Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Πληροφορική»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Υλοποίηση μηχανισμού εφαρμογής του Γ.Κ.Π.Δ. σε συνδυασμό με την οδηγία P.S.D.2 για χρηματοπιστωτικά ιδρύματα εντός της Ευρωπαϊκής ένωσης.** **Implementing G.D.P.R. and P.S.D.2 regulations for financial institutions under European Union.**
Ονοματεπώνυμο Φοιτητή	**Σαρίκος Γεώργιος**
Πατρώνυμο	**Μιχαήλ**
Αριθμός Μητρώου	**ΜΠΠΛ/17044**
Επιβλέπων	**Κωνσταντίνος Πατσάκης, Αναπληρωτής Καθηγητής**

Ημερομηνία Παράδοσης **Δεκέμβριος 2022**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Κωνσταντίνος Πατσάκης, Αναπληρωτής Καθηγητής	Αλέπης Ευθύμιος, Αναπληρωτής Καθηγητής	Σακκόπουλος Ευάγγελος, Αναπληρωτής Καθηγητής

Περιεχόμενα

Περίληψη

Από το Μάιο του 2018 εφαρμόζεται καθολικά ο Γενικός Κανονισμός για την Προστασία των Δεδομένων (General Data Protection Regulation GDPR) σε όλα τα κράτη μέλη της Ευρωπαϊκής ένωσης. Με αυτόν το κανονισμό οι όλες οι επιχειρήσεις που διαχειρίζονται δεδομένα προσωπικού χαρακτήρα είναι υποχρεωμένες να διασφαλίζουν τη προστασία αυτών, τη συγκατάθεση συλλογής και επεξεργασίας τους και την επιλογή διαγραφής τους από τα πρόσωπα που αφορά.

Περίπου την ίδια περίοδο εφαρμογής του ΓΚΠΔ, εφαρμόζεται σταδιακά και η οδηγία από όλα τα χρηματοπιστωτικά ιδρύματα της Ευρωπαϊκής ένωσης για όλα τα κράτη μέλη της, η PSD2. Η συγκεκριμένη οδηγία αποσκοπεί στην ενίσχυση της ασφάλειας των ηλεκτρονικών συναλλαγών πάσης φύσεως και στην εισαγωγή των Παρόχων Υπηρεσιών Πληρωμών οι οποίοι αναλαμβάνουν να διεκπεραιώνουν τις ηλεκτρονικές συναλλαγές πολιτών κατά ανάγκη αυτών σε συνεργασία με τις εκάστοτε τράπεζες που έχουν τους λογαριασμούς με τα χρήματα. Καλείται λοιπόν πλέον η κάθε τράπεζα να μοιράζεται με τους παρόχους αυτούς κατά απαίτηση των πελατών συγκεκριμένες πληροφορίες με τα στοιχεία τους.

Έτσι λοιπόν χρειάζονται να δημιουργηθούν νέες υλοποιήσεις στα συστήματα των τραπεζών και των παρόχων ώστε να διασφαλίζεται η ισχυρή ασφάλεια και η ισχυρή ταυτοποίηση των ατόμων που επιλέγουν να κάνουν ηλεκτρονικές συναλλαγές με αυτό το τρόπο.

Στη παρούσα εργασία θα αναφέρουμε κάποια σημεία του Γενικού Κανονισμού για την Προστασία των Δεδομένων (GDPR) τα οποία αποτελούν και θέμα προβληματισμού για την πλήρη εφαρμογή του νόμου σε ένα χρηματοπιστωτικό ίδρυμα. Στη συνέχεια επίσης θα επεκταθούμε σε κάποια σημεία της Payment Services Directive (PSD2) οδηγίας τα οποία επίσης αποτελούν θέμα προβληματισμού κατά την εφαρμογή της για τον ίδιο σκοπό. Τέλος περιγράφεται η υλοποίηση ενός συστήματος στα πλαίσια της παρούσας εργασίας που καλείται να εφαρμόσει κάποια σημεία του GDPR και της PSD2 και για τις ανάγκες του οποίου δημιουργήθηκαν δύο διεπαφές προγραμματισμού εφαρμογών ανεξάρτητες μεταξύ τους και ένα γραφικό επίσης ανεξάρτητο με τα δύο, περιβάλλον στο οποίο ο χρήστης εκτελεί τις συναλλαγές του. Οι διεπαφές εκ των οποίων η μία αφορά για παράδειγμα μια τράπεζα μη έχοντας καθόλου στοιχεία προσωπικού χαρακτήρα και η άλλη αφορά το πάροχο υπηρεσιών πληρωμών (για τους σκοπούς της εργασίας ας υποθέσουμε ότι είναι η Google) που έχει μόνο στοιχεία προσωπικού χαρακτήρα. Ο χρήστης με ένα απλό γραφικό περιβάλλον μπορεί να δημιουργήσει λογαριασμό στη τράπεζα χρησιμοποιώντας το λογαριασμό που έχει ήδη με τα προσωπικά του στοιχεία στη Google δίχως αυτά να χρειάζεται να τα κρατήσει η τράπεζα και να τα μοιραστεί μαζί της. Εν συνεχεία πραγματοποιεί συναλλαγές ηλεκτρονικές κατά τις οποίες για να ταυτοποιήσει η τράπεζα το χρήστη, επικοινωνεί με το πάροχο και εξουσιοδοτείται έτσι η πληρωμή. Στη συγκεκριμένη υλοποίηση χρησιμοποιείται αυστηρή εξακρίβωση ταυτότητας χρήστη (strong authentication) καθώς και διπλή ταυτοποίηση (two factor authentication). Ακόμα έχουν χρησιμοποιηθεί και όλα τα τελευταία πρότυπα κρυπτογράφησης και ασφάλειας των επικοινωνιών όπως η κρυπτογράφηση των προσωπικών στοιχείων εξακρίβωσης ταυτότητας δεδομένων με τη χρήση Argon2 hash generator.

Abstract

From May 2018 all organizations that have and manipulate personal data info of any kind inside the European Union are obliged to implement the General Data Protection Regulation GDPR. With this regulation institutions and organizations that have in their possession any personal data must ensure the protection against any leaks and give the right to the persons to delete them at their disposal.

At the same period of this implementation, another regulation is taking effect and that is PSD2. This is also a European regulation for electronic payment services. It seeks to make payments more secure in Europe, boost innovation and help banking services adapt to new technologies and up to date services. PSD2 is evidence of the increasing importance Application Program Interfaces (APIs) are acquiring in different financial sectors and all organizations must comply to that regulation.

In this master's thesis we will analyze some of the most important problems impacting the day to day business for all financial institutions implementing GDPR regulations and which are under serious consideration and are not implemented at all. As for the PSD2, we will also analyze key points and problems that occur for financial institutions implementation. In the end we will introduce and analyze this master's thesis project that will cover most of the problems that are analyzed before and is about having three independent systems that communicate to each other with APIs. One of them is the graphical user interface that gives the right to the end user to execute the business transaction. Another is the system that have the personal data and is supposed to simulate something like Google which holds all kind of personal information. The last is a simplified simulation of a financial organization like a bank that unlike what all banks are implementing at this point, do not hold any kind of personal information and is relying on the second system to authenticate the end user. All of these systems are implementing a full secured and encrypted environments. For these purposes we used a two factor authentication, strong authentication and Argon2 encryption for any sensitive data that are stored in the database.

1 Εισαγωγή – Σύντομη περιγραφή αντικειμένου

1.1 GDPR

Από το Μάιο του 2018 σταδιακά εφαρμόζεται από όλες τις επιχειρήσεις πάσης φύσεως ο ευρωπαϊκός κανονισμός GDPR που έχουν στη κατοχή τους προσωπικά δεδομένα και όσες δε συμμορφώνονται με αυτό το κανονισμό, μπορούν να τους επιβληθούν υπέρογκα ποσά προστίμων.

Ο κανονισμός αυτός όπως αναφέρεται συνοπτικά και στο άρθρο 1:

- Θεσπίζει κανόνες που αφορούν την προστασία των φυσικών προσώπων έναντι της επεξεργασίας δεδομένων προσωπικού χαρακτήρα και κανόνες που αφορούν την ελεύθερη κυκλοφορία των δεδομένων προσωπικού χαρακτήρα.
- Προστατεύει θεμελιώδη δικαιώματα και ελευθερίες των φυσικών προσώπων και ειδικότερα το δικαίωμά τους στην προστασία των δεδομένων προσωπικού χαρακτήρα.
- Επιτρέπει την ελεύθερη κυκλοφορία των δεδομένων προσωπικού χαρακτήρα εντός της Ένωσης όταν δεν περιορίζεται ούτε απαγορεύεται για λόγους που σχετίζονται με την προστασία των φυσικών προσώπων έναντι της επεξεργασίας δεδομένων προσωπικού χαρακτήρα.

Οι βασικές αρχές που διέπουν την επεξεργασία των προσωπικών δεδομένων όπως αναφέρονται και στο άρθρο 5 :

- υποβάλλονται σε σύννομη και θεμιτή επεξεργασία με διαφανή τρόπο σε σχέση με το υποκείμενο των δεδομένων («νομιμότητα, αντικειμενικότητα και διαφάνεια»)
- συλλέγονται για καθορισμένους, ρητούς και νόμιμους σκοπούς και δεν υποβάλλονται σε περαιτέρω επεξεργασία κατά τρόπο ασύμβατο προς τους σκοπούς αυτούς η περαιτέρω επεξεργασία για σκοπούς αρχειοθέτησης προς το δημόσιο συμφέρον ή σκοπούς επιστημονικής ή ιστορικής έρευνας ή στατιστικούς σκοπούς δεν θεωρείται ασύμβατη με τους αρχικούς σκοπούς σύμφωνα με το άρθρο 89 παράγραφος 1 («περιορισμός του σκοπού»)
- είναι κατάλληλα, συναφή και περιορίζονται στο αναγκαίο για τους σκοπούς για τους οποίους υποβάλλονται σε επεξεργασία («ελαχιστοποίηση των δεδομένων»)
- είναι ακριβή και όταν είναι αναγκαίο, επικαιροποιούνται πρέπει να λαμβάνονται όλα τα εύλογα μέτρα για την άμεση διαγραφή ή διόρθωση δεδομένων προσωπικού χαρακτήρα τα οποία είναι ανακριβή, σε σχέση με τους σκοπούς της επεξεργασίας («ακρίβεια»)
- διατηρούνται υπό μορφή που επιτρέπει την ταυτοποίηση των υποκειμένων των δεδομένων μόνο για το διάστημα που απαιτείται για τους σκοπούς της επεξεργασίας των δεδομένων προσωπικού χαρακτήρα τα δεδομένα προσωπικού χαρακτήρα μπορούν να αποθηκεύονται για μεγαλύτερα διαστήματα, εφόσον τα δεδομένα προσωπικού χαρακτήρα θα υποβάλλονται σε επεξεργασία μόνο για σκοπούς αρχειοθέτησης προς το δημόσιο συμφέρον, για σκοπούς επιστημονικής ή ιστορικής έρευνας ή για στατιστικούς σκοπούς, σύμφωνα με το άρθρο 89 παράγραφος 1 και εφόσον εφαρμόζονται τα

κατάλληλα τεχνικά και οργανωτικά μέτρα που απαιτεί ο παρών κανονισμός για τη διασφάλιση των δικαιωμάτων και ελευθεριών του υποκειμένου των δεδομένων («περιορισμός της περιόδου αποθήκευσης»)

- υποβάλλονται σε επεξεργασία κατά τρόπο που εγγυάται την ενδεδειγμένη ασφάλεια των δεδομένων προσωπικού χαρακτήρα, μεταξύ άλλων την προστασία τους από μη εξουσιοδοτημένη ή παράνομη επεξεργασία και τυχαία απώλεια, καταστροφή ή φθορά, με τη χρησιμοποίηση κατάλληλων τεχνικών ή οργανωτικών μέτρων («ακεραιότητα και εμπιστευτικότητα»).

1.1.1 Σημεία εφαρμογής GDPR

Στη παρούσα εργασία έχουμε εφαρμόσει τη πληθώρα των κανονιστικών πλαισίων που διέπει ο νόμος.

Αναλυτικότερα το άτομο που χρησιμοποιεί την εφαρμογή επιβάλλεται να δώσει έγγραφη συγκατάθεση αποθήκευσης και επεξεργασίας των προσωπικών του δεδομένων κατά τη δημιουργία του λογαριασμού του.

Διασφαλίζεται η εγκυρότητα και η σαφήνεια των δεδομένων του καθώς σε οποιαδήποτε στιγμή δύναται να τα παραμετροποιήσει.

Εφαρμόζονται όλα τα τελευταία πρότυπα της ασφάλειας αποθήκευσης της πληροφορίας καθώς όπως θα αναλύσουμε πιο κάτω χρησιμοποιούμε το .NET framework το οποίο φροντίζει με τα πιο σύγχρονα πρότυπα ασφάλειας να προστατεύει τα δεδομένα που αποθηκεύονται στη βάση των δεδομένων όπως επιθέσεις που χαρακτηρίζονται SQL Injections. Ταυτόχρονα με τη τεχνική που έχει χρησιμοποιηθεί για την αποθήκευση του κωδικού χρήστη password hashing and password salting είναι αδύνατο να υποκλαπεί ο κωδικός του χρήστη. Ακόμα και να υποκλαπεί ο επιτιθέμενος δεν έχει πρόσβαση σε όλα τα στοιχεία του χρήστη καθώς έχουν αποθηκευτεί σε δύο ξεχωριστά συστήματα, ένα για τα προσωπικά δεδομένα και ένα για τα χρηματοπιστωτικά, σε συνδυασμό πως αυτά τα συστήματα με τις πιο ευαίσθητες και σημαντικότερες πληροφορίες δεν δέχονται κλήσεις https από οποιαδήποτε τοποθεσία, το σύστημα θεωρείται από τα πιο ασφαλή.

