

ΙΟΝΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ



Πτυχιακή Εργασία

Ανάπτυξη ενός ευέλικτου, cloudless διαχειρίσιμη κωδικών
χρησιμοποιώντας τεχνολογίες web

Αναστάσιος Παπαλύρας Τζάκρης

Επιβλέπων: Μιχαήλ Στεφανιδάκης

10 Σεπτεμβρίου 2023

Επιβλέπων

Μιχαήλ Στεφανιδάκης, Αναπληρωτής Καθηγητής,
Ιόνιο Πανεπιστήμιο

Τριμελής Επιτροπή

Μιχαήλ Στεφανιδάκης, Αναπληρωτής Καθηγητής,
Ιόνιο Πανεπιστήμιο

Θεόδωρος Ανδρόνικος, Αναπληρωτής Καθηγητής,
Ιόνιο Πανεπιστήμιο

Εμμανουήλ Μάγκος, Καθηγητής,
Ιόνιο Πανεπιστήμιο

Αυτή η προσπάθεια αφιερώνεται στους φοιτητές του Τμήματος Πληροφορικής του Ιονίου
Πανεπιστημίου.

Πρόλογος και Ευχαριστίες

Οι εφαρμογές διαχείρισης κωδικών (Password Managers) αποτελούν ένα αναπόσπαστο κομμάτι της ηλεκτρονικής παρουσίας μιας σημαντικής μερίδας χρηστών [1], μιας και επιτρέπουν την αποθήκευση και ανάκτηση κωδικών για ιστοσελίδες και εφαρμογές εύκολα και με ασφαλή τρόπο. Για διάφορους λόγους, εταιρίες που κατασκευάζουν τέτοιες εφαρμογές συνήθως διατηρούν τον πηγαίο κώδικα απόρρητο, τόσο στις υπηρεσίες που προσφέρουν στο υπολογιστικό σύννεφο, όσο και στις εφαρμογές που εγκαθιστά ο κάθε χρήστης στον προσωπικό του υπολογιστή και τις κινητές συσκευές του.

Η παρούσα πτυχιακή εργασία είχε ως στόχο τη δημιουργία ενός ανοιχτού κώδικα (open source) διαχειριστή κωδικών πρόσβασης ο οποίος, χωρίς να έχει πρόσβαση στα δεδομένα του χρήστη καθ' αυτά, επιτρέπει τον διαμοιρασμό των κωδικών με άλλους χρήστες με ασφαλή τρόπο, παραμένοντας όμως εύκολος στη χρήση.

Για την υλοποίηση του διαχειριστή χρησιμοποιήθηκαν τεχνολογίες διαδικτύου (Go, TypeScript), επιτρέποντας την ταχεία δημιουργία προτύπων και συγγραφή του κώδικα. Για την επίτευξη της προστασίας των δεδομένων των χρηστών χρησιμοποιήθηκαν τεχνικές συμμετρικής κρυπτογράφησης και κρυπτογράφησης δημόσιου κλειδιού, καθώς και αρχιτεκτονική μηδενικής γνώσης (zero knowledge).

Αυτό το έργο δεν θα ήταν δυνατό χωρίς τη βοήθεια του επιβλέποντος καθηγητή κ. Στεφανιδάκη, και τον ευχαριστώ θερμά για την υποστήριξη και καθοδήγησή του.

Preface

Password managers are integral to the digital presence of a significant number of users [1], as they enable easy and secure storage and retrieval of passwords for websites and applications. For various reasons, companies that develop such password management applications typically keep their source code private, including both the services they provide in the cloud and the client applications that users install on their personal computers or mobile devices.

The goal of this thesis was to create an open-source password manager that could facilitate the synchronization of password collections between different users without being able to access their data, while also maintaining user-friendliness.

The implementation of the password manager involved the utilization of web technologies like Go and TypeScript, enabling rapid prototyping and fast development. To ensure the confidentiality of user data, various techniques, including symmetric encryption, public-key cryptography, and zero-knowledge architecture principles, were employed.

This work would not have been possible without the help of my supervisor, Professor Michael Stefanidakis, and I am grateful for his support and guidance.

Περιεχόμενα

	Σελίδα
1. Γενική Περιγραφή	7
1.1. Στόχοι	7
1.2. Καθορισμός απαιτήσεων	7
1.3. Ποια προβλήματα λύνει;	8
2. Εργαλεία και μεθοδολογίες	9
2.1. Go	9
2.2. Gin Web Framework	10
2.3. JavaScript / TypeScript	10
2.4. Svelte & HTML/CSS	11
2.5. MongoDB	12
2.6. Design patterns	12
3. Υλοποίηση	13
3.1. Εφαρμογή Πελάτη	13
3.1.1. Domain	15
3.1.2. Application	17
3.1.3. Διεπαφή Enclave	18
3.1.4. Διεπαφή Remote	19
3.1.5. Υπηρεσίες	20
3.1.6. Υλοποίηση του Enclave	32
3.1.7. Υλοποίηση του Remote	38
3.1.8. HTTP REST server	47
3.1.9. Scheduler	50
3.1.10. Main module	51
3.1.11. Διεπαφή χρήστη	53
3.2. Εφαρμογή Εξυπηρετητή	57
3.2.1. Domain	57
3.2.2. Application	59
3.2.3. HTTP REST server	69
3.2.4. Main module	71
4. Testing Infrastructure	73
5. Building & Deployment	84
5.1. Client	84
5.2. Server	85
5.3. Deployment	85
6. Συμπεράσματα	88
6.1. Μελλοντικές βελτιώσεις	89
7. Βιβλιογραφία	90

1. Γενική Περιγραφή

1.1. Στόχοι

Το λογισμικό που αναπτύχθηκε στα πλαίσια της συγκεκριμένης εργασίας είναι μια εφαρμογή διαχείρισης κωδικών με το όνομα Myst. Σχεδιάστηκε να είναι ένα ασφαλές μέσο αποθήκευσης κωδικών πρόσβασης για ιστοσελίδες, επιτρέποντας παράλληλα τον διαμοιρασμό κωδικών με άλλους χρήστες, όλα μέσω ενός εύκολου στη χρήση γραφικού περιβάλλοντος.

Η εφαρμογή σχεδιάστηκε γύρω από την ιδέα πως ο τελικός χρήστης πρέπει να είναι η μοναδική οντότητα η οποία έχει κυριότητα στα προσωπικά του δεδομένα, όπως κωδικούς πρόσβασης, σημειώσεις ή το ιστορικό πλοήγησης [2]. Τα δεδομένα αυτά πρέπει να αποθηκεύονται με τέτοιο τρόπο έτσι ώστε να μην μπορεί κάποιος άλλος να τα διαβάσει ή τροποποιήσει και, αν γίνει κάποια κακόβουλη τροποποίηση, αυτή πρέπει να ανιχνεύεται και να προειδοποιείται ο χρήστης [3].

Παράλληλα όμως, το λογισμικό θα πρέπει να μπορεί να υποστηρίξει την επέκταση των λειτουργιών του, με όσο το δυνατόν μικρότερη επίπτωση στην κυριότητα των δεδομένων του χρήστη [4]. Απώτερος στόχος είναι ο εμπλουτισμός των δυνατοτήτων του λογισμικού με λειτουργίες που το κάνουν πιο χρήσιμο προς τον χρήστη, όπως για παράδειγμα διαλειτουργικότητα με το υπολογιστικό νέφος ή άλλες συσκευές [5].

1.2. Καθορισμός απαιτήσεων

Αρχικά λοιπόν, ως κύριες απαιτήσεις του λογισμικού ορίστηκαν η δυνατότητα διαχείρισης της συλλογής των κωδικών του χρήστη, καθώς και η ομαδοποίηση των στοιχείων σε φακέλους (keystores), χωρίς τη σύνδεση στο διαδίκτυο. Τα keystores του διαχειριστή έχουν ένα όνομα που ορίζεται από τον χρήστη και αποτελούνται από ένα πλήθος στοιχείων, το καθένα από τα οποία περιλαμβάνει δεδομένα πρόσβασης για έναν ιστότοπο. Ενέργειες όπως η δημιουργία, ανάκτηση και επεξεργασία των στοιχείων αυτών θα πρέπει να μπορούν να γίνουν τοπικά στη συγκεκριμένη συσκευή χωρίς να γίνεται αποστολή δεδομένων στο διαδίκτυο [6].

Επιπλέον, σε περίπτωση που μία τέτοια σύνδεση είναι εδραιωμένη, θα πρέπει να περιορίζεται η επικοινωνία με πόρους που βρίσκονται εκτός του τοπικού δικτύου, και αν αυτή η επικοινωνία είναι αναγκαία, να μην εκθέτει τα προσωπικά δεδομένα του χρήστη με κανέναν τρόπο, ή να μπορεί να γίνει όταν ο χρήστης το επιλέξει (για παράδειγμα κατά τη δημιουργία της συλλογής των κωδικών) [7].

Επιπρόσθετα, θα πρέπει να δίνεται η επιλογή στον χρήστη να συνδεθεί στον εξυπηρετητή της εφαρμογής, με σκοπό την ενεργοποίηση δυνατοτήτων όπως τον διαμοιρασμό της συλλογής κωδικών του με άλλους χρήστες ή συσκευές. Αυτός ο διαμοιρασμός δεδομένων βέβαια πρέπει να λαμβάνει χώρα μέσω ενός ασφαλούς καναλιού, έτσι ώστε οποιοσδήποτε διαμεσολαβητής να μην έχει την ικανότητα να διαβάσει τα δεδομένα του χρήστη. Ο αποστολέας και ο παραλήπτης της πληροφορίας πρέπει να είναι οι μοναδικές οντότητες με την ικανότητα να τα αποκρυπτογραφήσουν, και η ταυτότητά τους πρέπει να έχει επιβεβαιωθεί πριν να ξεκινήσει η διαδικασία διαμοιρασμού των κωδικών [8].

1.3. Ποια προβλήματα λύνει;

Στο παρελθόν έχουν σημειωθεί πολλά περιστατικά στα οποία hackers απέκτησαν πρόσβαση στις βάσεις δεδομένων εταιρειών που διατηρούν τα αρχεία με τους κωδικούς των χρηστών [9]. Αυτή η διαρροή δεδομένων θα είχε αποφευχθεί εάν οι εν λόγω εταιρείες δεν είχαν εξ αρχής πρόσβαση σε τέτοια δεδομένα. Η λειτουργίες του διαχειριστή κωδικών Myst δεν εξαγουν καμία πληροφορία στο cloud χωρίς προηγουμένως να γίνει κρυπτογράφηση της χρησιμοποιώντας τον master password του χρήστη. Η εφαρμογή υιοθετεί μια αρχιτεκτονική “μηδενικής γνώσης” (zero-knowledge architecture) για να επιτευχθεί αυτή η απαίτηση [10].

Επιπρόσθετα, πολλοί χρήστες δεν γνωρίζουν πως τα μέσα μόνιμης αποθήκευσης συνήθως δεν διαγράφουν το περιεχόμενό τους όταν διαγράφεται ένα αρχείο ή γίνεται ολική επαναφορά του μέσου. Μεταδεδομένα, κομμάτια πληροφορίας ή και ολόκληρη η πληροφορία μπορεί να παραμένει αποθηκευμένη στο μέσο, με αποτέλεσμα κάποιος έμπειρος χρήστης που έτυχε να βρει ή να αγοράσει μεταχειρισμένο το αποθηκευτικό μέσο, να επιχειρήσει και καταφέρει ανάκτηση μέρους ή ολόκληρης της συλλογής κωδικών του χρήστη [11]. Μία λύση είναι η κρυπτογράφηση του ίδιου του σκληρού δίσκου (disk-level encryption), αλλά υπάρχει η περίπτωση το setup του χρήστη να μην υποστηρίζει τη λειτουργία ή να είναι προβληματική [12]. Ο διαχειριστής Myst κρυπτογραφεί και υπογράφει όλα τα δεδομένα του χρήστη πριν γραφτούν στον δίσκο, με αποτέλεσμα να είναι πρακτικά αδύνατη η ανάκτηση ή τροποποίησή τους από κακόβουλους χρήστες.

2. Εργαλεία και μεθοδολογίες

2.1. Go

Η Go (αλλιώς γνωστή ως Golang) [13] είναι μια γλώσσα προγραμματισμού γενικού σκοπού που δημιουργήθηκε το 2009 από μια ομάδα προγραμματιστών στην εταιρεία Google και χρησιμοποιείται κυρίως για εφαρμογές που εκτελούνται στο υπολογιστικό νέφος [14]. Είναι μία επιτακτική γλώσσα προγραμματισμού, με το συντακτικό της να μοιάζει με τη γλώσσα C, χωρίς ωστόσο να είναι όσο πολύπλοκη είναι η C (οι προδιαγραφές της γλώσσας αποτελούνται από μερικές δεκάδες σελίδες [15]). Η Go, αν και είναι μία γλώσσα της οποίας ο μεταγλωττιστής δημιουργεί προγράμματα σε γλώσσα μηχανής, έχει αυτόματο σύστημα απελευθέρωσης μνήμης, χρησιμοποιώντας έναν garbage collector.

Από την αρχή της υλοποίησής της είχε ως βασικό στόχο της να κάνει εύκολο τον ταυτοχρονισμό διαφορετικών τμημάτων ενός πολύπλοκου προγραμματιστικού συστήματος [16]. Αυτή η απαίτηση επιτυγχάνεται χρησιμοποιώντας τα λεγόμενα “goroutines”, τα οποία είναι εικονικά υπολογιστικά νήματα που μπορούν να δημιουργηθούν από το περιβάλλον εκτέλεσης πολύ γρήγορα, χωρίς άμεση εξάρτηση προς νήματα του λειτουργικού συστήματος. Η Go επίσης χρησιμοποιεί δομές ταυτοχρονισμού όπως τα channels (κανάλια), τα οποία επιτρέπουν εύκολα κατανοητή ανταλλαγή δεδομένων ανάμεσα σε goroutines.

Ένα αρεστό χαρακτηριστικό της γλώσσας Go είναι η απλότητά της. Δίχως να έχει περίπλοκους συντακτικούς κανόνες, πολλά προβλήματα λύνονται με παρόμοιο τρόπο από διαφορετικούς προγραμματιστές και σε βάθος χρόνου, κάνοντας έτσι εύκολη τη συγγραφή και ανάγνωση του κώδικα. Ακόμη, οι βασικές βιβλιοθήκες της γλώσσας είναι επαρκώς τεκμηριωμένες και προσφέρουν πολλές ανέσεις και σωστά δομημένες αφαιρέσεις, κρύβοντας έτσι αρκετές λεπτομέρειες υλοποίησης, χωρίς όμως να είναι περίπλοκες να κατανοηθούν [17].

Επιπρόσθετα, η Go είναι μια πολύ καλή επιλογή για τη δημιουργία εφαρμογών διαδικτύου. Περιλαμβάνει σαν βασική βιβλιοθήκη έναν πολύ ισχυρό διακομιστή HTTP, ο οποίος μπορεί να επεκταθεί από πληθώρα βιβλιοθηκών και μεσολογισμικών (middleware).

Γενικά, η Go είναι μια εξαιρετική επιλογή για τη συγγραφή εφαρμογών ιστού που απαιτούν υψηλές επιδόσεις, επεκτασιμότητα, ταυτοχρονισμό και ευκολία συντήρησης. Η εστίασή της στην απλότητα και την αναγνωσιμότητα, μαζί με την εκτεταμένη βιβλιοθήκη της την καθιστούν ένα ισχυρό εργαλείο για τη σύγχρονη ανάπτυξη εφαρμογών διαδικτύου.

2.2. Gin Web Framework

Το Gin είναι μια βιβλιοθήκη γραμμένη στη γλώσσα προγραμματισμού Go που χρησιμοποιείται στη συγγραφή εξυπηρετητών για εφαρμογές διαδικτύου [18]. Δημιουργήθηκε για να κάνει πιο απλή και πιο ευχάριστη τη διαδικασία προγραμματισμού τέτοιων εφαρμογών και συγκεκριμένα τη δημιουργία HTTP APIs, καθιστώντας το μία πολύ δημοφιλή λύση για τη δημιουργία ενός RESTful API.

Το Gin κατά κύριο λόγο προσφέρει στους προγραμματιστές έναν εύκολο τρόπο να ορίσουν τις διάφορες διαδρομές HTTP (HTTP routes) που εξυπηρετεί ο διακομιστής τους, καθώς και τη διαδικασία που εκτελείται κάθε φορά που ένας χρήστης καλεί το εκάστοτε route. Τα routes ορίζονται με HTTP ρήματα (GET, POST, PUT, DELETE, κ.ο.κ.) και ένα URL μοτίβο, και παράλληλα μπορούν να ρυθμιστούν και μεσολογισμικά για το κάθε route ή για μια ομάδα από routes.

Ένα ιδιαίτερο χαρακτηριστικό του Gin είναι η υποστήριξη μεσολογισμικών (middleware) και η πληθώρα έτοιμων μεσολογισμικών ως βιβλιοθήκες [19]. Τα μεσολογισμικά χρησιμοποιούνται για να προσθέσουν λειτουργικότητα και συγκεκριμένα pre- και post-processing στα διάφορα HTTP handlers που ορίζει ο χρήστης. Έτσι, διαδικασίες όπως η καταγραφή συμβάντων, η αυθεντικοποίηση και η διαχείριση ασφαμάτων, μπορούν να οριστούν ως μεσολογισμικά για πολλαπλά HTTP routes.

Το Gin επιπρόσθετα περιλαμβάνει διάφορα εργαλεία που βοηθούν την ανάλυση και ερμηνεία των δεδομένων που στέλνονται στα αιτήματα HTTP (HTTP request parsing), καθώς και την εξυπηρέτηση μεμονωμένων στατικών αρχείων ή συστημάτων αρχείων [20].

Το Gin εν κατακλείδι είναι ένα εύκολο στη χρήση, ευέλικτο και υψηλών επιδόσεων web framework, κάνοντας το έτσι μία πολύ καλή επιλογή για τη δημιουργία HTTP RESTful APIs για Go εφαρμογές.

2.3. JavaScript / TypeScript

Η JavaScript είναι μια διερμηνευόμενη γλώσσα προγραμματισμού που επιτρέπει στους προγραμματιστές να δημιουργούν δυναμικές, διαδραστικές ιστοσελίδες και ευέλικτες διεπαφές χρήστη. Η JavaScript χρησιμοποιείται κατά κόρον επειδή είναι συμβατή με τα περισσότερα προγράμματα περιήγησης ιστού, και συνήθως συνοδεύεται από τεχνολογίες όπως την HTML και τη CSS.

Η JavaScript είναι ευρέως διαδεδομένη εξαιτίας του μεγάλου αριθμού από εργαλεία και βιβλιοθήκες ανοιχτού κώδικα που σχεδιάστηκαν για να ενισχύσουν την κατασκευή εφαρμογών με καινούργιες δυνατότητες [21]. Βιβλιοθήκες όπως η React, η Angular και η VueJS, και εργαλεία όπως το Webpack [22] και το Vite [23], βοηθούν τους προγραμματιστές να δημιουργήσουν προοδευτικές διεπαφές χρήστη με μια σχετική ευκολία.

Η TypeScript [24], ως ένα υπερσύνολο της JavaScript, είναι μια γλώσσα προγραμματισμού η οποία ενισχύει το συντακτικό της JavaScript με ένα σύστημα τύπων (type system) και δυνατότητα μεταγλώττισης και πρόληψης λαθών. Περιλαμβάνει εργαλεία που διευκολύνουν την κατασκευή πολύπλοκων εφαρμογών ιστού, παρέχοντας μια ασφαλή εμπειρία ανάπτυξης στην οποία τα περισσότερα λάθη εντοπίζονται κατά τη μεταγλώττιση και όχι κατά την εκτέλεση του κώδικα.

Ένα από τα μεγαλύτερα πλεονεκτήματα της TypeScript σε σχέση με την JavaScript είναι πως προσφέρει ένα σύστημα επισημείωσης και επιβεβαίωσης των τύπων των μεταβλητών κατά τη διαδικασία της μεταγλώττισης, βοηθώντας έτσι τους προγραμματιστές να γράφουν κώδικα με μεγαλύτερη αυτοπεποίθηση. Προσφέρει πληθώρα επιπρόσθετων εργαλείων και συντακτικού, όπως για παράδειγμα διεπαφές (interfaces) και decorators, τα οποία επιτρέπουν στους προγραμματιστές να γράφουν πιο αρθρωτό και επαναχρησιμοποιήσιμο κώδικα [25]. Ακόμη, προσφέρει λειτουργίες όπως τα generics (parametric polymorphism) [26], που επιτρέπουν σε εργαλεία μεταγλώττισης να προσφέρουν με τη σειρά τους περισσότερες βοηθητικές λειτουργίες, για παράδειγμα μέσω ενός ολοκληρωμένου περιβάλλοντος ανάπτυξης εφαρμογών (IDE) [27].

2.4. Svelte & HTML/CSS

Το Svelte είναι ένα JavaScript framework που δημιουργήθηκε από τον Rich Harris το 2016 [28]. Χρησιμοποιείται στη δημιουργία εφαρμογών ιστού υψηλών επιδόσεων και μεγάλης πολυπλοκότητας. Σε αντίθεση με τα πιο γνωστά JavaScript frameworks όπως είναι η React, η Vue.js και η Angular, το Svelte δεν περιλαμβάνει κάποιο περιβάλλον εκτέλεσης, αλλά είναι ένας προ-επεξεργαστής (pre-processor) που μεταφράζει τον κώδικα που γράφει ο χρήστης σε JavaScript (μαζί με HTML και CSS), η οποία αλληλεπιδρά άμεσα με το Μοντέλο Αντικειμένων Εγγράφων (Document Object Model ή DOM) [29].

Διαθέτει ένα ευέλικτο συντακτικό, το οποίο επιτρέπει τη συγγραφή εφαρμογών ποικίλων δυνατοτήτων, χωρίς να έχει απώλειες απόδοσης από την τροποποίηση ενός εικονικού DOM (virtual DOM). Επίσης περιλαμβάνει built-in εργαλεία τα οποία διευκολύνουν τη διαχείριση της κατάστασης της εφαρμογής, χωρίς την εισαγωγή επιπλέον βιβλιοθηκών. Αν και σαν framework είναι σχετικά νέο, πολλοί προγραμματιστές το προτιμούν για την απλότητά του και την πολύ εύκολη και γρήγορη συγγραφή εφαρμογών διαδικτύου [30].

Σε συνδυασμό με το Svelte, χρησιμοποιήθηκαν οι τεχνολογίες HTML/CSS ώστε να περιγράψουν τη δομή των διάφορων frontend components, και τον τρόπο που αυτά εμφανίζονται στην οθόνη. Συγκεκριμένα, για το element styling χρησιμοποιήθηκε SCSS [31], και τα περισσότερα components υλοποιήθηκαν με component-scoped SCSS κανόνες [32].

2.5. MongoDB

Η Mongo είναι μια NoSQL βάση δεδομένων, η οποία έγινε source-available το 2009 από την εταιρεία 10gen που αργότερα μετονομάστηκε σε MongoDB Inc [33]. Η Mongo αποτελείται από ένα document store το οποίο περιλαμβάνει built-in δυνατότητες indexing και επιτρέπει τη δημιουργία εγγράφων σε μορφή παρόμοια με JSON, τα οποία δεν έχουν απαραίτητα κάποιο συγκεκριμένο schema, επιτρέποντας ευελιξία κατά την ανάπτυξη εφαρμογών [34]. Επιλέχθηκε για να αποθηκεύει μακροπρόθεσμα τα δεδομένα των χρηστών της εφαρμογής στο cloud, έτσι ώστε να είναι εύκολη η συγγραφή των αντίστοιχων συναρτήσεων που αποθηκεύουν και ενημερώνουν τις διάφορες οντότητες.

2.6. Design Patterns

Καθ' όλη την έκταση της εφαρμογής, χρησιμοποιήθηκαν διάφορα design patterns έτσι ώστε να επιτευχθεί η υλοποίησή της.

Καταρχάς, έγινε χρήση των SOLID principles σε όλη την έκταση του κώδικα της εφαρμογής, έτσι ώστε ο κώδικας να μην είναι "rigid", δηλαδή να μπορούν να αφαιρεθούν ή τροποποιηθούν components εύκολα, χωρίς να επηρεάζεται η υλοποίηση των υπολοίπων components. Συγκεκριμένα, έγινε η προσπάθεια χρήσης των αρχών Single-Responsibility και Dependency Inversion όπου αυτό ήταν δυνατό [35].

