developer Options→Enable Bluetooth HCI snoop log on the Android device. Unfortunately, not every manufacturer or device model implements this feature, so this approach might not be successful on every device. For the Bluetooth link packet data collection, we enabled the hci log file on the Android device and imported the file into Wireshark for display and further editing. Wireshark is a packet analyzer program. After filtering out any unnecessary packet information we extracted the log into CSV format file and imported it to Microsoft EXCEL for statistical analysis.

4.1.1 Packet Delay Results

First we tested the Classic Bluetooth link. We measured packet delay by calculating the time difference between two successive received packets. If the first packet was received at $t_{packet1}$ time and the second at $t_{packet2}$ time, then the delay between those packets is $t_{delay} = t_{packet2} - t_{packet1}$. By calculating the times using this formula for all the received packets we can find the mean value and the standard

deviation of the delay. The formulas for mean value and standard deviation calculation are:

- $\overline{x} = \frac{1}{N}\sum_{n=1}^{N} x_n$ (4.1)
- $S_n = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(x_n - \overline{x})^2}$ (4.2)

Where N is the sample size. In our experiments N=1000 packets.

We repeat the test and every time we change the distance between the two devices both in open space and with a wall between them.

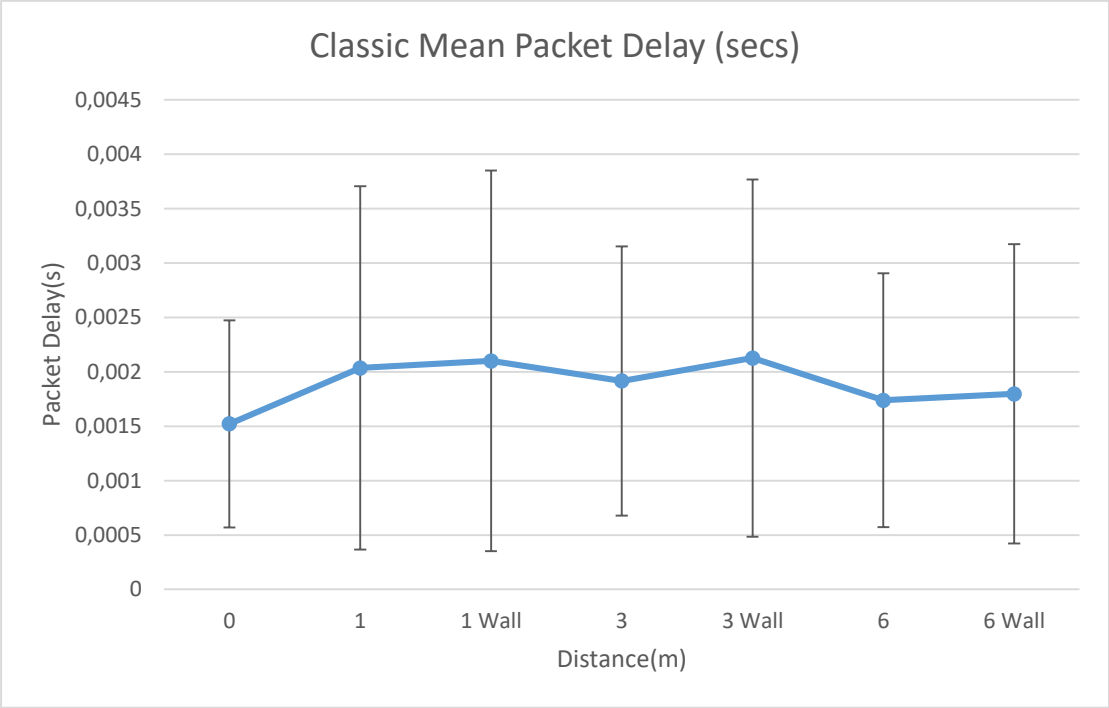


Figure 4.1: Bluetooth Classic Packet Delay Mean Value.

We followed a slightly different approach in calculating packet delay for the Bluetooth Low Energy communication. We took advantage of the GATT profile's specifications, so we measured the delay between the client's GATT read requests and the server's GATT read responses. Let $t_{request}$ be the time client sends the request and $t_{response}$ the time it receives the server's response, then the delay is calculated from the formula

$t_{delay} = t_{response} - t_{request}$. We used the formulas 4.1,4.2 again to calculate the mean value and the standard deviation of packet delay and followed the same testing procedure as in Bluetooth Classic communication. Like before the sample size for our experiment is N=1000 packets. Figures 4.3,4.4 shows the experimental results:

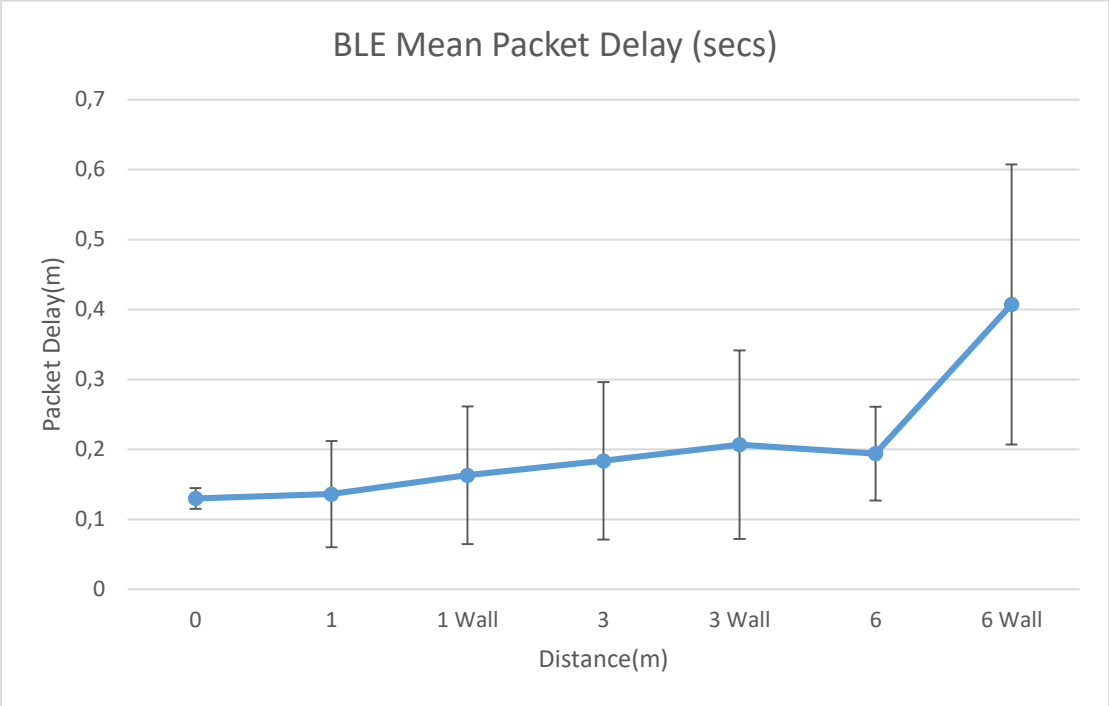


Figure 4.2: Bluetooth Low Energy Packet Delay Mean Value.