Εφαρμόζεται η πολιτική της ελαχιστοποίησης της πληροφορίας κατά την μεταφορά της από το ένα σύστημα στο άλλο, δηλαδή ακόμα και η πληροφορία που είναι κρυπτογραφημένη να διαρρεύσει και με κάποιο τρόπο να αποκρυπτογραφηθεί, δεν είναι αξιοποιήσιμη καθώς μεταφέρονται μόνο τα τμήματα της πληροφορίας με το ελάχιστο όγκο δεδομένων που χρειάζονται για τη επικοινωνία των συστημάτων. Αυτό με ένα παράδειγμα για επεξήγηση σημαίνει πως αν στη οθόνη χρειάζεται να εμφανιστεί η καρτέλα του χρήστη με όλα τα στοιχεία του (προσωπικά ή και χρηματοπιστωτικά), τότε επικοινωνεί με το πρώτο σύστημα για την αυθεντικοποίηση του χρήστη, μεταφέρεται μια φορά ο κωδικός χρήστη και ο κωδικός πρόσβασης στο σύστημα και έπειτα από επεξεργασία και ταυτοποίηση αποδίδεται ένα JWT TOKEN (θα επεξηγηθεί παρακάτω αναλυτικότερα) με το οποίο εφεξής γίνεται και η επικοινωνία μεταξύ των συστημάτων έως ότου λήξει σε ένα σύντομο χρονικό διάστημα. Στο JSON που μεταφέρεται η πληροφορία η ζητούμενη, κάθε φορά χρησιμοποιείται για τη ταυτοποίηση/αυθεντικοποίηση το TOKEN (έτσι δεν μεταφέρεται συνεχώς η πληροφορία

κωδικού και όνομα χρήστη) και έπειτα τα ελάχιστα δεδομένα αναλόγως τη φύση της οθόνης που ζητήθηκαν (δεν μεταφέρονται όλα τα στοιχεία προσωπικής και χρηματοπιστωτικής φύσης ταυτόχρονα και στη πληρότητα τους).

1.1.2 Προβληματισμοί

Με αυτές τις τεχνικές υλοποίησης εφαρμόζεται ο νόμος στη πλειονότητα του για τους σκοπούς της παρούσης εργασίας. Το σημείο που αξίζει να αναφέρουμε προς προβληματισμό είναι το δικαίωμα της ανάκλησης/διαγραφής των δεδομένων. Είναι λογικό να μη δύναται η ολική διαγραφή των δεδομένων που αφορούν χρηματοπιστωτικά στοιχεία για λόγους διαφάνειας των συναλλαγών. Όμως όπως περιγράφεται και στο άρθρο 89 παράγραφος 1, ο νόμος αφήνει ένα περιθώριο διατήρησης αποθήκευσης κάποιων δεδομένων **για κάποιο χρονικό διάστημα** και με την ανάλογη τεκμηρίωση. Έτσι και στη παρούσα εφαρμογή ο χρήστης μπορεί να κλείσει το λογαριασμό του και να διαγράψει τα στοιχεία του, όμως μόνο αυτά που αποτελούν καθαρά προσωπικής φύσεως δεδομένα και όχι οτιδήποτε αφορά ταυτοποίηση χρηματοπιστωτικών συναλλαγών.

1.2 PSD2

Η πρώτη οδηγία για τις Υπηρεσίες Πληρωμών (PSD1 οδηγία 2007/64/ΕΕ) που τέθηκε σε ισχύ 2009 και προσαρμόστηκε στην ελληνική νομοθεσία με το Ν.3862/2010 αποτελεί τη νομική βάση τής δημιουργίας μιας ενιαίας αγοράς υπηρεσιών πληρωμών εντός των κρατών μελών τής Ευρωπαϊκής Ένωσης.

Η δεύτερη οδηγία για τις Υπηρεσίες Πληρωμών (PSD2 οδηγία 2015/2366) που τέθηκε σε εφαρμογή το 2018 και προσαρμόστηκε στην ελληνική νομοθεσία με το Ν.4537/2018 αποτελεί επέκταση της PSD1. Πρόκειται για μια νέα αναθεωρημένη οδηγία, η οποία προκάλεσε έναν αξιοσημείωτο μετασχηματισμό για τον τραπεζικό τομέα.

Η PSD2 θέτει τους κανόνες για την εξασφάλιση της ασφάλειας των συναλλαγών και προστασίας των συναλλασσόμενων, ταυτόχρονα με την προστασία των χρηματοοικονομικών δεδομένων των χρηστών, θεσπίζοντας αυστηρές προδιαγραφές ασφαλείας αναφορικά με τις ηλεκτρονικές πληρωμές. Αποτελεί επέκταση της εμβέλειας της PSD1, συμπεριλαμβάνοντας τις περιπτώσεις συναλλαγών όπου τουλάχιστον ένα, και όχι πλέον και τα δύο συμβαλλόμενα μέρη βρίσκεται εντός των συνόρων της ΕΕ και επεκτείνει τον ορισμό του «Ιδρύματος Πληρωμών» σε νέους τύπους υπηρεσιών πληρωμών, αλλάζοντας έτσι τη μορφή του ανταγωνισμού στις συναλλαγές λιανικής τραπεζικής.

Για τους Παρόχους Υπηρεσιών Πληρωμών εισάγονται δύο υπηρεσίες, ο Πάροχος Υπηρεσιών Εκκίνησης Πληρωμών (PISP) και ο Πάροχος Υπηρεσιών Πληροφοριών Λογαριασμού (AISP). Για να είναι σε θέση οι πάροχοι αυτοί να παρέχουν τις υπηρεσίες τους πρέπει να αποκτήσουν πρόσβαση στις πληροφορίες των τραπεζικών λογαριασμών των πελατών τους. Συνεπώς οι τράπεζες είναι υποχρεωμένες να δώσουν τα επαρκή στοιχεία λογαριασμού στους παρόχους με τη συγκατάθεση των πελατών τους ώστε να μπορούν να ολοκληρωθούν οι συναλλαγές. Τεχνικά αυτή η απρόσκοπτη ανταλλαγή πληροφοριών γίνεται

μέσω των Application Programming Interface – API και απαιτούνται υψηλά πρότυπα ασφαλείας για την εξασφάλιση της προστασίας των δεδομένων των χρηστών καθώς και την ασφάλεια των συναλλαγών. Στην PSD2 τυποποιείται και ενισχύεται η προώθηση ηλεκτρονικών υπηρεσιών και συναλλαγών, για την ασφάλεια των οποίων τίθενται απαιτήσεις ισχυρής ταυτοποίησης πελάτη, ενώ απαιτούνται νέες τεχνολογίες και συστήματα για την επίτευξη πλήρους συμμόρφωσης (3D secure).

1.2.1 Σημεία εφαρμογής PSD2

Στη παρούσα εργασία έχουμε εφαρμόσει τη πληθώρα των κανονιστικών πλαισίων που διέπονται από την οδηγία PSD2. Πιο συγκεκριμένα μας απασχόλησαν προς υλοποίηση τα σημεία που αναφέρονται στα άρθρα 66,67 και 97. Στην εφαρμογή μας την Υπηρεσία εκκίνησης πληρωμών αποτελεί το BankAuth-SPA που είναι και το γραφικό περιβάλλον για τη πρόσβαση ενός χρήστη ενώ ο Πάροχος Υπηρεσιών Πληροφοριών Λογαριασμού αποτελεί το BankAuth.API. Τα κυριότερα και σημαντικότερα σημεία που υπάρχουν στα εν λόγω άρθρα αναφέρονται παρακάτω με τον τρόπο υλοποίησης τους.

Το BankAuth-SPA όπως αναφέρει το άρθρο 66 δεν διακρατά κεφάλαια του πληρωτή και δεν αποθηκεύει τα ευαίσθητα δεδομένα πληρωμής του χρήστη υπηρεσιών πληρωμών καθώς δεν αποθηκεύεται κανένα δεδομένο (δεν υφίσταται βάση δεδομένων).

Διασφαλίζει ότι τα εξατομικευμένα διαπιστευτήρια ασφαλείας του χρήστη υπηρεσιών πληρωμών, δεν είναι, με την εξαίρεση του χρήστη και του εκδότη των εξατομικευμένων διαπιστευτηρίων ασφαλείας, προσβάσιμα σε άλλα μέρη και ότι διαβιβάζονται από τον πάροχο υπηρεσιών εκκίνησης πληρωμών μέσω ασφαλών και αποτελεσματικών διαύλων. Αυτό επιτυγχάνεται όπως θα αναλυθεί τεχνικά και παρακάτω με την ταυτοποίηση του χρήστη καλώντας το BankAuth.API το οποίο του επιστρέφει μέσω JSON το TOKEN αυθεντικοποίησης μαζί με το id του χρήστη που ταυτίζεται με το id της τράπεζας. Η παραπάνω πληροφορία είναι πλήρως κρυπτογραφημένη και συμβαίνει μέσω https. Στη συνέχεια καλεί το BankData.API για να αντλήσει τα απαραίτητα χρηματοπιστωτικά στοιχεία με τα ίδια πρότυπα ασφαλείας μη διαρρέοντας σημαντικές πληροφορίες σε άλλα μέρη.

Διασφαλίζει ότι οποιεσδήποτε άλλες πληροφορίες για τον χρήστη των υπηρεσιών πληρωμών, που λαμβάνονται κατά την παροχή υπηρεσιών εκκίνησης πληρωμών, χορηγούνται μόνο στον δικαιούχο και μόνο με τη ρητή συγκατάθεση του χρήστη υπηρεσιών πληρωμών. Αυτό το επιτυγχάνει η εφαρμογή κατά τη δημιουργία του λογαριασμού χρήστη για το BankAuth.API που είναι υποχρεωμένος ο χρήστης να δώσει συγκατάθεση για πληροφορίες για τη πρόσβαση στο χρηματοπιστωτικό του λογαριασμό.

Κάθε φορά που διενεργείται έναρξη πληρωμής, ταυτοποιείται στον πάροχο υπηρεσιών πληρωμών εξυπηρέτησης λογαριασμού του πληρωτή και επικοινωνεί με τον πάροχο υπηρεσιών πληρωμών εξυπηρέτησης λογαριασμού, τον πληρωτή και τον δικαιούχο κατά τρόπο ασφαλή, με τη τεχνική που αναφέραμε παραπάνω.

Συνεπώς το BankAuth.API διασφαλίζει ότι τα εξατομικευμένα διαπιστευτήρια ασφαλείας του χρήστη υπηρεσιών πληρωμών, δεν είναι, με την εξαίρεση του χρήστη και του εκδότη των εξατομικευμένων διαπιστευτηρίων ασφαλείας, προσβάσιμα σε άλλα μέρη και ότι κατά τη

διαβίβασή τους γίνεται μέσω ασφαλών και αποτελεσματικών διαύλων (τεχνικές υλοποίησης που αναλύονται παρακάτω και αφορούν JWT, https, JSON).

Για κάθε κύκλο επικοινωνίας, ταυτοποιείται ο χρήστης στο BankAuth.API και επικοινωνεί ασφαλώς με τον πάροχο ή τους παρόχους υπηρεσιών πληρωμών εξυπηρέτησης λογαριασμού και τον χρήστη υπηρεσιών πληρωμών. Ορίζεται ως κύκλος επικοινωνίας ένα εύλογο σύντομο χρονικό διάστημα το οποίο χρειάζεται για να ολοκληρωθεί μια συναλλαγή κατά τη διάρκεια του οποίου το TOKEN έχει ισχύ. Μετά το πέρας αυτού του κύκλου το TOKEN λήγει και χρειάζεται εκ νέου αυθεντικοποίηση του χρήστη.

Όπως αναφέρεται στο άρθρο 97 ο πάροχος υπηρεσιών πληρωμών οφείλει να εφαρμόζει αυστηρή εξακρίβωση της ταυτότητας του πελάτη όταν ο πληρωτής έχει πρόσβαση στο λογαριασμό πληρωμών του διαδικτυακά, διενεργεί την έναρξη πράξης πληρωμής ηλεκτρονικά και εκτελεί οποιαδήποτε ενέργεια, μέσω εξ αποστάσεως διαύλου, η οποία μπορεί να ενέχει κίνδυνο απάτης στον τομέα των πληρωμών ή άλλων παραβιάσεων. Αυτήν την απαίτηση την καλύπτει η εφαρμογή μας καθώς χρησιμοποιείται το μοντέλο της διπλής ταυτοποίησης χρήστη. Για το σκοπό αυτό ο χρήστης κατά την εισαγωγή των διαπιστευτηρίων του, αποστέλλεται SMS με μοναδικό τυχαίο κωδικό που παράγεται από το σύστημα και του αποστέλλεται στο κινητό του τηλέφωνο που έχει δηλώσει κατά τη δημιουργία του λογαριασμού του. Ο κωδικός έχει διάρκεια λήξης το ένα λεπτό και μετά το πέρας αυτού δεν γίνεται αποδεκτός από το σύστημα, καθιστώντας υποχρεωτικό στο χρήστη να επαναλάβει τη διαδικασία εκ νέου.

1.2.2 Προβληματισμοί – Μελλοντικές υλοποιήσεις

Στη εργασία μας υλοποιείται το μεγαλύτερο μέρος της PSD2 καθιστώντας το σύστημα μας ένα ικανό μοντέλο για να ικανοποιήσει τις ηλεκτρονικές χρηματοπιστωτικές συναλλαγές ενός χρήστη με ασφάλεια και επάρκεια. Αξίζει να αναφέρουμε πως σε μελλοντικό χρόνο επέκταση θα μπορούσε να αποτελέσει η δημιουργία εφαρμογής για κινητές συσκευές αντικαθιστώντας έτσι λειτουργίες αυθεντικοποίησης που έχουμε ήδη υλοποιήσει. Ο λόγος για τον οποίο είναι σκόπιμη αυτή η επέκταση είναι πως θα μπορούσε μια κινητή συσκευή να ενσωματώσει ένα πρότυπο αυθεντικοποίησης FIDO.