Επιπρόσθετα, έγινε χρήση του Hexagonal/Clean Architecture, δηλαδή του διαχωρισμού των διάφορων components της εφαρμογής σε "στρώσεις" (rings), με πυρήνα τα διάφορα domain entities και το business logic της εφαρμογής. Κάθε layer έχει συγκεκριμένες υποχρεώσεις, κάνοντας πιο εύκολο τον συλλογισμό για την λειτουργία της εφαρμογής, και κάνοντας "pluggable" τα διάφορα implementations [36], για παράδειγμα αυτά των entity repositories.

Επίσης, σε κάποιο βαθμό, ακολουθήθηκαν οι αρχές Domain Driven Design, και συγκεκριμένα χρησιμοποιήθηκαν patterns όπως τα domain repositories, και η διαχείριση των διάφορων οντοτήτων ως Value Objects ή Aggregates [37]. Ακόμη, έγινε η προσπάθεια να γίνει διαχωρισμός του domain από τις λεπτομέρειες υλοποίησης της εφαρμογής, δηλαδή τα domain services να μην εξαρτώνται στα διάφορα adapters, και να γίνεται dependency injection, όπου αυτό είναι δυνατό.

Τέλος, χρησιμοποιήθηκαν patterns όπως το Facade [38], για παράδειγμα για τον ορισμό και υλοποίηση του interface της εφαρμογής, έτσι ώστε να είναι πιο εύκολη η διαχείριση της εφαρμογής ως ένα collection από service methods από όλα τα διάφορα ports της εφαρμογής.

3. Υλοποίηση

Η δημιουργία του διαχειριστή κωδικών αποτελείται από την υλοποίηση δύο διαφορετικών προγραμμάτων, ενός προγράμματος πελάτη (Myst client) και ενός προγράμματος εξυπηρετητή (Myst server). Η συγγραφή και των δύο προγραμμάτων έγινε με χρήση τεχνολογιών διαδικτύου. Συγκεκριμένα, για την εφαρμογή πελάτη χρησιμοποιούνται οι γλώσσες προγραμματισμού Go και TypeScript καθώς και εργαλεία όπως HTML, SCSS και η βιβλιοθήκη Svelte, και για τον εξυπηρετητή γίνεται χρήση της γλώσσας Go.

Η συγγραφή του πυρήνα και των δύο προγραμμάτων έγινε στη γλώσσα Go. Η επιλογή της Go έγινε αρχικά επειδή είναι μια ικανή, ώριμη γλώσσα, με χαμηλού επιπέδου λειτουργικότητα και εξαιρετικές επιδόσεις εκτέλεσης στα μεταγλωττισμένα προγράμματα, καθώς και επειδή μπορεί να δημιουργήσει εφαρμογές οι οποίες δεν εξαρτώνται σε κάποιο περιβάλλον εκτέλεσης που πρέπει να εγκατασταθεί παράλληλα με την εφαρμογή. Έτσι έγινε εύκολη η υποστήριξη όλων των δημοφιλών λειτουργικών συστημάτων και αρχιτεκτονικών επεξεργαστή.

Όσον αφορά στην εφαρμογή πελάτη, χρησιμοποιήθηκε η γλώσσα Go σε συνδυασμό με τη γλώσσα TypeScript και το Svelte framework. Η TypeScript επέτρεψε τη συγγραφή κώδικα με μια σχετική ασφάλεια, εξαιτίας του συστήματος τύπων και τον έλεγχο εγκυρότητας του κώδικα που περιλαμβάνει. Το Svelte μαζί με τις γλώσσες HTML και SCSS βοήθησε στην πολύ γρήγορη υλοποίηση του περιβάλλοντος χρήσης της εφαρμογής πελάτη, καθώς και την επίτευξη μιας φιλικής διεπαφής χρήστη.

Τα δύο διαφορετικά προγράμματα είχαν το καθένα τις δικές του απαιτήσεις εκτέλεσης, περιβάλλοντος εγκατάστασης και τρόπου χρήσης, και έτσι η περαιτέρω ανάλυσή τους θα γίνει ξεχωριστά.

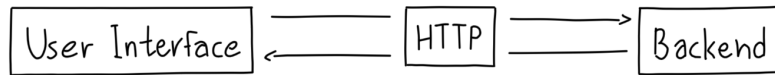
3.1. Εφαρμογή Πελάτη

Η εφαρμογή πελάτη απαρτίζεται από δύο διαφορετικά δομικά στοιχεία, την εφαρμογή παρασκήνιου (backend) και το περιβάλλον χρήσης (user interface). Αυτά τα δύο στοιχεία διασυνδέονται μέσω μιας RESTful HTTP διεπαφής. Κύρια ευθύνη της εφαρμογής είναι να επιτρέπει τη διαχείριση των ακόλουθων οντοτήτων:

- των συλλογών διαπιστευτηρίων ιστοσελίδων του χρήστη (keystores),
- του λογαριασμού του χρήστη (user, credentials), και

- των προσκλήσεων ανάμεσα στον τοπικό χρήστη και σε άλλους χρήστες (invitations).

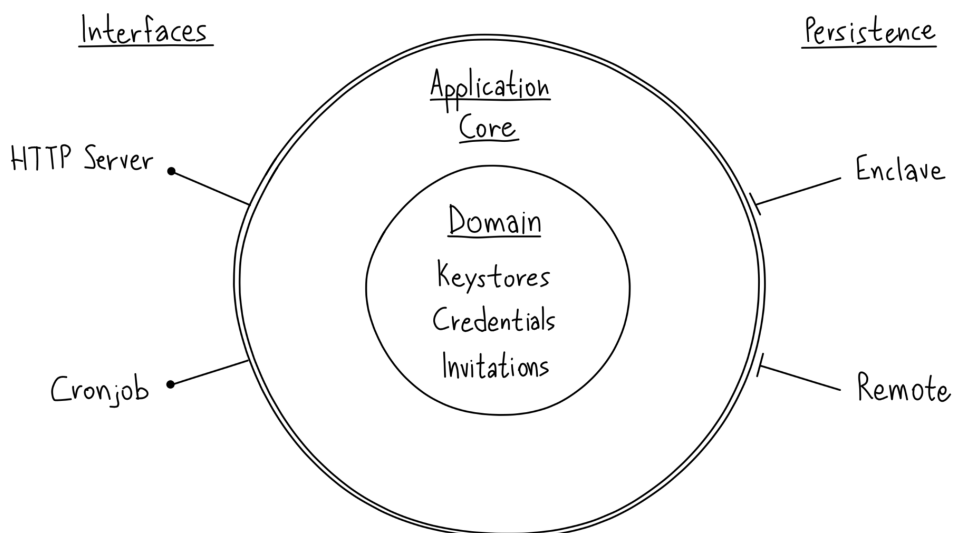
Η εφαρμογή, επιπλέον, είναι υπεύθυνη για διαδικασίες όπως η κρυπτογράφηση των παραπάνω δεδομένων στον τοπικό δίσκο χρησιμοποιώντας έναν “κύριο” κωδικό (master password), και τον συγχρονισμό επιλεγμένων από τον χρήστη στοιχείων με άλλους χρήστες.



Διάγραμμα 3.1. Δομή εφαρμογής πελάτη

Για την αρχιτεκτονική του backend της εφαρμογής χρησιμοποιήθηκε η μεθοδολογία Clean/Hexagonal Architecture σε συνδυασμό με ιδιότητες Domain-Driven Design. Ακολουθώντας τη συγκεκριμένη μεθοδολογία, η εφαρμογή αποτελείται από τον πυρήνα, ο οποίος περιέχει ορισμούς από όλες τις οντότητες για τις οποίες η εφαρμογή έχει κυριότητα (domain entities) και συναρτήσεις (functions) με τις οποίες αυτές οι οντότητες αλληλεπιδρούν μεταξύ τους. Προκειμένου να επιτραπεί η αλληλεπίδραση της εφαρμογής με τον “έξω κόσμο”, ορίζονται συγκεκριμένες υποδοχές (ports), στις οποίες προσκολλώνται με τη χρήση έγχυσης εξαρτήσεων (dependency injection) προσαρμογείς (adapters). Η υλοποίηση/ανάπτυξη των προσαρμογέων δεν επηρεάζει τη λειτουργία της εφαρμογής, εφόσον το συμβόλαιο (contract) παραμένει σταθερό.

Λαμβάνοντας τα παραπάνω υπ’ όψιν, ακολουθεί ένα σχεδιάγραμμα με τη βασική δομή του backend της εφαρμογής πελάτη (Διάγραμμα 3.2)



Διάγραμμα 3.2. Backend εφαρμογής πελάτη

Στη συνέχεια θα μελετήσουμε εκτενώς τη δομή του backend, ξεκινώντας από τον πυρήνα της εφαρμογής και προχωρώντας προς τα εξωτερικά layers.

3.1.1. Domain

Η βασική λειτουργικότητα της εφαρμογής πελάτη αφορά στη διαχείριση του κύκλου ζωής της οντότητας Keystore. Ένα keystore είναι μια συλλογή από καταχωρήσεις διαπιστευτηρίων ιστοσελίδων (entries), η οποία περιέχει επιπρόσθετα μεταδεδομένα (metadata), επιτρέποντας τον διαμοιρασμό της δομής αυτής με άλλους χρήστες. Η δομή ενός keystore παρουσιάζεται στο διάγραμμα 3.3 παρακάτω:

```
type Keystore struct {
    Id          string
    RemoteId    string
    Name        string
    ReadOnly    bool
    Version     int
    Entries     map[string]entry.Entry
    Key         []byte
    CreatedAt   time.Time
    UpdatedAt   time.Time
}
```

Διάγραμμα 3.3. Keystore struct

Ένα keystore μπορεί να αναγνωριστεί με δύο τρόπους: με το Id, το οποίο ορίζεται τοπικά κατά τη δημιουργία του keystore, ή με το RemoteId, το οποίο προέρχεται από τον server, εφόσον ο χρήστης επιλέξει να συγχρονίσει το keystore αυτό με το cloud. Επιπλέον, ένα keystore διαθέτει ένα όνομα (Name), μια έκδοση (Version), η οποία αλλάζει κάθε φορά που γίνεται τροποποίηση του keystore, και καταχωρήσεις με κωδικούς για ιστοσελίδες (Entries). Ακόμη, ορίζονται τα πεδία CreatedAt και UpdatedAt, τα οποία είναι οι ημερομηνίες δημιουργίας και τελευταίας ενημέρωσης του keystore αντιστοίχως. Τέλος, υπάρχουν τα πεδία ReadOnly και Key, τα οποία είναι απαραίτητα για λειτουργίες σχετικές με την αποθήκευση τοπικά αλλά και τον συγχρονισμό του keystore. Το ReadOnly υποδεικνύει εάν ο χρήστης μπορεί να τροποποιήσει το keystore (σε περίπτωση που είναι ο ιδιοκτήτης) ή όχι (σε περίπτωση που κάποιος άλλος χρήστης του παρέχει πρόσβαση ανάγνωσης). Το Key χρησιμοποιείται για την κρυπτογράφηση του keystore κατά την αποθήκευση στον τοπικό δίσκο ή κατά τον συγχρονισμό στο cloud.

Μία ακόμη σημαντική λειτουργία της εφαρμογής πελάτη είναι η δυνατότητα σύνδεσης με έναν απομακρυσμένο server. Για την επίτευξη αυτής της λειτουργίας έχουν οριστεί οι δομές Credentials και User (Διάγραμμα 3.4).

```

type Credentials struct {
    Address      string
    Username     string
    Password     string
    PublicKey    []byte
    PrivateKey []byte
}

type User struct {
    Id          string
    Username    string
    PublicKey   []byte
}

```

Διάγραμμα 3.4. Credentials & User structs

Στα credentials αποθηκεύονται πληροφορίες για τη σύνδεση και ταυτοποίηση του τοπικού χρήστη με έναν απομακρυσμένο server. Το πεδίο Address υποδεικνύει την URL διεύθυνση του server, και τα πεδία Username και Password χρησιμοποιούνται για την επαλήθευση της ταυτότητας του χρήστη. Παράλληλα, τα πεδία PublicKey και PrivateKey είναι τα κλειδιά ασύμμετρης κρυπτογράφησης του χρήστη, τα οποία δημιουργούνται τοπικά κατά την αρχικοποίηση της εφαρμογής, και χρησιμοποιούνται για την εδραίωση της διαδικασίας διαμοιρασμού των keystores με άλλους χρήστες.

Παράλληλα, η δομή User χρησιμοποιείται για την περιγραφή των διαφόρων χρηστών. Εμπεριέχει ένα Id, το οποίο είναι ένας μοναδικός κωδικός ταυτοποίησης ενός χρήστη σε σχέση με τον server, καθώς και το Username και το δημόσιο κλειδί ενός χρήστη.

Προκειμένου να πραγματοποιηθεί ο συγχρονισμός των keystores με έναν ασφαλή τρόπο, χρησιμοποιείται ένα σύστημα προσκλήσεων ανάμεσα στους χρήστες, το οποίο επιτυγχάνεται μέσω της δομής Invitation. Η δομή Invitation παρουσιάζεται στο Διάγραμμα 3.5.

```

type Invitation struct {
    Id                      string
    RemoteKeystoreId       string
    KeystoreName           string
    Inviter                 user.User
    Invitee                user.User
    EncryptedKeystoreKey []byte
    Status                 Status
    CreatedAt              time.Time
    UpdatedAt              time.Time
    DeletedAt              time.Time
    DeclinedAt             time.Time
    AcceptedAt             time.Time
    CancelledAt            time.Time
    FinalizedAt            time.Time
}

```

Διάγραμμα 3.5. Invitation struct

Η δομή Invitation αναγνωρίζεται μοναδικά μέσω του πεδίου Id, και αφορά στην πρόσκληση από τον χρήστη Inviter προς τον χρήστη Invitee, προκειμένου ο Invitee να αποκτήσει read-only πρόσβαση σε ένα Keystore. Ένα invitation χαρακτηρίζεται από το πεδίο Status, το οποίο υποδεικνύει εάν το invitation έχει αποδεχθεί ή όχι από τον άλλο χρήστη, και ελέγχει τι ενέργειες μπορούν να εκτελεστούν. Εάν ο invitee αποδεχθεί το invitation, ο inviter αυτόματα ολοκληρώνει τη διαδικασία ορίζοντας το EncryptedKeystoreKey, το οποίο είναι το Key του Keystore κρυπτογραφημένο με το δημόσιο κλειδί του invitee. Τέλος, ορίζονται τα πεδία CreatedAt, UpdatedAt, AcceptedAt, DeclinedAt, και DeletedAt, τα οποία είναι οι ημερομηνίες κατά τις οποίες ολοκληρώθηκε κάποια αντίστοιχη ενέργεια σχετική με το invitation.

3.1.2. Application

Έχοντας υπ' όψιν το domain, υλοποιήθηκαν οι διάφορες υπηρεσίες (services) της εφαρμογής που ρυθμίζουν τη βασική λειτουργία της. Οι λειτουργίες αυτές, σε συνδυασμό με το domain, οριοθετούν τον πυρήνα (core) της εφαρμογής.

Πιο συγκεκριμένα, η εφαρμογή πελάτη αφορά στον ορισμό και την υλοποίηση της παρακάτω διεπαφής:

```
type Application interface {
    Initialize(password string) (sessionId []byte, err error)
    IsInitialized(sessionId []byte) (bool, error)
    Authenticate(password string) (sessionId []byte, err error)
    HealthCheck(sessionId []byte) error

    CreateKeystore(sessionId []byte, name string) (keystore.Keystore, error)
    DeleteKeystore(sessionId []byte, id string) error
    Keystore(sessionId []byte, id string) (keystore.Keystore, error)
    Keystores(sessionId []byte) (map[string]keystore.Keystore, error)

    CreateEntry(sessionId []byte,
        keystoreId string, website, username, password, notes string) (entry.Entry, error)
    UpdateEntry(sessionId []byte,
        keystoreId string, entryId string, opts UpdateEntryOptions) (entry.Entry, error)
    DeleteEntry(sessionId []byte,
        keystoreId, entryId string) error

    Register(sessionId []byte, username, password string) (user.User, error)
    CurrentUser(sessionId []byte) (*user.User, error)
    SharedSecret(sessionId []byte, userId string) ([]byte, error)

    CreateInvitation(sessionId []byte,
        keystoreId string, inviteeUsername string) (invitation.Invitation, error)
    AcceptInvitation(sessionId []byte, id string) (invitation.Invitation, error)
    DeleteInvitation(sessionId []byte, id string) (invitation.Invitation, error)
    FinalizeInvitation(sessionId []byte,
        invitationId, remoteKeystoreId string, inviteePublicKey []byte) (invitation.Invitation, error)
    Invitations(sessionId []byte) (map[string]invitation.Invitation, error)
    Invitation(sessionId []byte, id string) (invitation.Invitation, error)

    Sync() error
}
```

Διάγραμμα 3.6. Client Application interface

Η δομή application, όπως θα δούμε παρακάτω, υλοποιεί τις μεθόδους της διεπαφής Application, και περιέχει αναφορές προς δύο διεπαφές, τη διεπαφή Enclave και τη διεπαφή Remote. Η διεπαφή Enclave αφορά στο abstraction των λειτουργιών που γίνονται στα τοπικά δεδομένα του χρήστη, ενώ η διεπαφή Remote αφορά στην επικοινωνία με τον server της εφαρμογής μέσω του διαδικτύου.

```
type application struct {
    enclave Enclave
    remote  Remote
    sessions map[string]time.Time
    key      []byte
    mux      sync.Mutex
}
```

Διάγραμμα 3.7. Client application struct

3.1.3. Διεπαφή Enclave

Το Enclave είναι ένα repository που έχει ως ρόλο την αποθήκευση και ανάκτηση των δεδομένων του χρήστη από και προς τον τοπικό δίσκο.

```
type Enclave interface {
    Initialize(password string) (key []byte, err error)
    IsInitialized() (bool, error)
    Authenticate(password string) (key []byte, err error)
    CreateKeystore(key []byte, k keystore.Keystore) (keystore.Keystore, error)
    Keystore(key []byte, id string) (keystore.Keystore, error)
    UpdateKeystore(key []byte, k keystore.Keystore) (keystore.Keystore, error)
    Keystores(key []byte) (map[string]keystore.Keystore, error)
    DeleteKeystore(key []byte, id string) error
    UpdateCredentials(key []byte, creds credentials.Credentials) (credentials.Credentials, error)
    Credentials(key []byte) (credentials.Credentials, error)
}
```

Διάγραμμα 3.8. Enclave interface

Απαιτεί μία αρχικοποίηση (μέθοδος Initialize) πριν να μπορεί να χρησιμοποιηθεί. Αυτό γίνεται έτσι ώστε ο χρήστης να μπορεί να ρυθμίσει τον “master password” του. Για την ανάγνωση των δεδομένων εντός του enclave, σε οποιοδήποτε στάδιο μετά την αρχικοποίηση, χρειάζεται η επαλήθευση της ταυτότητας του χρήστη (μέθοδος Authenticate), η οποία γίνεται με την εισαγωγή του master password. Με την ολοκλήρωση της διαδικασίας επαλήθευσης, δίνεται άμεση πρόσβαση στα δεδομένα του enclave. Η πρόσβαση αυτή όμως έχει ένα χρονικό όριο, μετά το πέρας του οποίου αναιρείται, και οποιαδήποτε εξωτερική διεπαφή θα πρέπει να καλεί τη μέθοδο HealthCheck ανά τακτά χρονικά διαστήματα προκειμένου να μην αφαιρεθεί η πρόσβαση.

Στο enclave αποθηκεύονται όλα τα δεδομένα του χρήστη, και η διεπαφή του περιλαμβάνει συναρτήσεις οι οποίες επιτρέπουν την τροποποίηση των δεδομένων αυτών. Συγκεκριμένα, ορίζονται οι συναρτήσεις CreateKeystore, Keystore, UpdateKeystore και DeleteKeystore, οι οποίες αφορούν στη δημιουργία, ανάκτηση, ενημέρωση και διαγραφή ενός keystore του χρήστη, καθώς και η συνάρτηση Keystores, οι οποία επιστρέφει όλα τα keystores του χρήστη.

Επιπρόσθετα, στο enclave αποθηκεύονται τα στοιχεία πρόσβασης του χρήστη, εφόσον ο χρήστης έχει επιλέξει να χρησιμοποιήσει τις υπηρεσίες της εφαρμογής που λειτουργούν στο υπολογιστικό νέφος. Ορίζονται οι συναρτήσεις SetCredentials και Credentials, οι οποίες επιτρέπουν την ενημέρωση και ανάκτηση των δεδομένων αυτών.

3.1.4. Διεπαφή Remote

Το Remote είναι μια διεπαφή που περιγράφει τη λειτουργικότητα που προσφέρει ο εξυπηρετητής (server) της εφαρμογής στο υπολογιστικό νέφος. Διαθέτει τη συνάρτηση Address, η οποία επιστρέφει το URL του απομακρυσμένου server με τον οποίο θα γίνει η επικοινωνία. Όπως και στη διεπαφή Enclave, ορίζονται οι συναρτήσεις CreateKeystore, Keystore, UpdateKeystore, DeleteKeystore και Keystores, επιτρέποντας τη διαχείριση των keystores του χρήστη στο cloud, καθώς και την πρόσβαση σε keystores άλλων χρηστών, εφόσον έχουν δοθεί τα απαραίτητα δικαιώματα. Τα keystores αυτά όμως δεν είναι αναγνώσιμα χωρίς ένα keystoreKey για το κάθε keystore, το οποίο οι χρήστες πρέπει να γνωρίζουν προκειμένου να βρίσκονται σε θέση να διαβάσουν τα περιεχόμενα ενός keystore.

Μέσω του Remote γίνεται, επίσης, διαχείριση των προσκλήσεων από και προς άλλους χρήστες για πρόσβαση σε keystores, και η ανταλλαγή του keystoreKey που αναφέρθηκε παραπάνω. Συγκεκριμένα είναι εφικτή:

- Η δημιουργία μιας πρόσκλησης (CreateInvitation)
- Η αποδοχή μιας πρόσκλησης (AcceptInvitation)
- Η ακύρωση (εάν είναι ο αποστολέας) ή απόρριψη (εάν είναι ο παραλήπτης) μιας πρόσκλησης (DeclineOrCancelInvitation)
- Η “οριστικοποίηση” μιας πρόσκλησης (FinalizeInvitation). Η οριστικοποίηση πραγματοποιείται από τον δημιουργό του invitation προκειμένου να γίνει ανταλλαγή των credentials που θα δώσουν πρόσβαση στα δεδομένα του keystore.

```

type Remote interface {
    Address() string
    Authenticate(username, password string) error
    Register(username, password string, publicKey []byte) (user.User, error)
    Authenticated() bool
    CurrentUser() *user.User
    CreateKeystore(k keystore.Keystore) (keystore.Keystore, error)
    UpdateKeystore(k keystore.Keystore) (keystore.Keystore, error)
    Keystores(privateKey []byte) (map[string]keystore.Keystore, error)
    DeleteKeystore(id string) error
    CreateInvitation(keystoreRemoteId, inviteeUsername string) (invitation.Invitation, error)
    Invitation(id string) (invitation.Invitation, error)
    AcceptInvitation(id string) (invitation.Invitation, error)
    DeleteInvitation(id string) (invitation.Invitation, error)
    FinalizeInvitation(invitationId string, privateKey, inviteePublicKey []byte,
        k keystore.Keystore) (invitation.Invitation, error)
    Invitations() (map[string]invitation.Invitation, error)
    SharedSecret(privateKey []byte, userId string) ([]byte, error)
}

```

Διάγραμμα 3.9. Remote interface

3.1.5. Υπηρεσίες

Οι υπηρεσίες της εφαρμογής πελάτη αλληλεπιδρούν εξ' ολοκλήρου με τις διεπαφές Enclave και Remote. Ακολουθεί εκτενής ανάλυση της υλοποίησης όλων των υπηρεσιών που προσφέρει η εφαρμογή.