In order to simulate real life operation conditions, all the experiments were conducted in a non-isolated environment with Wi-Fi networks present, so interference from Wi-Fi signals was taken into account and is represented in the measurements.

4.1.2 IoT Implementation

In this scope we chose to implement the Bluetooth monitor application on the Raspberry Pi 3 model B. The platform is built around the Broadcom BCM2387 chipset and its technical specifications are:

- **Processor:** 1.2GHz Quad-Core ARM Cortex-A53.
- **GPU:** Dual Core VideoCore IV Multimedia Co-Processor.
- **Memory:** 1GB LPDDR2.
- **GPIO Connector:** 40-pin 2.54 mm (100 mil) expansion header: 2x20 strip Providing 27 GPIO pins as well as +3.3 V, +5 V and GND supply lines.
- **Networking:** 802.11 b/g/n Wireless LAN and Bluetooth 4.1 (Bluetooth Classic and LE), 10/100 BaseT Ethernet.
- **Dimensions:** 85 x 56 x 17mm.

The platform boots from an SD card running any compatible version of the Linux operating system or Windows 10 IoT. We used the Raspbian Jessie Lite version 4.4 as operating system. The Lite distribution has no graphical interface so it needs less storage space on the SD card and utilizes less system resources, as a result OS power consumption is minimal. In order to make the BLE part of our application to operate on the board we cross-compiled the Qt library using the gcc-4.7-linaro-rpi-gnueabihf cross-compiler. We also configured the Qt Creator to be able to compile and deploy our project directly on the board, so we managed to create a fully functional toolchain with an IDE for developing and debugging applications on the Raspberry Pi 3 board. We used the top Linux command in 100 milliseconds interval to measure the CPU utilization of our application. First we test the BLE part of the application. Figures 4.5,4.6 show the results.

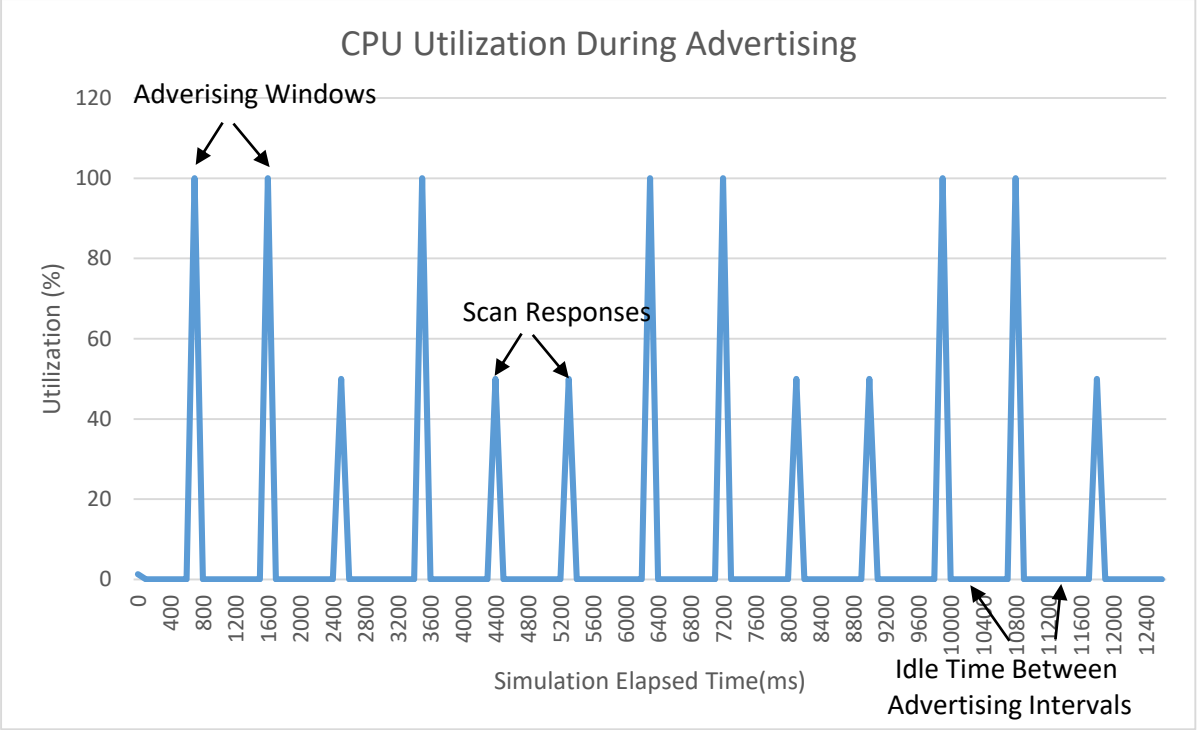


Figure 4.3: Advertising CPU Utilization.

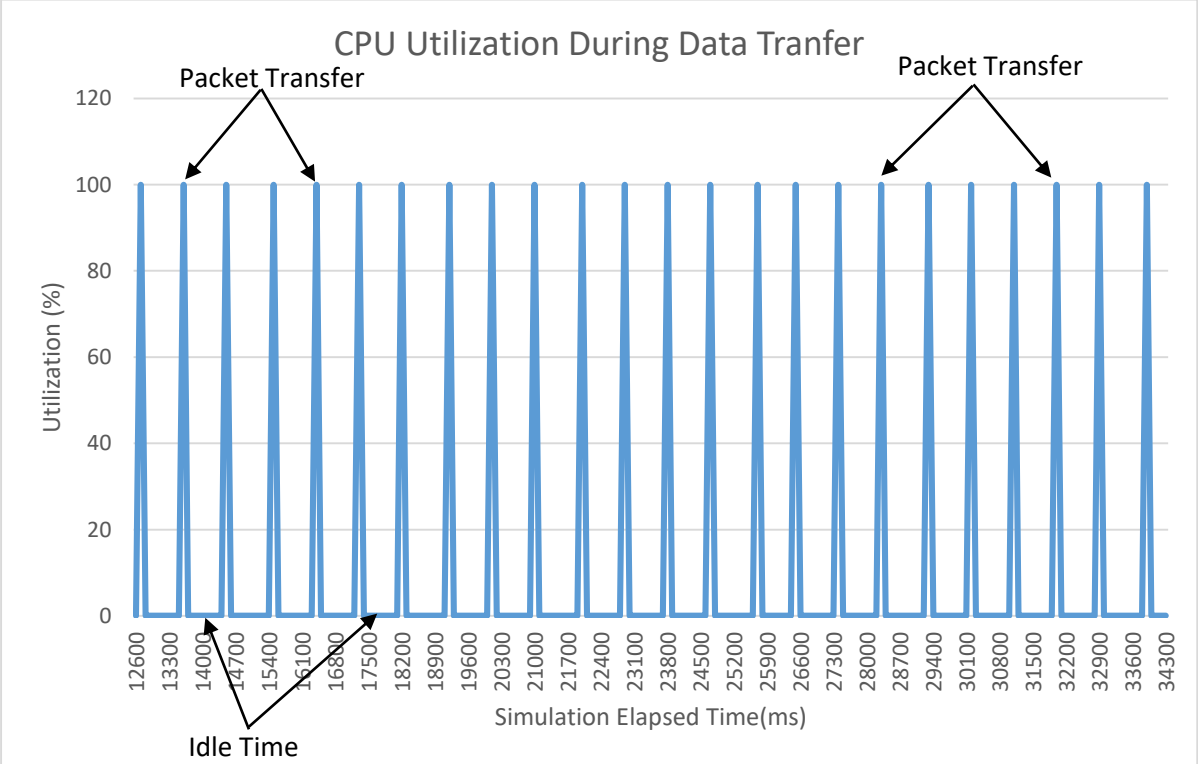


Figure 4.4: Low Energy Data Transfer CPU Utilization.