Το FIDO είναι ένα εναλλακτικό πρότυπο αυθεντικοποίησης το οποίο χρησιμοποιεί κρυπτογραφία δημοσίου και ιδιωτικού κλειδιού σε συνδυασμό με την εφαρμογή της αυθεντικοποίησης μέσω των βιομετρικών χαρακτηριστικών του χρήστη. Σε μια τέτοια υλοποίηση ο χρήστης δεν χρειάζεται να εισάγει κωδικό χρήστη και κωδικό πρόσβασης. Μπορεί απλά να εισάγει για παράδειγμα το δακτυλικό του αποτύπωμα και έπειτα η εφαρμογή να δώσει την έγκριση της ταυτοποίησης για να συνεχιστεί η συναλλαγή. Αυτό συνεπάγεται πως δεν είναι αναγκαία και η αποστολή μοναδικού προσωρινού κωδικού μέσω SMS που εφαρμόζουμε στην υπάρχουσα εργασία.

Με το πρότυπο FIDO επίσης επιτυγχάνεται η ισχυρή ταυτοποίηση του χρήστη η οποία και είναι προϋπόθεση να πληρείται ώστε σε ένα ολοκληρωμένο σύστημα ηλεκτρονικών χρηματοπιστωτικών συναλλαγών να εφαρμόζεται η PSD2.

2 Λειτουργικές προδιαγραφές

2.1 Εισαγωγή

Στη παρούσα εργασία δημιουργήθηκαν τρία ανεξάρτητα συστήματα (stand alone applications) εκ των οποίων ένα είναι το γραφικό περιβάλλον, το δεύτερο είναι το σύστημα που έχει τη λειτουργία αποθήκευσης των προσωπικών δεδομένων και το τρίτο περιλαμβάνει όλα τα δεδομένα που αφορούν τις χρηματικές συναλλαγές όπως για παράδειγμα το διαθέσιμο υπόλοιπο ενός χρηματικού λογαριασμού.

2.2 Γραφικό περιβάλλον

Για λόγους χρηστικότητας δημιουργήθηκε ένα απλό γραφικό περιβάλλον πάνω στο οποίο ο χρήστης μπορεί να εκτελεί τις χρηματικές του συναλλαγές. Ο χρήστης έχει τη δυνατότητα να δημιουργεί λογαριασμό και έπειτα να χειρίζεται τις συναλλαγές του έχοντας ένα διαθέσιμο χρηματικό υπόλοιπο.

Για την υλοποίηση αυτή χρησιμοποιήθηκε το Angular Framework το οποίο αποτελεί ένα από τα τελευταίας τεχνολογίας γραφικά περιβάλλοντα. Είναι γρήγορο, εφαρμόζει όλα τα τελευταία πρότυπα ασφάλειας της πληροφορίας και είναι διαθέσιμο για υλοποίηση και εφαρμογή σε διάφορες πλατφόρμες. Διαθέτει γραφικό περιβάλλον που προβάλλεται τόσο από όλες τις κινητές συσκευές καθώς και από όλους τους υπολογιστές όσο και τις έξυπνες συσκευές.

Το project αυτό ονομάστηκε BankAuth-SPA και με την ονομασία αυτή θα αναφερόμαστε στο συγκεκριμένο πρόγραμμα.

2.3 Σύστημα υπεύθυνο για την αποθήκευση των προσωπικών δεδομένων

Η τεχνολογία που επιλέχθηκε για το συγκεκριμένο σύστημα προς ανάπτυξη είναι το .Net Core της Microsoft. Είναι ένα “cross platform framework” το οποίο σημαίνει πως σου δίνει τη δυνατότητα να το εγκαταστήσεις πάνω σε οποιοδήποτε λογισμικό διαθέτεις όπως Microsoft Windows, Linux καθώς και Mac OS. Χρησιμοποιεί τη γλώσσα προγραμματισμού C# η οποία ενσωματώνει όλες τις τελευταίες τεχνοτροπίες προγραμματισμού όπως ασύγχρονος προγραμματισμός καθώς και τη γλώσσα Linq για πιο ασφαλή και γρήγορη προς ανάπτυξη επικοινωνία με τη βάση που αποθηκεύονται τα δεδομένα.

Δημιουργήθηκε με αυτές τις τεχνολογίες ένα API (πρόγραμμα διεπαφής) το οποίο επικοινωνεί με το BankAuth-SPA όποτε δεχθεί κλήση μέσω https. Συνεπώς για να δέχεται κλήσεις και να απαντάει έχει στηθεί ανεξάρτητο (stand alone app) σε δικό του server και δέχεται κλήσεις σε συγκεκριμένη διεύθυνση που έχει προγραμματιστεί. Ενσωματώνει για την ασφάλεια αυτής της επικοινωνίας (transaction) Bearer Token Authenticaton και το επιστρέφει έπειτα από ισχυρή ταυτοποίηση μέσω two factor authentication (strong authentication) αφού

έχει θεωρηθεί η κλήση authorized. Εδώ βρίσκονται αποθηκευμένα όλα τα προσωπικής φύσεως δεδομένα ενός χρήστη προσομοιάζοντας έτσι ένα οργανισμό όπως η Google. Τα δεδομένα αυτά βρίσκονται σε ασφαλή τοποθεσία εσωτερικά του server και δέχεται κλήσεις μόνο από το συγκεκριμένο API σε τοπική τοποθεσία. Ενσωματώνεται εδώ η εφαρμογή του κανονισμού GDPR καθώς δίνεται η δυνατότητα στο χρήστη να τροποποιεί ή να διαγράφει τα δεδομένα του όποτε το ζητήσει. Να σημειωθεί σε αυτό το σημείο πως όλη η επικοινωνία μεταξύ των προγραμμάτων γίνεται με την ανταλλαγή μηνυμάτων σε μορφή JSON. Η ισχυρή ταυτοποίηση υλοποιήθηκε χρησιμοποιώντας πάροχο ανταλλαγής μηνυμάτων μέσω SMS για τον οποίο και έχει δημιουργηθεί αντίστοιχο service. Ο χρήστης μόλις ζητήσει να κάνει login (αφού πρώτα έχει προηγηθεί η δημιουργία του λογαριασμού του με τα προσωπικά δεδομένα) του έρχεται ένα SMS μέσω τρίτης υπηρεσίας το οποίο του δίνει ένα προσωρινό κωδικό ο οποίος και λήγει έπειτα από ένα λεπτό και παράγεται τυχαία κάθε φορά από το σύστημα. Η αποθήκευση του κωδικού πρόσβαση του χρήστη γίνεται με τη χρήση της βιβλιοθήκης Argon2 η οποία και αποθηκεύει το hash του password και όχι το ίδιο το password εμποδίζοντας έτσι οποιοδήποτε εκτός του ίδιου του χρήστη να μπορεί να γνωρίζει το κωδικό του.

Το project αυτό ονομάστηκε BankAuth.API και στο εξής θα αναφερόμαστε σε αυτό με τη συγκεκριμένη ονομασία.

2.4 Σύστημα που προσομοιάζει ένα χρηματοπιστωτικό ίδρυμα

Το σύστημα το τελευταίο που υλοποιήθηκε προσομοιάζει τις λειτουργίες μια τράπεζας καθώς ενσωματώνει λειτουργίες όπως λογαριασμό χρήστη που έχει χρηματικό υπόλοιπο και μπορεί να υλοποιεί χρηματικές ηλεκτρονικές συναλλαγές.

Αναπτύχθηκε και αυτό όπως και το προηγούμενο project ως stand alone app πάνω στις ίδιες τεχνολογίες που περιγράψαμε προηγουμένως και συγκεκριμένα σε .Net Core. Είναι και αυτό ένα API πρόγραμμα διεπαφής υπεύθυνο να δέχεται κλήσεις με την ίδια τεχνολογία όπως και το BankAuth.API με τη διαφορά ότι επιστρέφει δεδομένα μη προσωπικής φύσης αλλά χρηματικής.

Με αυτή την υλοποίηση εφαρμόζεται και η ευρωπαϊκή οδηγία PSD2 καθώς στη βάση του συγκεκριμένου συστήματος δεν αποθηκεύονται προσωπικά δεδομένα. Η ταυτοποίηση του χρήστη γίνεται μέσω ασφαλής κλήσης στο BankAuth.API το οποίο και επιστρέφει μαζί με το authorization token και το μοναδικό κλειδί ταυτοποίησης του χρήστη το οποίο και ταυτίζεται με τη βάση του συγκεκριμένου project. Με τη σειρά του το συγκεκριμένο project κάνει μια τελευταία κλήση στο BankAuth-SPA μέσω JSON το οποίο και μεταφέρει τη πληροφορία του χρηματικού λογαριασμού του χρήστη και είναι υπεύθυνο για την γραφική διεπαφή.

Το project αυτό ονομάστηκε BankData.API και στο εξής έτσι θα αναφερόμαστε σε αυτό.

3 Τεχνικές προδιαγραφές

3.1 Σύντομη περιγραφή των τεχνολογιών υλοποίησης

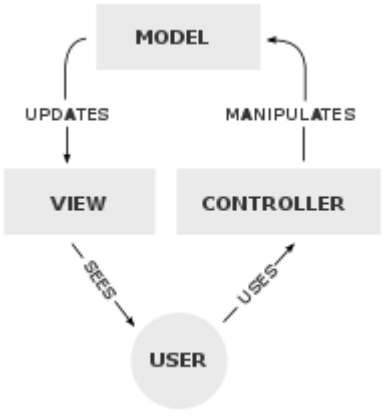
3.1.1 .Net Core 3.1

Το .NET Core είναι ένα σύνολο στοιχείων χρόνου εκτέλεσης, βιβλιοθήκης και μεταγλωττιστή. Μπορούν να χρησιμοποιηθούν σε διάφορες ρυθμίσεις παραμέτρων για φόρτους εργασίας συσκευής και cloud. Η γλώσσα προγραμματισμού πάνω στην οποία γράφεται η υλοποίηση είναι η C#. Υποστηρίζεται σε πολλές πλατφόρμες ανοιχτού κώδικα και παρέχει ένα μοντέλο ελαφριάς ανάπτυξης καθώς και την ευελιξία χρήσης διάφορων πλατφορμών λειτουργικών συστημάτων για εργαλεία ανάπτυξης. Το συγκεκριμένο framework είναι διαθέσιμο στο GitHub με την άδεια χρήσης του MIT. Τέλος αναφέρεται σε διάφορες τεχνολογίες συμπεριλαμβανομένων των .Net Core, ASP.Net Core και Entity Framework Core.

Το .NET Core ενσωματώνει το μοντέλο αρχιτεκτονικής Model-View-Controller το οποίο χρησιμοποιείται για τη δημιουργία περιβαλλόντων αλληλεπίδρασης χρήστη.

3.1.2 Model – View – Controller

Η συγκεκριμένη αρχιτεκτονική λογισμικού διαιρείται σε τρία διασυνδεδεμένα μέρη ώστε να διαχωριστεί η παρουσίαση της πληροφορίας στο τελικό χρήστη από τη μορφή που αποθηκεύεται στο σύστημα. Το κυρίως μέρος του μοντέλου - Model διαχειρίζεται την ανάκτηση και την αποθήκευση των δεδομένων στο σύστημα. Το αντικείμενο - View παρουσιάζει τη πληροφορία στο χρήστη συνήθως με γραφικό τρόπο και το τρίτο και τελευταίο μέρος ο - Controller είναι αυτός που δέχεται την είσοδο και στέλνει τις εντολές στο Model και στο View.



3.1.3 Argon2 – Password Hasher

Το Argon2 αποτελεί έναν από τους τελευταίους μηχανισμούς password hashing καθότι είναι πιο ασφαλής ενσωματώνοντας τα τελευταία πρότυπα κρυπτογράφησης της πληροφορίας και είναι ιδιαίτερα γρήγορος.

Σχεδιάστηκε από τους Alex Biryukov, Daniel Dinu, και Dmitry Khovratovich στο πανεπιστήμιο του Λουξεμβούργου. Διατίθεται κάτω από την άδεια χρήσης της Creative Commons CC0 ή Apache License 2.0. Μερικά από τα χαρακτηριστικά του είναι πως μεγιστοποιεί την δυνατότητα αντίστασης ενός συστήματος κάτω από επιθέσεις χρησιμοποιόντας GPU (GPU cracking attacks).