Η πρώτη διαδικασία που πραγματοποιείται από την εφαρμογή πελάτη είναι η αρχικοποίηση του enclave. Προκειμένου οποιοσδήποτε χρήστης της εφαρμογής να βρίσκεται σε θέση να γνωρίζει εάν αυτή η αρχικοποίηση έχει ήδη πραγματοποιηθεί ή όχι, υλοποιήθηκε η συνάρτηση `IsInitialized`, η οποία επιστρέφει `boolean true` μόνο εάν έχει ήδη δημιουργηθεί το enclave.

```

func (app *application) IsInitialized(sessionId []byte) (bool, error) {
    app.mux.Lock()
    defer app.mux.Unlock()

    isInit, err := app.enclave.IsInitialized()
    if err != nil {
        return false, errors.WithMessage(err, "failed to query enclave initialization status")
    }

    if isInit && !app.sessionActive(sessionId) : false, ErrAuthenticationRequired }

    return isInit, nil
}

```

Διάγραμμα 3.10. Συνάρτηση `IsInitialized`

Η αρχικοποίηση επιτυγχάνεται με τη συνάρτηση `Initialize`. Η συνάρτηση `Initialize` δέχεται σαν παράμετρο το master password που επιλέγει ο χρήστης, και δημιουργεί το enclave εάν δεν έχει ήδη δημιουργηθεί (Διάγραμμα 3.11).

```
func (app *application) Initialize(password string) ([]byte, error) {
    app.mux.Lock()
    defer app.mux.Unlock()

    key, err := app.enclave.Initialize(password)
    if err != nil : nil, errors.WithMessage(err, "failed to initialize enclave")

    id, err := app.newSession()
    if err != nil : nil, errors.WithMessage(err, "failed to create new session")

    app.key = key

    return id, nil
}
```

Διάγραμμα 3.11. Συνάρτηση `Initialize`

Έχοντας ολοκληρώσει τη διαδικασία αρχικοποίησης του enclave, ο χρήστης καλείται να χρησιμοποιήσει τη συνάρτηση `Authenticate`, προκειμένου να “ξεκλειδώσει” το enclave και να μπορεί να διαβάσει και τροποποιήσει τα keystores του. Η συνάρτηση `Authenticate` επίσης ελέγχει εάν ο χρήστης έχει συνδεθεί στο παρελθόν με τον remote server, και γίνεται προσπάθεια επανασύνδεσης, εφόσον βέβαια η διεύθυνση με την οποία έγινε η προηγούμενη σύνδεση παραμένει ίδια.

```
func (app *application) Authenticate(password string) ([]byte, error) {
    app.mux.Lock()
    defer app.mux.Unlock()

    key, err := app.enclave.Authenticate(password)
    if err != nil : nil, errors.WithMessage(err, "failed to authenticate against enclave")

    rem, err := app.enclave.Credentials(key)
    if err != nil : nil, errors.WithMessage(err, "failed to query credentials")

    if rem.Address != "" {
        if rem.Address != app.remote.Address() : nil, ErrRemoteAddressMismatch

        err = app.remote.Authenticate(rem.Username, rem.Password)
        if err != nil : nil, errors.WithMessage(err, "failed to authenticate against remote")
    }

    sessionId, err := app.newSession()
    if err != nil : nil, errors.WithMessage(err, "failed to create new session")

    app.key = key

    return sessionId, nil
}
```

Διάγραμμα 3.12. Συνάρτηση `Authenticate`

Έχοντας ολοκληρώσει τη διαδικασία του authentication, το περιβάλλον χρήσης θα πρέπει να “ειδοποιεί” το application ότι η σύνδεση θα πρέπει να παραμείνει ανοιχτή. Αυτό γίνεται έτσι ώστε, σε περίπτωση που ο χρήστης παραμείνει αδρανής για κάποιο χρονικό διάστημα, να κλειδώνει αυτόματα το enclave, έτσι ώστε να μην είναι εύκολη η πρόσβαση από άλλους χρήστες που τυχαίνει να χρησιμοποιήσουν αργότερα τη συσκευή. Η HealthCheck συνάρτηση προσφέρεται για αυτόν ακριβώς το λόγο.

```
func (app *application) HealthCheck(sessionId []byte) error {  
    app.mux.Lock()  
    defer app.mux.Unlock()  
  
    key := string(sessionId)  
    app.sessions[key] = time.Now()  
    return nil  
}
```

Διάγραμμα 3.13. Συνάρτηση HealthCheck

Στην περίπτωση που ο χρήστης δεν έχει ήδη δημιουργήσει λογαριασμό στον server, αυτό μπορεί να γίνει με τη συνάρτηση Register.

```

func (app *application) Register(sessionId []byte, username, password string) (user.User, error) {
    app.mux.Lock()
    defer app.mux.Unlock()

    if !app.sessionActive(sessionId) {
        return user.User{}, ErrForbidden
    }

    var mustInit bool

    creds, err := app.enclave.Credentials(app.key)
    if errors.Is(err, ErrCredentialsNotFound) {
        mustInit = true
    } else if err != nil {
        return user.User{}, errors.WithMessage(err, "failed to query credentials")
    }

    if !mustInit && creds.Address != "" && creds.Address != app.remote.Address() {
        return user.User{}, ErrRemoteAddressMismatch
    }

    u, err := app.remote.Register(username, password, creds.PublicKey)
    if err != nil {
        return user.User{}, errors.WithMessage(err, "failed to register user")
    }

    creds.Address = app.remote.Address()
    creds.Username = username
    creds.Password = password

    _, err = app.enclave.UpdateCredentials(app.key, creds)
    if err != nil {
        return user.User{}, errors.WithMessage(err, "failed to update credentials")
    }

    return u, nil
}

```

Διάγραμμα 3.14. Συνάρτηση Register

Η συνάρτηση Register δημιουργεί ένα καινούργιο λογαριασμό στον server και αυτομάτως συνδέει τον χρήστη. Κατά το initialization έχει δημιουργηθεί ένα ζεύγος (keypair) δημόσιου-ιδιωτικού κλειδιού, από το οποίο το δημόσιο κλειδί, μαζί με το επιλεγμένο username και password, αποστέλλονται στον server προκειμένου να δημιουργηθεί ο λογαριασμός. Εάν η δημιουργία του λογαριασμού ήταν επιτυχής, τότε ενημερώνονται τα credentials του χρήστη και αποθηκεύονται στο enclave.

Η συνάρτηση CurrentUser επιτρέπει τον έλεγχο της σύνδεσης με τον απομακρυσμένο server. Εάν η σύνδεση είναι εδραιωμένη, τότε επιστρέφεται nil (κανένας χρήστης), διαφορετικά επιστρέφεται ο λογαριασμός του χρήστη που έχει συνδεθεί (User struct).

```

func (app *application) CurrentUser(sessionId []byte) (*user.User, error) {
    app.mux.Lock()
    defer app.mux.Unlock()

    if !app.sessionActive(sessionId) : nil, ErrForbidden ↵

    rem, err := app.enclave.Credentials(app.key)
    if err != nil {
        return nil, errors.WithMessage(err, "failed to query credentials")
    }

    if rem.Address == "" {
        return nil, ErrCredentialsNotFound
    }

    u := app.remote.CurrentUser()
    if u == nil : nil, nil ↵

    u.PublicKey = rem.PublicKey

    return u, nil
}

```

Διάγραμμα 3.15. Συνάρτηση CurrentUser

Όσον αφορά στα keystores, η συνάρτηση CreateKeystore δημιουργεί το keystore και το αποθηκεύει στο enclave, κάνοντας πρώτα τους απαραίτητους ελέγχους, για παράδειγμα για τη εγκυρότητα και μοναδικότητα του ονόματός του.

```

func (app *application) CreateKeystore(sessionId []byte, name string) (keystore.Keystore, error) {
    app.mux.Lock()
    defer app.mux.Unlock()

    if !app.sessionActive(sessionId) : keystore.Keystore{}, ErrForbidden ↵

    name = strings.TrimSpace(name)

    if len(name) == 0 || len(name) > 24 : keystore.Keystore{}, ErrInvalidKeystoreName ↵

    ks, err := app.enclave.Keystores(app.key)
    if err != nil {
        return keystore.Keystore{}, errors.WithMessage(err, "failed to query keystores")
    }

    // do not allow keystores with the same name
    for _, k2 := range ks {
        if k2.Name == name : keystore.Keystore{}, ErrInvalidKeystoreName ↵
    }

    k := keystore.New(keystore.WithName(name))

    k, err = app.enclave.CreateKeystore(app.key, k)
    if err != nil : keystore.Keystore{}, errors.WithMessage(err, "failed to create keystore") ↵

    return k, nil
}

```

Διάγραμμα 3.16. Συνάρτηση CreateKeystore

Η συνάρτηση DeleteKeystore αφαιρεί το keystore από το enclave.

```
func (app *application) DeleteKeystore(sessionId []byte, id string) error {
    app.mux.Lock()
    defer app.mux.Unlock()

    if !app.sessionActive(sessionId) : ErrForbidden ↵

    k, err := app.enclave.Keystore(app.key, id)
    if err != nil : errors.WithMessage(err, "failed to get keystore") ↵

    if k.RemoteId != "" {
        err = app.remote.DeleteKeystore(k.RemoteId)
        if err != nil : errors.WithMessage(err, "failed to delete keystore from remote") ↵
    }

    err = app.enclave.DeleteKeystore(app.key, id)
    if err != nil : errors.WithMessage(err, "failed to delete keystore from enclave") ↵

    return nil
}
```

Διάγραμμα 3.17. Συνάρτηση DeleteKeystore

Για την ανάκτηση των keystores χρησιμοποιούνται κατά βάση οι συναρτήσεις Keystore και Keystores. Η συνάρτηση Keystore απλώς επιστρέφει το Keystore, έτσι όπως είναι στο enclave.

```
func (app *application) Keystore(sessionId []byte, id string) (keystore.Keystore, error) {
    app.mux.Lock()
    defer app.mux.Unlock()

    if !app.sessionActive(sessionId) : keystore.Keystore{}, ErrForbidden ↵

    k, err := app.enclave.Keystore(app.key, id)
    if err != nil : keystore.Keystore{}, errors.WithMessage(err, "failed to get keystore") ↵

    return k, nil
}
```

Διάγραμμα 3.18. Συνάρτηση Keystore

Η συνάρτηση Keystores αρχικά ανακτά όλα τα keystores που έχουν αποθηκευτεί τοπικά. Εάν ο χρήστης έχει συνδεθεί στον server, τότε ανακτά και όλα τα keystores στα οποία ο χρήστης έχει πρόσβαση, και ακολούθως επιστρέφονται όλα τα keystores. Στην περίπτωση που ένα keystore υπάρχει και τοπικά και στον sever, επιστρέφεται η τοπική έκδοση.

```

func (app *application) Keystores(sessionId []byte) (map[string]keystore.Keystore, error) {
    app.mux.Lock()
    defer app.mux.Unlock()

    isInit, err := app.enclave.IsInitialized()
    if err != nil : nil, errors.WithMessage(err, "failed to query enclave initialization status")

    if isInit && !app.sessionActive(sessionId) : nil, ErrAuthenticationRequired

    ks, err := app.enclave.Keystores(app.key)
    if err != nil : nil, errors.WithMessage(err, "failed to get keystores")

    for id, k := range ks {
        ks[id] = k
    }

    if app.remote.Authenticated() {
        // add remote keystores too
        rem, err := app.enclave.Credentials(app.key)
        if err != nil : nil, errors.WithMessage(err, "failed to get credentials")

        rks, err := app.remote.Keystores(rem.PrivateKey)
        if err != nil : nil, errors.WithMessage(err, "failed to get remote keystores")

        for _, rk := range rks {
            if _, ok := ks[rk.Id]; !ok {
                ks[rk.Id] = rk
            }
        }
    }

    return ks, nil
}

```

Διάγραμμα 3.19. Συνάρτηση Keystores

Προκειμένου να τροποποιηθούν τα entries εντός ενός keystore, χρησιμοποιούνται οι συναρτήσεις CreateEntry, UpdateEntry και DeleteEntry. Και οι τρεις αυτές συναρτήσεις πρώτα ανακτούν το keystore από το enclave, ενημερώνουν το keystore αναλόγως, και τέλος καλούν τη συνάρτηση UpdateKeystore του enclave για να αποθηκευτεί το keystore. Να σημειωθεί πως στις συναρτήσεις CreateEntry και UpdateEntry γίνεται επιπρόσθετος έλεγχος για το password, έτσι ώστε να μην αποθηκευτεί entry με κενό κωδικό πρόσβασης.

```

func (app *application) CreateEntry(
    sessionId []byte,
    keystoreId string, website, username, password, notes string) (entry.Entry, error) {
    app.mux.Lock()
    defer app.mux.Unlock()

    if !app.sessionActive(sessionId) : entry.Entry{}, ErrForbidden ↵

    // do not allow empty fields for website, username, password
    if strings.TrimSpace(website) == "" : entry.Entry{}, ErrInvalidWebsite ↵

    if strings.TrimSpace(username) == "" : entry.Entry{}, ErrInvalidUsername ↵

    if strings.TrimSpace(password) == "" : entry.Entry{}, ErrInvalidPassword ↵

    e := entry.New(
        entry.WithWebsite(website),
        entry.WithUsername(username),
        entry.WithPassword(password),
        entry.WithNotes(notes),
    )

    k, err := app.enclave.Keystore(app.key, keystoreId)
    if err != nil : entry.Entry{}, errors.WithMessage(err, "failed to get keystore") ↵

    k.Entries[e.Id] = e

    _, err = app.enclave.UpdateKeystore(app.key, k)
    if err != nil : entry.Entry{}, errors.WithMessage(err, "failed to update keystore") ↵

    return e, nil
}

```

Διάγραμμα 3.20. Συνάρτηση CreateEntry

```

func (app *application) UpdateEntry(
    sessionId []byte,
    keystoreId, entryId string, opts UpdateEntryOptions) (entry.Entry, error) {
    app.mux.Lock()
    defer app.mux.Unlock()

    if !app.sessionActive(sessionId) : entry.Entry{}, ErrForbidden ↵

    // do not allow empty password
    if opts.Password != nil && strings.TrimSpace(*opts.Password) == "" {
        return entry.Entry{}, ErrInvalidPassword
    }

    k, err := app.enclave.Keystore(app.key, keystoreId)
    if err != nil : entry.Entry{}, errors.WithMessage(err, "failed to get keystore") ↵

    e, ok := k.Entries[entryId]
    if !ok : entry.Entry{}, ErrEntryNotFound ↵

    if opts.Password != nil {
        e.Password = *opts.Password
    }

    if opts.Notes != nil {
        e.Notes = *opts.Notes
    }

    k.Entries[e.Id] = e

    _, err = app.enclave.UpdateKeystore(app.key, k)
    if err != nil : entry.Entry{}, errors.WithMessage(err, "failed to update keystore") ↵

    return e, nil
}

```

Διάγραμμα 3.21. Συνάρτηση UpdateEntry

```

func (app *application) DeleteEntry(sessionId []byte, keystoreId, entryId string) error {
    app.mux.Lock()
    defer app.mux.Unlock()

    if !app.sessionActive(sessionId) : ErrForbidden ↵

    k, err := app.enclave.Keystore(app.key, keystoreId)
    if err != nil : errors.WithMessage(err, "failed to get keystore") ↵

    if _, ok := k.Entries[entryId]; !ok : ErrEntryNotFound ↵

    delete(k.Entries, entryId)

    _, err = app.enclave.UpdateKeystore(app.key, k)
    if err != nil : errors.WithMessage(err, "failed to update keystore") ↵

    return nil
}

```

Διάγραμμα 3.22. Συνάρτηση DeleteEntry

Συνεχίζοντας με τα invitations, αρχικά χρησιμοποιείται η συνάρτηση CreateInvitation, η οποία δημιουργεί ένα invitation προς έναν άλλο χρήστη (invitee) για πρόσβαση σε ένα keystore. Σαν παραμέτρους δέχεται το id του keystore, καθώς και το username του invitee, το οποίο θα πρέπει να γνωρίζει ο τοπικός χρήστης εξ' αρχής. Η συγκεκριμένη συνάρτηση πρώτα ανακτά το keystore από το enclave, και στη συνέχεια ελέγχει το πεδίο RemoteId. Εάν το πεδίο είναι κενό, τότε "ανεβάζει" το keystore στον απομακρυσμένο server. Κατά το ανέβασμα του keystore δίνεται τιμή στο πεδίο RemoteId, και το keystore με το πεδίο RemoteId πλέον ορισμένο, αποθηκεύεται τοπικά στο enclave. Αμέσως μετά, το invitation δημιουργείται στον remote server και επιστρέφεται.

```
func (app *application) CreateInvitation(
    sessionId []byte,
    keystoreId string, inviteeUsername string) (invitation.Invitation, error) {
    app.mux.Lock()
    defer app.mux.Unlock()

    if !app.sessionActive(sessionId) : invitation.Invitation{}, ErrForbidden }

    k, err := app.enclave.Keystore(app.key, keystoreId)
    if err != nil {
        return invitation.Invitation{}, errors.WithMessage(err, "failed to get keystore")
    }

    // if keystore's remoteId is empty, then upload the keystore and save remoteId
    if k.RemoteId == "" {
        k, err = app.remote.CreateKeystore(k)
        if err != nil {
            return invitation.Invitation{}, errors.WithMessage(err, "failed to create keystore")
        }

        k, err = app.enclave.UpdateKeystore(app.key, k)
        if err != nil {
            return invitation.Invitation{}, errors.WithMessage(err, "failed to update keystore")
        }
    }

    inv, err := app.remote.CreateInvitation(k.RemoteId, inviteeUsername)
    if err != nil {
        return invitation.Invitation{}, errors.WithMessage(err, "failed to create invitation")
    }

    return inv, nil
}
```

Διάγραμμα 3.23. Συνάρτηση CreateInvitation

Ακολουθώντας τον κύκλο ζωής ενός invitation, το επόμενο βήμα είναι η αποδοχή του invitation από τον παραλήπτη (invitee). Συγκεκριμένα για την αποδοχή, χρησιμοποιείται η συνάρτηση AcceptInvitation, η οποία χρησιμοποιώντας τη διεπαφή remote αποδέχεται το invitation εκ μέρους του παραλήπτη.

```

func (app *application) AcceptInvitation(sessionId []byte, id string) (invitation.Invitation, error) {
    app.mux.Lock()
    defer app.mux.Unlock()

    if !app.sessionActive(sessionId) : invitation.Invitation{}, ErrForbidden }

    inv, err := app.remote.AcceptInvitation(id)
    if err != nil : invitation.Invitation{}, errors.WithMessage(err, "failed to accept invitation") }

    return inv, nil
}

```

Διάγραμμα 3.24. Συνάρτηση AcceptInvitation

Ένα invitation δεν αρκεί να είναι αποδεκτό από τον παραλήπτη, αλλά χρειάζεται και να είναι “οριστικοποιημένο” (finalized) από τον αποστολέα, προκειμένου να γίνει η κατάλληλη ανταλλαγή των κλειδιών κρυπτογράφησης. Για την επίτευξη αυτού, γίνεται χρήση της συνάρτησης FinalizeInvitation. Η FinalizeInvitation λειτουργεί με τον ακόλουθο τρόπο:

1. Ανακτά το invitation από τον απομακρυσμένο server
2. Ανακτά τα credentials του χρήστη από το enclave
3. Ανακτά το keystore από το enclave βάσει του RemoteId του invitation
4. Δημιουργεί ένα ασύμμετρο κλειδί χρησιμοποιώντας το ιδιωτικό κλειδί του τοπικού χρήστη σε συνδυασμό με το δημόσιο κλειδί του invitee
5. Κρυπτογραφεί το κλειδί κρυπτογράφησης του keystore χρησιμοποιώντας το ασύμμετρο κλειδί
6. Καλεί τη συνάρτηση FinalizeInvitation του remote, η οποία θα δώσει πρόσβαση του encryptedKeystoreKey στον invitee.

```

func (app *application) FinalizeInvitation(sessionId []byte, invitationId, remoteKeystoreId string,
    inviteePublicKey []byte) (invitation.Invitation, error) {
    app.mux.Lock()
    defer app.mux.Unlock()

    if !app.sessionActive(sessionId) {
        return invitation.Invitation{}, ErrForbidden
    }

    k, err := app.keystoreByRemoteId(remoteKeystoreId)
    if err != nil {
        return invitation.Invitation{}, errors.WithMessage(err, "failed to get keystore by remoteId")
    }

    creds, err := app.enclave.Credentials(app.key)
    if err != nil {
        return invitation.Invitation{}, errors.WithMessage(err, "failed to get credentials")
    }

    inv, err := app.remote.FinalizeInvitation(invitationId, creds.PrivateKey, inviteePublicKey, k)
    if err != nil {
        return invitation.Invitation{}, errors.WithMessage(err, "failed to finalize invitation")
    }

    return inv, nil
}

```

Διάγραμμα 3.25. Συνάρτηση FinalizeInvitation

Σε αυτό το σημείο το invitation έχει οριστικοποιηθεί, και ο invitee μπορεί πλέον να ανακτήσει το keystore από τον remote server.

Παράλληλα με τις παραπάνω μεθόδους, ορίζεται και η συνάρτηση DeleteInvitation, η οποία επιτρέπει είτε στον inviter ή στον invitee να ακυρώσει/απορρίψει το invitation αντίστοιχα. Η συνάρτηση αυτή καλεί την αντίστοιχη μέθοδο του remote.

```
func (app *application) DeleteInvitation(sessionId []byte, id string) (invitation.Invitation, error) {
    app.mux.Lock()
    defer app.mux.Unlock()

    if !app.sessionActive(sessionId) : invitation.Invitation{}, ErrForbidden }

    inv, err := app.remote.DeleteInvitation(id)
    if err != nil {
        return invitation.Invitation{}, errors.WithMessage(err, "failed to delete invitation")
    }

    return inv, nil
}
```

Διάγραμμα 3.26. Συνάρτηση DeleteInvitation

Ακόμη, έχουν υλοποιηθεί οι συναρτήσεις Invitation και Invitations, οι οποίες αντίστοιχα επιστρέφουν ένα συγκεκριμένο invitation (εφόσον ο χρήστης έχει τα δικαιώματα να το δει) ή όλα τα invitations που αφορούν στον συνδεδεμένο χρήστη.

```
func (app *application) Invitation(sessionId []byte, id string) (invitation.Invitation, error) {
    app.mux.Lock()
    defer app.mux.Unlock()

    if !app.sessionActive(sessionId) : invitation.Invitation{}, ErrForbidden }

    inv, err := app.remote.Invitation(id)
    if err != nil : invitation.Invitation{}, errors.WithMessage(err, "failed to get invitation") }

    return inv, nil
}

func (app *application) Invitations(sessionId []byte) (map[string]invitation.Invitation, error) {
    app.mux.Lock()
    defer app.mux.Unlock()

    if !app.sessionActive(sessionId) : nil, ErrForbidden }

    invs, err := app.remote.Invitations()
    if err != nil : nil, errors.WithMessage(err, "failed to get invitations") }

    return invs, nil
}
```

Διάγραμμα 3.27. Συναρτήσεις Invitation και Invitations

Τέλος, η συνάρτηση Sync συγχρονίζει τα keystores από και προς τον απομακρυσμένο server. Η συνάρτηση Sync, εφόσον ο χρήστης είναι ήδη συνδεδεμένος με τον server, ανακτά όλα τα keystores του χρήστη από το enclave, και εφόσον σε κάποιο από αυτά έχει δοθεί ένα remoteld, τότε ξεκινά τη διαδικασία συγχρονισμού.

Αρχικά, ανακτάται το remote keystore, και συγκρίνονται τα πεδία Version και για τα δύο keystores (του local και του remote). Στην περίπτωση που η έκδοση του remote keystore είναι μεγαλύτερη, τότε το local keystore αντικαθιστάται από την remote έκδοση, ενώ στην περίπτωση που είναι μικρότερη, το remote keystore ενημερώνεται με την τρέχουσα έκδοση. Αν οι εκδόσεις είναι ίδιες τότε δεν χρειάζεται κάποια αλλαγή.

Εφόσον η διαδικασία συγχρονισμού ολοκληρωθεί για όλα τα keystores, τότε η συνάρτηση επιστρέφει.

3.1.6. Υλοποίηση του Enclave

Η υλοποίηση του Enclave αφορά στην υλοποίηση των μεθόδων που ορίζονται στο Enclave interface από το `enclaverepo.Repository struct`. Το `Repository struct` περιέχει το πεδίο `path`, το οποίο χρησιμοποιείται για να υποδηλώσει τον φάκελο στον οποίο εμπεριέχεται το enclave.