Figures 4.7,4.8 show the testing results for Classic Bluetooth when we transfer the data as a stream without any delay intervals.

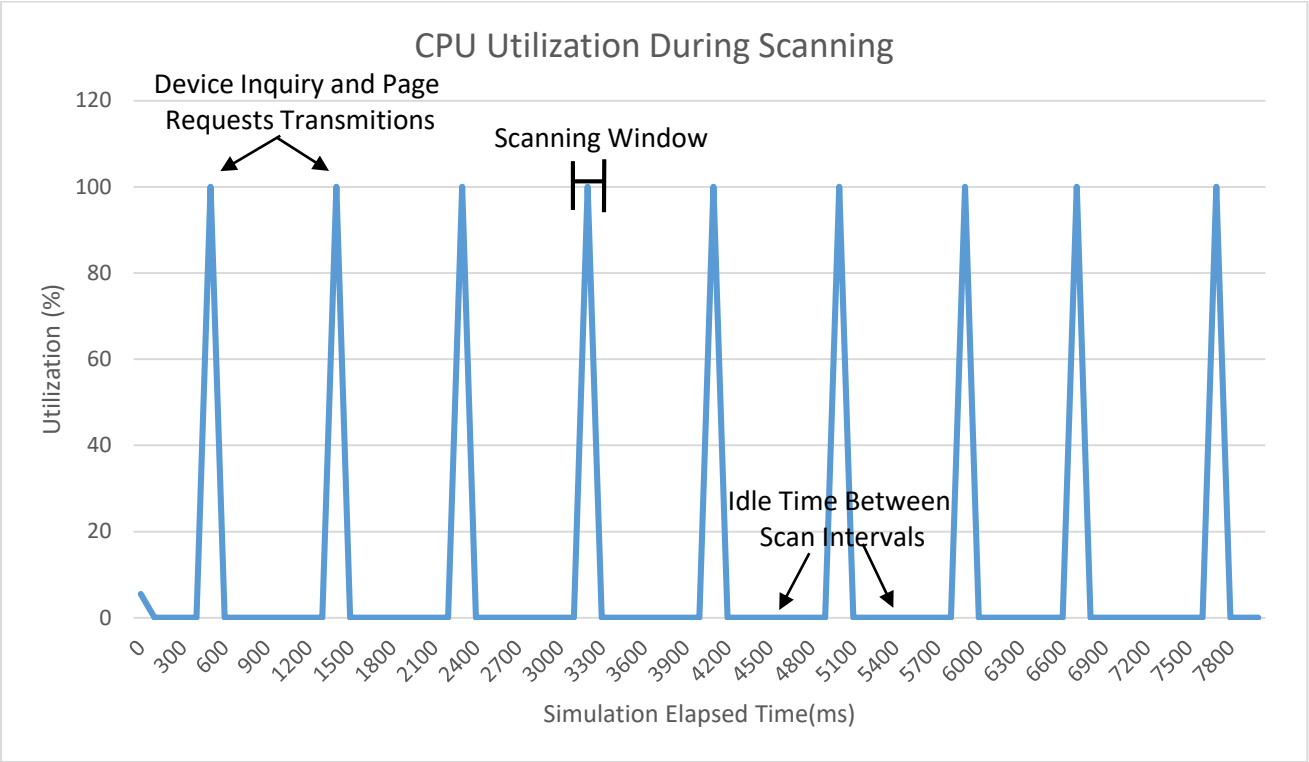


Figure 4.5: Scanning CPU Utilization.

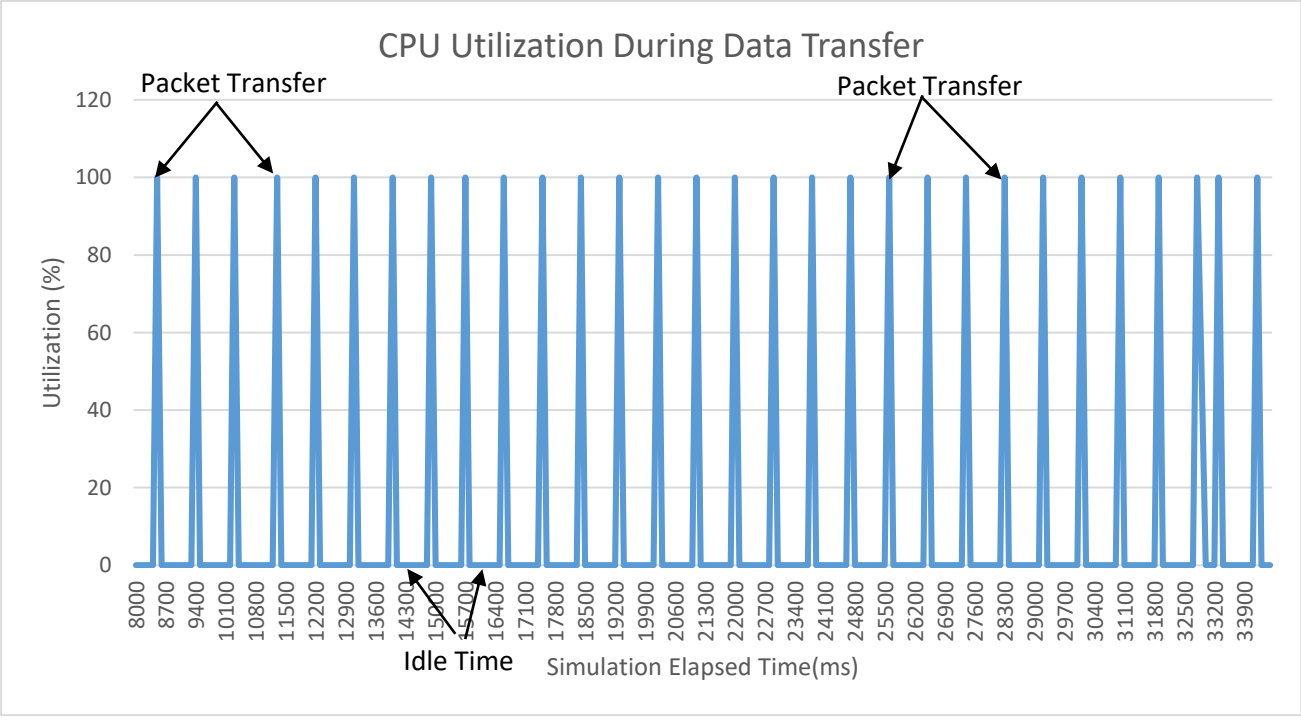


Figure 4.6: Classic Data Transfer CPU Utilization.