Ο αλγόριθμος που χρησιμοποιεί παρατίθεται παρακάτω.

```
Function Argon2
   Inputs:
      password (P):       Bytes (0..2^32-1)    Password (or message) to be hashed
      salt (S):           Bytes (8..2^32-1)    Salt (16 bytes recommended for password hashing)
      parallelism (p):    Number (1..2^24-1)   Degree of parallelism (i.e. number of threads)
      tagLength (T):      Number (4..2^32-1)   Desired number of returned bytes
      memorySizeKB (m):   Number (8p..2^32-1)  Amount of memory (in kibibytes) to use
      iterations (t):     Number (1..2^32-1)   Number of iterations to perform
      version (v):        Number (0x13)        The current version is 0x13 (19 decimal)
      key (K):            Bytes (0..2^32-1)    Optional key (Errata: PDF says 0..32 bytes, RFC says 0..2^32 bytes)
      associatedData (X): Bytes (0..2^32-1)    Optional arbitrary extra data
      hashType (y):       Number (0=Argon2d, 1=Argon2i, 2=Argon2id)
   Output:
      tag:                Bytes (tagLength)    The resulting generated bytes, tagLength bytes long

   Generate initial 64-byte block H_0.
    All the input parameters are concatenated and input as a source of additional entropy.
    Errata: RFC says H_0 is 64-bits; PDF says H_0 is 64-bytes.
    Errata: RFC says the Hash is H^, the PDF says it's 𝒽 (but doesn't document what 𝒽 is). It's actually Blake2b.
    Variable length items are prepended with their length as 32-bit little-endian integers.
   buffer ← parallelism ‖ tagLength ‖ memorySizeKB ‖ iterations ‖ version ‖ hashType
         ‖ Length(password)       ‖ Password
         ‖ Length(salt)           ‖ salt
         ‖ Length(key)            ‖ key
         ‖ Length(associatedData) ‖ associatedData
   H_0 ← Blake2b(buffer, 64) //default hash size of Blake2b is 64-bytes

   Calculate number of 1 KB blocks by rounding down memorySizeKB to the nearest multiple of 4*parallelism kibibytes
   blockCount ← Floor(memorySizeKB, 4*parallelism)

   Allocate two-dimensional array of 1 KiB blocks (parallelism rows x columnCount columns)
   columnCount ← blockCount / parallelism;   //In the RFC, columnCount is referred to as q
```

```
Compute the first and second block (i.e. column zero and one ) of each lane (i.e. row)
for i ← 0 to parallelism-1 do for each row
   B_i[0] ← Hash(H_0 ‖ 0 ‖ i, 1024) //Generate a 1024-byte digest
   B_i[1] ← Hash(H_0 ‖ 1 ‖ i, 1024) //Generate a 1024-byte digest

Compute remaining columns of each lane
for i ← 0 to parallelism-1 do //for each row
   for j ← 2 to columnCount-1 do //for each subsequent column
      //i' and j' indexes depend if it's Argon2i, Argon2d, or Argon2id (See section 3.4)
      i', j' ← GetBlockIndexes(i, j)  //the GetBlockIndexes function is not defined
      B_i[j] = G(B_i[j-1], B_i'[j']) //the G hash function is not defined

Further passes when iterations > 1
for nIteration ← 2 to iterations do
   for i ← 0 to parallelism-1 do for each row
     for j ← 0 to columnCount-1 do //for each subsequent column
        //i' and j' indexes depend if it's Argon2i, Argon2d, or Argon2id (See section 3.4)
        i', j' ← GetBlockIndexes(i, j)
        if j == 0 then
          B_i[0] = B_i[0] xor G(B_i[columnCount-1], B_i'[j'])
        else
          B_i[j] = B_i[j] xor G(B_i[j-1], B_i'[j'])

Compute final block C as the XOR of the last column of each row
C ← B_0[columnCount-1]
for i ← 1 to parallelism-1 do
   C ← C xor B_i[columnCount-1]

Compute output tag
return Hash(C, tagLength)
```

Παρατίθεται επίσης και ο μηχανισμός Variable-length hash function ο οποίος χρησιμοποιείται για την ολοκλήρωση παραγωγής του hash.

```
Function Hash(message, digestSize)
   Inputs:
      message:         Bytes (0..2^32-1)     Message to be hashed
      digestSize:      Integer (1..2^32)     Desired number of bytes to be returned
   Output:
      digest:          Bytes (digestSize)    The resulting generated bytes, digestSize bytes long

   Hash is a variable-length hash function, built using Blake2b, capable of generating
   digests up to 2^32 bytes.

   If the requested digestSize is 64-bytes or lower, then we use Blake2b directly
   if (digestSize <= 64) then
      return Blake2b(digestSize ‖ message, digestSize) //concatenate 32-bit little endian digestSize with the message bytes

   For desired hashes over 64-bytes (e.g. 1024 bytes for Argon2 blocks),
   we use Blake2b to generate twice the number of needed 64-byte blocks,
   and then only use 32-bytes from each block

   Calculate the number of whole blocks (knowing we're only going to use 32-bytes from each)
   r ← Ceil(digestSize/32)-1;

   Generate r whole blocks.
   Initial block is generated from message
   V_1 ← Blake2b(digestSize ‖ message, 64);
   Subsequent blocks are generated from previous blocks
   for i ← 2 to r do
      V_i ← Blake2b(V_i-1, 64)
   Generate the final (possibly partial) block
   partialBytesNeeded ← digestSize - 32*r;
   V_r+1 ← Blake2b(V_r, partialBytesNeeded)

   Concatenate the first 32-bytes of each block V_i
   (except the possibly partial last block, which we take the whole thing)
   Let A_i represent the lower 32-bytes of block V_i
   return A_1 ‖ A_2 ‖ ... ‖ A_r ‖ V_r+1
```

3.1.4 JSON

Στη πληροφορική το JSON (JavaScript Object Notation) είναι ένα ανοικτό μορφότυπο το οποίο χρησιμοποιεί κείμενο που διαβάζεται από άνθρωπο για τη μετάδοση αντικειμένων δεδομένων σε ζεύγη χαρακτηριστικών – τιμών και τύπου συστοιχιών ή όποιας άλλης σειριοποίησης τιμής. Είναι ένα κοινό μορφότυπο δεδομένων το οποίο χρησιμοποιείται για την ασύγχρονη επικοινωνία περιηγητή – διακομιστή αντικαθιστώντας τα συνήθη έως τώρα μορφότυπα XML. Ένα παράδειγμα τέτοιας επικοινωνίας αποτελεί το παρακάτω.

```
{
    "glossary": {
        "title": "example glossary",
                "GlossDiv": {
            "title": "S",
                        "GlossList": {
                "GlossEntry": {
                    "ID": "SGML",
                                        "SortAs": "SGML",
                                        "GlossTerm": "Standard Generalized Markup Language",
                                        "Acronym": "SGML",
                                        "Abbrev": "ISO 8879:1986",
                                        "GlossDef": {
                        "para": "A meta-markup language, used to create markup languages such as DocBook.",
                                                "GlossSeeAlso": ["GML", "XML"]
                    },
                                        "GlossSee": "markup"
                }
            }
        }
    }
}
```

Το αντίστοιχο σε XML θα ήταν της μορφής:

```
<!DOCTYPE glossary PUBLIC "-//OASIS//DTD DocBook V3.1//EN">
 <glossary><title>example glossary</title>
  <GlossDiv><title>S</title>
   <GlossList>
    <GlossEntry ID="SGML" SortAs="SGML">
     <GlossTerm>Standard Generalized Markup Language</GlossTerm>
     <Acronym>SGML</Acronym>
     <Abbrev>ISO 8879:1986</Abbrev>
     <GlossDef>
      <para>A meta-markup language, used to create markup
languages such as DocBook.</para>
      <GlossSeeAlso OtherTerm="GML">
      <GlossSeeAlso OtherTerm="XML">
     </GlossDef>
     <GlossSee OtherTerm="markup">
    </GlossEntry>
   </GlossList>
  </GlossDiv>
 </glossary>
```

3.1.5 SQLite

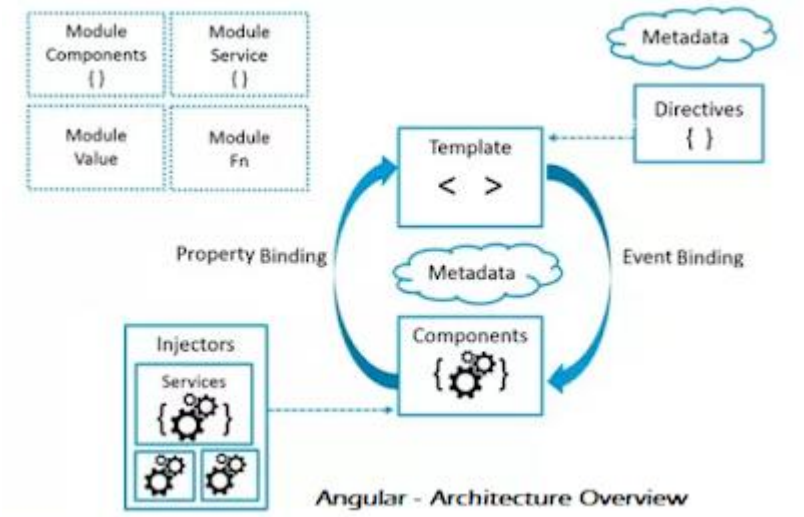
Είναι ένα σύστημα διαχείρισης σχεσιακών βάσεων δεδομένων που περιέχεται σε μια C προγραμματιστική βιβλιοθήκη. Πάνω σε αυτή τη τεχνολογία βάσεων αποθηκεύτηκαν όλα μας τα δεδομένα.

3.1.6 Angular 5

Η Angular είναι ένα framework το οποίο χρησιμοποιείται στη πληθώρα των σημερινών front - end (γραφικών περιβαλλόντων) προγραμμάτων. Αποσκοπεί στην δημιουργία μιας σελίδας γραφικού περιβάλλοντος (single-page client app) χρησιμοποιώντας HTML και TypeScript σε αντίθεση με τα συνήθη γραφικά περιβάλλοντα στον ιστό που χρησιμοποιούν HTML, CSS και JavaScript.

Ο προγραμματιστής καλείται να δημιουργήσει βασικά μπλοκ επαναχρησιμοποιούμενου κώδικα τα οποία καλούνται components και είναι συγκεντρωμένα και οργανωμένα σε NgModules. Το βασικό κομμάτι της εκάστοτε εφαρμογής γράφεται σε TypeScript έναντι του κλασσικού Css και JavaScript ή της πλέον διαδεδομένης για κλασσικά συστήματα με php, Apache κτλ. Jquery βιβλιοθήκης. Το πιο σημαντικό κομμάτι του συγκεκριμένου framework είναι πως εναλλάσσει τα μπλοκ κώδικα (ανανεώνει τμήματα της σελίδας) με ένα βέλτιστο τρόπο αντικαθιστώντας το κλασσικό AJAX πάνω στο οποίο στηρίχτηκε η λειτουργία ανανέωσης συγκεκριμένων κομματιών της ιστοσελίδας αποφεύγοντας να ανανεωθεί ολόκληρη με (postback στο server). Έτσι επιτυγχάνει να αποσυμφορήσει το server καθώς στέλνει μόνο κομμάτια της html και όχι ολόκληρη τη σελίδα κάθε φορά που ανανεώνεται οτιδήποτε γραφικό και βοηθάει δραματικά όταν για παράδειγμα σε ένα Web App ταυτόχρονα χρησιμοποιείται από πάρα πολλούς χρήστες (scalability).

Συνεπώς με αυτή την αρχιτεκτονική δε χρειάζεται να επαναληφθεί κώδικας που χρησιμοποιείται και σε άλλες περιοχές και εμφανίζεται με το ίδιο μπλοκ κώδικα. Είναι ένα σύστημα αρχιτεκτονικής διαφορετικό από το MVC χρησιμοποιώντας Components, Templates, Metadata, Data Binding, Directives, Services και Dependency Injection. Παρατίθεται και το σχεδιάγραμμα αυτής της αρχιτεκτονικής.



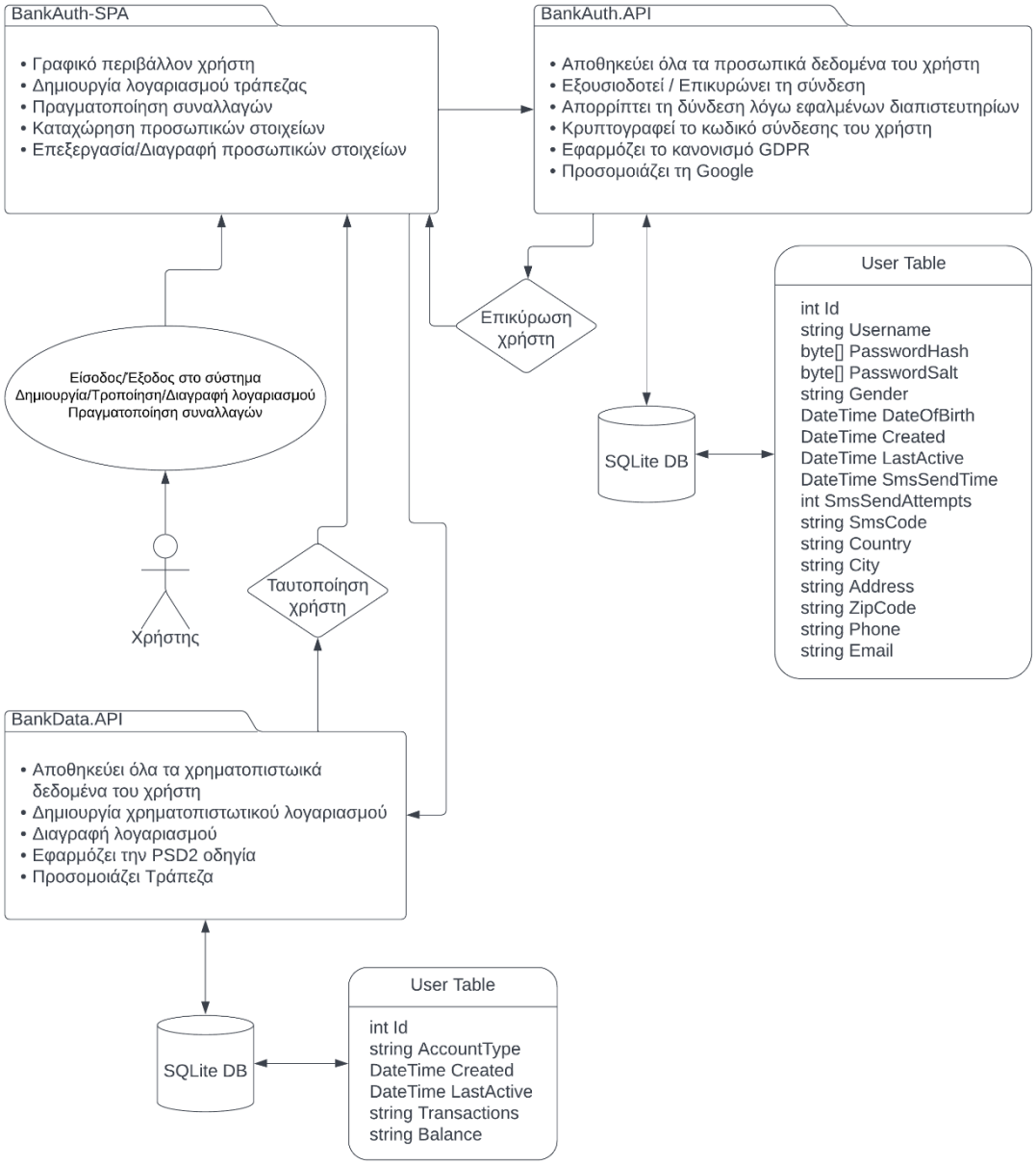
4 Τεχνική ανάλυση

4.1 Σχεδιάγραμμα ροής της υλοποίησης

Ακολουθεί ένα συνοπτικό σχεδιάγραμμα της λογικής ροής που ακολουθείται περιγράφοντας την επικοινωνία των συστημάτων μεταξύ τους. Περιλαμβάνονται και τα τρία συστήματα BankAuth-SPA, BankAuth.API και BankData.API τα οποία όπως προαναφέρθηκε είναι ανεξάρτητα μεταξύ τους και επικοινωνούν μέσω κλήσεις στον ιστό με κρυπτογραφημένη επικοινωνία https. Περιγράφονται οι βάσεις αποθήκευσης των δεδομένων τους με συνοπτική περιγραφή των πινάκων και των στοιχείων που περιέχουν.