Βασική προϋπόθεση για τη λειτουργία του `Repository` είναι η κλήση της συνάρτησης `Initialize`. Η `Initialize` πρώτα ελέγχει εάν υπάρχει ήδη το αρχείο του enclave. Εάν δεν υπάρχει, θα το δημιουργήσει στη συνέχεια. Αρχικά, ανακτώνται από το `crypto package` τα `default parameters` της διαδικασίας κρυπτογράφησης, και συγκεκριμένα του αλγορίθμου `Argon2id`. Στη συνέχεια, δημιουργούνται δύο τυχαία `salts` προκειμένου να γίνουν `append` στο `master password` για να είναι πιο δύσκολη η ανάκτηση του κωδικού με τεχνικές `brute force`, και συγκεκριμένα για να αποφευχθούν επιθέσεις που χρησιμοποιούν `rainbow tables`. Τα `salts` και το `password` δίνονται ως `input` στη συνάρτηση `Argon2id`, η οποία θα δημιουργήσει δύο αντίστοιχα `one-way cryptographic hashes`, ένα `encryption key` και ένα `signing key`. Τα `salts` θα αποθηκευτούν μέσα στο enclave, μιας και είναι απαραίτητα για τον μελλοντικό υπολογισμό των `encryption/signing keys`, και το enclave θα “κλειδωθεί” (κρυπτογραφηθεί) χρησιμοποιώντας τα `keys` αυτά χρησιμοποιώντας τη συνάρτηση `sealAndWrite`.

Σε αυτό το σημείο αξίζει να αναλυθεί η συνάρτηση `sealAndWrite`, η οποία κρυπτογραφεί το enclave και το αποθηκεύει στον δίσκο. Σε αυτή τη συνάρτηση χρησιμοποιείται η τεχνική `Encrypt-then-MAC`, στην οποία το `plaintext` κείμενο πρώτα θα κρυπτογραφηθεί και μετά θα δημιουργηθεί η υπογραφή του κρυπτοκειμένου ¹³. Η τεχνική αυτή χρησιμοποιείται επειδή μπορεί να αποδειχθεί πως το `ciphertext` είναι έγκυρο, ακόμη και σε περιπτώσεις χρήσης `malleable` αλγορίθμων κρυπτογράφησης, όπως είναι ο `AES` σε `CBC mode` που χρησιμοποιείται στη συγκεκριμένη περίπτωση.

Αρχικά, το enclave κωδικοποιείται στη μορφή `JSON`, και αμέσως μετά κρυπτογραφείται με το `encryption key` που δόθηκε σαν όρισμα. Στη συνέχεια, υπολογίζεται ο κωδικός ελέγχου ταυτότητας μηνύματος (`Message Authentication code` ή `MAC`) του κρυπτοκειμένου με βάση το `signing key`, και γράφονται όλα τα πεδία (`encryptionSalt`, `signingSalt`, `mac`, `ciphertext`) στο αρχείο σε `binary` μορφή.

```

func (r *Repository) Initialize(password string) ([]byte, error) {
    exists, err := r.enclaveExists()
    if err != nil {
        return nil, errors.WithMessage(err, "failed to check if enclave exists")
    }

    if exists : nil, application.ErrEnclaveExists ↵

    p := crypto.DefaultArgon2IdParams

    encSalt, err := crypto.GenerateRandomBytes(uint(p.SaltLength))
    if err != nil : nil, errors.WithMessage(err, "failed to generate salt") ↵

    signSalt, err := crypto.GenerateRandomBytes(uint(p.SaltLength))
    if err != nil : nil, errors.WithMessage(err, "failed to generate salt") ↵

    var encKey, signKey []byte

    var wg sync.WaitGroup
    wg.Add(2)

    go func() {
        defer wg.Done()

        encKey = crypto.Argon2Id([]byte(password), encSalt)
    }()

    go func() {
        defer wg.Done()

        signKey = crypto.Argon2Id([]byte(password), signSalt)
    }()

    wg.Wait()

    publicKey, privateKey, err := crypto.NewCurve25519Keypair()
    if err != nil {
        return nil, errors.WithMessage(err, "failed to generate keypair")
    }

    e := newEnclave(encSalt, signSalt, publicKey, privateKey)

    keypair := append(encKey, signKey...)

    err = r.sealAndWrite(keypair, e)
    if err != nil {
        return nil, errors.WithMessage(err, "failed to seal enclave")
    }

    return keypair, nil
}

```

Διάγραμμα 3.28. Συνάρτηση Initialize

```

func (r *Repository) sealAndWrite(keypair []byte, e *enclave) error {
    b, err := enclaveToJSON(e)
    if err != nil {
        return errors.WithMessage(err, "failed to marshal enclave")
    }

    p := crypto.DefaultArgon2IdParams

    encKey := keypair[:p.KeyLength]
    signKey := keypair[p.KeyLength:]

    b, err = crypto.AES256CBC_Encrypt(encKey, b)
    if err != nil : errors.WithMessage(err, "failed to encrypt enclave") ␣

    // authenticate
    mac := crypto.HMAC_SHA256(signKey, b)

    // prepend salts and mac to the ciphertext
    b = append(e.encSalt, append(e.signSalt, append(mac, b...)...)...)

    err = os.WriteFile(r.enclavePath(), b, 0600)
    if err != nil : errors.Wrap(err, "failed to write enclave") ␣

    return nil
}

```

Διάγραμμα 3.29. Συνάρτηση sealAndWrite

Προκειμένου να ελεγχθεί (από κάποια διεπαφή χρήστη) η κατάσταση του enclave, παρέχεται η συνάρτηση IsInitialized, η οποία επιστρέφει boolean true μόνο εάν υπάρχει το enclave.

```

func (r *Repository) IsInitialized() (bool, error) {
    exists, err := r.enclaveExists()
    if err != nil {
        return false, errors.WithMessage(err, "failed to check if enclave exists")
    }

    if !exists : false, nil ␣

    return true, nil
}

func (r *Repository) enclaveExists() (bool, error) {
    _, err := os.Stat(r.enclavePath())
    if errors.Is(err, os.ErrNotExist) {
        return false, nil
    } else if err != nil {
        return false, errors.Wrap(err, "failed to stat enclave")
    }

    return true, nil
}

```

Διάγραμμα 3.30. Συνάρτηση IsInitialized

Η συνάρτηση `Authenticate` επιτρέπει την επιβεβαίωση του password του χρήστη σε σχέση με το αποθηκευμένο enclave. Αρχικά, διαβάζεται στη μνήμη το κρυπτογραφημένο enclave, και ακολούθως εξάγονται τα encryption και signing salts και υπολογίζονται τα encryption και signing κλειδιά αντίστοιχα. Τέλος, επιστρέφονται τα κλειδιά για να χρησιμοποιηθούν σε ενέργειες που θα πραγματοποιηθούν αργότερα από τον χρήστη, έτσι ώστε να μην υπολογίζονται εκ νέου μιας και ο υπολογισμός τους είναι κοστοβόρος.

```
func (r *Repository) Authenticate(password string) (keypair []byte, err error) {
    // we don't defer unlock because hash computation is slow

    b, err := os.ReadFile(r.enclavePath())
    if errors.Is(err, fs.ErrNotExist) {
        return nil, application.ErrInitializationRequired
    } else if err != nil {
        return nil, errors.Wrap(err, "failed to read enclave")
    }

    encSalt, signSalt := getSaltsFromData(b)

    var encKey, signKey []byte

    var wg sync.WaitGroup
    wg.Add(2)

    go func() {
        defer wg.Done()

        encKey = crypto.Argon2Id([]byte(password), encSalt)
    }()

    go func() {
        defer wg.Done()

        signKey = crypto.Argon2Id([]byte(password), signSalt)
    }()

    wg.Wait()

    p := crypto.DefaultArgon2IdParams

    mac := b[p.SaltLength*2 : sha256.Size+p.SaltLength*2]
    b = b[p.SaltLength*2+sha256.Size:]

    valid := crypto.VerifyHMAC_SHA256(signKey, mac, b)
    if !valid {
        return nil, application.ErrAuthenticationFailed
    }

    return append(encKey, signKey...), nil
}
```

Διάγραμμα 3.31. Συνάρτηση `Authenticate`

Παράλληλα με τις συναρτήσεις αυτές, το enclave repository προσφέρει επίσης συναρτήσεις για τη διαχείριση των keystores και των credentials του χρήστη. Αυτές οι συναρτήσεις, εφόσον τους δοθεί το σωστό pair encryption και signing κλειδιών, ανακτούν πρώτα το enclave, στη συνέχεια διαβάζουν ή τροποποιούν τα αντίστοιχα πεδία, και σε περίπτωση τροποποίησης του enclave, καλούν τη συνάρτηση `sealAndWrite`, αποθηκεύοντας έτσι την όποια αλλαγή.

```

func (r *Repository) CreateKeystore(keypair []byte, k keystore.Keystore) (keystore.Keystore, error) {
    e, err := r.enclave(keypair)
    if err != nil : keystore.Keystore{}, errors.WithMessage(err, "failed to get enclave") ␣

    err = e.addKeystore(k)
    if err != nil {
        return keystore.Keystore{}, errors.WithMessage(err, "failed to add keystore")
    }

    err = r.sealAndWrite(keypair, e)
    if err != nil {
        return keystore.Keystore{}, errors.WithMessage(err, "failed to seal enclave")
    }

    return k, nil
}

```

Διάγραμμα 3.32. Συνάρτηση CreateKeystore

```

func (r *Repository) Keystore(keypair []byte, id string) (keystore.Keystore, error) {
    e, err := r.enclave(keypair)
    if err != nil : keystore.Keystore{}, errors.WithMessage(err, "failed to get enclave") ␣

    k, err := e.keystore(id)
    if err != nil : keystore.Keystore{}, errors.WithMessage(err, "failed to get keystore") ␣

    return k, nil
}

```

Διάγραμμα 3.33. Συνάρτηση Keystore

```

func (r *Repository) UpdateKeystore(keypair []byte, k keystore.Keystore) (keystore.Keystore, error) {
    e, err := r.enclave(keypair)
    if err != nil : keystore.Keystore{}, errors.WithMessage(err, "failed to get enclave") ␣

    k, err = e.updateKeystore(k)
    if err != nil {
        return keystore.Keystore{}, errors.WithMessage(err, "failed to update keystore")
    }

    err = r.sealAndWrite(keypair, e)
    if err != nil : keystore.Keystore{}, errors.WithMessage(err, "failed to seal enclave") ␣

    return k, nil
}

```

Διάγραμμα 3.34. Συνάρτηση UpdateKeystore

```

func (r *Repository) Keystores(keypair []byte) (map[string]keystore.Keystore, error) {
    _, err := os.ReadFile(r.enclavePath())
    if errors.Is(err, os.ErrNotExist) {
        return nil, application.ErrInitializationRequired
    } else if err != nil {
        return nil, errors.Wrap(err, "failed to read enclave")
    }

    e, err := r.enclave(keypair)
    if err != nil {
        return nil, errors.WithMessage(err, "failed to get enclave")
    }

    ks, err := e.keystoresWithKeys()
    if err != nil {
        return nil, errors.WithMessage(err, "failed to get keystores")
    }

    return ks, nil
}

```

Διάγραμμα 3.35. Συνάρτηση Keystores

```

func (r *Repository) DeleteKeystore(keypair []byte, id string) error {
    e, err := r.enclave(keypair)
    if err != nil : errors.WithMessage(err, "failed to get enclave") ␣

    err = e.deleteKeystore(id)
    if err != nil : errors.WithMessage(err, "failed to delete keystore") ␣

    err = r.sealAndWrite(keypair, e)
    if err != nil : errors.WithMessage(err, "failed to seal enclave") ␣

    return nil
}

```

Διάγραμμα 3.36. Συνάρτηση DeleteKeystore

```

func (r *Repository) Credentials(keypair []byte) (credentials.Credentials, error) {
    e, err := r.enclave(keypair)
    if err != nil : credentials.Credentials{}, errors.WithMessage(err, "failed to get enclave") ␣

    return e.creds, nil
}

```

Διάγραμμα 3.37. Συνάρτηση Credentials

```

func (r *Repository) UpdateCredentials(keypair []byte, creds credentials.Credentials,
) (credentials.Credentials, error) {
    e, err := r.enclave(keypair)
    if err != nil : credentials.Credentials{}, errors.WithMessage(err, "failed to get enclave") }

    e.creds = creds

    err = r.sealAndWrite(keypair, e)
    if err != nil : credentials.Credentials{}, errors.WithMessage(err, "failed to seal enclave") }

    return creds, nil
}

```

Διάγραμμα 3.38. Συνάρτηση UpdateCredentials

3.1.7. Υλοποίηση του Remote

Η δομή Remote αφορά στην υλοποίηση του Remote interface, και συγκεκριμένα περιλαμβάνει συναρτήσεις που διαχειρίζονται τον λογαριασμό του χρήστη και τα δεδομένα του, όπως keystores και invitations προς και από άλλους χρήστες. Οι συναρτήσεις του Remote κατά κύριο λόγο μετατρέπουν τις δομές από το domain της εφαρμογής στα αντίστοιχα αντικείμενα JSON, καλούν το εκάστοτε endpoint του remote server, και επιστρέφουν το αποτέλεσμα αφότου γίνει μετατροπή πίσω σε domain objects όπου χρειάζεται.

Στο Remote, προκειμένου να είναι πιο εύκολη και ασφαλής η αλληλεπίδραση με τον server, χρησιμοποιείται ένας dynamically generated HTTP client. Ο client αυτός δημιουργείται από το OpenAPI specification του server με την εντολή go generate, χρησιμοποιώντας τη βιβλιοθήκη oapi-codegen της εταιρείας DeepMap.

Εντός του Remote struct ορίζονται τα ακόλουθα πεδία:

- address (string): η απομακρυσμένη διεύθυνση URL του server
- client (*generated.ClientWithResponses): ο generated HTTP client
- bearerToken (string): το bearer token που ταυτοποιεί τον χρήστη
- user (*user.User): το προφίλ του συνδεδεμένου χρήστη
- username (string): το username του χρήστη
- password (string): το password του χρήστη

Καταρχάς, σχετικά με τον λογαριασμό χρήστη ορίζεται η συνάρτηση Register, η οποία δέχεται ως παραμέτρους το username, το password και το publicKey του χρήστη, και στη συνέχεια καλεί την αντίστοιχη μέθοδο του HTTP client, και αποθηκεύει εντός του Remote struct τον δημιουργημένο χρήστη και το bearer token που θα χρησιμοποιηθεί ακολούθως για την ταυτοποίηση του χρήστη. Επίσης αποθηκεύονται εντός του struct τα πεδία username και password, προκειμένου να γίνει εκ νέου η αυθεντικοποίηση του χρήστη με τον server στο παρασκήνιο ανά τακτά χρονικά διαστήματα.

```

func (r *Remote) Register(username, password string, publicKey []byte) (user.User, error) {
    ctx, cancel := context.WithTimeout(context.Background(), 10*time.Second)
    defer cancel()

    res, err := r.client.RegisterWithResponse(
        ctx, generated.RegisterJSONRequestBody{
            Username: username,
            Password: password,
            PublicKey: publicKey,
        },
    )
    if err != nil : user.User{}, errors.Wrap(err, "failed to register")

    if res.StatusCode() != http.StatusCreated || res.JSON201 == nil {
        return user.User{}, errors.New("invalid response")
    }

    if res.JSON201.Token == "" : user.User{}, errors.New("invalid token")

    u := user.User{
        Id: res.JSON201.User.Id,
        Username: res.JSON201.User.Username,
        PublicKey: publicKey,
    }

    r.user = &u
    r.bearerToken = res.JSON201.Token

    return u, nil
}

```

Διάγραμμα 3.39. Συνάρτηση Register

Επίσης, η συνάρτηση Authenticate χρησιμοποιείται για να αυθεντικοποιηθεί ένας ήδη εγγεγραμμένος χρήστης με τον remote server. Η συνάρτηση αυτή καλεί το login endpoint του server και αποθηκεύει, ομοίως με τη συνάρτηση Register, τον χρήστη το bearer token, και τα πεδία username και password εντός του struct. Χρησιμοποιείται η private ρουτίνα authenticate, έτσι ώστε να μπορεί να γίνει ανά πάσα στιγμή μέσω της συνάρτησης Reauthenticate η εκ νέου αυθεντικοποίηση του χρήστη, μιας και το access token που παρέχεται από τον server έχει κάποιο ορισμένο Time To Live (TTL) μετά το πέρας του οποίου το bearer token σταματά να είναι έγκυρο.

```

func (r *Remote) Authenticate(username, password string) error {
    // store for future re-authentication
    r.username = username
    r.password = password

    err := r.authenticate()
    if err != nil : errors.WithMessage(err, "failed to authenticate") ␣

    return nil
}

func (r *Remote) authenticate() error {
    ctx, cancel := context.WithTimeout(context.Background(), 10*time.Second)
    defer cancel()

    res, err := r.client.LoginWithResponse(
        ctx, generated.LoginJSONRequestBody{
            Username: r.username,
            Password: r.password,
        },
    )
    if err != nil : errors.Wrap(err, "login failed") ␣

    if res.StatusCode() != http.StatusOK || res.JSON200 == nil {
        return errors.New("invalid response")
    }

    if res.JSON200.Token == "" : errors.New("invalid token") ␣

    u := user.User{
        Id:         res.JSON200.User.Id,
        Username:   res.JSON200.User.Username,
        PublicKey:  res.JSON200.User.PublicKey,
    }

    r.user = &u
    r.bearerToken = res.JSON200.Token

    return nil
}

```

Διάγραμμα 3.40. Συνάρτηση Authenticate

Αμέσως μετά την επιτυχή σύνδεση με τον server, μπορούν να χρησιμοποιηθούν οι συναρτήσεις Authenticated και CurrentUser για να γίνει έλεγχος της τωρινής κατάστασης σύνδεσης με τον server και, αντιστοίχως, του συνδεδεμένου χρήστη.

```

func (r *Remote) Authenticated() bool {
    return r.bearerToken != ""
}

```

Διάγραμμα 3.41. Συνάρτηση Authenticated

```
func (r *Remote) CurrentUser() *user.User {
    return r.user
}
```

Διάγραμμα 3.42. Συνάρτηση CurrentUser

Σχετικά με τη διαχείριση των keystores, ορίζονται οι συναρτήσεις CreateKeystore, UpdateKeystore, DeleteKeystore και Keystores.

Πρωτίστως, στη συνάρτηση CreateKeystore, εφόσον γίνει έλεγχος για την σύνδεση με τον server, γίνεται κρυπτογράφηση του keystore χρησιμοποιώντας το αντίστοιχο ζεύγος encryption και signing key του keystore. Στη συνέχεια, γίνεται ένα HTTP request που θα δημιουργήσει το keystore στον remote server, παρέχοντας στον server μόνο το όνομα του keystore και το κρυπτογραφημένο φορτίο (payload) του περιεχομένου του keystore. Με την επιτυχία της δημιουργίας αυτής, ορίζεται στο keystore το πεδίο RemoteId, που θα είναι το UUID που όρισε ο server κατά τη δημιουργία, και θα βοηθήσει αργότερα στην αναγνώριση του keystore στον remote server.

```
func (r *Remote) CreateKeystore(k keystore.Keystore) (keystore.Keystore, error) {
    if !r.Authenticated() : keystore.Keystore{}, application.ErrAuthenticationRequired }

    jk := keystoreToJSON(k)

    b, err := json.Marshal(jk)
    if err != nil : keystore.Keystore{}, errors.Wrap(err, "failed to marshal keystore") }

    b, err = crypto.AES256CBC_Encrypt(k.Key, b)
    if err != nil : keystore.Keystore{}, errors.WithMessage(err, "aes256cbc encrypt failed") }

    res, err := r.client.CreateKeystoreWithResponse(
        context.Background(), generated.CreateKeystoreJSONRequestBody{
            Name: k.Name,
            Payload: b,
        },
    )
    if err != nil : keystore.Keystore{}, errors.Wrap(err, "failed to create keystore") }

    if res.StatusCode() != http.StatusCreated || res.JSON201 == nil {
        return keystore.Keystore{}, errors.New("invalid response")
    }

    k.RemoteId = (*res.JSON201).Id

    return k, nil
}
```

Διάγραμμα 3.43. Συνάρτηση CreateKeystore

Με παρόμοιο τρόπο λειτουργεί και η συνάρτηση UpdateKeystore, ενημερώνοντας το υπάρχον κρυπτογραφημένο φορτίο και το όνομα του keystore (στην περίπτωση που ο χρήστης αλλάξει το όνομα του keystore ενώ υπάρχει κάποιο εκκρεμές invitation).

```

func (r *Remote) UpdateKeystore(k keystore.Keystore) (keystore.Keystore, error) {
    if !r.Authenticated() : keystore.Keystore{}, application.ErrAuthenticationRequired }

    jk := keystoreToJSON(k)

    b, err := json.Marshal(jk)
    if err != nil : keystore.Keystore{}, errors.Wrap(err, "failed to marshal keystore") }

    b, err = crypto.AES256CBC_Encrypt(k.Key, b)
    if err != nil : keystore.Keystore{}, errors.WithMessage(err, "aes256cbc encrypt failed") }

    res, err := r.client.UpdateKeystoreWithResponse(context.Background(),
        k.RemoteId, generated.UpdateKeystoreRequest{
            Name:      &k.Name,
            Payload: &b,
        })
    if err != nil : keystore.Keystore{}, errors.Wrap(err, "failed to update keystore") }

    if res.StatusCode() != http.StatusOK || res.JSON200 == nil {
        return keystore.Keystore{}, errors.New("invalid response")
    }

    _, err = KeystoreFromJSON(*res.JSON200, k.Key)
    if err != nil : keystore.Keystore{}, errors.WithMessage(err, "failed to parse keystore") }

    return k, nil
}

```

Διάγραμμα 3.44. Συνάρτηση UpdateKeystore

Ομοίως η συνάρτηση DeleteKeystore διαγράφει ένα keystore από τον remote server. Χρησιμοποιείται για να “καθαρίσει” το keystore από τον remote server, έτσι ώστε να μην καταναλώνει αποθηκευτικό χώρο.

```

func (r *Remote) DeleteKeystore(id string) error {
    if !r.Authenticated() : application.ErrAuthenticationRequired }

    res, err := r.client.DeleteKeystoreWithResponse(context.Background(), id)
    if err != nil : errors.Wrap(err, "failed to delete keystore") }

    if res.StatusCode() != http.StatusOK : errors.New("invalid response") }

    return nil
}

```

Διάγραμμα 3.45. Συνάρτηση DeleteKeystore

Ιδιαίτερο ενδιαφέρον παρουσιάζει η συνάρτηση Keystores, η οποία επιστρέφει όλα τα keystores στα οποία έχει πρόσβαση ο χρήστης. Προκειμένου να αποκρυπτογραφηθεί το περιεχόμενό τους, συλλέγονται τα invitations που σχετίζονται με αυτόν τον χρήστη. Εφόσον το ζεύγος των κλειδιών encryption/signing του keystore μπορεί να αποκρυπτογραφηθεί χρησιμοποιώντας το shared secret του χρήστη, τότε το εκάστοτε keystore συμπεριλαμβάνεται στη λίστα που θα επιστραφεί.

```

func (r *Remote) Keystores(privateKey []byte) (map[string]keystore.Keystore, error) {
    if !r.Authenticated() : nil, application.ErrAuthenticationRequired }

    res, err := r.client.KeystoresWithResponse(context.Background())
    if err != nil : nil, errors.Wrap(err, "failed to get keystores") }

    if res.StatusCode() != http.StatusOK || res.JSON200 == nil {
        return nil, errors.New("invalid response")
    }

    restKeystores := *res.JSON200

    invs, err := r.Invitations()
    if err != nil : nil, errors.WithMessage(err, "failed to get invitations") }

    keystores := make(map[string]keystore.Keystore)

    for _, restKeystore := range restKeystores {
        var keystoreKey []byte

        // find the finalized invitation for this keystore and decrypt its payload
        for _, inv := range invs {
            if inv.RemoteKeystoreId == restKeystore.Id && inv.Finalized() {
                var pub []byte
                if inv.Inviter.Id == r.CurrentUser().Id {
                    pub = inv.Invitee.PublicKey
                } else {
                    pub = inv.Inviter.PublicKey
                }

                sharedSecret, err := curve25519.X25519(privateKey, pub)
                if err != nil : nil, errors.Wrap(err, "failed to derive shared secret") }

                keystoreKey, err = crypto.AES256CBC_Decrypt(sharedSecret, inv.EncryptedKeystoreKey)
                if err != nil : nil, errors.WithMessage(err, "failed to decrypt keystore key") }

                break
            }
        }

        if keystoreKey == nil {
            continue
        }

        k, err := KeystoreFromJSON(restKeystore, keystoreKey)
        if err != nil : nil, errors.WithMessage(err, "failed to parse keystore") }

        keystores[k.Id] = k
    }

    return keystores, nil
}

```

Διάγραμμα 3.46. Συνάρτηση Keystores

Το remote struct προσφέρει επίσης συναρτήσεις για τη δημιουργία και διαχείριση των invitations από και προς άλλους χρήστες. Πιο συγκεκριμένα, ορίζονται οι συναρτήσεις CreateInvitation, AcceptInvitation, DeleteInvitation, FinalizeInvitation, καθώς και τα queries Invitation και Invitations.