We also measured the current consumed by the board from the USB port by using a USB test adapter in order to get an estimation for the power consumption of our application. The board operates at a constant 4,98 V voltage. Let I to be the current drawn from the USB port, so the power consumption can be calculated by the formula $P = V * I$.

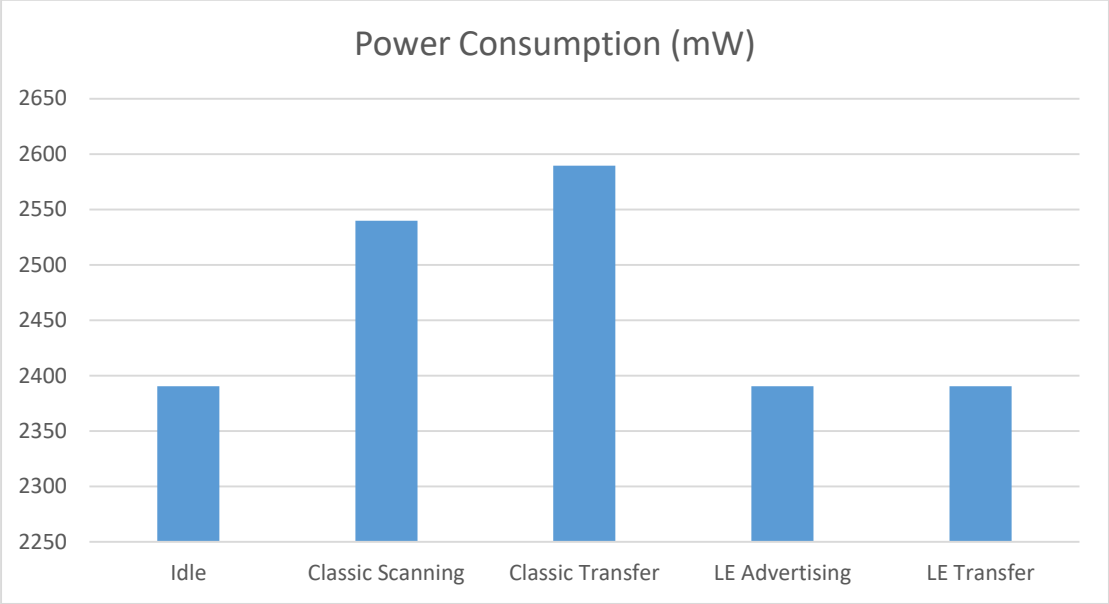


Figure 4.7: Raspberry Power Consumption.

4.2 Android Application Profiling

Android Studio offers a range of tools to track connected devices CPU and GPU utilization, memory management and network usage while an application is running. We used these tools to monitor the amount of resources needed by our application to be able to extract useful conclusions about its operation. We tested the BLE connection first. Figure 4.8 shows the CPU and memory utilization. From 13 seconds to 21 seconds the application is scanning for Bluetooth devices. There is a rise in CPU usage that is caused by the frequency hopping algorithm of the Bluetooth adapter. At 24 seconds we have a connection event and after that the device receives data. During the

connection event there is a spike in CPU utilization because the application exchanges information with the remote device in order to set up a connection link between them. When the connection is established the operating system allocates more memory to the application in order to receive the data. This allocation is represented in the memory graph in Figure 4.8 at 25 seconds just when the connection is established. The arrows on the Figures show these points of interest.

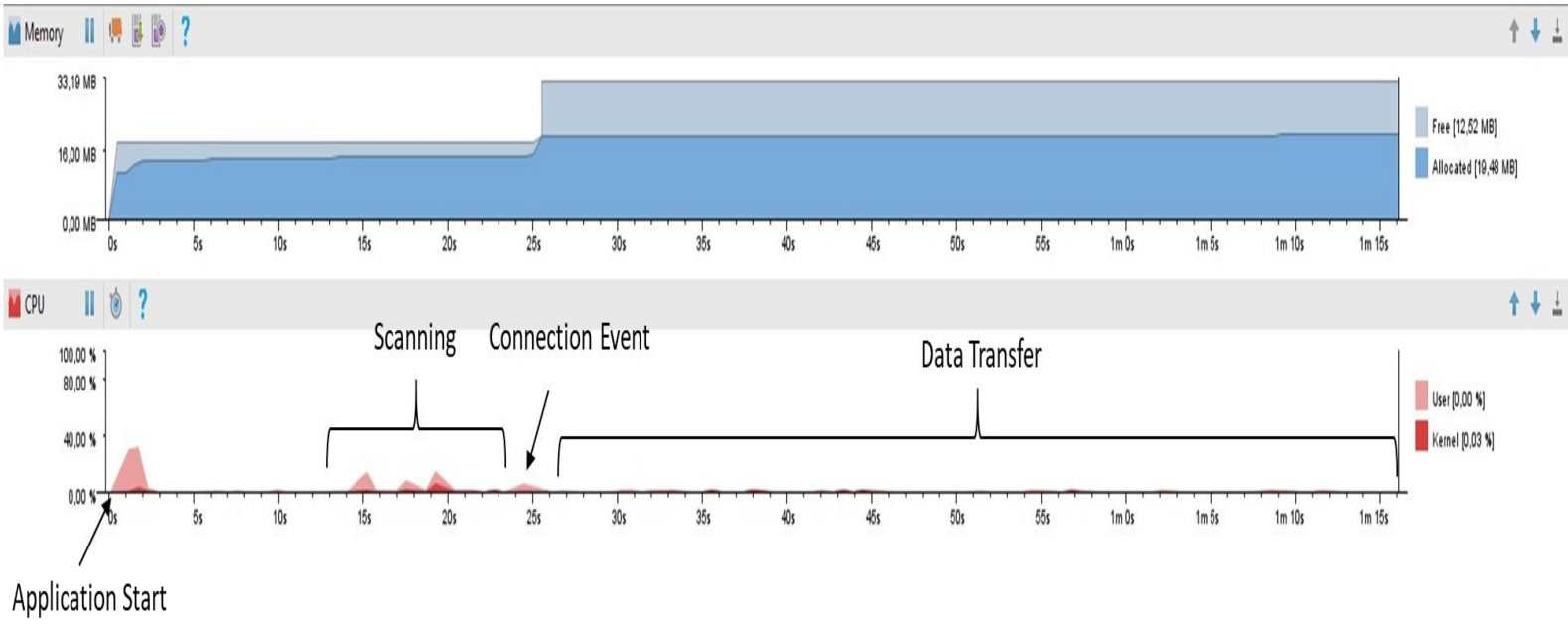


Figure 4.8: BLE CPU and Memory Utilization.