Οι βάσεις επικοινωνούν αποκλειστικά σε επίπεδο τοπικού δικτύου (απομονωμένες από το διαδίκτυο) για ασφάλεια διαρροής των δεδομένων και μόνο με το εκάστοτε σύστημα για το οποίο και έχουν δημιουργηθεί. Το BankAuth-SPA επικοινωνεί με το BankAuth.API για να δοθεί εξουσιοδότηση στο χρήστη ενσωματώνοντας το πρότυπο της ισχυρής εξακρίβωσης χρήστη με την αποστολή μοναδικού πιστοποίησης μέσω SMS και έπειτα με το BankData.API που έχοντας γίνει η ταυτοποίηση μέσω του κλειδιού χρήστη από τη βάση του BankAuth.API επιστρέφονται τα δεδομένα του χρηματοπιστωτικού λογαριασμού του χρήστη πίσω στο γραφικό περιβάλλον. Τηρείται το μοντέλο της ελάχιστης μετάδοσης της πληροφορίας που αφορά τα στοιχεία του χρήστη που εξυπηρετούν αποκλειστικά τις ανάγκες της συναλλαγής. Στο γραφικό περιβάλλον οι κλήσεις στο API πιστοποίησης γίνονται ως service καθώς υπάρχει το ενδεχόμενο σε μια συνεδρία να γίνουν πολλές κλήσεις μεταξύ των δύο συστημάτων για επαναπιστοποίηση του χρήστη όπως σε περίπτωση που λήξει το μοναδικό κλειδί πιστοποίησης jwt token.

Σε πλήρη τήρηση κομβικών σημείων της PSD2 για την λειτουργία του συστήματος απαραίτητη προϋπόθεση είναι να βρίσκεται ο χρήστης με πλήρη πρόσβαση στο διαδίκτυο. Ταυτόχρονα δεν αποθηκεύονται οποιαδήποτε δεδομένα που αφορούν τη συναλλαγή όπως στοιχεία χρήστη προσωπικής και χρηματοπιστωτικής φύσεως σε κανένα σύστημα εκτός του αρμόδιου χρηματοπιστωτικού ιδρύματος που υφίσταται ο λογαριασμός. Είναι αναγκαίο για την ταυτοποίηση του πελάτη να υπάρξει αμφίδρομη επικοινωνία του συστήματος με το πάροχο υπηρεσιών πληρωμών καθώς και το πάροχο υπηρεσιών πληρωμών.

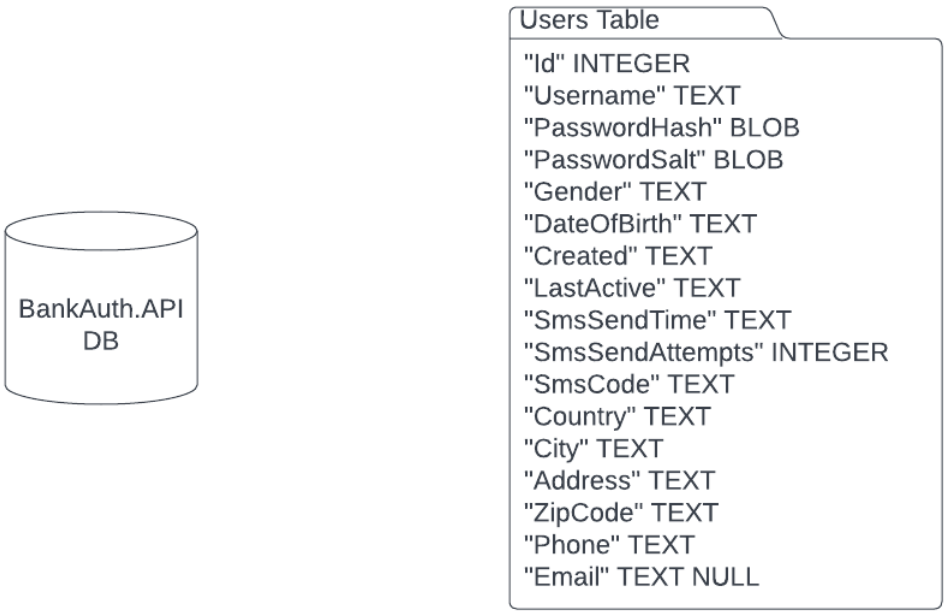


5 Υλοποίηση

5.1 Βάσεις δεδομένων

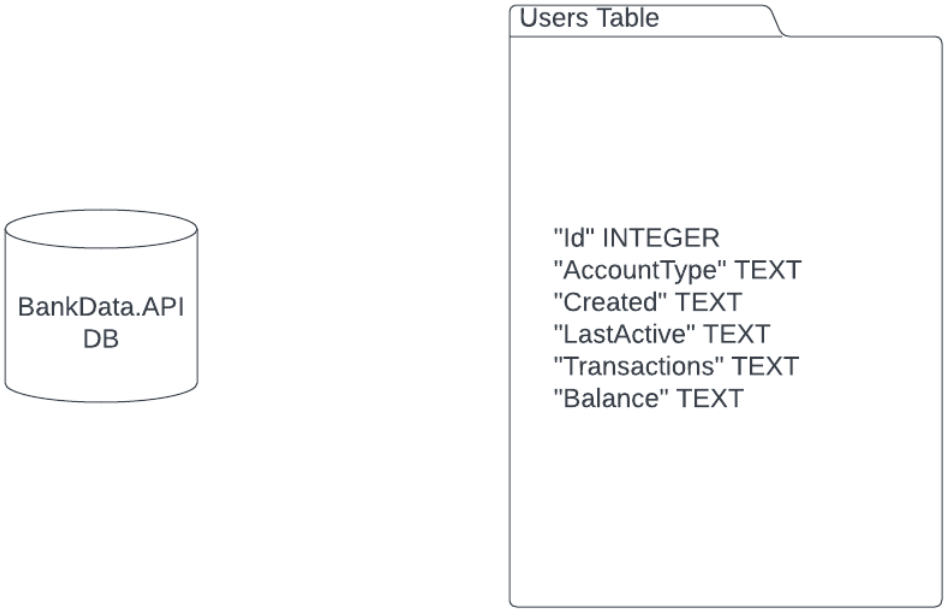
5.1.1 Βάση BankAuth.API

Ακολουθεί σχεδιάγραμμα υλοποίησης της βάσης του BankAuth.API.



5.1.2 Βάση BankData.API

Ακολουθεί σχεδιάγραμμα υλοποίησης της βάσης του BankData.API.



5.2 BankAuth-SPA

Το συγκεκριμένο project υλοποιήθηκε πάνω στο framework Angular 5. Η δομή που απαιτείται στο συγκεκριμένο framework αποτελείται από επαναχρησιμοποιούμενα μπλοκ (κομμάτια) κώδικα τα οποία ονομάζονται components. Στη συγκεκριμένη περίπτωση δημιουργήθηκαν τα εξής:

- Home
- Cards
- Investments
- Nav
- Register
- Account
- App

Για την ορθή λειτουργία επίσης δημιουργήθηκαν άλλες δύο λειτουργίες που δεν ανήκουν στα ανωτέρω μπλοκ (reusable components) καθώς υπάρχει η ανάγκη να τρέχουν σε όλη τη διάρκεια που τρέχει η εφαρμογή στη μνήμη και αφορούν τα εξής:

- Guards
- Services

5.2.1 Home Component

Το συγκεκριμένο component αποτελεί την αρχική οθόνη του γραφικού περιβάλλοντος την οποία συναντάει ο οποιοσδήποτε χρήστης στο διαδίκτυο και στην οποία έχει πρόσβαση να εκτελέσει είσοδο στο σύστημα αν είναι ήδη χρήστης ή να δημιουργήσει λογαριασμό αν είναι νέος.

Για λόγους σύνοψης της εργασίας αυτής και αποφυγή παράθεσης μακροσκελών τμημάτων του κώδικα της υλοποίησης θα αποφευχθούν τμήματα που αφορούν τα γραφικά μέρη του προγράμματος (html κτλ) όπως και όλα τα λειτουργικά μέρη κώδικά που χρειάζεται το framework για να λειτουργήσει. Θα υπάρξει όμως ανάλυση στα πιο ενδιαφέροντα λειτουργικά τμήματα.

Παρατίθεται ο κώδικας της υλοποίησης για το συγκεκριμένο component.

```
import { Component, OnInit } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css']
})
export class HomeComponent implements OnInit {
  registerMode = false;
  // values: any;

  constructor(private http: HttpClient) { }

  ngOnInit() {
    // this.getValues();
  }

  registerToggle() {
    this.registerMode = !this.registerMode;
  }

  // getValues() {
  //   this.http.get('http://localhost:5000/api/values').subscribe(response => {
  //     this.values = response;
  //   }, error => {
  //     console.log(error);
  //   });
  // }

  cancelRegisterMode(registerMode: boolean) {
    this.registerMode = registerMode;
  }


}
```

5.2.2 Cards Component

Η συγκεκριμένη υλοποίηση αφορά καθαρά τα στοιχεία μιας κάρτας που μπορεί να έχει κάποιος πελάτης μιας τράπεζας η οποία και είναι άμεσα συνδεδεμένη με το λογαριασμό πίστωσης του. Σε αυτήν την οθόνη μπορεί να δει λεπτομέρειες που αφορούν τη κάρτα του όπως τελευταίες κινήσεις, εκκρεμείς και ολοκληρωμένες συναλλαγές που εκτελέστηκαν με αυτή τη κάρτα κ.ά.

5.2.3 Investments Component

Το συγκεκριμένο component αφορά την οθόνη που εμφανίζει τις διάφορες επενδύσεις που μπορεί να κάνει κάποιος χρήστης καθώς περιέχει και συνοπτική ανάλυση των ήδη υπάρχοντων επενδύσεων.

5.2.4 Nav Component

Το συγκεκριμένο component αφορά βασικό λειτουργικό τμήμα της εφαρμογής καθώς λόγω του ότι όπως φαίνεται και από το τίτλο του αφορά τη περιήγηση μεταξύ των διάφορων τμημάτων της εφαρμογής περιέχει στην υλοποίηση του λογική εισόδου στο σύστημα, απόρριψης της αίτησης σύνδεσης, ικανοποίησης αιτήματος δημιουργίας λογαριασμού, εκτέλεση της διαδικασίας πιστοποίησης μέσω μηνυμάτων SMS (strong authentication – two factor authentication) καθώς και επανασύνδεση ή αποσύνδεση από το σύστημα. Βρίσκεται στο πάνω μέρος της οθόνης και υπάρχει σε όλες τις οθόνες αφότου ο χρήστης αιτηθεί δημιουργία λογαριασμού ή συνδεθεί καθώς προϋπάρχει λογαριασμός.

```
import { Component, OnInit } from '@angular/core';
import { AuthService } from '../_services/auth.service';
import { AlertifyService } from '../_services/alertify.service';
import { Router } from '@angular/router';

@Component({
  selector: 'app-nav',
  templateUrl: './nav.component.html',
  styleUrls: ['./nav.component.css']
})
export class NavComponent implements OnInit {
  model: any = {};
  sendSms = false;

  constructor(
    public authService: AuthService,
    private alertify: AlertifyService,
    private router: Router
  ) {}

  ngOnInit() {}

  login() {
    this.authService.login(this.model).subscribe(
      next => {
        this.alertify.success('Logged in succesfully');
      },
      error => {
        this.alertify.error(error);
      },
      () => {
        this.router.navigate(['/account']);
      }
    );
  }
```

```
loggedIn() {
  return this.authService.loggedIn();
  // const token = localStorage.getItem('token');
  // return !!token;
}

requestLogin() {
  this.sendSms = true;
  this.authService.requestLogin(this.model).subscribe(
    next => {
      this.alertify.success('SMS sent succesfully');
    },
    error => {
      this.alertify.error(error);
      this.sendSms = false;
    }
  );
}

logout() {
  // this.login();
  localStorage.removeItem('token');
  this.alertify.message('logged out');
  this.router.navigate(['/home']);
  this.sendSms = false;
}

sendSMSfunction() {
  if (
    this.model.username === null ||
    this.model.password === null ||
    this.model.phone === null ||
    this.model.username === undefined ||
    this.model.password === undefined ||
    this.model.phone === undefined
  ) {
      return (this.sendSms = false);
    } else {
      return (this.sendSms = true);
    }
  }

  hideLoginDiv(event) {
    if (event.action === 'done') {
      this.sendSms = false;
    }
  }
}
```