Η συνάρτηση `CreateInvitation` δημιουργεί ένα `invitation` για ένα συγκεκριμένο `keystore` (το οποίο αναγνωρίζεται από το `RemoteId` του) προς έναν συγκεκριμένο χρήστη.

```
func (r *Remote) CreateInvitation(keystoreRemoteId, inviteeUsername string,
) (invitation.Invitation, error) {
    if !r.Authenticated() : invitation.Invitation{}, application.ErrAuthenticationRequired }

    res, err := r.client.CreateInvitationWithResponse(
        context.Background(), keystoreRemoteId, generated.CreateInvitationJSONRequestBody{
            Invitee: inviteeUsername,
        },
    )
    if err != nil : invitation.Invitation{}, errors.Wrap(err, "failed to create invitation") }

    if res.StatusCode() != http.StatusCreated || res.JSON201 == nil {
        return invitation.Invitation{}, errors.New("invalid response")
    }

    inv, err := InvitationFromJSON(*res.JSON201)
    if err != nil : invitation.Invitation{}, errors.WithMessage(err, "failed to parse invitation") }

    return inv, nil
}
```

Διάγραμμα 3.47. Συνάρτηση `CreateInvitation`

Στη συνέχεια, έχουμε τη συνάρτηση `AcceptInvitation`, η οποία καλείται από τον χρήστη που λαμβάνει την πρόσκληση (`invitee`), και καλεί το `endpoint` που αλλάζει το `status` του `invitation` σε “accepted” στον server.

```
func (r *Remote) AcceptInvitation(invitationId string) (invitation.Invitation, error) {
    res, err := r.client.AcceptInvitationWithResponse(
        context.Background(), invitationId,
    )
    if err != nil : invitation.Invitation{}, errors.Wrap(err, "failed to accept invitation") }

    if res.StatusCode() == http.StatusNotFound {
        return invitation.Invitation{}, application.ErrInvitationNotFound
    } else if res.StatusCode() == http.StatusForbidden {
        return invitation.Invitation{}, application.ErrForbidden
    } else if res.StatusCode() != http.StatusOK || res.JSON200 == nil {
        return invitation.Invitation{}, errors.New("invalid response")
    }

    inv, err := InvitationFromJSON(*res.JSON200)
    if err != nil : invitation.Invitation{}, errors.WithMessage(err, "failed to parse invitation") }

    return inv, nil
}
```

Διάγραμμα 3.48. Συνάρτηση `AcceptInvitation`

Η `DeleteInvitation` ομοίως στέλνει αίτημα διαγραφής του `invitation` προς στον server.

```

func (r *Remote) DeleteInvitation(id string) (invitation.Invitation, error) {
    if !r.Authenticated() : invitation.Invitation{}, application.ErrAuthenticationRequired }

    res, err := r.client.DeleteInvitationWithResponse(
        context.Background(), id,
    )
    if err != nil : invitation.Invitation{}, errors.Wrap(err, "failed to delete invitation") }

    if res.StatusCode() != http.StatusOK || res.JSON200 == nil {
        return invitation.Invitation{}, errors.New("invalid response")
    }

    inv, err := InvitationFromJSON(*res.JSON200)
    if err != nil : invitation.Invitation{}, errors.WithMessage(err, "failed to parse invitation") }

    return inv, nil
}

```

Διάγραμμα 3.49. Συνάρτηση DeleteInvitation

Η FinalizeInvitation από την άλλη καλείται από τον inviter προκειμένου να οριστικοποιήσει ένα invitation. Κατά τη διαδικασία οριστικοποίησης, υπολογίζεται το shared secret χρησιμοποιώντας το private key του inviter και το public key του invitee, και το κλειδί αυτό στη συνέχεια χρησιμοποιείται για να κρυπτογραφήσει το keystore key και να αποθηκεύσει το κρυπτογραφημένο keystore key εντός του invitation.

```

func (r *Remote) FinalizeInvitation(
    invitationId string, privateKey []byte, inviteePublicKey []byte, k keystore.Keystore,
) (invitation.Invitation, error) {
    // derive shared secret using the user's private key and the invitee's public key
    sharedSecret, err := curve25519.X25519(privateKey, inviteePublicKey)
    if err != nil : invitation.Invitation{}, errors.Wrap(err, "failed to derive shared secret") }

    // encrypt the keystore key with the asymmetric key
    encryptedKeystoreKey, err := crypto.AES256CBC_Encrypt(sharedSecret, k.Key)
    if err != nil {
        return invitation.Invitation{}, errors.WithMessage(err, "failed to encrypt keystore key")
    }

    res, err := r.client.FinalizeInvitationWithResponse(context.Background(),
        invitationId, generated.FinalizeInvitationJSONRequestBody{
            KeystoreKey: encryptedKeystoreKey,
        },
    )
    if err != nil : invitation.Invitation{}, errors.Wrap(err, "failed to finalize invitation") }

    if res.StatusCode() != http.StatusOK || res.JSON200 == nil {
        return invitation.Invitation{}, errors.New("invalid response")
    }

    inv, err := InvitationFromJSON(*res.JSON200)
    if err != nil : invitation.Invitation{}, errors.WithMessage(err, "failed to parse invitation") }

    return inv, nil
}

```

Διάγραμμα 3.50. Συνάρτηση FinalizeInvitation

Σχετικά με τα query-related functions του remote, οι συναρτήσεις Invitation και Invitations απλώς επιστρέφουν, αντίστοιχα, ένα συγκεκριμένο invitation εφόσον ο χρήστης έχει πρόσβαση σε αυτό, ή όλα τα invitations συσχετισμένα με τον συνδεδεμένο χρήστη.

```
func (r *Remote) Invitation(id string) (invitation.Invitation, error) {
    if !r.Authenticated() : invitation.Invitation{}, application.ErrAuthenticationRequired }

    res, err := r.client.GetInvitationWithResponse(context.Background(), id)
    if err != nil : invitation.Invitation{}, errors.Wrap(err, "failed to get invitation") }

    if res.StatusCode() == http.StatusNotFound {
        return invitation.Invitation{}, application.ErrInvitationNotFound
    }

    if res.StatusCode() != http.StatusOK || res.JSON200 == nil {
        return invitation.Invitation{}, errors.New("invalid response")
    }

    inv, err := InvitationFromJSON(*res.JSON200)
    if err != nil {
        return invitation.Invitation{}, errors.WithMessage(err, "failed to parse invitation")
    }

    return inv, nil
}
```

Διάγραμμα 3.51. Συνάρτηση Invitation

```
func (r *Remote) Invitations() (map[string]invitation.Invitation, error) {
    if !r.Authenticated() : nil, application.ErrAuthenticationRequired }

    res, err := r.client.InvitationsWithResponse(context.Background())
    if err != nil : nil, errors.Wrap(err, "failed to get invitations") }

    if res.StatusCode() != http.StatusOK || res.JSON200 == nil {
        return nil, errors.New("invalid response")
    }

    invs := map[string]invitation.Invitation{}
    for _, inv := range *res.JSON200 {
        inv, err := InvitationFromJSON(inv)
        if err != nil : nil, errors.WithMessage(err, "failed to parse invitation") }

        invs[inv.Id] = inv
    }

    return invs, nil
}
```

Διάγραμμα 3.52. Συνάρτηση Invitations

Τέλος, παρέχεται η συνάρτηση SharedSecret η οποία επιστρέφει το κοινό κλειδί κρυπτογράφησης ανάμεσα στον συνδεδεμένο χρήστη και έναν οποιοδήποτε χρήστη με βάση το username.

```

func (r *Remote) SharedSecret(privateKey []byte, userId string) ([]byte, error) {
    invs, err := r.Invitations()
    if err != nil : nil, errors.WithMessage(err, "failed to query invitations")

    for _, inv := range invs {
        if inv.Inviter.Id != userId && inv.Invitee.Id != userId {
            continue
        }

        var pub []byte
        if inv.Inviter.Id == r.CurrentUser().Id {
            pub = inv.Invitee.PublicKey
        } else {
            pub = inv.Inviter.PublicKey
        }

        sharedSecret, err := curve25519.X25519(privateKey, pub)
        if err != nil : nil, errors.Wrap(err, "failed to derive shared secret")

        return sharedSecret, nil
    }

    return nil, application.ErrSharedSecretNotFound
}

```

Διάγραμμα 3.53. Συνάρτηση SharedSecret

3.1.8. HTTP REST server

Μια διεπαφή χρήστη μπορεί να αλληλεπιδράσει με τον πυρήνα της εφαρμογής του Client μέσω του HTTP REST server. Ο HTTP server χρησιμοποιεί το Gin web framework και προσφέρει REST endpoints τα οποία καλούν τις αντίστοιχες συναρτήσεις του application, φροντίζοντας να μεταβιβάσουν στο application το ανάλογο context. Σχεδόν όλες οι συναρτήσεις του Client μπορούν να κληθούν από τον HTTP server. Επιπρόσθετα, ο HTTP server χρησιμοποιεί ένα OpenAPI specification, με τη βοήθεια του οποίου δημιουργούνται τα μοντέλα αλληλεπίδρασης με τον server και ένας HTTP Client, ο οποίος μπορεί να χρησιμοποιηθεί, για παράδειγμα, σε integration tests.

Ειδικότερα, ο Server αποτελεί επέκταση της δομής gin.Engine, και εντός της δομής του αποθηκεύεται μια αναφορά στο instance του Client, καθώς και ο Go HTTP server ο οποίος θα κάνει serve όλους τους handlers του Gin.

```

type Server struct {
    *gin.Engine
    app    application.Application
    server *server.Server
}

```

Διάγραμμα 3.54. Server struct

Κατά τη δημιουργία του HTTP REST server δημιουργείται το αντίστοιχο Gin router instance, ρυθμίζονται τα διάφορα μεσολογισμικά του HTTP server όπως το panic recovery, το logger και το CORS (Cross-Origin-Resource-Sharing) middleware, και ορίζονται όλα τα routes και τα μεσολογισμικά τους. Παράλληλα, ρυθμίζεται το noRouteMiddleware το οποίο, στην περίπτωση που το requested route δεν είναι ένα από τα προκαθορισμένα και εφόσον η διαδρομή δεν αφορά στο REST API, τότε επιστρέφει στον εκάστοτε HTTP client το index.html αρχείο της διεπαφής χρήστη (UI) της εφαρμογής πελάτη.

```
r.Use(recoveryMiddleware)
r.Use(loggerMiddleware)
r.NoRoute(noRouteMiddleware("/", embedFolder(ui, "static")))
r.Use(
    cors.New(
        cors.Options{...},
    ),
)
```

Διάγραμμα 3.55. Router Middleware

Τελικά, ορίζονται τα διάφορα HTTP routes, τα επιμέρους middleware τους, όπως το authenticationMiddleware το οποίο χρησιμοποιείται για να κάνει τους απαραίτητους ελέγχους για το authentication session του χρήστη, και οι handlers του κάθε route.

```
api := r.Group("api")

api.POST("/authenticate", s.Authenticate)
api.POST("/enclave", s.CreateEnclave)

sec := api.Group("/")
sec.Use(s.authenticationMiddleware())

sec.GET("/enclave", s.Enclave)
sec.GET("/health", s.HealthCheck)
sec.POST("/auth/register", s.Register)
sec.POST("/keystores", s.CreateKeystore)
sec.GET("/keystores", s.Keystores)
sec.GET("/keystore/:keystoreId", s.Keystore)
sec.DELETE("/keystore/:keystoreId", s.DeleteKeystore)
sec.POST("/keystore/:keystoreId/entries", s.CreateEntry)
sec.PATCH("/keystore/:keystoreId/entry/:entryId", s.UpdateEntry)
sec.DELETE("/keystore/:keystoreId/entry/:entryId", s.DeleteEntry)
sec.GET("/invitations", s.GetInvitations)
sec.GET("/invitation/:invitationId", s.GetInvitation)
sec.POST("/keystore/:keystoreId/invitations", s.CreateInvitation)
sec.PATCH("/invitation/:invitationId", s.AcceptInvitation)
sec.DELETE("/invitation/:invitationId", s.DeclineOrCancelInvitation)
sec.POST("/invitation/:invitationId", s.FinalizeInvitation)
sec.GET("/user", s.CurrentUser)
```

Διάγραμμα 3.56. HTTP routes

```

func (s *Server) authenticationMiddleware() gin.HandlerFunc {
    return func(c *gin.Context) {
        err := s.authenticateWithToken(c)
        if err != nil {
            log.Error(err)
            Error(c, http.StatusForbidden)
            c.Abort()
        }
    }
}

func (s *Server) authenticateWithToken(c *gin.Context) error {
    auth := c.GetHeader("Authorization")
    if auth == "" : nil

    parts := strings.Split(auth, "Bearer")
    if len(parts) != 2 : errors.New("authentication failed")
    auth = strings.TrimSpace(parts[1])

    if auth == "" : errors.New("authentication required")

    sid, err := base64.StdEncoding.DecodeString(auth)
    if err != nil : errors.Wrap(err, "invalid token")

    c.Set("sessionId", sid)

    return nil
}

```

Διάγραμμα 3.57. Authentication middleware

Όλα τα route handlers του HTTP server έχουν παρόμοια μορφή, και ο βασικός τους ρόλος είναι να δώσουν στο application το authentication context που χρειάζεται για να αυθεντικοποιήσει τον χρήστη, καθώς και να μετατρέψουν το αίτημα HTTP σε μια μορφή συμβατή με το application model, και να επιστρέψει την αντίστοιχα συμβατή με τον HTTP server απάντηση στη διεπαφή χρήστη. Παρατίθεται παρακάτω παράδειγμα της μεθόδου UpdateEntry, η οποία ενημερώνει ένα συγκεκριμένο keystore entry.

```

func (s *Server) UpdateEntry(c *gin.Context) {
    sid := sessionId(c)
    keystoreId := c.Param("keystoreId")
    entryId := c.Param("entryId")

    var req generated.UpdateEntryRequest
    err := c.ShouldBindJSON(&req)
    if err != nil {
        Error(c, http.StatusBadRequest)
        return
    }

    e, err := s.app.UpdateEntry(sid, keystoreId, entryId,
        application.UpdateEntryOptions{
            Password: req.Password, Notes: req.Notes,
        },
    )
    if err != nil {
        log.Error(err)
        Error(c, http.StatusInternalServerError)
        return
    }

    c.JSON(http.StatusOK, generated.Entry{
        Id:      e.Id,
        Website: e.Website,
        Username: e.Username,
        Password: e.Password,
        Notes:   e.Notes,
    })
}

```

Διάγραμμα 3.58. UpdateEntry route handler

3.1.9. Scheduler

Ένα ακόμη component του backend της εφαρμογής πελάτη είναι ο Scheduler, ο οποίος έχει ρόλο να προγραμματίζει επαναλαμβανόμενα events. Αυτά είναι ο συγχρονισμός των keystores του χρήστη με τον απομακρυσμένο server, και η περιοδική ανανέωση του authentication token. Η βιβλιοθήκη gocron χρησιμοποιείται για να διευκολύνει τον ορισμό των επαναλαμβανόμενων events αυτών.

```

type Scheduler struct {
    app    application.Application
    remote *remote.Remote
    sched  *gocron.Scheduler
    log    *logger.Logger
}

func New(app application.Application, remote *remote.Remote) (*Scheduler, error) {
    s := &Scheduler{
        app:    app,
        remote: remote,
        sched:  gocron.NewScheduler(time.UTC),
        log:    logger.New("scheduler", logger.DefaultColor),
    }

    _, err := s.sched.Every(10).Second().
        StartAt(time.Now()).
        Do(s.sync)
    if err != nil {
        return nil, errors.Wrap(err, "failed to schedule sync")
    }

    _, err = s.sched.Every(10).Minute().
        StartAt(time.Now()).
        Do(s.reauthenticate)
    if err != nil {
        return nil, errors.Wrap(err, "failed to schedule silentAuth")
    }

    return s, nil
}

func (s *Scheduler) Start() error {
    s.sched.StartAsync()

    return nil
}

```

Διάγραμμα 3.59. Scheduler package

3.1.10. Main module

Εν τέλει, όλα τα επιμέρους components της εφαρμογής πελάτη συνδέονται μεταξύ τους με την τεχνική του dependency injection, μέσω του αντίστοιχου main module του client. Η βασική του λειτουργικότητα είναι να κάνει parse τα διάφορα command-line interface (CLI) flags και να κάνει instantiate το κάθε module με τα ανάλογα options. Επίσης, ευθύνεται για το graceful shutdown των components εάν η ζητηθεί ο τερματισμός λειτουργίας της εφαρμογής από τον χρήστη.

```

func run() (cleanup func() error, err error) {
    cfg := parseFlags()

    err = createDataDir(cfg.DataDir)
    if err != nil : nil, errors.WithMessage(err, "unable to create data directory") ↵

    logger.EnableDebug = true

    enc := enclaverepo.New(cfg.DataDir)

    rem, err := remote.New(
        remote.WithAddress(cfg.RemoteAddress),
    )
    if err != nil : nil, errors.WithMessage(err, "unable to create remote repository") ↵

    app := application.New(
        application.WithEnclave(enc),
        application.WithRemote(rem),
    )

    sched, err := scheduler.New(app, rem)
    if err != nil : nil, errors.WithMessage(err, "unable to create scheduler") ↵

    server := rest.NewServer(app, static)

    err = sched.Start()
    if err != nil : nil, errors.WithMessage(err, "unable to start scheduler") ↵

    err = server.Start(fmt.Sprintf(":%d", cfg.Port))
    if err != nil : nil, errors.WithMessage(err, "unable to start server") ↵

    return func() error {
        err = server.Stop()
        if err != nil : errors.WithMessage(err, "unable to stop server") ↵

        err = sched.Stop()
        if err != nil : errors.WithMessage(err, "unable to stop scheduler") ↵

        return nil
    }, nil
}

```

Διάγραμμα 3.60. Υπορουτίνα run

```

func main() {
    cleanup, err := run()
    if err != nil {
        log.Error(err)
        os.Exit(1)
        return
    }

    c := make(chan os.Signal, 1)
    signal.Notify(c, os.Interrupt)

    <-c

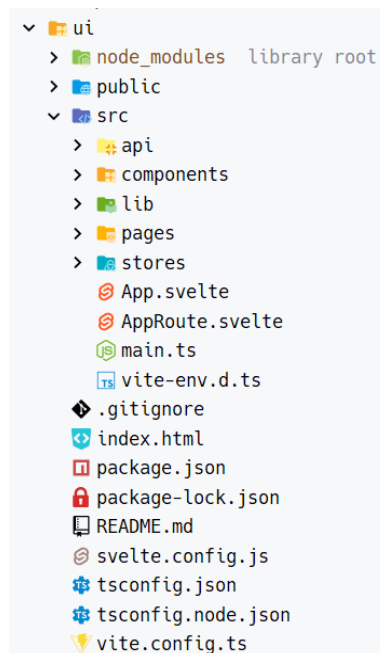
    err = cleanup()
    if err != nil {
        log.Error(err)
        os.Exit(1)
        return
    }
}

```

Διάγραμμα 3.61. Η συνάρτηση main

3.1.11. Διεπαφή χρήστη

Η διεπαφή χρήστη της εφαρμογής πελάτη δημιουργήθηκε χρησιμοποιώντας το Svelte framework, και αποτελείται από διάφορα TypeScript modules τα οποία οργανώνονται σε pages, components, stores και helper libraries. Βρίσκεται εξ' ολοκλήρου στο φάκελο ui εντός του project.

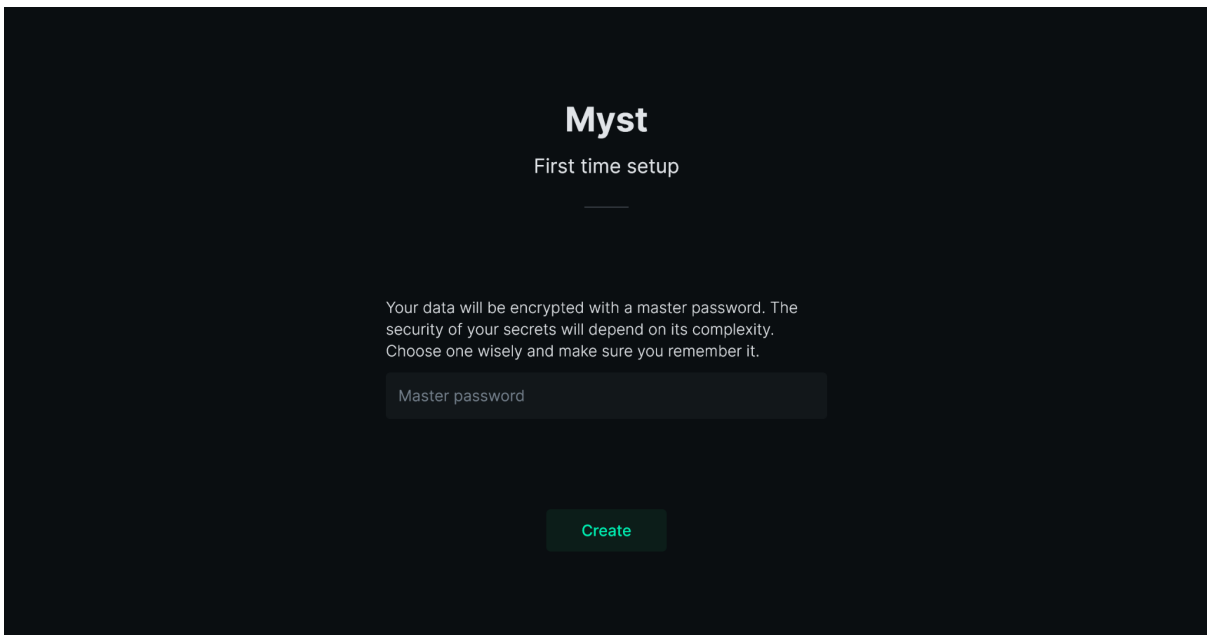


Διάγραμμα 3.62. UI directory

Η εφαρμογή χρησιμοποιεί το built-in Svelte router για navigation, και το entrypoint της αποτελείται από τα αρχεία App.svelte και AppRoute.svelte. Αυτά τα components ελέγχουν την κατάσταση του enclave και της σύνδεσης του χρήστη, και κάνουν render το αντίστοιχο component ανάλογα με το authentication state. Στην περίπτωση που το enclave δεν έχει αρχικοποιηθεί, γίνεται render του OnboardingForm component, το οποίο επιτρέπει στον χρήστη να δημιουργήσει το enclave με ένα master password.