The Classic connection results are shown at Figure 4.9. At 6 seconds the application initiates the server, represented by the small spike in CPU usage and starts to listen for connections, while listening CPU usage is almost 0%. At 35 seconds the connection is established, after that data transfer commences. During the transmission CPU utilization reaches up to 10%.

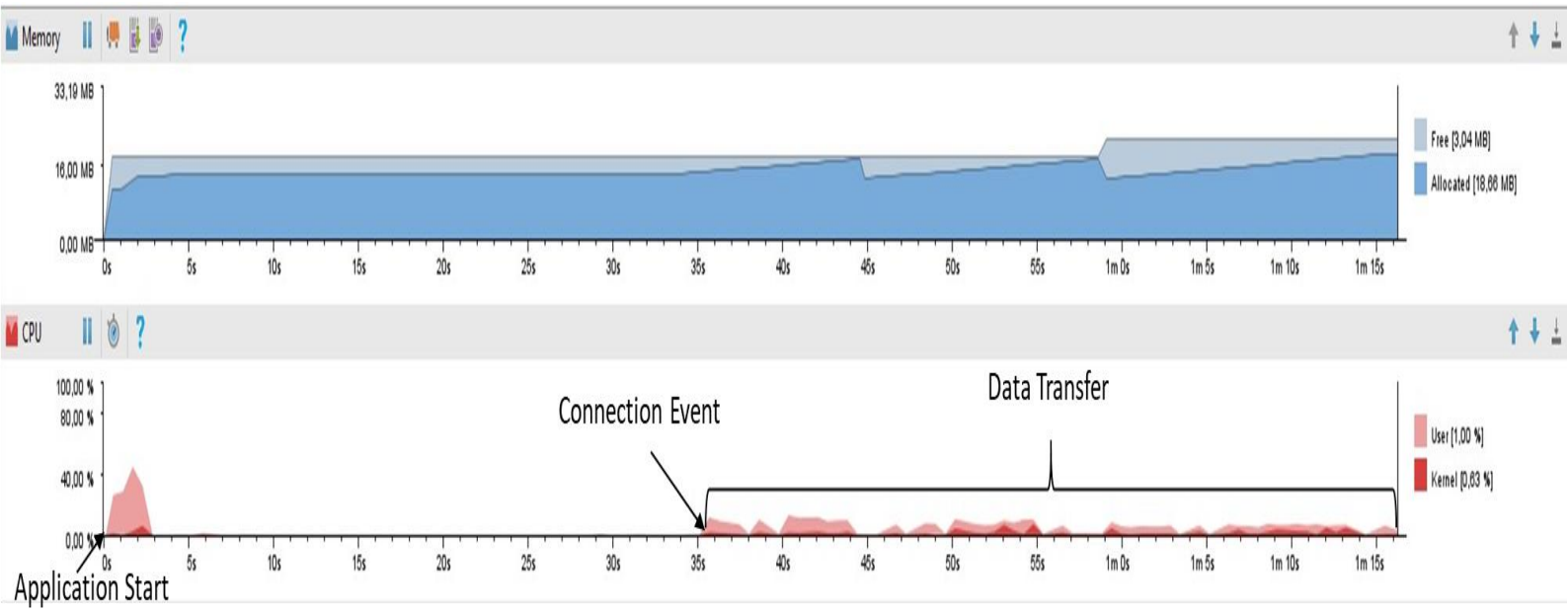


Figure 4.9: Bluetooth Classic CPU and Memory Utilization.

In figure 4.10 we can see the GPU utilization during the ECG dynamic plotting during data transfer. The plotting starts at 18 seconds, before that the small spikes represent the activity's UI screen creation. The spikes represent the time needed to complete graphical processes during dynamic plotting in order for the operating system to render and update the screen.

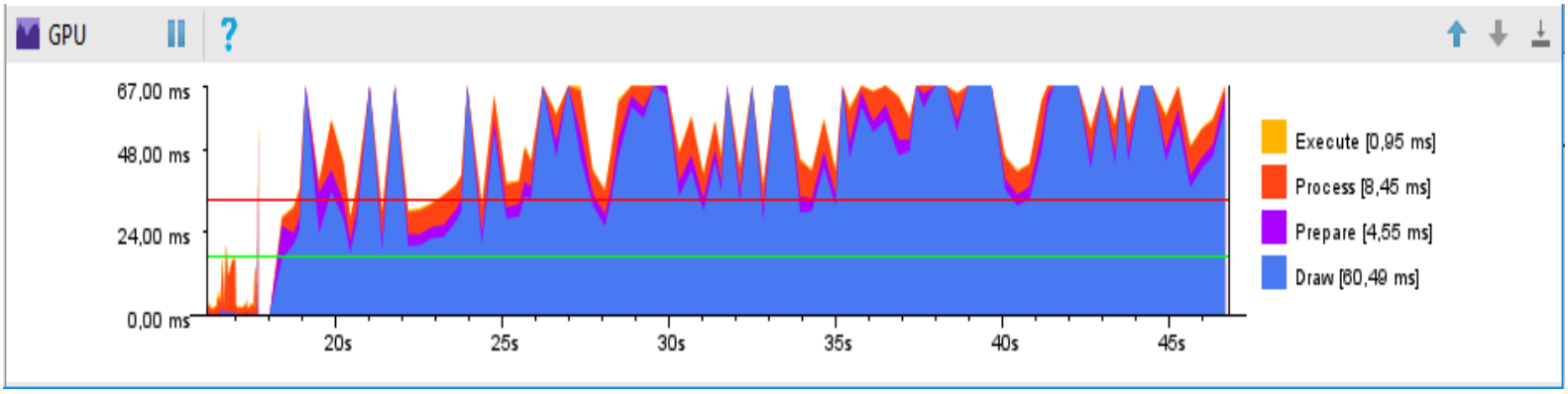


Figure 4.10: BLE GPU Utilization.

4.3 Database Comparison

We test the system's web service with both MySQL and MongoDB in order to compare their performance and how well its of them synergizes with the rest of the system. For the MySQL database we use a schema with two tables, one for the patients information and one that holds every patient's ECG data. The Patients table has the columns ID,Longitude and Latitude. The ECG table has the columns ID and Vals. The ID column in both tables contains the patients unique ids. Because of the No-SQL database's ability to handle dynamic data there was no need to define any schema in MongoDB. In order to test the databases we sent POST requests to the web servide and inside the JSON file we added the "db" key that selects a database. We measured how much time in milliseconds is needed to insert or update one patient's record into the database. All the tests have been conducted by using a laptop equipped with 2,6GHz Intel i7 6700HQ CPU, 8GB DDR4 RAM and SSD hard disk drive as server.

4.3.1 MySQL Testing Results

For all the experiments we used the ID column as index. We tried different configurations in order to improve performance. First we tested the database with just the indexing, then we enabled the web container's connection pooling feature that creates and manages database connections and distributes them to the clients. Lastly, we improved our code by sending all the INSERT and UPDATE commands as batch. That practically means that we do not commit any changes to the database until all the write commands have been completed.