5.2.5 Register Component

Το συγκεκριμένο component αφορά τη λογική δημιουργίας λογαριασμού για ένα νέο χρήστη στο σύστημα.

```
import { Component, OnInit, Input, Output, EventEmitter } from '@angular/core';
import { AuthService } from '../_services/auth.service';
import { AlertifyService } from '../_services/alertify.service';
import { Router } from '@angular/router';

@Component({
  selector: 'app-register',
  templateUrl: './register.component.html',
  styleUrls: ['./register.component.css']
})
export class RegisterComponent implements OnInit {
  // @Input() valuesFromHome: any;
  @Output() cancelRegister = new EventEmitter();
  model: any = {};

  constructor(
    private authService: AuthService,
    private alertify: AlertifyService
  ) {}

  ngOnInit() {}

  register() {
    this.authService.register(this.model).subscribe(
      next => {
        this.alertify.success('registration successful');
      },
      error => {
        this.alertify.error(error);
      }
    );
  }

  cancel() {
    this.cancelRegister.emit(false);
  }
}
```

5.2.6 Account Component

Το component αυτό αναλαμβάνει να εμφανίσει στην οθόνη τα χρηματοπιστωτικά στοιχεία του λογαριασμού ενός χρήστη. Για να το επιτεύξει αυτό, χρειάζεται εφόσον έχει καλεστεί το service της πιστοποίησης του χρήστη (κλήση στο BankAuth.API) να καλέσει το service της τράπεζας (κλήση στο BankData.API) όπου και θα αντληθούν τα δεδομένα προς επίδειξη.

```
import { Component, OnInit } from '@angular/core';
import { JwtHelperService } from '@auth0/angular-jwt';
import { HttpClient } from '@angular/common/http';
import { AuthService } from '../_services/auth.service';
import { AlertifyService } from '../_services/alertify.service';
import { Router } from '@angular/router';
import { map } from 'rxjs/operators';

@Component({
  selector: 'app-account',
  templateUrl: './account.component.html',
  styleUrls: ['./account.component.css']
})
export class AccountComponent implements OnInit {
  baseUrl = 'http://localhost:5100/user/';
  jwtHelper = new JwtHelperService();
  userId: string;
  model: any;
  userData: any;

  constructor(
    private http: HttpClient,
    public authService: AuthService,
    private alertify: AlertifyService,
    private router: Router
  ) {}

  ngOnInit() {
    this.getAccountData();
  }

  getAccountData() {
    this.userId = this.authService.decodedToken.nameid;

    return this.http
      .post(this.baseUrl + this.userId, this.model)
      .pipe(
        map((response: any) => {
          const data = response;
          if (data) {
            this.userData = data;
          }
        })
      )
      .subscribe(
        next => {
          // this.alertify.success('SMS sent succesfully');
        },
        error => {
          this.alertify.error(error);
        }
      );
  }
}
```

5.2.7 App Component

Το component αυτό με τη συγκεκριμένη ονομασία περιέχεται σε όλες τις υλοποιήσεις εφαρμογών πάνω στο συγκεκριμένο framework και περιέχει λογική η οποία εκτελείται αναλόγως τη περίπτωση σε όλη τη διάρκεια ζωής της εφαρμογής.

5.2.7.1 App Component

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
  jwtHelper = new JwtHelperService();

  constructor(private authService: AuthService) {}

  ngOnInit() {
    const token = localStorage.getItem('token');
    if (token) {
      this.authService.decodedToken = this.jwtHelper.decodeToken(token);
    }
  }
}
```

5.2.7.2 App Module

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { HttpClientModule } from '@angular/common/http';
import { FormsModule } from '@angular/forms';
import { BsDropdownModule } from 'ngx-bootstrap';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
import { RouterModule } from '@angular/router';
import { JwtModule } from '@auth0/angular-jwt';

import { AppComponent } from './app.component';
import { NavComponent } from './nav/nav.component';
import { AuthService } from './_services/auth.service';
import { HomeComponent } from './home/home.component';
import { RegisterComponent } from './register/register.component';
import { ErrorInterceptorProvider } from './_services/error.interceptor';
import { InvestmentsComponent } from './investments/investments.component';
import { AccountComponent } from './account/account.component';
import { CardsComponent } from './cards/cards.component';
import { appRoutes } from './routes';
import { CountdownModule } from 'ngx-countdown';

export function tokenGetter() {
  return localStorage.getItem('token');
}

@NgModule({
  declarations: [
    AppComponent,
    NavComponent,
    HomeComponent,
    RegisterComponent,
    InvestmentsComponent,
    AccountComponent,
    CardsComponent
  ],
  imports: [
    BrowserModule,
    HttpClientModule,
    FormsModule,
    BsDropdownModule.forRoot(),
    BrowserAnimationsModule,
    RouterModule.forRoot(appRoutes),
    CountdownModule,
```

```
    JwtModule.forRoot({
      config: {
        tokenGetter: tokenGetter,
        whitelistedDomains: ['localhost:5000', 'localhost:5100'],
        blacklistedRoutes: ['localhost:5000/api/auth']
      }
    })
  ],
  providers: [AuthService, ErrorInterceptorProvider],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

5.2.8 Guard

Η συγκεκριμένη υλοποίηση αφορά μια υπηρεσία ανακατεύθυνσης του χρήστη εφόσον δεν εγκριθούν τα διαπιστευτήρια του.

```
import { Injectable } from '@angular/core';
import { CanActivate, Router } from '@angular/router';
import { AuthService } from '../_services/auth.service';
import { AlertifyService } from '../_services/alertify.service';

@Injectable({
  providedIn: 'root'
})
export class AuthGuard implements CanActivate {
  constructor(
    private authService: AuthService,
    private router: Router,
    private alertify: AlertifyService
  ) {}

  canActivate(): boolean {
    if (this.authService.loggedIn()) {
      return true;
    }

    this.alertify.error(
      'Unauthorized. Please login with your valid credentials.'
    );
    this.router.navigate(['/home']);
    return false;
  }
}
```

5.2.9 Services

Οι υλοποιήσεις κάτω από το γενικό πλαίσιο με την ονομασία services αφορούν λογική κώδικα που εκτελείται συνεχώς στη μνήμη κατά απαίτηση από κάποιο component και είναι υπεύθυνες να πιστοποιήσουν το χρήστη (auth.service) ή να ειδοποιήσουν με κάποιο μήνυμα το χρήστη (alertify.service)

5.2.9.1 Alertify service

Υπεύθυνη υπηρεσία για την εμφάνιση μηνυμάτων στην οθόνη του χρήστη.

```
import { Injectable } from '@angular/core';
import * as alertify from 'alertifyjs';

@Injectable({
  providedIn: 'root'
})
export class AlertifyService {
  constructor() {}

  confirm(message: string, okCallback: () => any) {
    alertify.confirm(message, (e: any) => {
      if (e) {
        okCallback();
      } else {
      }
    });
  }

  success(message: string) {
    alertify.success(message);
  }

  error(message: string) {
    alertify.error(message);
  }

  warning(message: string) {
    alertify.warning(message);
  }

  message(message: string) {
    alertify.message(message);
  }
}
```

5.2.9.2 Auth service

Υπεύθυνη υπηρεσία πιστοποίησης του χρήστη. Πραγματοποιεί κλήσεις στο BankAuth.API και αποθηκεύει κατόπιν πιστοποίησης το μοναδικό κλειδί στη περιοχή τοπικής αποθήκευσης του εκάστοτε περιηγητή (local storage).

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { map } from 'rxjs/operators';
import { JwtHelperService } from '@auth0/angular-jwt';

@Injectable({
  providedIn: 'root'
})
export class AuthService {
  baseUrl = 'http://localhost:5000/api/auth/';
  jwtHelper = new JwtHelperService();
  decodedToken: any;

  constructor(private http: HttpClient) {}

  login(model: any) {
    return this.http.post(this.baseUrl + 'login', model).pipe(
      map((response: any) => {
        const user = response;
        if (user) {
          localStorage.setItem('token', user.token);
          this.decodedToken = this.jwtHelper.decodeToken(user.token);
          // console.log(this.decodedToken);
        }
      })
    );
  }

  requestLogin(model: any) {
    return this.http.post(this.baseUrl + 'requestlogin', model);
  }

  register(model: any) {
    return this.http.post(this.baseUrl + 'register', model);
  }

  loggedIn() {
    const token = localStorage.getItem('token');
    return !this.jwtHelper.isTokenExpired(token);
  }
}
```

5.2.10 Σύνοψη

Το γραφικό περιβάλλον καθώς και η διαλειτουργικότητα της εφαρμογής που υλοποιήθηκαν στο συγκεκριμένο framework όπως και οι τεχνικές υλοποίησης των επιμέρους τμημάτων αποτελούν αυτή τη δεδομένη στιγμή μερικά από τα πιο σύγχρονα πρότυπα υλοποίησης. Οι τεχνικές που ακολουθήθηκαν είναι βασισμένες στα πρότυπα σύγχρονης αρχιτεκτονικής καθώς περιέχουν ασύγχρονες κλήσεις σε διάφορες ανεξάρτητες προγραμματιστικές διεπαφές και χρήση πιστοποίησης μέσω jwt – token.

Προτείνεται το συγκεκριμένο σύστημα για κατασκευή τέτοιου είδους διαδικτυακών προγραμμάτων καθώς προσφέρει με ευκολία τη δημιουργία των τμημάτων που απαιτούνται σύμφωνα με τα τελευταία πρότυπα web, ασφάλειας των δεδομένων και της κρυπτογράφησης

της πληροφορίας στοχεύοντας στην ασφάλεια της μεταφοράς διαδικτυακής πληροφορίας από υποκλοπές.

5.3 BankAuth.API

Η διεπαφή προγραμματισμού BankAuth.API υλοποιήθηκε πάνω στο .NET Core framework. Είναι μια ανεξάρτητη υλοποίηση που ως στόχο έχει να αποθηκεύει τα προσωπικά δεδομένα ενός χρήστη εφαρμόζοντας κάθε πτυχή του κανονισμού GDPR καθώς και να τα προσφέρει με ασφάλεια στις πιστοποιημένες για το σύστημα εφαρμογές.

Εφαρμόζοντας το μοντέλο αρχιτεκτονικής MVC στη υλοποίηση έχουν δημιουργηθεί controllers καθώς και models της βάσης. Εδώ επίσης εφαρμόζεται μια μοντελοποίηση των πινάκων της βάσης σε αντικείμενα (objects) ώστε ο προγραμματισμός να γίνεται πιο εύκολος και ευανάγνωστος για τον άνθρωπο. Εν συνεχεία μοντελοποιούνται και τα αντικείμενα που μεταφέρουν τη πληροφορία στα διάφορα σημεία του προγράμματος και ονομάζονται Dtos (Data Transfer Objects). Όλα αυτά περικλείουν το Entity Framework πάνω στο οποίο γράφεται η επικοινωνία με τη βάση με τη γλώσσα Linq. Έτσι ο προγραμματιστής αποφεύγει να γράφει queries για επικοινωνία με τη βάση και αυτό βελτιστοποιεί το προγραμματισμό σε ταχύτητα καθώς και σε ασφάλεια. Προστατεύεται έτσι το πρόγραμμα από τυχόν κυβερνο - επιθέσεις που βασίζονται σε sql injections.