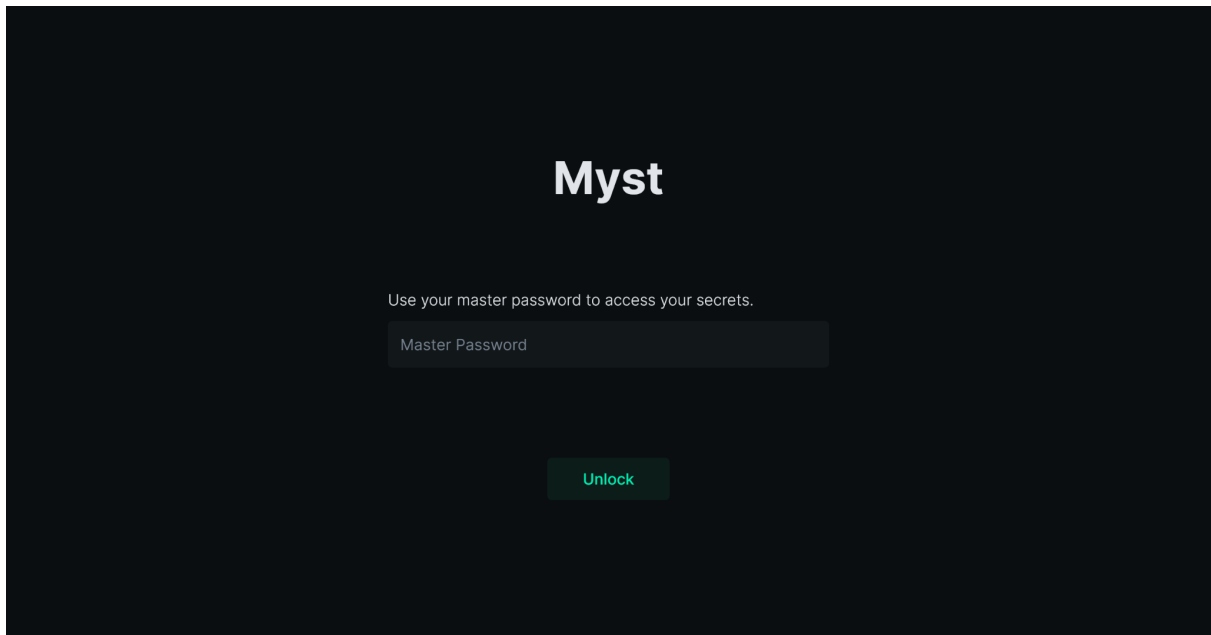
```
{#if $authState !== undefined}
  {#if $authState === AuthState.Onboarding}
    <OnboardingForm on:initialized={async () => {
      await checkState()
    }}/>
  {:else if $authState === AuthState.SignedOut}
    <LoginForm on:login={async () => {
      await checkState()
    }}/>
  {:else if $authState === AuthState.SignedIn}
    <Main/>
  {/if}
{/if}
```

Διάγραμμα 3.63. Auth state conditional rendering



Διάγραμμα 3.64. Onboarding form

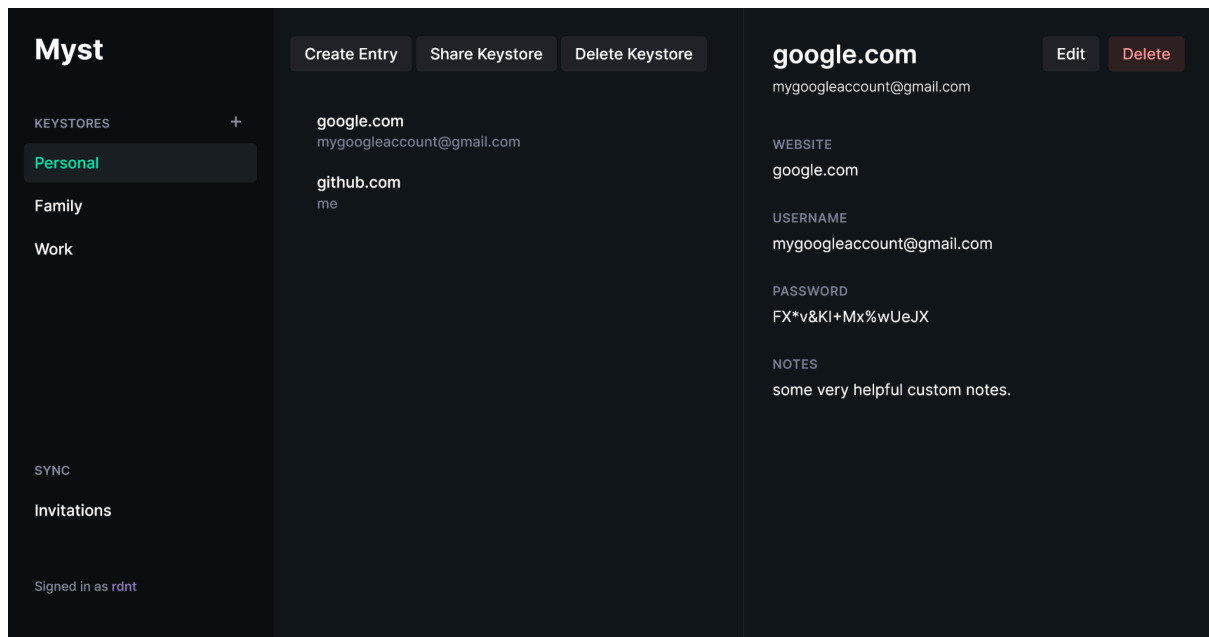
Εάν ο χρήστης έχει προηγουμένως αρχικοποιήσει το enclave, τότε εμφανίζεται το LoginForm component, στο οποίο ο χρήστης μπορεί να συνδεθεί χρησιμοποιώντας το master password που είχε επιλέξει κατά τη διαδικασία του onboarding.



Διάγραμμα 3.65. Login form

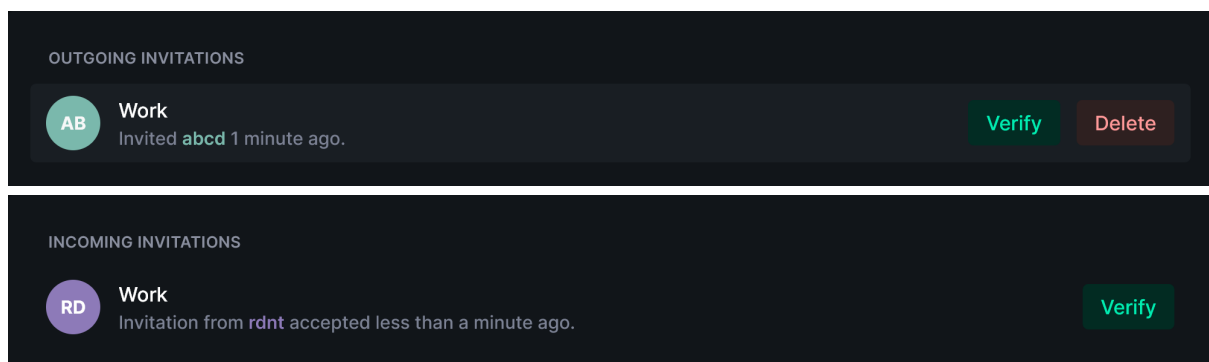
Εφόσον ο χρήστης εισάγει το master password επιτυχώς, τότε γίνεται “Unlock” του enclave, και εμφανίζεται η κεντρική οθόνη της εφαρμογής, και συγκεκριμένα γίνεται rendere του Main component. Σε αυτό το component εμφανίζεται ένα sidebar με τη λίστα των keystores του χρήστη και κάποιες λεπτομέρειες σχετικές με τον συγχρονισμό, όπως το user status και τα invitations του χρήστη, καθώς και η λίστα από τα entries του πρώτου keystore που εμφανίζεται στο sidebar. Από εκεί και πέρα ο χρήστης μπορεί να δημιουργήσει καινούργια keystores, να προσθέσει, επεξεργαστεί ή διαγράψει entries ενός keystore, και εάν έχει δημιουργήσει έναν online λογαριασμό, να μοιραστεί keystores από τη συλλογή του με άλλους χρήστες.

Ακόμη, στην περίπτωση που γίνει ανανέωση του browser tab κατά την περιήγηση, τότε γίνεται αυτόματη αποσύνδεση από το backend του client, και ο χρήστης πρέπει να εισάγει εκ νέου το master password για να συνδεθεί. Τέλος, ο χρήστης μπορεί να είναι συνδεδεμένος από πολλαπλά διαφορετικά tabs, χωρίς κάποιο πρόβλημα διαφθοράς των δεδομένων του, μιας και όλα τα operations στο application προστατεύονται με ένα mutex.

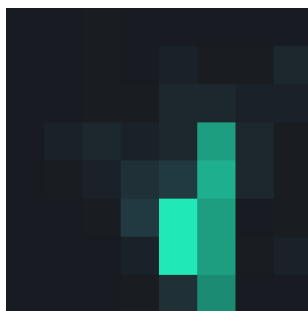


Διάγραμμα 3.66. Main component

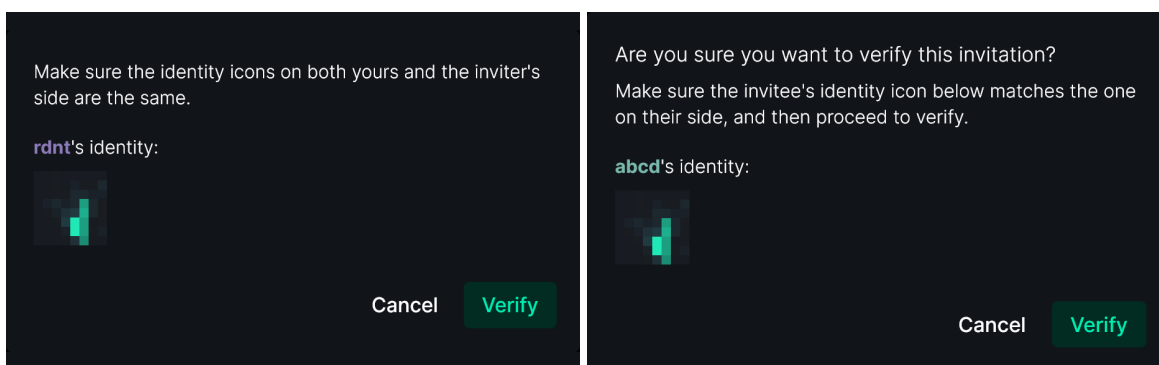
Κατά τη διαδικασία διαμοιρασμού ενός keystore, οι δύο χρήστες μπορούν να επιβεβαιώσουν την εγκυρότητα των κλειδιών κρυπτογράφησης συγκρίνοντας τα “hashicons”, τα οποία είναι εικονίδια που εμφανίζονται όταν ο εκάστοτε χρήστης επιλέξει να αποδεχθεί ή οριστικοποιήσει ένα invitation. Τα hashicons βασίζονται στον αλγόριθμο The Drunken Bishop [39], αποτελούνται από μια SVG εικόνα, και αντιπροσωπεύουν ένα οπτικά αναγνωρίσιμο hash ενός κλειδιού κρυπτογράφησης. Χρησιμοποιώντας τα, οι χρήστες μπορούν με μία ματιά να βεβαιωθούν πως το μυστικό κλειδί είναι έγκυρο πριν προχωρήσουν.



Διάγραμμα 3.67. Incoming/outgoing invitations



Διάγραμμα 3.68. Hashicon of shared secret



Διάγραμμα 3.69. Hashicon verification

3.2. Εφαρμογή Εξυπηρετητή

Η εφαρμογή εξυπηρετητή, ομοίως με το backend της εφαρμογής πελάτη, αφορά στη διαχείριση των οντοτήτων keystores, invitations και users. Οι βασικές λειτουργίες της είναι η αυθεντικοποίηση των χρηστών σε ένα κεντρικό υπολογιστικό σύστημα, και η επίτευξη ενός κοινού καναλιού επικοινωνίας ανάμεσα σε χρήστες, προκειμένου να είναι δυνατός στη συνέχεια ο διαμοιρασμός των keystores ανάμεσά τους.

3.2.1. Domain

Το domain της εφαρμογής εξυπηρετητή αποτελείται από τις δομές User, Keystore, και Invitation. Η δομή User αντιπροσωπεύει έναν χρήστη του συστήματος, και περιέχει πεδία όπως το username, το hash του κωδικού πρόσβασης, καθώς και το δημόσιο κλειδί με το οποίο ο χρήστης δημιούργησε τον λογαριασμό του.

```

type User struct {
    Id            string
    Username      string
    PasswordHash  string
    PublicKey     []byte
    CreatedAt     time.Time
    UpdatedAt     time.Time
}

```

Διάγραμμα 3.70. User struct

Ακολούθως, η δομή Keystore περικλείει τα δεδομένα του κάθε keystore που επιτρέπουν τον συγχρονισμό τους, όπως το όνομα, την αντιστοίχιση με τον κάτοχο του keystore, και το κρυπτογραφημένο payload του keystore.

```

type Keystore struct {
    Id          string
    Name        string
    Payload     []byte
    OwnerId     string
    CreatedAt   time.Time
    UpdatedAt   time.Time
}

```

Διάγραμμα 3.71. Keystore struct

Τέλος, η δομή Invitation περιέχει την κατάσταση μιας πρόσκλησης από έναν χρήστη σε κάποιον άλλο προκειμένου να αποκτήσει ο προσκεκλημένος χρήστης πρόσβαση στο keystore.

```

type Invitation struct {
    Id                string
    KeystoreId        string
    InviterId         string
    InviteeId         string
    EncryptedKeystoreKey []byte
    Status            Status
    CreatedAt         time.Time
    UpdatedAt         time.Time
    DeletedAt         time.Time
    DeclinedAt        time.Time
    AcceptedAt        time.Time
    CancelledAt       time.Time
    FinalizedAt       time.Time
}

```

Διάγραμμα 3.72. Δομή Invitation

3.2.2. Application

Η εφαρμογή του server σχετίζεται κυρίως με τη δημιουργία και αυθεντικοποίηση των χρηστών, τη διαχείριση των keystores (δημιουργία, ενημέρωση), καθώς και την απόκτηση read-only πρόσβασης στους χρήστες που έχουν ολοκληρώσει το invitation process. Το API surface της εφαρμογής απεικονίζεται στο παρακάτω interface.

```
type Application interface {
    CreateUser(username, password string, publicKey []byte) (user.User, error)
    AuthenticateUser(username, password string) (user.User, error)
    User(id string) (user.User, error)
    UserByUsername(username string) (user.User, error)

    CreateKeystore(name, ownerId string, payload []byte) (keystore.Keystore, error)
    Keystore(id string) (keystore.Keystore, error)
    UserKeystore(userId, keystoreId string) (keystore.Keystore, error)
    Keystores() ([]keystore.Keystore, error)
    UpdateKeystore(userId, keystoreId string, opts UpdateKeystoreOptions) (keystore.Keystore, error)
    DeleteKeystore(userId, keystoreId string) error
    UserKeystores(userId string) ([]keystore.Keystore, error)

    CreateInvitation(keystoreId, inviterId, inviteeUsername string) (invitation.Invitation, error)
    AcceptInvitation(userId, invitationId string) (invitation.Invitation, error)
    DeleteInvitation(userId, invitationId string) (invitation.Invitation, error)
    FinalizeInvitation(usrId, invId string, encKeystoreKey []byte) (invitation.Invitation, error)
    UserInvitation(userId, invitationId string) (invitation.Invitation, error)
    UserInvitations(userId string, opts UserInvitationsOptions) ([]invitation.Invitation, error)
}
```

Διάγραμμα 3.73. Interface εφαρμογής εξυπηρετητή

Οι απαιτήσεις της εφαρμογής αποτελούνται από τρία data repositories, ένα για κάθε οντότητα.

```

type InvitationRepository interface {
    CreateInvitation(invitation.Invitation) (invitation.Invitation, error)
    Invitation(id string) (invitation.Invitation, error)
    UpdateInvitation(invitation.Invitation) (invitation.Invitation, error)
    DeleteInvitation(id string) error
    Invitations() ([]invitation.Invitation, error)
}

type KeystoreRepository interface {
    CreateKeystore(keystore.Keystore) (keystore.Keystore, error)
    Keystore(id string) (keystore.Keystore, error)
    UpdateKeystore(k keystore.Keystore) (keystore.Keystore, error)
    DeleteKeystore(id string) error
    Keystores() ([]keystore.Keystore, error)
}

type UserRepository interface {
    CreateUser(user.User) (user.User, error)
    User(id string) (user.User, error)
    UserByUsername(username string) (user.User, error)
    UpdateUser(user.User) (user.User, error)
    Users() ([]user.User, error)
}

```

Διάγραμμα 3.74. Application Repositories

Ξεκινώντας με την υλοποίηση του Application interface, η συνάρτηση που θα κληθεί για πρώτη φορά είναι η CreateUser, η οποία εφόσον είναι διαθέσιμο το username, θα προβεί στη δημιουργία του λογαριασμού του χρήστη. Από τον κωδικό του χρήστη υπολογίζεται ένα hash χρησιμοποιώντας μια one-way hashing συνάρτηση, έτσι ώστε να είναι δυνατή η αυθεντικοποίηση του χρήστη σε επόμενες επισκέψεις του.

```

func (app *application) CreateUser(username, password string, publicKey []byte) (user.User, error) {
    if username == "" : user.User{}, ErrInvalidUsername }

    if password == "" : user.User{}, ErrInvalidPassword }

    hash, err := crypto.HashPassword(password)
    if err != nil : user.User{}, errors.WithMessage(err, "failed to hash password") }

    // make sure username is not taken
    users, err := app.users.Users()
    if err != nil : user.User{}, errors.WithMessage(err, "failed to get users") }

    for _, u := range users {
        if u.Username == username : user.User{}, ErrUsernameTaken }
    }

    u, err := app.users.CreateUser(user.New(
        user.WithUsername(username),
        user.WithPasswordHash(hash),
        user.WithPublicKey(publicKey),
    ))
    if err != nil : user.User{}, errors.WithMessage(err, "failed to create user") }

    return u, nil
}

```

Διάγραμμα 3.75. Συνάρτηση CreateUser

Η συνάρτηση `AuthenticateUser` μπορεί να χρησιμοποιηθεί στη συνέχεια για να επιβεβαιώσει την αυθεντικότητα των διαπιστευτηρίων σύνδεσης του χρήστη.

```
func (app *application) AuthenticateUser(username, password string) (user.User, error) {
    u, err := app.users.UserByUsername(username)
    if errors.Is(err, ErrUserNotFound) {
        return user.User{}, ErrAuthenticationFailed
    } else if err != nil {
        return user.User{}, errors.WithMessage(err, "failed to get user by username")
    }

    ok, err := crypto.VerifyPassword(password, u.PasswordHash)
    if err != nil : user.User{}, errors.WithMessage(err, "failed to verify password")

    if !ok : user.User{}, ErrAuthenticationFailed

    return u, nil
}
```

Διάγραμμα 3.76. Συνάρτηση `AuthenticateUser`

Έχοντας συνδεθεί, ο client που αντιπροσωπεύει τον χρήστη μπορεί ακολούθως να διαχειριστεί τα keystores του χρήστη, δηλαδή να ανεβάσει/δημιουργήσει (`CreateKeystore`), ενημερώσει (`UpdateKeystore`), διαγράψει (`DeleteKeystore`) ή ανακτήσει (`UserKeystores`) το εκάστοτε keystore. Επίσης προσφέρεται η συνάρτηση `UserKeystores`, η οποία επιστρέφει όλα τα keystores του χρήστη.

```
func (app *application) CreateKeystore(
    name, ownerId string, payload []byte) (keystore.Keystore, error) {
    u, err := app.users.User(ownerId)
    if err != nil : keystore.Keystore{}, errors.WithMessage(err, "failed to get user")

    k := keystore.New(
        keystore.WithName(name),
        keystore.WithOwnerId(u.Id),
        keystore.WithPayload(payload),
    )

    k, err = app.keystores.CreateKeystore(k)
    if err != nil : keystore.Keystore{}, errors.WithMessage(err, "failed to create keystore")

    return k, nil
}
```

Διάγραμμα 3.77. Συνάρτηση `CreateKeystore`

```

func (app *application) UpdateKeystore(
    userId, keystoreId string, opts UpdateKeystoreOptions) (keystore.
    Keystore,
    error) {
    k, err := app keystores.Keystore(keystoreId)
    if err != nil : keystore.Keystore{}, errors.WithMessage(err, "failed to get keystore") }

    // check to see if the user should be able to see this keystore
    if k.OwnerId != userId {
        invs, err := app.UserInvitations(
            userId,
            UserInvitationsOptions{Status: optional.Ref(invitation.Finalized)},
        )
        if err != nil {
            return keystore.Keystore{}, errors.WithMessage(err, "failed to get user invitations")
        }

        var found bool
        for _, inv := range invs {
            if inv.KeystoreId == keystoreId {
                found = true
                break
            }
        }

        if found : keystore.Keystore{}, ErrForbidden } else {
            return keystore.Keystore{}, ErrKeystoreNotFound
        }
    }

    // update the keystore's properties
    if opts.Name != nil {
        k.Name = *opts.Name
    }

    if opts.Payload != nil {
        k.Payload = *opts.Payload
    }

    k, err = app keystores.UpdateKeystore(k)
    if err != nil : keystore.Keystore{}, errors.WithMessage(err, "failed to update keystore") }

    return k, nil
}

```

Διάγραμμα 3.78. Συνάρτηση UpdateKeystore

```

func (app *application) DeleteKeystore(userId string, keystoreId string) error {
    _, err := app.users.User(userId)
    if err != nil : errors.WithMessage(err, "failed to get user") ␣

    k, err := app.keystores.Keystore(keystoreId)
    if err != nil : errors.WithMessage(err, "failed to get keystore") ␣

    // check to see if the user should be able to delete this keystore
    if k.OwnerId != userId {
        invs, err := app.UserInvitations(
            userId,
            UserInvitationsOptions{Status: optional.Ref(invitation.Finalized)},
        )
        if err != nil : errors.WithMessage(err, "failed to get user invitations") ␣

        var found bool
        for _, inv := range invs {
            if inv.KeystoreId == keystoreId {
                found = true
                break
            }
        }

        if found : ErrForbidden ␣ else : ErrKeystoreNotFound ␣
    }

    invs, err := app.invitations.Invitations()
    if err != nil : errors.WithMessage(err, "failed to get invitations") ␣

    // delete any associated invitations
    for _, inv := range invs {
        if inv.KeystoreId == keystoreId {
            err = app.invitations.DeleteInvitation(inv.Id)
            if err != nil : errors.WithMessage(err, "failed to delete invitation") ␣
        }
    }

    err = app.keystores.DeleteKeystore(keystoreId)
    if err != nil : errors.WithMessage(err, "failed to delete keystore") ␣

    return nil
}

```

Διάγραμμα 3.79. Συνάρτηση DeleteKeystore

```

func (app *application) UserKeystore(userId, keystoreId string) (keystore.Keystore, error) {
    k, err := app keystores.Keystore(keystoreId)
    if err != nil : keystore.Keystore{}, errors.WithMessage(err, "failed to get keystore") }

    // check to see if the user should be able to see this keystore
    if k.OwnerId != userId {
        invs, err := app.UserInvitations(
            userId,
            UserInvitationsOptions{Status: optional.Ref(invitation.Finalized)},
        )
        if err != nil {
            return keystore.Keystore{}, errors.WithMessage(err, "failed to get user invitations")
        }

        var found bool
        for _, inv := range invs {
            if inv.KeystoreId == keystoreId {
                found = true
                break
            }
        }

        if found : k, nil } else : keystore.Keystore{}, ErrKeystoreNotFound }
    }

    return k, nil
}

```

Διάγραμμα 3.80. Συνάρτηση UserKeystore

```

func (app *application) UserKeystores(userId string) ([]keystore.Keystore, error) {
    u, err := app.users.User(userId)
    if err != nil : nil, errors.WithMessage(err, "failed to get user") }

    invs, err := app.UserInvitations(
        u.Id,
        UserInvitationsOptions{Status: optional.Ref(invitation.Finalized)},
    )
    if err != nil : nil, errors.WithMessage(err, "failed to get user invitations") }

    keystores := []keystore.Keystore{}
    for _, inv := range invs {
        k, err := app keystores.Keystore(inv.KeystoreId)
        if err != nil : nil, errors.WithMessage(err, "failed to get keystore") }

        keystores = append(keystores, k)
    }

    return keystores, nil
}

```

Διάγραμμα 3.81. Συνάρτηση UserKeystores

Επιπρόσθετα, παρέχονται οι συναρτήσεις Keystore και Keystores, προκειμένου να είναι δυνατή η ανάκτηση ενός ή όλων των keystores αντίστοιχα, χωρίς την εξουσιοδότηση κάποιου συγκεκριμένου χρήστη.

```
func (app *application) Keystore(keystoreId string) (keystore.Keystore, error) {
    k, err := app.keystores.Keystore(keystoreId)
    if err != nil : keystore.Keystore{}, errors.WithMessage(err, "failed to get keystore") }

    return k, nil
}
```

Διάγραμμα 3.82. Συνάρτηση Keystore

```
func (app *application) Keystores() ([]keystore.Keystore, error) {
    ks, err := app.keystores.Keystores()
    if err != nil : nil, errors.WithMessage(err, "failed to get keystores") }

    return ks, nil
}
```

Διάγραμμα 3.83. Συνάρτηση Keystores

Ακόμη, η εφαρμογή εξυπηρετητή διαθέτει συναρτήσεις που επιτρέπουν τη δημιουργία invitations για ένα keystore από έναν χρήστη προς κάποιον άλλο (CreateInvitation), συναρτήσεις μεταβολής της κατάστασης του invitation ή διαγραφής του (AcceptInvitation, FinalizeInvitation, DeleteInvitation) αλλά και συναρτήσεις για την ανάκτηση όλων ή ενός συγκεκριμένου invitation από την οπτική γωνία του εκάστοτε συνδεδεμένου χρήστη.