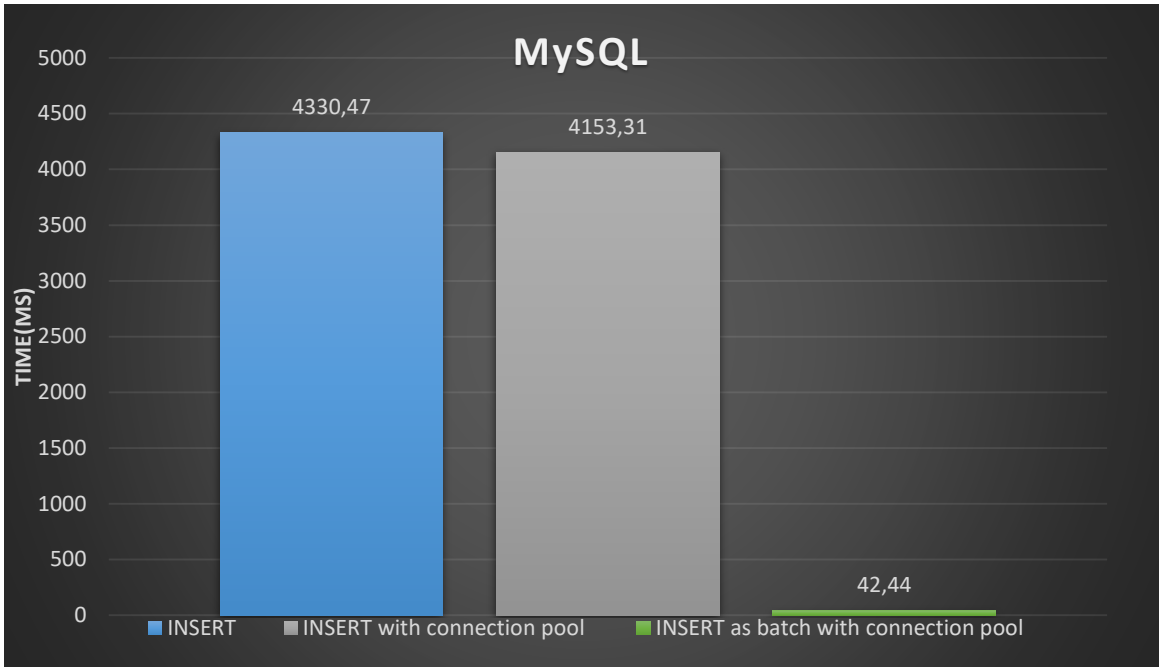


Figure 4.11: MySQL INSERT Test Results.

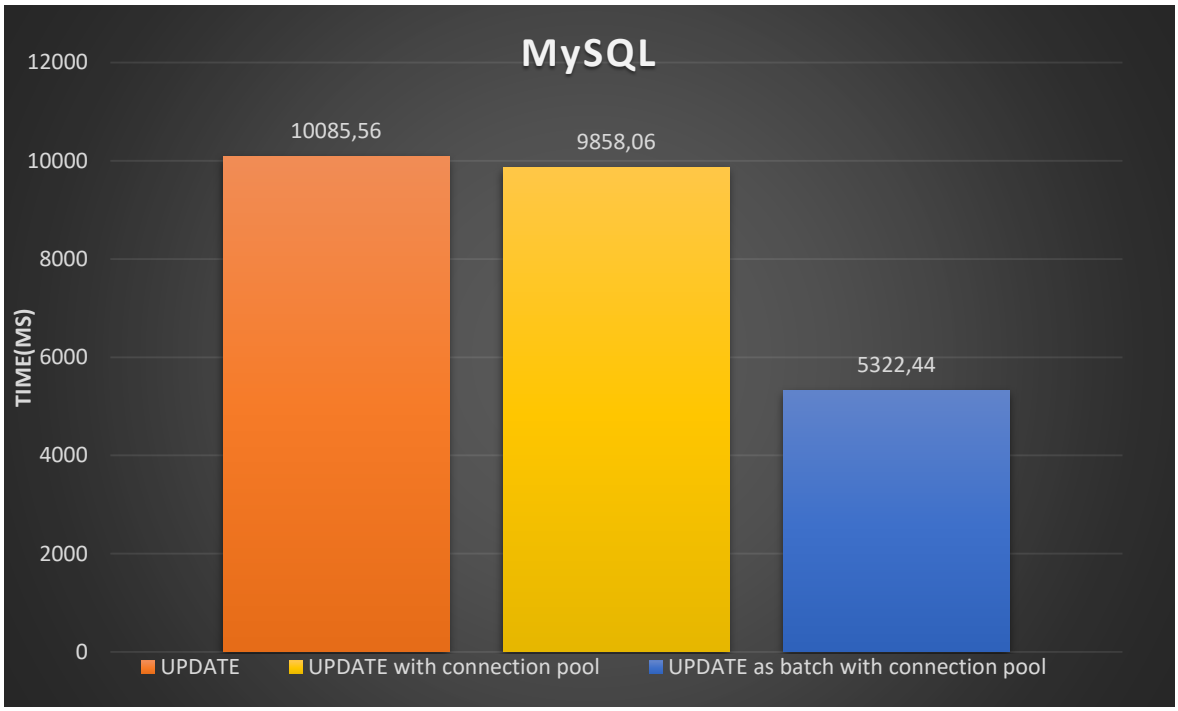


Figure 4.12: MySQL UPDATE Test Results.

We also measured the time needed to retrieve one patient from the database. We queried the database with the patient's ID in order to join the two tables to retrieve all the information.

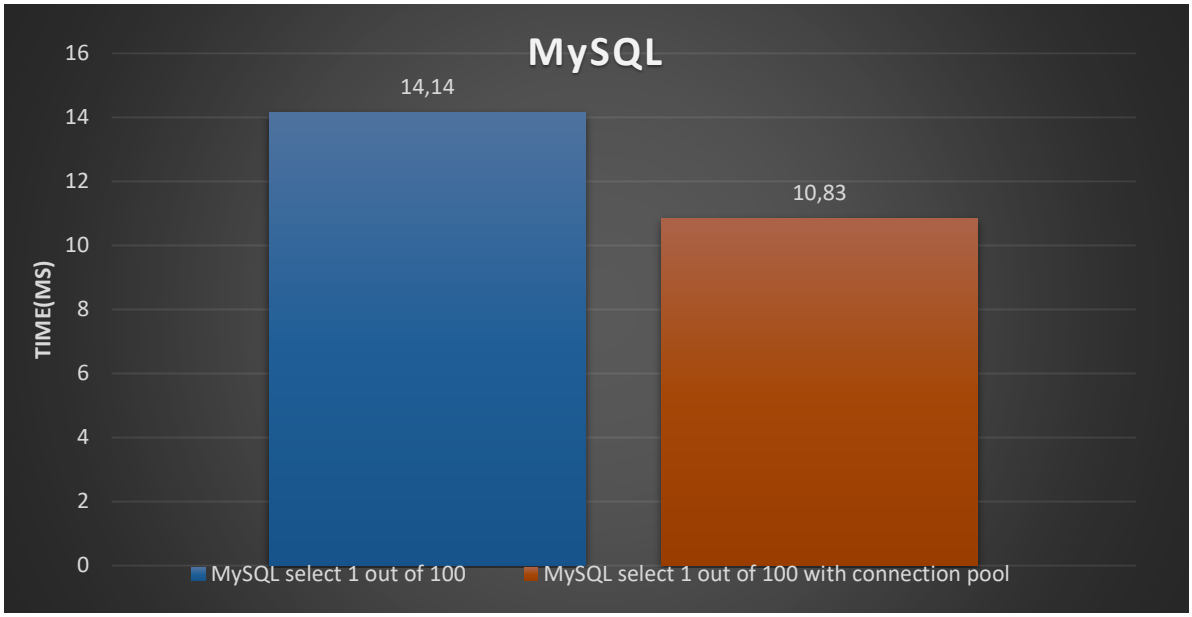


Figure 4.13: MySQL SELECT Test Results.