Controllers που υλοποιήθηκαν:

- AuthController

Class που εφαρμόστηκαν:

- AuthRepository
- DataContext
- IAuthRepository (Interface)

Dtos προγραμματος:

- UserForLoginDto
- UserForRegisterDto
- UserForRequestLogin

Models βάσης:

- User

Services:

- MessageServices

5.3.1 AuthController

Ξεκινώντας από τους Controllers που η δουλειά τους είναι να εξυπηρετούν τις κλήσεις που γίνονται στη προγραμματιστική διεπαφή παραθέτουμε το κώδικα που όπως φαίνεται παρακάτω ικανοποιεί τις κλήσεις που αφορούν δημιουργία λογαριασμού – register, είσοδος στο σύστημα από χρήστη που έχει ήδη λογαριασμό – requestlogin που σε αυτό το σημείο αποστέλλεται ο προσωρινός κωδικός μέσω SMS (two factor authentication) και έπειτα είσοδος – login.

```
using System;
using System.IdentityModel.Tokens.Jwt;
using System.Net.Http;
using System.Security.Claims;
using System.Text;
using System.Threading.Tasks;
using BankAuth.API.Data;
using BankAuth.API.Dtos;
using BankAuth.API.Models;
using BankAuth.API.Services;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Configuration;
using Microsoft.IdentityModel.Tokens;

namespace BankAuth.API.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class AuthController : ControllerBase
    {
        private readonly IAuthRepository _repo;
        private readonly IConfiguration _config;
        private readonly DataContext _context;
        private readonly IHttpClientFactory _clientFactory;
        public AuthController(IAuthRepository repo, IConfiguration config, DataContext context, IHttpClientFactory clientFactory)
        {
            _clientFactory = clientFactory;
            _context = context;
            _config = config;
            _repo = repo;
        }

        [HttpPost("register")]
        public async Task<IActionResult> Register(UserForRegisterDto userForRegisterDto)
        {

            userForRegisterDto.Username = userForRegisterDto.Username.ToLower();

            if (await _repo.UserExists(userForRegisterDto.Username))
                return BadRequest("Username already exists");

            var createdUser = await _repo.Register(userForRegisterDto);



            var request = new HttpRequestMessage(HttpMethod.Post,
                          "http://localhost:5100/user/register");

            // request.Headers.Add("Content-Type", "application/json");

            var client = _clientFactory.CreateClient();

            var response = await client.SendAsync(request);

            //         if (response.IsSuccessStatusCode)
            // return StatusCode(201)


            return response.IsSuccessStatusCode ? StatusCode(201) : StatusCode(403);

        }
```

```
[HttpPost("requestlogin")]
public async Task<IActionResult> RequestLogin(UserForRequestLogin userForRequestLoginDto)
{
    var userFromRepo = await _repo.Login(userForRequestLoginDto.Username.ToLower(), userForRequestLoginDto.Password);

    if (userFromRepo == null)
        return Unauthorized("Unauthorized");

    if (userFromRepo.SmsSendAttempts > 3 && (DateTime.Now - userFromRepo.SmsSendTime).TotalMinutes < 10)
    {
        if (!string.IsNullOrWhiteSpace(userFromRepo.SmsCode))
        {
            userFromRepo.SmsCode = null;
            await _context.SaveChangesAsync();
        }

        var timespanFromLastTry = Math.Floor((10 - (DateTime.Now - userFromRepo.SmsSendTime).TotalMinutes)) == 0 ? 10 -
        (DateTime.Now - userFromRepo.SmsSendTime).TotalMinutes : Math.Floor((10 - (DateTime.Now - userFromRepo.SmsSendTime).TotalMinutes));

        return Unauthorized($"Too many attempts for login. Retry in {timespanFromLastTry} minutes");
    }
    else
    {
        if ((DateTime.Now - userFromRepo.SmsSendTime).TotalSeconds < 60)
        {
            userFromRepo.SmsSendAttempts = userFromRepo.SmsSendAttempts + 1;
            await _context.SaveChangesAsync();
            return Ok();
        }
        else if ((DateTime.Now - userFromRepo.SmsSendTime).TotalMinutes > 10)
            userFromRepo.SmsSendAttempts = 0;

        userFromRepo.SmsSendAttempts = userFromRepo.SmsSendAttempts + 1;
        userFromRepo.SmsSendTime = DateTime.Now;
        Random generator = new Random();
        string code = generator.Next(0, 999999).ToString("D6");
        userFromRepo.SmsCode = code;
        await _context.SaveChangesAsync();

        Console.WriteLine("*****************************************************");
        Console.WriteLine($"{userForRequestLoginDto.Phone} ---------> Your verification Code is: {code}");
        Console.WriteLine("*****************************************************");

    }

    return Ok();
}
    [HttpPost("login")]
    public async Task<IActionResult> Login(UserForLoginDto userForLoginDto)
    {
        var userFromRepo = await _repo.Login(userForLoginDto.Username.ToLower(), userForLoginDto.Password);

        if (userFromRepo == null || userForLoginDto.SmsCode != userFromRepo.SmsCode || (DateTime.Now - userFromRepo.SmsSendTime).TotalSeconds > 60)
            return Unauthorized("Unauthorized");


        var claims = new[]
        {
            new Claim(ClaimTypes.NameIdentifier, userFromRepo.Id.ToString()),
            new Claim(ClaimTypes.Name, userFromRepo.Username)
        };

        var key = new SymmetricSecurityKey(Encoding.UTF8
            .GetBytes(_config.GetSection("AppSettings:Token").Value));

        var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha512Signature);

        var tokenDescriptor = new SecurityTokenDescriptor
        {
            Subject = new ClaimsIdentity(claims),
            Expires = DateTime.Now.AddDays(1),
            SigningCredentials = creds
        };

        var tokenHandler = new JwtSecurityTokenHandler();

        var token = tokenHandler.CreateToken(tokenDescriptor);

        return Ok(new
        {
            token = tokenHandler.WriteToken(token)
        });
    }

}
```

5.3.2 AuthRepository

Κώδικας που περιέχει τη λογική που εφαρμόζεται βάση των κλήσεων μέσω του controller. Εδώ περιέχεται και η λογική παραγωγής - μετατροπής του κρυπτογραφημένου κωδικού λογαριασμού χρήστη μέσω της Argon2 τεχνικής.

```
using System;
using System.Globalization;
using System.Linq;
using System.Security.Cryptography;
using System.Text;
using System.Threading.Tasks;
using AutoMapper;
using BankAuth.API.Dtos;
using BankAuth.API.Models;
using Konscious.Security.Cryptography;
using Microsoft.EntityFrameworkCore;
using static BankAuth.API.Services.MessageServices;

namespace BankAuth.API.Data
{
    public class AuthRepository : IAuthRepository
    {
        private readonly DataContext _context;
        private readonly IMapper _mapper;
        public AuthRepository(DataContext context, IMapper mapper)
        {
            _mapper = mapper;
            _context = context;
        }
        public async Task<User> Login(string username, string password)
        {
            var user = await _context.Users.FirstOrDefaultAsync(x => x.Username == username);

            if (user == null)
                return null;

            if (!VerifyPasswordHash(password, user.PasswordHash, user.PasswordSalt))
                return null;

            return user;
        }
```

```
        public async Task<User> Register(UserForRegisterDto user)
        {
            byte[] passwordHash, passwordSalt;
            CreatePasswordHash(user.Password, out passwordHash, out passwordSalt);

            var newUser = new User();

            newUser = _mapper.Map<User>(user);

            newUser.PasswordHash = passwordHash;
            newUser.PasswordSalt = passwordSalt;

            await _context.Users.AddAsync(newUser);
            await _context.SaveChangesAsync();

            return newUser;
        }

        private void CreatePasswordHash(string password, out byte[] passwordHash, out byte[] passwordSalt)
        {
            var buffer = new byte[16];

            using (var rng = new RNGCryptoServiceProvider())
            {
                rng.GetBytes(buffer);
                passwordSalt = buffer;

                var argon2 = new Argon2id(Encoding.UTF8.GetBytes(password));
                argon2.Salt = passwordSalt;
                argon2.DegreeOfParallelism = 8; // four cores
                argon2.Iterations = 4;
                argon2.MemorySize = 1024 * 1024; // 1 GB
                passwordHash = argon2.GetBytes(16);
            }
        }
```

```
        private bool VerifyPasswordHash(string password, byte[] passwordHash, byte[] passwordSalt)
        {
            var argon2 = new Argon2id(Encoding.UTF8.GetBytes(password));
            argon2.Salt = passwordSalt;
            argon2.DegreeOfParallelism = 8; // four cores
            argon2.Iterations = 4;
            argon2.MemorySize = 1024 * 1024; // 1 GB
            var newpasswordHash = argon2.GetBytes(16);

            return passwordHash.SequenceEqual(newpasswordHash);
        }

        public async Task<bool> UserExists(string username)
        {
            if (await _context.Users.AnyAsync(x => x.Username == username))
                return true;

            return false;
        }
    }
}
```

5.3.3 DataContext

```
using BankAuth.API.Models;
using Microsoft.EntityFrameworkCore;

namespace BankAuth.API.Data
{
    public class DataContext : DbContext
    {
        public DataContext(DbContextOptions<DataContext> options) : base(options) { }

        public DbSet<Value> Values { get; set; }
        public DbSet<User> Users { get; set; }
    }
}
```

5.3.4 IAuthRepository

```
using System.Threading.Tasks;
using BankAuth.API.Dtos;
using BankAuth.API.Models;

namespace BankAuth.API.Data
{
    public interface IAuthRepository
    {
        Task<User> Register(UserForRegisterDto user);
        Task<User> Login(string username, string password);
        Task<bool> UserExists(string username);
    }
}
```

5.3.5 UserForLoginDto

```
namespace BankAuth.API.Dtos
{
    public class UserForLoginDto
    {
        public string Username { get; set; }
        public string Password { get; set; }
        public string SmsCode { get; set; }

    }
}
```

5.3.6 UserForRegisterDto

```
using System;
using System.ComponentModel.DataAnnotations;

namespace BankAuth.API.Dtos
{
    public class UserForRegisterDto
    {
        [Required]
        public string Username { get; set; }

        [Required]
        [StringLength(8, MinimumLength = 4, ErrorMessage = "You must specify password between 4 and 8 characters")]
        public string Password { get; set; }
        public string Gender { get; set; }
        public DateTime DateOfBirth { get; set; }
        public DateTime Created = DateTime.Now;
        public DateTime LastActive = DateTime.Now;
        public string Country { get; set; }
        public string City { get; set; }
        public string Address { get; set; }
        public string ZipCode { get; set; }
        public string Phone { get; set; }
        public string Email { get; set; }
    }
}
```

5.3.7 UserForRequestLogin

```
namespace BankAuth.API.Dtos
{
    public class UserForRequestLogin
    {
        public string Username { get; set; }
        public string Password { get; set; }
        public string Phone { get; set; }

    }
}
```

5.3.8 User Model for DB

```
using System;

namespace BankAuth.API.Models
{
    public class User
    {
        public int Id { get; set; }
        public string Username { get; set; }
        public byte[] PasswordHash { get; set; }
        public byte[] PasswordSalt { get; set; }
        public string Gender { get; set; }
        public DateTime DateOfBirth { get; set; }
        public DateTime Created { get; set; }
        public DateTime LastActive { get; set; }
        public DateTime SmsSendTime { get; set; }
        public int SmsSendAttempts { get; set; }
        public string SmsCode { get; set; }
        public string Country { get; set; }
        public string City { get; set; }
        public string Address { get; set; }
        public string ZipCode { get; set; }
        public string Phone { get; set; }
        public string Email { get; set; }
    }
}
```

5.3.9 MessageServices

Εδώ βρίσκεται η υλοποίηση της υπηρεσίας αποστολής των SMS μηνυμάτων που αφορούν την αρχιτεκτονική διπλής ισχυρής ταυτοποίησης. Όπως είναι λογικό αυτό εξυπηρετήθηκε από υπηρεσία «τρίτων» (third party software) και συγκεκριμένα από την Αμερικάνικη εταιρεία Twilio μέσω Rest API.

```
using Microsoft.AspNetCore.Identity.UI.Services;
using Microsoft.Extensions.Options;
using System.Threading.Tasks;
using Twilio;
using Twilio.Rest.Api.V2010.Account;
using Twilio.Types;

namespace BankAuth.API.Services
{
    public class MessageServices
    {
        public Task SendSmsAsync(string number, string message)
        {
            var accountSid = "AC71c963371c741917712df0b838700e1e";

            var authToken = "a4133dc46522a702d275bd3616b21c16";

            TwilioClient.Init(accountSid, authToken);

            return MessageResource.CreateAsync(
              to: new PhoneNumber(number),
              from: new PhoneNumber("+18653510190"),
              body: message);
        }
    }
}
```

5.3.10 Σύνοψη

Παραπάνω το DataContext, IAuthRepository, UserForLoginDto, UserForRegisterDto, UserForRequestLogin και User model της βάσης του προγράμματος που παρατέθηκαν δεν επεξηγήθηκαν περαιτέρω καθώς δεν περιέχουν ιδιαίτερης λογικής σημασίας υλοποίηση. Αυτό σημαίνει πως δημιουργήθηκαν καθαρά για την εσωτερική διαδικαστική λειτουργικότητα της εφαρμογής και δεν αποτελούν ιδιαίτερο ενδιαφέρον για περισσότερη ανάλυση.

5.4 BankData.API

Η διεπαφή που δημιουργήθηκε με την ονομασία BankData.API είναι υπεύθυνη για την αποθήκευση των χρηματοπιστωτικών δεδομένων ενός χρήστη δίχως να περιέχει προσωπικά στοιχεία, κάτι που τη καθιστά πρότυπο χρηματοπιστωτικό σύστημα καθώς εφαρμόζει την PSD2 οδηγία. Είναι υπεύθυνη για την ασφαλή αποθήκευση αυτών των δεδομένων καθώς και για την εξυπηρέτηση κλήσεων με αποστολή των στοιχείων αυτών σε πιστοποιημένο σύστημα.

Αναπτύχθηκε πάνω στο .Net Core σύστημα όπως και η BankAuth.API ακριβώς με τα ίδια πρότυπα αρχιτεκτονικής. Συνεπώς και εδώ έχουμε MVC αρχιτεκτονική, Entity Framework, C# και Linq (Language Integrated Query) γλώσσες προγραμματισμού. Αξίζει εδώ να σημειωθεί πως και οι δύο προγραμματιστικές διεπαφές προσφέρουν τις υπηρεσίες τους βάση της αρχιτεκτονικής με Restfull Web Services ή αλλιώς REST API – Representational State Transfer API. Η συγκεκριμένη αρχιτεκτονική προτιμάται και οι περισσότερες προγραμματιστικές διεπαφές σχεδιάζονται με βάση αυτό το πρότυπο καθώς χρησιμοποιούν μικρότερου όγκου επικοινωνία μεταξύ τους (less bandwidth), χρησιμοποιούν κείμενο για επικοινωνία και μεταφορά πληροφορίας σε μορφή JSON το οποίο μεταφέρεται με ασφάλεια καθώς είναι κρυπτογραφημένο μέσω https.