Η συνάρτηση CreateInvitation, εφόσον γίνει έλεγχος εγκυρότητας του αιτήματος δημιουργίας, όπως έλεγχος για τα δικαιώματα του χρήστη και της απουσίας υπάρχουσας πρόσκλησης που αφορά στους συγκεκριμένους χρήστες για αυτό το keystore, προβαίνει στη δημιουργία του invitation.

```

func (app *application) CreateInvitation(
    keystoreId, inviterId, inviteeUsername string) (invitation.Invitation, error) {
    inviter, err := app.users.User(inviterId)
    if errors.Is(err, ErrUserNotFound) {
        return invitation.Invitation{}, ErrInviterNotFound
    } else if err != nil {
        return invitation.Invitation{}, errors.WithMessage(err, "failed to get inviter by id")
    }

    k, err := app keystores.Keystore(keystoreId)
    if err != nil {
        return invitation.Invitation{}, errors.WithMessage(err, "failed to get keystore by id")
    }

    if k.OwnerId != inviter.Id : invitation.Invitation{}, ErrForbidden }

    invitee, err := app.users.UserByUsername(inviteeUsername)
    if errors.Is(err, ErrUserNotFound) {
        return invitation.Invitation{}, ErrInviteeNotFound
    } else if err != nil {
        return invitation.Invitation{}, errors.WithMessage(err, "failed to get invitee by username")
    }

    if inviter.Id == invitee.Id {
        return invitation.Invitation{}, ErrInvalidInvitee
    }

    invs, err := app.invitations.Invitations()
    if err != nil {
        return invitation.Invitation{}, errors.WithMessage(err, "failed to get invitations")
    }

    for _, inv := range invs {
        if inv.InviterId == inviter.Id && inv.InviteeId == invitee.Id && inv.KeystoreId == k.Id {
            return invitation.Invitation{}, ErrForbidden
        }
    }

    inv := invitation.New(
        invitation.WithKeystoreId(k.Id),
        invitation.WithInviterId(inviter.Id),
        invitation.WithInviteeId(invitee.Id),
    )

    inv, err = app.invitations.CreateInvitation(inv)
    if err != nil {
        return invitation.Invitation{}, errors.WithMessage(err, "failed to create invitation")
    }

    return inv, nil
}

```

Διάγραμμα 3.84. Συνάρτηση CreateInvitation

Εφόσον το invitation έχει δημιουργηθεί, ο invitee μπορεί να το αποδεχθεί χρησιμοποιώντας τη συνάρτηση AcceptInvitation. Με την κλήση της από τον σωστό χρήστη, το state του invitation αλλάζει στην τιμή “accepted”.

```

func (app *application) AcceptInvitation(
    userId string, invitationId string) (invitation.Invitation, error) {
    inv, err := app.UserInvitation(userId, invitationId)
    if err != nil {
        return invitation.Invitation{}, errors.WithMessage(err, "failed to get user invitation")
    }

    if userId != inv.InviteeId : invitation.Invitation{}, ErrForbidden }

    if inv.Status != invitation.Pending : invitation.Invitation{}, ErrForbidden }

    inv.Status = invitation.Accepted
    inv.AcceptedAt = time.Now()

    inv, err = app.invitations.UpdateInvitation(inv)
    if err != nil {
        return invitation.Invitation{}, errors.WithMessage(err, "failed to update invitation")
    }

    return inv, nil
}

```

Διάγραμμα 3.85. Συνάρτηση AcceptInvitation

Ο δημιουργός του invitation μπορεί ακολούθως να ολοκληρώσει τη διαδικασία καλώντας τη συνάρτηση FinalizeInvitation. Η συνάρτηση αυτή δέχεται ως όρισμα το encryptedKeystoreKey, που είναι ένα κλειδί που προσφέρει πρόσβαση στο keystore, το οποίο όμως είναι το ίδιο κρυπτογραφημένο από τον δημιουργό του invitation, με ένα κλειδί συμφωνημένο με τον παραλήπτη.

```

func (app *application) FinalizeInvitation(
    userId, invitationId string, encryptedKeystoreKey []byte) (invitation.Invitation, error) {
    inv, err := app.invitations.Invitation(invitationId)
    if err != nil {
        return invitation.Invitation{}, errors.WithMessage(err, "failed to get invitation")
    }

    if userId != inv.InviterId && userId != inv.InviteeId {
        return invitation.Invitation{}, ErrInvitationNotFound
    }

    if userId != inv.InviterId : invitation.Invitation{}, ErrForbidden }

    if !inv.Accepted() : invitation.Invitation{}, ErrForbidden }

    inv.EncryptedKeystoreKey = encryptedKeystoreKey
    inv.Status = invitation.Finalized
    inv.FinalizedAt = time.Now()

    inv, err = app.invitations.UpdateInvitation(inv)
    if err != nil {
        return invitation.Invitation{}, errors.WithMessage(err, "failed to update invitation")
    }

    return inv, nil
}

```

Διάγραμμα 3.86. Συνάρτηση FinalizeInvitation

Οποιοσδήποτε χρήστης ανά πάσα στιγμή μπορεί να ακυρώσει τη διαδικασία αυτή χρησιμοποιώντας τη συνάρτηση `DeleteInvitation`, εάν δεν επιθυμεί να αποδεχθεί την πρόσκληση ή αν παρατηρήσει κάτι παράξενο, για παράδειγμα στην περίπτωση μιας επίθεσης `Man-In-The-Middle`. Η συνάρτηση `DeleteInvitation`, ανάλογα με την κατάσταση της πρόσκλησης και τον χρήστη που την κάλεσε, είτε ακυρώνει, ή απορρίπτει, ή διαγράφει την πρόσκληση.

```
func (app *application) DeleteInvitation(
    userId, invitationId string) (invitation.Invitation, error) {
    inv, err := app.invitations.Invitation(invitationId)
    if err != nil {
        return invitation.Invitation{}, errors.WithMessage(err, "failed to get invitation")
    }

    if userId != inv.InviterId && userId != inv.InviteeId {
        return invitation.Invitation{}, ErrInvitationNotFound
    }

    if userId == inv.InviterId {
        inv, err = app.deleteOrCancelInvitation(inv)
        if err != nil {
            return invitation.Invitation{}, errors.WithMessage(err,
                "failed to delete or cancel invitation")
        }
    } else if userId == inv.InviteeId {
        inv, err = app.declineInvitation(inv)
        if err != nil {
            return invitation.Invitation{}, errors.WithMessage(err,
                "failed to decline invitation")
        }
    }

    return inv, nil
}
```

Διάγραμμα 3.87. Συνάρτηση `DeleteInvitation`

```

func (app *application) deleteOrCancelInvitation(inv invitation.Invitation,
) (invitation.Invitation, error) {
    if inv.Pending() {
        inv.Status = invitation.Deleted
        inv.DeletedAt = time.Now()
    } else if inv.Accepted() {
        inv.Status = invitation.Cancelled
        inv.CancelledAt = time.Now()
    } else : invitation.Invitation{}, ErrForbidden }

    var err error
    inv, err = app.invitations.UpdateInvitation(inv)
    if err != nil {
        return invitation.Invitation{}, errors.WithMessage(err, "failed to update invitation")
    }

    return inv, nil
}

func (app *application) declineInvitation(inv invitation.Invitation) (invitation.Invitation, error) {
    if !inv.Pending() : invitation.Invitation{}, ErrForbidden }

    inv.Status = invitation.Declined
    inv.DeclinedAt = time.Now()

    var err error
    inv, err = app.invitations.UpdateInvitation(inv)
    if err != nil {
        return invitation.Invitation{}, errors.WithMessage(err, "failed to update invitation")
    }

    return inv, nil
}

```

Διάγραμμα 3.88. Υπορουτίνες deleteOrCancelInvitation και declineInvitation

Με το finalization του invitation, ο invitee πλέον μπορεί να ανακτήσει το keystore χρησιμοποιώντας τη συνάρτηση Keystores που αναφέρθηκε νωρίτερα.

3.2.3. HTTP REST Server

Όλες οι λειτουργίες του server είναι προσβάσιμες χρησιμοποιώντας το HTTP REST API του server module. Το API αυτό, ομοίως με το HTTP API του client, χρησιμοποιεί το Gin web framework για τον ορισμό των επιμέρους routes αλλά και middleware για logging, panic recovery και αυθεντικοποίηση. Για την ταυτοποίηση των χρηστών χρησιμοποιούνται JSON Web Tokens (JWT), τα οποία κωδικοποιούν την ταυτότητα του χρήστη και “υπογράφονται” χρησιμοποιώντας ένα κλειδί που γνωρίζει μόνο ο server, με σκοπό την μελλοντική επαλήθευση της ταυτότητας του χρήστη.

```

func NewServer(app application.Application, jwtSigningKey []byte) *Server {
    s := &Server{
        app:          app,
        jwtSigningKey: jwtSigningKey,
    }

    // Set gin mode
    gin.SetMode(gin.DebugMode)

    gin.DefaultWriter = io.Discard
    gin.DefaultErrorWriter = io.Discard

    r := gin.New()
    s.Engine = r

    // Do not redirect folders to trailing slash
    r.RedirectTrailingSlash = true
    r.RedirectFixedPath = true

    // Custom PrintRouteFunc
    gin.DebugPrintRouteFunc = printRoutes

    r.Use(recoveryMiddleware)
    r.Use(loggerMiddleware)
    r.NoRoute(noRouteMiddleware)

    r.GET("/", func(c *gin.Context) {
        c.String(http.StatusOK, ":)")
    })

    s.initRoutes()

    return s
}

```

Διάγραμμα 3.89. HTTP Server constructor

```

func (s *Server) initRoutes() {
    api := s.Group("/api")

    api.GET("/health", func(c *gin.Context) {
        c.JSON(http.StatusOK, map[string]any{
            "status": "healthy",
        })
    })

    api.POST("/auth/login", s.Login)
    api.POST("/auth/register", s.Register)

    sec := api.Group("/")
    sec.Use(s.authenticationMiddleware())

    sec.GET("/user", s.UserByUsername)

    sec.POST("/keystores", s.CreateKeystore)
    sec.GET("/keystore/:keystoreId", s.Keystore)
    sec.PATCH("/keystore/:keystoreId", s.UpdateKeystore)
    sec.DELETE("/keystore/:keystoreId", s.DeleteKeystore)
    sec.GET("/keystores", s.Keystores)

    sec.POST("/keystore/:keystoreId/invitations", s.CreateInvitation)
    sec.GET("/invitation/:invitationId", s.Invitation)
    sec.PATCH("/invitation/:invitationId", s.AcceptInvitation)
    sec.POST("/invitation/:invitationId", s.FinalizeInvitation)
    sec.DELETE("/invitation/:invitationId", s.DeleteInvitation)
    sec.GET("/invitations", s.Invitations)
}

```

Διάγραμμα 3.90. Υπορουτίνα initRoutes

3.2.4. Main module

Στη συνάρτηση main γίνεται parse των CLI flags του server, όπως το JWT signing key, database credentials, καθώς και το HTTP port. Επίσης γίνεται inject των domain entity repositories, και η αρχικοποίηση του ίδιου του application και του HTTP server.

```

func run() (cleanup func() error, err error) {
    cfg := parseFlags()

    jwtSigningKey, err := base64.StdEncoding.DecodeString(cfg.JWTSigningKey)
    if err != nil : nil, errors.Wrap(err, "could not decode jwt signing key") ↵

    logger.EnableDebug = true

    repo, err := mongorepo.New(cfg.MongoAddress, cfg.MongoDatabase)
    if err != nil : nil, errors.WithMessage(err, "could not create mongo repository") ↵

    app := application.New(
        application.WithKeystoreRepository(repo),
        application.WithUserRepository(repo),
        application.WithInvitationRepository(repo),
    )

    server := rest.NewServer(app, jwtSigningKey)

    err = server.Start(":8080")
    if err != nil : nil, errors.WithMessage(err, "could not start server") ↵

    return func() error {
        err := server.Stop()
        if err != nil : errors.WithMessage(err, "could not stop server") ↵

        return nil
    }, nil
}

```

Διάγραμμα 3.91. Main module

4. Testing Infrastructure

Προκειμένου να γίνει πιο γρήγορη η υλοποίηση νέων λειτουργιών και η επιβεβαίωση πως δεν έχει επηρεαστεί η λειτουργικότητα υπαρχόντων, δημιουργήθηκε μια σουίτα με tests εντοποίησης (integration test suite). Η συγκεκριμένη σουίτα αποτελείται από συναρτήσεις που δημιουργούν πολλαπλά runtime instances του client και ένα του server, έτσι ώστε να είναι δυνατή η συγγραφή tests τα οποία προσομοιώνουν ενέργειες πολλαπλών χρηστών ενάντιας του server. Για την συγγραφή της σουίτας και των tests χρησιμοποιήθηκε το built-in testing πακέτο της βιβλιοθήκης της Go σε συνδυασμό με το helper library `gotest.tools/assert`.

```

func New(t *testing.T) *Suite {
    t.Parallel()

    ctx, cancel := context.WithTimeout(context.Background(), 30*time.Second)

    logger.EnableDebug = true

    ports, err := freeport.GetFreePorts(4)
    assert.NoError(t, err)

    addrs := lo.Map(ports, func(port int, _ int) string {
        return fmt.Sprintf("127.0.0.1:#{port}")
    })

    var server *Server
    var client1, client2, client3 *Client

    server = newServer(t, addrs[0])
    client1 = newClient(t, addrs[0], addrs[1])
    client2 = newClient(t, addrs[0], addrs[2])
    client3 = newClient(t, addrs[0], addrs[3])

    server.start(t)
    client1.start(t)
    client2.start(t)
    client3.start(t)

    t.Cleanup(func() {
        cancel()

        client1.stop(t)
        client2.stop(t)
        client3.stop(t)

        server.stop(t)
    })

    return &Suite{
        Ctx:    ctx,
        Server: server,
        Client1: client1,
        Client2: client2,
        Client3: client3,
    }
}

```

Διάγραμμα 4.1. Integration tests suite

Χρησιμοποιώντας τη συγκεκριμένη σουίτα, δημιουργήθηκαν διάφορα tests, για παράδειγμα tests για την αυθεντικοποίηση των χρηστών, για τα invitations ανάμεσα σε χρήστες, καθώς και για τη δημιουργία keystores και τον συγχρονισμό τους.

```
f 🗄 TestKeystoreSync(t *testing.T)
f 🗄 TestAuthentication(t *testing.T)
f 🗄 TestKeystores(t *testing.T)
f 🗄 TestInvitationCreate(t *testing.T)
f 🗄 TestInvitationDelete(t *testing.T)
f 🗄 TestInvitationDecline(t *testing.T)
f 🗄 TestInvitationAccept(t *testing.T)
f 🗄 TestInvitationFinalize(t *testing.T)
```

Διάγραμμα 4.2. Integration tests

Όλα τα tests δημιουργήθηκαν χρησιμοποιώντας το AAA pattern (Arrange-Act-Assert), και περιλαμβάνουν την αρχικοποίηση του περιβάλλοντος (arrange), την εκτέλεση μίας ή περισσότερων ενεργειών (act) και τον έλεγχο της κατάστασης των εφαρμογών (assert).

Η βασική λειτουργικότητα της εφαρμογής είναι η διαχείριση των keystores του χρήστη. Η συνάρτηση TestKeystores ελέγχει την λειτουργικότητα των διαφόρων συναρτήσεων οι οποίες σχετίζονται με τη δημιουργία, ενημέρωση (προσθήκη και επεξεργασία από entries) και διαγραφή των keystores.

```

func TestKeystores(t *testing.T) {
    s := suite.New(t)

    s1id, err := s.Client1.App.Initialize(s.Client1.MasterPassword)
    assert.NoError(t, err)
    s.Client1.SessionId = s1id

    t.Run("keystoresWithKeys are empty", func(t *testing.T) {...})

    keystoreName := strings.TrimSpace(s.Random(t))
    var keystoreId string

    t.Run("keystore created", func(t *testing.T) {
        res, err := s.Client1.Client.CreateKeystoreWithResponse(s.Ctx,
            generated.CreateKeystoreJSONRequestBody{Name: keystoreName},
        )
        assert.NoError(t, err)
        assert.Assert(t, res.JSON201 != nil)
        assert.Equal(t, res.JSON201.Name, keystoreName)

        keystoreId = (*res.JSON201).Id
    })

    t.Run("keystore can be queried", func(t *testing.T) {...})

    t.Run("keystoresWithKeys contain created keystore", func(t *testing.T) {...})

    website := s.Random(t)
    username := s.Random(t)
    password := s.Random(t)
    notes := s.Random(t)
    var entryId string

    t.Run("Entry created", func(t *testing.T) {
        res, err := s.Client1.Client.CreateEntryWithResponse(s.Ctx, keystoreId,
            generated.CreateEntryJSONRequestBody{
                Website: website, Username: username, Password: password, Notes: notes,
            },
        )
        assert.NoError(t, err)
        assert.Assert(t, res.JSON201 != nil)
        assert.Equal(t, res.JSON201.Website, website)
        assert.Equal(t, res.JSON201.Username, username)
        assert.Equal(t, res.JSON201.Password, password)
        assert.Equal(t, res.JSON201.Notes, notes)

        entryId = (*res.JSON201).Id
    })

    t.Run("Entry can be queried", func(t *testing.T) {...})

    t.Run("Entry can be updated", func(t *testing.T) {...})

    t.Run("Entry can be deleted", func(t *testing.T) {...})

    t.Run("keystore can be deleted", func(t *testing.T) {...})
}

```

Διάγραμμα 4.3. Συνάρτηση TestKeystores

Σχετικά με το authentication και γενικά την λειτουργικότητα με το cloud, υλοποιήθηκε η test συνάρτηση TestAuthentication, στην οποία γίνεται initialization του keystore, και ακολούθως registration του χρήστη.

```

func TestAuthentication(t *testing.T) {
    s := suite.New(t)

    sessionId, err := s.Client1.App.Initialize(s.Client1.MasterPassword)
    assert.NoError(t, err)

    _, err = s.Client1.App.Register(sessionId, s.Client1.Username, s.Client1.Password)
    assert.NoError(t, err)

    u, err := s.Client1.App.CurrentUser(sessionId)
    assert.NoError(t, err)
    assert.Equal(t, u.Username, s.Client1.Username)
}

```

Διάγραμμα 4.4. Authentication test

Για τα invitations υλοποιήθηκαν tests που ελέγχουν όλο το lifecycle ενός invitation. Πιο συγκεκριμένα, υλοποιήθηκαν τα tests TestInvitationCreate, TestInvitationDelete, TestInvitationDecline, TestInvitationAccept και TestInvitationFinalize.

Στη συνάρτηση TestInvitationCreate δημιουργούνται δύο clients, και ο ένας από τους δύο δημιουργεί ένα keystore και αμέσως μετά δημιουργεί ένα invitation προς τον άλλο χρήστη. Στη συνέχεια γίνεται έλεγχος για τα δικαιώματα πρόσβασης του κάθε χρήστη.

```

func TestInvitationCreate(t *testing.T) {
    s := suite.New(t)

    sessionId, err := s.Client1.App.Initialize(s.Client1.MasterPassword)
    assert.NoError(t, err)
    s.Client1.SessionId = sessionId

    _, err = s.Client1.App.Register(sessionId, s.Client1.Username, s.Client1.Password)
    assert.NoError(t, err)

    session2Id, err := s.Client2.App.Initialize(s.Client2.MasterPassword)
    assert.NoError(t, err)
    s.Client2.SessionId = session2Id

    _, err = s.Client2.App.Register(session2Id, s.Client2.Username, s.Client2.Password)
    assert.NoError(t, err)

    session3Id, err := s.Client3.App.Initialize(s.Client3.MasterPassword)
    assert.NoError(t, err)
    s.Client3.SessionId = session3Id

    _, err = s.Client3.App.Register(session3Id, s.Client3.Username, s.Client3.Password)
    assert.NoError(t, err)

    keystore := s.CreateKeystore(t)

    var invitationId string
    t.Run("Invitation is created", func(t *testing.T) {
        res, err := s.Client1.Client.CreateInvitationWithResponse(s.Ctx,
            keystore.Id,
            generated.CreateInvitationJSONRequestBody{
                Invitee: s.Client2.Username,
            },
        )
        assert.NoError(t, err)
        assert.Assert(t, res.JSON201 != nil)

        invitationId = res.JSON201.Id
    })

    t.Run("The inviter has access to the invitation", func(t *testing.T) {...})

    t.Run("The invitee has access to the invitation", func(t *testing.T) {...})

    t.Run("Another user doesn't have access to the invitation", func(t *testing.T) {...})
}

```

Διάγραμμα 4.5. Συνάρτηση TestInvitationCreate

Στο test TestInvitationDelete δημιουργείται με παρόμοιο τρόπο ένα invitation, το οποίο όμως διαγράφεται πριν μπορέσει να γίνει accept από τον invited χρήστη.

```

func TestInvitationDelete(t *testing.T) {
    s := suite.New(t)

    s1id, err := s.Client1.App.Initialize(s.Client1.MasterPassword)
    assert.NoError(t, err)
    s.Client1.SessionId = s1id

    _, err = s.Client1.App.Register(s1id, s.Client1.Username, s.Client1.Password)
    assert.NoError(t, err)

    s2id, err := s.Client2.App.Initialize(s.Client2.MasterPassword)
    assert.NoError(t, err)
    s.Client2.SessionId = s2id

    _, err = s.Client2.App.Register(s2id, s.Client2.Username, s.Client2.Password)
    assert.NoError(t, err)

    keystore := s.CreateKeystore(t)
    invitationId := s.CreateInvitation(t, keystore.Id)

    t.Run("The inviter deletes the invitation", func(t *testing.T) {
        res, err := s.Client1.Client.DeclineOrCancelInvitationWithResponse(
            s.Ctx, invitationId,
        )
        assert.NoError(t, err)
        assert.Assert(t, res.JSON200 != nil)
        assert.Equal(t, res.JSON200.Status, generated.Deleted)
        assert.Assert(t, res.JSON200.DeletedAt != time.Time{})
    })

    t.Run("Only the inviter has access to the deleted invitation", func(t *testing.T) {...})
}

```

Διάγραμμα 4.6. Συνάρτηση TestInvitationDelete

Αντ' αυτού, στο TestInvitationDecline ο invitee αρνείται το invitation, μαρκάροντας το ως "declined", και το invitation παραμένει αναγνώσιμο και από τους δύο χρήστες.

```

func TestInvitationDecline(t *testing.T) {
    s := suite.New(t)

    s1id, err := s.Client1.App.Initialize(s.Client1.MasterPassword)
    assert.NoError(t, err)
    s.Client1.SessionId = s1id

    _, err = s.Client1.App.Register(s1id, s.Client1.Username, s.Client1.Password)
    assert.NoError(t, err)

    s2id, err := s.Client2.App.Initialize(s.Client2.MasterPassword)
    assert.NoError(t, err)
    s.Client2.SessionId = s2id

    _, err = s.Client2.App.Register(s2id, s.Client2.Username, s.Client2.Password)
    assert.NoError(t, err)

    keystore := s.CreateKeystore(t)
    invitationId := s.CreateInvitation(t, keystore.Id)

    t.Run("The invitee declines the invitation", func(t *testing.T) {
        res, err := s.Client2.Client.DeclineOrCancelInvitationWithResponse(
            s.Ctx, invitationId,
        )
        assert.NoError(t, err)
        assert.Assert(t, res.JSON200 != nil)
        assert.Equal(t, res.JSON200.Status, generated.Declined)
        assert.Assert(t, res.JSON200.DeclinedAt != time.Time{})
    })

    t.Run("Both users see invitation as declined", func(t *testing.T) {...})
}