4.3.2 MongoDB Testing Results

We conducted the same tests for the MongoDB. One difference is that connection pooling is enabled by default and is managed directly by the database instead of the web container, as a result the throughput and speed of the connection manager is a lot better compared to MySQL. We used again the patients' IDs as index and took no actions for further performance improvement.

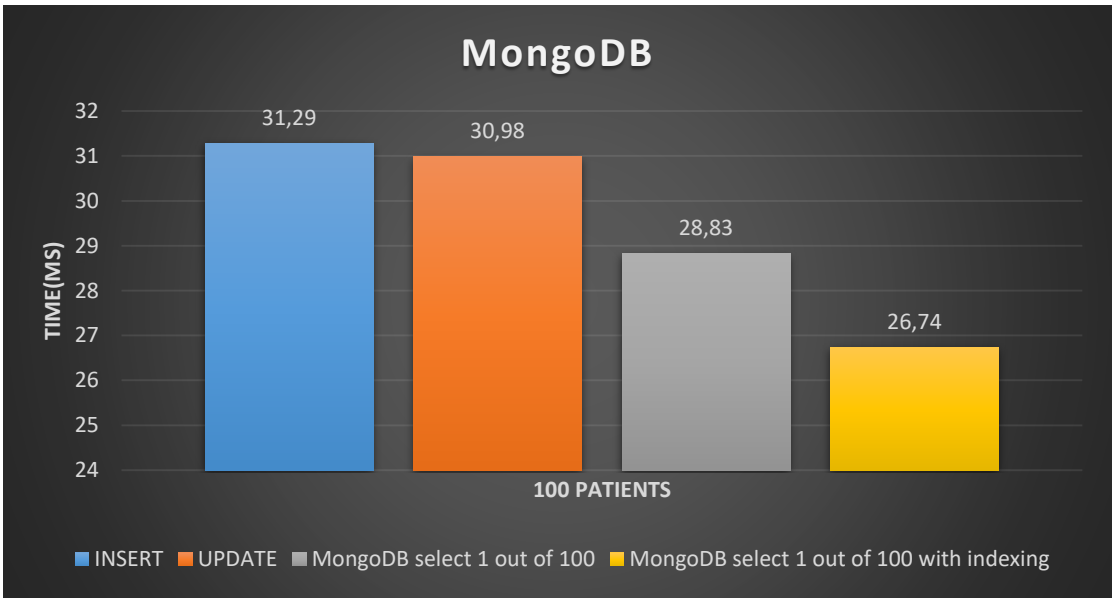


Figure 4.14: MongoDB Test Results.

CHAPTER 5

Conclusion

5.1 Summary

In this thesis we designed, developed and implemented onto an IoT platform a fully functional ECG monitoring system that can operate from the individual patient data acquisition level up to the remote data storing into a central medical database level. By conducting tests and acquiring experimental data we were able to fully evaluate the system and to find solutions to different use case scenarios. When we compared the resource utilization, packet delay and the power consumption results we reached the conclusion that Bluetooth Low Energy is better suited for a wearable device with limited power resources that operates in close range with the android device. Classic Bluetooth is recommended for stationary devices with constant power supply, it caused almost 10% power consumption rise on the Raspberry Pi, or for use cases that involve continuous monitoring of multiple patients at the same time over larger distances, so it would be more useful in hospitals in order to monitor patients in treatment rooms or ICUs 24 hours a day. For the database comparison we chose MongoDB as a better solution because it was a lot faster at storing records without any code improvements whereas we needed to devote a lot of development time in improving MySQL's performance and even then insert times were still slower compared to MongoDB's. Also disabling the write commit until all the write operations where finished can cause performance drops for clients who request data acquisition operations because the database lock will be held for longer periods of time. MySQL was a little faster in retrieving records but it had to join only two tables. If the database grows, so it will contain more tables the retrieving performance will drop significantly. In the next chapter we propose a system extension that makes use of SQL databases.

5.2 Future Work

5.2.1 Database Extension System for Further Statistical Analysis

Although NoSQL databases are very scalable and can handle dynamic data their architecture lacks in representing relations between the data. These databases are suitable to operate alongside web services for fast client serving as they are designed to run efficiently on clusters with no need for expensive hardware. We propose an extension to the system that answers the need to establish relations between the data. Medical applications need to connect ECG, EEG, blood pressure and many more human bio signal data with deceases and patient's history to statistically analyze them and reach to conclusions with greater accuracy by taking account as many factors are available.

The extended system will have NoSQL databases serve as caches for data that are accessed very frequently and can be located and maintained by individual hospitals. Every hospital will be able to store its patients' data in their own database for quick access. At the top level of the system there will be an SQL database with a proper schema that can relate the data acquired by the individual hospitals in order to be analyzed by researchers and can be located in a research center. The two levels will be connected by a manager that will transfer data between them efficiently. This system can achieve two things. Firstly, it improves medical data analysis by adding a relational architecture to data storing. Secondly, the two levels will serve as backup in case one of the databases lose some or all of its records, they will be able to retrieve them from the other level. Lastly, the proposed system can operate alongside the system we presented in this thesis without the need for changes or any redesigning of its parts.