Για την υλοποίηση αυτή δημιουργήθηκαν τα εξής:

Controllers:

- UserController

Class:

- DataContext

Models:

- User

5.4.1 UserController

Αναλαμβάνει όπως και στο προηγούμενο API να εξυπηρετήσει τις κλήσεις και να επιστρέψει τα στοιχεία που χρειάζονται βάσει id χρήστη (μοναδικός χρήστης για το σύστημα και ταυτοποιείται ο λογαριασμός αυτός του χρήστη μέσω του BankAuth.API). Τέλος εξυπηρετεί

και τη μοναδική λειτουργία για «ανώνυμο» χρήστη, μη ταυτοποιημένου από το σύστημα μόνο για δημιουργία λογαριασμού.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Claims;
using System.Threading.Tasks;
using BankData.API.Data;
using BankData.API.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Logging;

namespace BankData.API.Controllers
{
    [Authorize]
    [Route("[controller]")]
    [ApiController]
    public class UserController : ControllerBase
    {
        private readonly DataContext _context;
        public UserController(DataContext context)
        {
            _context = context;
        }

        [HttpPost("{id}")]
        public async Task<IActionResult> GetData(int id)
        {
            string tokenUserId = User.FindFirstValue(ClaimTypes.NameIdentifier);

            if (tokenUserId != id.ToString())
                return Unauthorized("Unauthorized");

            var user = await _context.Users.FirstOrDefaultAsync(x => x.Id == id);

            user.LastActive = DateTime.Now;

            await _context.SaveChangesAsync();

            if (user == null)
                return Unauthorized("Unauthorized");

            return Ok(user);

        }
        [AllowAnonymous]
        [HttpPost("register")]
        public async Task<IActionResult> Register()
        {
            var user = new User();

            user.AccountType = "Basic";

            user.Created = DateTime.Now;

            user.LastActive = DateTime.Now;

            user.Transactions = "no current transactions";

            user.Balance = "0 ευρώ";

            await _context.Users.AddAsync(user);
            await _context.SaveChangesAsync();

            return StatusCode(201);

        }

    }
}
```

5.4.2 DataContext

```
using BankData.API.Models;
using Microsoft.EntityFrameworkCore;

namespace BankData.API.Data
{
    public class DataContext : DbContext
    {
        public DataContext(DbContextOptions<DataContext> options) : base(options) { }

        public DbSet<User> Users { get; set; }
    }
}
```

5.4.2 User Model for DB

```
using System;

namespace BankData.API.Models
{
    public class User
    {
        public int Id { get; set; }
        public string AccountType { get; set; }
        public DateTime Created { get; set; }
        public DateTime LastActive { get; set; }
        public string Transactions { get; set; }
        public string Balance { get; set; }
    }
}
```

5.4.3 Σύνοψη

Ολοκληρώνοντας την υλοποίηση αυτής της εργασίας με το συγκεκριμένο API βλέπουμε ένα τρόπο επικοινωνίας μεταξύ τριών ανεξάρτητων συστημάτων εφαρμόζοντας και τους ζητούμενους κανόνες της Ευρωπαϊκής Ένωσης. Είναι σημαντικό να επισημάνουμε πως ο λόγος για τον οποίο οι τράπεζες τουλάχιστον εντός Ελλάδας δεν έχουν πλήρως εφαρμόσει την οδηγία PSD2 πιθανόν να οφείλεται στο γεγονός ότι πρόκειται για οργανισμούς με πάρα πολλούς πελάτες, συνεπώς και τεράστιο όγκο δεδομένων αλλά και ιδιαίτερα πολύπλοκων συστημάτων. Η μετάβαση στη πλήρη εφαρμογή της οδηγίας είναι έργο ιδιαίτερα δύσκολο, απαιτεί χρόνο και συνεπώς είναι αρκετά δαπανηρό.

6 Συμπεράσματα

Η χρήση προγραμματιστικών διεπαφών αποτελεί μια αρχιτεκτονική η οποία είναι και η πιο βέλτιστη από πλευράς υλοποίησης αλλά και η πιο διαδεδομένη στις διαδικτυακές εφαρμογές. Εφαρμόζει όλα τα πρότυπα της ασφάλειας μεταφοράς της πληροφορίας, είναι ιδιαίτερα αποτελεσματική καθώς είναι γρήγορη αυτή η επικοινωνία και αποτελεί ένα χαμηλού κόστους μέσο διάδοσης αυτής για την οποιαδήποτε μεγέθους εταιρείας. Αυτό πρακτικά σημαίνει πως μπορεί να βρεθεί σαν υλοποίηση σε μικρού μεγέθους επιχειρήσεις με σχετικά μικρό όγκο δεδομένων αλλά και σε υπερμεγέθεις εταιρείες με ιδιαίτερα μεγάλο όγκο δεδομένων (Big Data). Το .NET Core Framework είναι αρκετά δημοφιλές αυτή τη στιγμή καθώς εκτός του ότι παρέχεται δίχως χρέωση, μπορεί να εγκατασταθεί σε όλες τις πλατφόρμες με τα πιο δημοφιλή λογισμικά και τέλος είναι ανοιχτού κώδικα (Cross-platform, Open source).

Η Angular 5 επίσης είναι ένα framework Cross-platform το οποίο είναι ιδιαίτερα επίσης δημοφιλές τα τελευταία χρόνια για την ανάπτυξη διαδικτυακών εφαρμογών καθώς είναι πάρα πολύ γρήγορο στην επεξεργασία και στην μετάδοση της πληροφορίας. Παρέχεται και αυτό δωρεάν και είναι από την αρχή δημιουργίας οποιασδήποτε web εφαρμογής responsive. Responsive καλείται μια διαδικτυακή εφαρμογή η οποία είναι σωστά υλοποιημένη για εμφάνιση της πληροφορίας σε όλους του τύπους και μεγέθη οθονών που υπάρχουν. Αυτό πρακτικά σημαίνει πως είτε ένας χρήστης συνδέεται μέσω ενός σταθερού υπολογιστή ή φορητού, είτε συνδέεται μέσω ενός κινητού έξυπνου τηλεφώνου είτε συνδέεται μέσω μια έξυπνης τηλεόρασης, η πληροφορία που θα δει θα εμφανίζεται σωστά προς ανάγνωση σε όλες τις περιπτώσεις.

Στη συγκεκριμένη υλοποίηση όπως και στις περιπτώσεις που η ανάγκη χρήζει την αποθήκευση σχετικά μεγάλου όγκου δεδομένων με πολλούς πίνακες σχεσιακών βάσεων η Angular χρησιμοποιείται καθαρά ως frontend, δηλαδή για σχεδιασμό μόνο γραφικών περιβαλλόντων. Όμως μπορεί να αποτελέσει κατά περίπτωση και σύστημα αυτόνομο δίχως διασυνδέσεις όπως σε περιπτώσεις ενός απλού διαδικτυακού ιστότοπου.

Στην παρούσα εργασία εφαρμόστηκαν τα πιο δομικά στοιχεία του GDPR και της PSD2 καθιστώντας το εν λόγω σύστημα επαρκές για την εξυπηρέτηση χρηματοπιστωτικών συναλλαγών πάσης φύσεως και πρόταση αρχιτεκτονικής συστήματος για εφαρμογή σε οποιοδήποτε χρηματοπιστωτικό ίδρυμα. Ενσωματώνοντας και σε μελλοντική επέκταση το πρότυπο FIDO εκμεταλλεύεται όλες τις σύγχρονες μεθόδους ισχυρής ταυτοποίησης χρήστη προσφέροντας ασφάλεια σύμφωνα με τα τελευταία πρότυπα.

Συνεπώς μια τράπεζα πρόθυμη να ενσωματώσει το μοντέλο του Open Banking με αναβαθμισμένες υποδομές για την αμερόληπτη παροχή φιλικών προς το χρήστη ψηφιακών υπηρεσιών μπορεί να αποκτήσει ανταγωνιστικό πλεονέκτημα επωφελούμενη από τη συνεργασία με τους Παρόχους Υπηρεσιών Πληρωμών.

7 Βιβλιογραφία

[1] GDPR: Τι είναι; Ορισμός, Πεδίο Εφαρμογής και Παραδείγματα

https://www.niriis.gr/gdpr/gdpr-ti-einai/

[2] Τι είναι ο Γενικός Κανονισμός για την Προστασία Δεδομένων (GDPR);

https://gdpr-consult.gr/gdpr-%ce%ba%ce%b1%ce%bd%ce%bf%ce%bd%ce%b9%cf%83%ce%bc%ce%bf%cf%82-%ce%b5%ce%bb%ce%bb%ce%b1%ce%b4%ce%b1/

[3] PSD2: η αναθεωρημένη οδηγία υπηρεσίας πληρωμών και μετασχηματισμός των τραπεζών

https://home.kpmg/gr/el/home/insights/2019/12/eu-payments-services-directive-psd2.html

[4] Ευρωπαϊκή Οδηγία σχετικά με τις υπηρεσίες πληρωμών (PSD2)

https://www.bankofcyprus.com/home-gr/bank_gr/forms_gr/PSD-gr/

[5] What is GDPR, the EU's new data protection law?

https://gdpr.eu/what-is-gdpr/

[6] Everything you need to know about PSD2

https://www.bbva.com/en/everything-need-know-psd2/

[7] Build any app with .NET

https://dotnet.microsoft.com/en-us/

[8] Συνήθεις ερωτήσεις για τον κύκλο ζωής - .NET Core

https://learn.microsoft.com/el-gr/lifecycle/faq/dotnet-core

[9] Argon2

https://en.wikipedia.org/wiki/Argon2

[10] Model-view-controller

https://el.wikipedia.org/wiki/Model-view-controller

[11] JSON

https://el.wikipedia.org/wiki/JSON

[12] Introduction to Angular concepts

https://angular.io/guide/architecture

[13] What Is Architecture Overview of Angular?

https://www.code-sample.com/2018/01/angular-4-and-5-architecture-overview.html

[14] Twilio Customer Engagement Platform

https://www.twilio.com

[15] ΚΑΝΟΝΙΣΜΟΣ (ΕΕ) 2016/ 679 ΤΟΥ ΕΥΡΩΠΑΪΚΟΥ ΚΟΙΝΟΒΟΥΛΙΟΥ ΚΑΙ ΤΟΥ ΣΥΜΒΟΥΛΙΟΥ

https://eur-lex.europa.eu/legal-content/EL/TXT/PDF/?uri=CELEX:32016R0679&from=EL

[16] Νομοθετικό Πλαίσιο για την Προστασία Δεδομένων

https://www.hba.gr/info/gdpr

[17] Τι είναι ο «GDPR»– Ισχύουσα Νομοθεσία

https://www.taxheaven.gr/circulars/27607/arora-ti-einai-o-gdpr-kai-poies-oi-ypoxrewseis-twn-epixeirhsewn

[18] Polasik, Michał, et al. "The impact of Payment Services Directive 2 on the PayTech sector development in Europe." Journal of Economic Behavior & Organization 178 (2020): 385-401.

https://www.sciencedirect.com/science/article/pii/S0167268120302328

[19] Wich, Tobias, Daniel Nemmert, and Detlef Hühnlein. "Towards secure and standard-compliant implementations of the PSD2 Directive." Open Identity Summit 2017 (2017).

https://dspace.gi.de/bitstream/handle/20.500.12116/3581/proceedings-05.pdf?sequence=1&isAllowed=y

[20] Wolters, Pieter TJ, and Bart PF Jacobs. "The security of access to accounts under the PSD2." Computer law & security review 35.1 (2019): 29-41.

https://www.sciencedirect.com/science/article/abs/pii/S0267364918302620

[21] Politou, Eugenia, Efthimios Alepis, and Constantinos Patsakis. "Profiling tax and financial behaviour with big data under the GDPR." Computer law & security review 35.3 (2019): 306-329.

https://www.sciencedirect.com/science/article/abs/pii/S026736491830133X

[22] Politou, Eugenia, et al. "Backups and the right to be forgotten in the GDPR: An uneasy relationship." Computer Law & Security Review 34.6 (2018): 1247-1257. https://www.sciencedirect.com/science/article/abs/pii/S0267364918301389

[23] Politou, Eugenia, Efthimios Alepis, and Constantinos Patsakis. "Forgetting personal data and revoking consent under the GDPR: Challenges and proposed solutions." Journal of cybersecurity 4.1 (2018): tyy001. https://academic.oup.com/cybersecurity/article/4/1/tyy001/4954056

[24] Grammatopoulos, Athanasios Vasileios. FIDO2/WebAuthn implementation and analysis in terms of PSD2. MS thesis. Πανεπιστήμιο Πειραιώς, 2022. https://dione.lib.unipi.gr/xmlui/bitstream/handle/unipi/14260/GramThanos%20-%20FIDO2_WebAuthn_implementation_and_analysis_in_terms_of_PSD2.pdf?sequence=1

[25] PSD2: η αναθεωρημένη οδηγία υπηρεσίας πληρωμών και μετασχηματισμός των τραπεζών

https://home.kpmg/gr/el/home/insights/2019/12/eu-payments-services-directive-psd2.html

[26] PSD2: Τι είναι η Δεύτερη Οδηγία Υπηρεσιών Πληρωμών (Payment Services Directive). Ορισμός, Εφαρμογή και Παραδείγματα

https://www.niriis.gr/psd2/psd2-%CF%84%CE%B9-%CE%B5%CE%AF%CE%BD%CE%B1%CE%B9-%CE%B7-%CE%B4%CE%B5%CF%8D%CF%84%CE%B5%CF%81%CE%B7-%CE%BF%CE%B4%CE%B7%CE%B3%CE%AF%CE%B1-%CF%85%CF%80%CE%B7%CF%81%CE%B5%CF%83%CE%B9%CF%8E%CE%BD-%CF%80/

[27] ΟΔΗΓΊΑ (ΕΕ) 2015/2366 ΤΟΥ ΕΥΡΩΠΑΪΚΟΎ ΚΟΙΝΟΒΟΥΛΊΟΥ ΚΑΙ ΤΟΥ ΣΥΜΒΟΥΛΊΟΥ

https://eur-lex.europa.eu/legal-content/EL/TXT/?uri=celex%3A32015L2366