```

Διάγραμμα 4.7. Συνάρτηση TestInvitationDecline

Στο test TestInvitationAccept, ο invitee χρήστης αποδέχεται το invitation και γίνεται επιβεβαίωση πως κανένας χρήστης δεν μπορεί να αλλάξει το status του invitation σε accepted.

```

func TestInvitationAccept(t *testing.T) {
    s := suite.New(t)

    s1id, err := s.Client1.App.Initialize(s.Client1.MasterPassword)
    assert.NoError(t, err)
    s.Client1.SessionId = s1id

    _, err = s.Client1.App.Register(s1id, s.Client1.Username, s.Client1.Password)
    assert.NoError(t, err)

    s2id, err := s.Client2.App.Initialize(s.Client2.MasterPassword)
    assert.NoError(t, err)
    s.Client2.SessionId = s2id

    _, err = s.Client2.App.Register(s2id, s.Client2.Username, s.Client2.Password)
    assert.NoError(t, err)

    keystore := s.CreateKeystore(t)
    invitationId := s.CreateInvitation(t, keystore.Id)

    t.Run("The inviter cannot accept the invitation", func(t *testing.T) {...})

    t.Run("The invitee accepts the invitation", func(t *testing.T) {
        res, err := s.Client2.Client.AcceptInvitationWithResponse(s.Ctx, invitationId)
        assert.NoError(t, err)
        assert.Assert(t, res.JSON200 != nil)

        res2, err := s.Client1.Client.GetInvitationWithResponse(s.Ctx, invitationId)
        assert.NoError(t, err)
        assert.Assert(t, res2.JSON200 != nil)
        assert.Equal(t, res2.JSON200.Status, generated.Accepted)

        res2, err = s.Client2.Client.GetInvitationWithResponse(s.Ctx, invitationId)
        assert.NoError(t, err)
        assert.Assert(t, res2.JSON200 != nil)
        assert.Equal(t, res2.JSON200.Status, generated.Accepted)
    })

    t.Run("Invitation cannot now be accepted by either user", func(t *testing.T) {...})
}

```

Διάγραμμα 4.8. Συνάρτηση TestInvitationAccept

Τέλος, σχετικά με τα invitations υλοποιήθηκε η συνάρτηση TestInvitationFinalize, στην οποία ο χρήστης που δημιουργήσει το invitation το κάνει finalize, δίνοντας στον invitee το EncryptedKeystoreKey.

```

func TestInvitationFinalize(t *testing.T) {
    s := suite.New(t)

    s1id, err := s.Client1.App.Initialize(s.Client1.MasterPassword)
    assert.NoError(t, err)
    s.Client1.SessionId = s1id

    _, err = s.Client1.App.Register(s1id, s.Client1.Username, s.Client1.Password)
    assert.NoError(t, err)

    s2id, err := s.Client2.App.Initialize(s.Client2.MasterPassword)
    assert.NoError(t, err)
    s.Client2.SessionId = s2id

    _, err = s.Client2.App.Register(s2id, s.Client2.Username, s.Client2.Password)
    assert.NoError(t, err)

    keystore := s.CreateKeystore(t)
    invitationId := s.CreateInvitation(t, keystore.Id)
    var inviteePublicKey []byte
    var keystoreRemoteId string

    t.Run("The invitee accepts the invitation", func(t *testing.T) {...})

    t.Run("The inviter finalizes the invitation", func(t *testing.T) {
        res, err := s.Client1.Client.FinalizeInvitationWithResponse(s.Ctx, invitationId,
            generated.FinalizeInvitationRequest{
                InviteePublicKey: inviteePublicKey,
                RemoteKeystoreId: keystoreRemoteId,
            })
        assert.NoError(t, err)
        assert.Assert(t, res.JSON200 != nil)
    })

    t.Run("Both users see invitation as finalized", func(t *testing.T) {...})
}

```

Διάγραμμα 4.9. Συνάρτηση TestInvitationFinalize

Εφόσον έχει γίνει το creation ενός invitation, μπορεί να ξεκινήσει η διαδικασία του συγχρονισμού ανάμεσα στους δύο χρήστες. Η συνάρτηση TestKeystoreSync δημιουργήθηκε για αυτόν ακριβώς τον σκοπό, και συγκεκριμένα κατά την εκτέλεσή της, δημιουργείται ένα invitation, και αμέσως μετά γίνεται επιβεβαίωση πως το encrypted keystore payload έχει ανέβει στον server.

```

func TestKeystoreSync(t *testing.T) {
    s := suite.New(t)

    s1id, err := s.Client1.App.Initialize(s.Client1.MasterPassword)
    assert.NoError(t, err)
    s.Client1.SessionId = s1id

    _, err = s.Client1.App.Register(s1id, s.Client1.Username, s.Client1.Password)
    assert.NoError(t, err)

    s2id, err := s.Client2.App.Initialize(s.Client2.MasterPassword)
    assert.NoError(t, err)
    s.Client2.SessionId = s2id

    _, err = s.Client2.App.Register(s2id, s.Client2.Username, s.Client2.Password)
    assert.NoError(t, err)

    var keystore generated.Keystore
    t.Run("keystore is created", func(t *testing.T) {...})

    t.Run("keystore is not uploaded yet", func(t *testing.T) {...})

    t.Run("invitation is created for this keystore", func(t *testing.T) {...})

    t.Run("keystore is uploaded", func(t *testing.T) {
        rks, err := s.Server.App.Keystores()
        assert.NoError(t, err)
        assert.Equal(t, len(rks), 1)
        assert.Assert(t, rks[0].Id != "")
    })

    t.Run("keystore is synced", func(t *testing.T) {
        err := s.Client1.App.Sync()
        assert.NoError(t, err)

        rks, err := s.Server.App.Keystores()
        assert.NoError(t, err)
        assert.Equal(t, len(rks), 1)
        assert.Assert(t, rks[0].Id != "")

        res, err := s.Client1.Client.KeystoresWithResponse(s.Ctx)
        assert.NoError(t, err)
        assert.Assert(t, res.JSON200 != nil)
        assert.Assert(t, len(*res.JSON200) == 1)
        assert.Equal(t, (*res.JSON200)[0].RemoteId, rks[0].Id)

        ks, err := s.Client1.App.Keystores(s1id)
        assert.NoError(t, err)
        assert.Assert(t, len(ks) == 1)
        assert.Equal(t, ks[keystore.Id].Version, 3)
    })

    t.Run("Additional sync noop", func(t *testing.T) {...})
}

```

Διάγραμμα 4.10. Συνάρτηση TestKeystoreSync

5. Building & Deployment

Για τη διευκόλυνση της διαδικασίας building, δημιουργήθηκε το Myst Build CLI, το οποίο είναι ουσιαστικά ένα portable build script γραμμένο σε Go, έτσι ώστε να μπορεί εύκολα να τρέξει σε όλα τα δημοφιλή λειτουργικά χωρίς την ανάγκη συντήρησης διαφορετικών build scripts για το καθένα. Το build και των δύο components, της εφαρμογής πελάτη και της εφαρμογής εξυπηρετητή, γίνεται χρησιμοποιώντας αυτό το build CLI. Εντός αυτού ορίζονται τα ακόλουθα CLI commands:

- Bootstrap: επιβεβαίωση των build dependencies των εφαρμογών, δηλαδή της Go και των Node.js/npm, και εγκατάσταση περαιτέρω dependencies όπως των εργαλείων oapi-codegen και openapi-typescript-codegen
- Lint: Ένα linting task το οποίο επιβεβαιώνει την ποιότητα του Go codebase χρησιμοποιώντας το εργαλείο golangci-lint [40] και ορισμένων επιμέρους linters.
- Build: Το build command διαγράφει τα build directories και στη συνέχεια προβαίνει στο building και την εξαγωγή των build artifacts στα προκαθορισμένα directories.

Για τον ορισμό των commands και της δομής του CLI έγινε χρήση της βιβλιοθήκης cobra [41].

5.1. Client

Το Myst Client τρέχει natively στα Windows χωρίς κάποιο runtime dependency εκτός από την ύπαρξη προ-εγκατεστημένου web browser. Έγινε η επιλογή να μην γίνει bundle κάποιου web browser και embedding του εντός του binary ώστε, καταρχάς, να περιοριστεί το μέγεθος του ίδιου του binary, και δευτερευόντως να μπορεί ο χρήστης να χρησιμοποιήσει το web browser της επιλογής του. Επιπρόσθετα, το Myst με τον τρόπο που έχει δομηθεί, καθιστά δυνατή τη χρήση ενός native γραφικού περιβάλλοντος, εφόσον πραγματοποιηθούν πρώτα κάποιες αναγκαίες μετατροπές στο build process, όπως για παράδειγμα το bundling κάποιων περαιτέρω dependencies.

Κατά τη διαδικασία του building, αφότου διαγραφούν τα output directories, γίνεται πρώτα build του Svelte UI με output το “static” directory και, χρησιμοποιώντας το go:embed directive [42], γίνεται injection του UI payload μέσα στο client binary κατά την κλήση του go build.

Αποτέλεσμα είναι ένα μοναδικό binary της μορφής “client-os-arch(.exe)”, το οποίο μπορεί να εκτελεστεί για να ξεκινήσει ο Myst Client.

5.2. Server

Κατά το building του Myst Server εξάγεται με παρόμοιο τρόπο ένα Go binary της μορφής “server-os-arch”, το οποίο στη συνέχεια γίνεται bundle σε ένα OSI container image με τη βοήθεια του docker, χρησιμοποιώντας το αντίστοιχο Dockerfile. Το image αυτό μπορεί ακολούθως να γίνει deploy σε κάποιο cloud provider ή on-premise, έχοντας σαν runtime dependency πρόσβαση σε ένα MongoDB instance για persistency.

5.3. Deployment

Για τους σκοπούς της πτυχιακής και το πιο ρεαλιστικό manual end-to-end testing με πραγματικούς χρήστες, δημιουργήθηκε ένα deployment περιβάλλον του Myst Server στην πλατφόρμα DigitalOcean Apps [43]. Κατά το merging ενός pull request με το main branch εντός του repository του κώδικα της εφαρμογής, εκτελείται ένα workflow το οποίο δημιουργεί τα binaries του client και του server, αφενός για την επιβεβαίωση της εγκυρότητας του κώδικα, και αφετέρου για το deployment του server στο cloud. Σε αυτό το workflow δημιουργείται το OSI container image του server και προωθείται στο DigitalOcean Container Registry, και αμέσως μετά δίνεται η εντολή στη δημιουργία ενός αιτήματος re-deployment προς το DigitalOcean Apps Platform. Αυτό το re-deployment γίνεται με αυτόματο τρόπο, δίχως downtime (zero-downtime deployment) και με αυτόματο failover στην περίπτωση που υπάρξει κάποιο πρόβλημα με το container. Για τον εντοπισμό προβλημάτων έχει οριστεί ένα HTTP health check endpoint, το οποίο εάν δεν επιστρέφει HTTP Status Code 200 (OK) σηματοδοτεί πως το container είναι unhealthy, οδηγώντας στην εκκίνηση του failover μηχανισμού της πλατφόρμας.

Επιπρόσθετα, έχει δημιουργηθεί ένα managed MongoDB database instance εντός του ίδιου λογαριασμού DigitalOcean, το οποίο χρησιμοποιείται για persistence των δεδομένων. Το instance αυτό είναι προσβάσιμο μόνο από συγκεκριμένα containers εντός του λογαριασμού DigitalOcean.

6. Συμπεράσματα

Στη συγκεκριμένη πτυχιακή εργασία έγινε η ανάπτυξη ενός διαχειριστή κωδικών ο οποίος προσφέρει άνεση στη χρήση, αλλά και δυνατότητες συγχρονισμού και διασύνδεσης με το υπολογιστικό νέφος χρησιμοποιώντας τεχνολογίες που δεν αφαιρούν από τους χρήστες τα δικαιώματα ιδιοκτησίας των δεδομένων τους. Ακόμη, έγινε η προσπάθεια της συνύπαρξης αυτών των ιδιοτήτων με όσο το δυνατόν μικρότερο αντίκτυπο στη χρήση, επιτρέποντας την ασφαλέστερη αποθήκευση και διαμοιρασμό των στοιχείων πρόσβασης με άλλους χρήστες, καθώς και τη δυνατότητα προσθήκης λειτουργιών στο μέλλον με κάποια σχετική ευκολία.

Σχετικά με τις τεχνολογίες κρυπτογράφησης, υλοποιήθηκε μια ασφαλής και επεκτάσιμη μορφή αποθήκευσης των προσωπικών δεδομένων των χρηστών, έτσι ώστε τα δεδομένα του εκάστοτε χρήστη να παραμένουν ασφαλή ενώ βρίσκονται αποθηκευμένα στον σκληρό δίσκο, κατά την εκτέλεση της εφαρμογής, αλλά και κατά τον συγχρονισμό με άλλους χρήστες.

Επιπρόσθετα, η εφαρμογή αποτελεί μια ολοκληρωμένη υλοποίηση μιας desktop εφαρμογής η οποία χρησιμοποιεί τεχνολογίες όπως η γλώσσα προγραμματισμού Go και το Svelte framework, με ξεκάθαρη δομή και ευκολία στην μεταγλώττιση και εκτέλεση της πραγματικής εφαρμογής αλλά και των tests που επικυρώνουν τη σωστή λειτουργία της.

6.1. Μελλοντικές βελτιώσεις

Μία σημαντική παράλειψη στην υλοποίηση της εφαρμογής διαχείρισης κωδικών είναι η έλλειψη λειτουργιών που κάνουν τη διαχείριση των κωδικών μια πιο ευχάριστη διαδικασία. Κάποια από αυτά θα μπορούσαν να είναι ένα πεδίο αναζήτησης κωδικών με φίλτρα, λειτουργία ταξινόμησης και ομαδοποίησης των κωδικών εντός του ίδιου keystore, ένα password generator και επίσης αποθήκευση κωδικών μίας χρήσης (One-Time passwords - OTP).

Ακόμη, ο τρόπος που υλοποιήθηκε το backend της εφαρμογής εξυπηρετητή ίσως δεν είναι βέλτιστος. Για την ανάκτηση και επεξεργασία όλων των δομών χρησιμοποιείται το repository pattern, ενώ ίσως θα ήταν επιθυμητό να χρησιμοποιούνται aggregates με strong consistency boundaries, έτσι ώστε να βελτιωθεί η απόδοση της εφαρμογής και να μην υπάρχουν περιθώρια να υπάρχει περίπτωση να χαθεί η συνοχή των δεδομένων. Επίσης, οι έλεγχοι εξουσιοδότησης στο backend του server γίνονται εντός του application, και ίσως θα έπρεπε να υπάρχει άλλο ένα layer εξωτερικά του application που να πραγματοποιεί τους ελέγχους αυτούς.

Επιπλέον, αρκετές από τις λειτουργίες του client σε σχέση με τον συγχρονισμό των keystores πραγματοποιούνται λαμβάνοντας υπ όψιν το state του backend server, το οποίο καθιστά τον client ανοιχτό σε επιθέσεις. Θα ήταν ίσως θεμιτό να αποθηκεύεται οποιαδήποτε αλλαγή στο state σε σχέση με τον server και στον client, ώστε η πηγή της αλήθειας (source of truth) να βρίσκεται στον client. Μία τέτοια υλοποίηση θα μπορούσε να υποστηρίξει peer-to-peer ανταλλαγή των κωδικών, για παράδειγμα εντός του ίδιου τοπικού δικτύου, χωρίς να χρειάζεται απαραίτητα η διαμεσολάβηση με τον remote server.

Τέλος, η τωρινή υλοποίηση του συγχρονισμού δεν επιτρέπει τον συγχρονισμό των κωδικών ενός χρήστη με άλλες συσκευές του ίδιου. Οι τροποποιήσεις για να υποστηρίξουν μία τέτοια λειτουργία ηθελημένα παραλείφθηκαν για την πιο εύκολη υλοποίηση του Minimum Viable Product (MVP), αλλά θα μπορούσαν να προστεθούν σαν λειτουργία στο μέλλον.

Τέτοιες βελτιώσεις, αν και επιθυμητές, δεν αποτελούν κομμάτι της τωρινής υλοποίησης, μιας και η ανάπτυξη της εφαρμογής έγινε κυρίως για την μελέτη και υλοποίηση των βασικών πρωτοκόλλων συγχρονισμού, καθώς και τη δημιουργία μιας πλήρους βάσης κώδικα η οποία θα μπορεί εύκολα να επεκταθεί. Έτσι, έχοντας ένα αρχικό MVP, θα μπορεί να γίνει συλλογή επικοδομητικών σχολίων από τους χρήστες που θα συμμετέχουν σε κάποια διαδικασία BETA testing, με σκοπό την καλύτερη προτεραιοποίηση των λειτουργιών αυτών.

7. Βιβλιογραφία

- [1] LastPass (2022), *LastPass Recognized Again by G2 as a Leading Password Manager*,
<https://www.lastpass.com/company/newsroom/press-release/lastpass-recognized-again-by-g2-as-leading-password-manager>
- [2] Alina Dubinina (2019), *Factors Affecting Password Manager Adoption among European University Students*,
https://www.utupub.fi/bitstream/handle/10024/148428/Thesis_Alina_Dubinina.pdf
- [3] Anthony Luvanda (2014), *Identifying Threats Associated With Man-In-The-Middle Attacks during Communication between a Mobile Device and the Back End Server in Mobile Banking Applications*,
<https://www.jkuat.ac.ke/campuses/kigali/wp-content/uploads/2014/04/publication-Identifying-threats-associated-with-MANA.pdf>
- [4] James Simmons, Oumar Diallo, Sean Oesch, Scott Ruoti (2021), *Systematization of Password Manager Use Cases and Design Paradigms*,
<https://dl.acm.org/doi/pdf/10.1145/3485832.3485889>
- [5] Sascha Fahl, Marian Harbach, Marten Oltrogge, Thomas Muders, and Matthew Smith (2013), *Hey, You, Get Off of My Clipboard: On How Usability Trumps Security in Android Password Managers*,
https://link.springer.com/chapter/10.1007/978-3-642-39884-1_12
- [6] Ambarish Karole, Nitesh Saxena, and Nicolas Christin (2010), *A Comparative Usability Evaluation of Traditional Password Managers*,
https://link.springer.com/chapter/10.1007/978-3-642-24209-0_16
- [7] Joseph A. Tomain (2014), *Online Privacy and the First Amendment: An Opt-In Approach to Data Processing*,
<https://www.repository.law.indiana.edu/cgi/viewcontent.cgi?article=3649&context=facpub>
- [8] Sonia Chiasson, Robert Biddle (2007), *Issues in User Authentication*,
https://hotsoft.carleton.ca/~sonia/content/Chiasson_CHI2007Workshop_Issues_in_User_Authentication.pdf
- [9] LastPass (2023), *Security Incident Update and Recommended Actions*,
<https://blog.lastpass.com/2023/03/security-incident-update-recommended-actions/>

- [10] Cam Pedersen, David Dahl (2014), *Crypton: Zero-Knowledge Application Framework*, <https://campedersen.com/crypton.pdf>
- [11] Charles H. Sobey (2006), *Drive-independent data recovery: The current state-of-the-art*, https://www.researchgate.net/profile/Charles-Sobey/publication/3111743_Drive-independent_data_recovery_The_current_state-of-the-art/links/544783340cf2d62c30507ccd/Drive-independent-data-recovery-The-current-state-of-the-art.pdf
- [12] Sumit Narayan (2010), *Extendable Storage Framework for Reliable Clustered Storage Systems*, https://snsi.engr.uconn.edu/wp-content/uploads/sites/2477/2019/09/Extendable_storage_framework_f.pdf
- [13] go.dev (2023), *Build simple, secure, scalable systems with Go*, <https://go.dev/>
- [14] The README Project, GitHub (2022), *The top programming languages*, <https://octoverse.github.com/2022/top-programming-languages>
- [15] go.dev (2023), *The Go Programming Language Specification*, <https://go.dev/ref/spec>
- [16] Rob Pike (2012), *Go at Google: Language Design in the Service of Software Engineering*, <https://go.dev/talks/2012/splash.article>
- [17] Moumita Asad, Rafed Muhammad Yasir, Shihab Shahriar, Nadia Nahar, Md. Nurul Ahad Tawhid (2021), *Analyzing Program Comprehensibility of Go Projects*, https://www.researchgate.net/profile/Md-Nurul-Ahad-Tawhid/publication/354616857_Analyzing_Program_Comprehensibility_of_Go_Projects/links/6142a9f27d081355cce0ae2/Analyzing-Program-Comprehensibility-of-Go-Projects.pdf
- [18] gin-gonic.com (2023), *Gin Web Framework*. <https://gin-gonic.com/>
- [19] github/gin-gonic (2023), *Collection of middlewares created by the community*, <https://github.com/gin-gonic/contrib>
- [20] gin-gonic.com (2023), *Serving static files*, <https://gin-gonic.com/docs/examples/serving-static-files/>
- [21] docs.npmjs.com (2023), *About npm*, <https://docs.npmjs.com/about-npm>
- [22] webpack.js.org (2023), *webpack is a static module bundler for modern JavaScript applications*, <https://webpack.js.org/concepts/>
- [23] vitejs.dev (2023), *Why Vite*, <https://vitejs.dev/guide/why.html>
- [24] typescriptlang.org (2023), *TypeScript is JavaScript with syntax for types*, <https://www.typescriptlang.org/>

- [25] typescriptlang.org (2023), *Object Types*,
<https://www.typescriptlang.org/docs/handbook/2/objects.html>
- [26] typescriptlang.org (2023), *Generics*,
<https://www.typescriptlang.org/docs/handbook/2/generics.html>
- [27] Bc. Tat Dat Duong (2023), *Math expression evaluator for literal types in TypeScript*,
<https://dspace.cvut.cz/bitstream/handle/10467/108835/F8-DP-2023-Duong-Tat%20D-at-thesis.pdf>
- [28] svelte.dev (2023), *Svelte: cybernetically enhanced web apps*, <https://svelte.dev/>
- [29] svelte.dev (2018), *Virtual DOM is pure overhead*,
<https://svelte.dev/blog/virtual-dom-is-pure-overhead>
- [30] insights.stackoverflow.com (2021), *Web frameworks: Loved vs. Dreaded*,
<https://insights.stackoverflow.com/survey/2021#section-most-loved-dreaded-and-wanted-web-frameworks>
- [31] sass-lang.com (2023), *CSS with superpowers*, <https://sass-lang.com/>
- [32] svelte.dev (2023), *Svelte components: style component*,
<https://svelte.dev/docs/svelte-components#style>
- [33] mongodb.com (2023), *The developer data platform*, <https://www.mongodb.com/>
- [34] Anjali Chauhan (2019), *A Review on Various Aspects of MongoDB Databases*,
<https://www.ijert.org/research/a-review-on-various-aspects-of-mongodb-databases-IJERTV8IS050031.pdf>
- [35] Daniel Stenger (2023), *GO Design Patterns: An Introduction to SOLID*,
<https://hackernoon.com/go-design-patterns-an-introduction-to-solid>
- [36] Herberto Graça (2017), *DDD, Hexagonal, Onion, Clean, CQRS, ... How I put it all together*,
<https://herbertograca.com/2017/11/16/explicit-architecture-01-ddd-hexagonal-onion-clean-cqrs-how-i-put-it-all-together/>
- [37] lostechies.com (2008), *Entities, Value Objects, Aggregates and Roots*,
<https://lostechies.com/jimmybogard/2008/05/21/entities-value-objects-aggregates-and-roots/>
- [38] Brian Mulloy, *API Facade Pattern*,
<https://cloud.google.com/files/apigee/apigee-api-facade-pattern-ebook.pdf>

- [39] Dirk Loss, Tobias Limmer, Alexander von Gernler (2009), *The drunken bishop: An analysis of the OpenSSH fingerprint visualization algorithm*, <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=c01fab30d9022a39db4c8c7afe4fa592c00f4f4e>
- [40] golangci-lint.run (2023), *GolangCI-Lint: Introduction*, <https://golangci-lint.run/>
- [41] github/spf13 (2023), *A Commander for modern Go CLI interactions*, <https://github.com/spf13/cobra>
- [42] pkg.go.dev (2023), *Package embed provides access to files embedded in the running Go program*, <https://pkg.go.dev/embed>
- [43] digitalocean.com (2023), *Fully-managed infrastructure*, <https://www.digitalocean.com/products/app-platform>