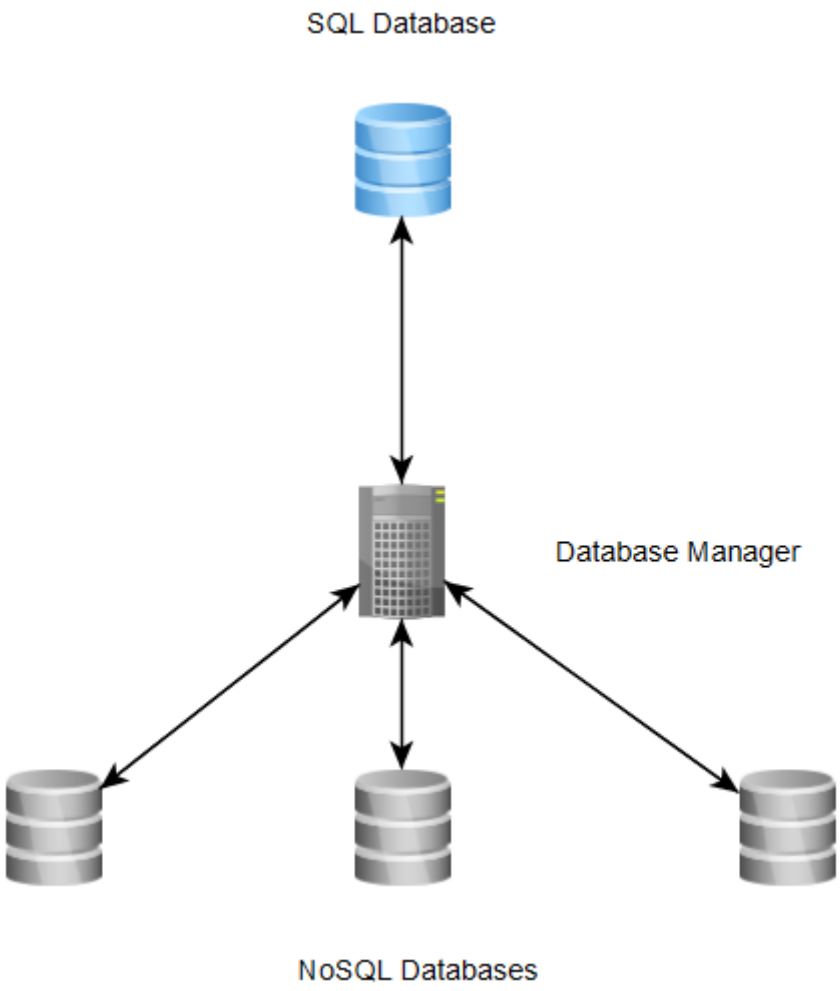


Figure 5.1: Proposed System Hierarchy.

5.2.2 Android Application Improvement

As we have stated before the Android OS does not have a way to manage and schedule commands send to the BLE controller by an application. The development of such manager-scheduler can enhance the IoT capabilities of our application. Also it will be very useful for every application that uses BLE. In addition, the application's UI can be further improved to become more user friendly.

References

Books, Articles & Guides:

[1] Collen, Morris F. 2012. Computer Medical Databases. 1st ed. London: Springer-Verlag London Ltd.

[2] Hareva, David, Clarissa Wijaya, Samuel Lukas, and Hendra Tjahyadi. 2014. "Developing Healthcare Apps On Different Mobile Phone Platform". International Journal Of Information Technology & Computer Science 17 (1): 15-20.

[3] Hillestad, R., J. Bigelow, A. Bower, F. Girosi, R. Meili, R. Scoville, and R. Taylor. 2005. "Can Electronic Medical Record Systems Transform Health Care? Potential Health Benefits, Savings, And Costs". Health Affairs 24 (5): 1103-1117. doi:10.1377/hlthaff.24.5.1103.

[4] Li, Shancang, Li Da Xu, and Shanshan Zhao. 2014. "The Internet Of Things: A Survey". Information Systems Frontiers 17 (2): 243-259. doi:10.1007/s10796-014-9492-7.

[5] McGrath, Michael J, and Cliodhna Ní Scanaill. 2013. Sensor Technologies. 1st ed. Berkeley, CA: Apress.

[6] Meyfroidt, Geert, Fabian Güiza, Jan Ramon, and Maurice Bruynooghe. 2009. "Machine Learning Techniques To Examine Large Patient Databases". Best Practice & Research Clinical Anaesthesiology 23 (1): 127-143. doi:10.1016/j.bpa.2008.09.003.

[7] Perera, Charith, Rajiv Ranjan, Lizhe Wang, Samee U. Khan, and Albert Y. Zomaya. 2015. "Big Data Privacy In The Internet Of Things Era". IT Professional 17 (3): 32-39. doi:10.1109/mitp.2015.34.

[8] Vermesan, Ovidiu, and Peter Friess. 2014. Internet Of Things Applications - From Research And Innovation To Market Deployment. 1st ed. Aalborg: River Publishers.

[9] Weber, Rolf H, and Romana Weber. 2014. Internet Of Things. 1st ed. Berlin: Springer Berlin.

[10] Choi, Jong Soo, Woo Baik Lee, and Poong-Lyul Rhee. 2013. "Cost-Benefit Analysis Of Electronic Medical Record System At A Tertiary Care Hospital". Healthcare Informatics Research 19 (3): 205. doi:10.4258/hir.2013.19.3.205.

[11] Bluetooth Core Specification Version 4.2, Bluetooth SIG, 2014.

[12] App Development: An NHS Guide for Developing Mobile Healthcare Applications, NHS, 2014.

[13] Azariadi, D., Tsoutsouras, V., Xydis, S., & Soudris, D. (2016, May). ECG signal analysis and arrhythmia detection on IoT wearable medical devices. In Modern Circuits and Systems Technologies (MOCAST), 2016 5th International Conference on (pp. 1-4). IEEE.

Web Resources:

[14] Analysis and Interpretation of the Electrocardiogram. Meds.queensu.ca. Retrieved from https://meds.queensu.ca/central/assets/modules/ts-ecg/

[15] Android, I. Introduction to Android | Android Developers. Developer.android.com. Retrieved from https://developer.android.com/guide/

[16] Behind The Numbers: Growth in the Internet of Things. Ncta.com. Retrieved 5 January 2017, from https://www.ncta.com/platform/broadband-internet/behind-the-numbers-growth-in-the-internet-of-things/

[17] Developer Tools & Resources | Bluetooth Technology Website. Bluetooth.com. Retrieved 25 May 2016, from https://www.bluetooth.com/develop-with-bluetooth/developer-resources-tools

[18] Java MongoDB Driver. Docs.mongodb.com. Retrieved 3 July 2016, from https://docs.mongodb.com/ecosystem/drivers/java/

[19] JSON vs XML. W3schools.com. Retrieved 23 August 2016, from https://www.w3schools.com/js/js_json_xml.asp

[20] MySQL Connector/J 5.1 Developer Guide. Dev.mysql.com. Retrieved 2 November 2016, from https://dev.mysql.com/doc/connector-j/5.1/en/

[21] Path to Internet Enabled BLE. (2014). Thevoidptr.wordpress.com. Retrieved 17 September 2016, from https://thevoidptr.wordpress.com/tag/6lowpan/

[22] Raspberry Pi Documentation. Raspberrypi.org. Retrieved 8 December 2016, from https://www.raspberrypi.org/documentation/

[23] Servlets Tutorial. www.tutorialspoint.com. Retrieved 27 June 2016, from https://www.tutorialspoint.com/servlets/

[24] The future is written with Qt: Cross-platform software development for embedded & desktop. Qt. Retrieved 4 July 2016, from https://www.qt.io/

[25] Three flavors of Bluetooth: Which one to choose? (2013). Edn.com. Retrieved from http://www.edn.com/Home/PrintView?contentItemId=4405960

[26] Tutorials - Nordic Developer Zone. Devzone.nordicsemi.com. Retrieved 17 April 2016, from https://devzone.nordicsemi.com/tutorials/17/