ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ – ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Προηγμένα Συστήματα Πληροφορικής - Ανάπτυξη Λογισμικού και Τεχνητής Νοημοσύνης»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Διαδικτυακό περιβάλλον κοινωνικής δραστηριότητας για την οργάνωση αθλητικών δραστηριοτήτων.** **A web-based social activity platform for organizing sports activities.**
Ονοματεπώνυμο Φοιτητή	**Καραχρήστος Ευάγγελος**
Πατρώνυμο	**Αθανάσιος**
Αριθμός Μητρώου	**ΜΠΣΠ2212**
Επιβλέπων	**Ευάγγελος Σακκόπουλος, Αναπληρωτής καθηγητής**

Ημερομηνία Παράδοσης **Ιούλιος 2024**

Τριμελής Εξεταστική Επιτροπή

Ευάγγελος Σακκόπουλος	Ι. Τασούλας	Διονύσιος Σωτηρόπουλος
Αναπληρωτής Καθηγητής	Επίκουρος Καθηγητής	Επίκουρος Καθηγητής

Περίληψη

Σκοπός της παρούσας εργασίας είναι η ανάπτυξη διαδικτυακού περιβάλλοντος κοινωνικής δραστηριότητας για την οργάνωση αθλητικών δραστηριοτήτων μεταξύ αγνώστων. Πρόκειται για μια εφαρμογή στην οποία θα μπορεί ο χρήστης εύκολα να επιλέξει μια δραστηριότητα που θα ήθελε να κάνει(προκαθορισμένες δραστηριότητες) σε κάποια συγκεκριμένη χρονική στιγμή και κάποιος άλλος χρήστης να τον επιλέξει ώστε να καταλήξουν να κάνουν την ίδια δραστηριότητα την ίδια χρονική στιγμή μαζί.

Το παραδοτέο αποτελείται από μια full-stack εφαρμογή, δηλαδή υλοποίηση front-end (FE), back-end (BE) και μια σχεσιακή βάση δεδομένων SQL.

Ο συνδυασμός των τριών αυτών αντικειμένων επιτυγχάνει το ζητούμενο αποτέλεσμα, επιτρέποντας στον χρήστη να έχει γραφική απεικόνιση της εφαρμογής και επικοινωνία του BE με την σχεσιακή μας βάση δεδομένων, σε συνδυασμό πάντα την ασφάλεια των δεδομένων του χρήστη.

Βασικό στοιχείο της εφαρμογής, εκτός της λογικής της που έχει αναπτυχθεί, είναι ο χρήστης να μπορεί να την αναπαράγει από οποιαδήποτε συσκευή διαθέτει, με την βέλτιστη απεικόνιση στην οθόνη του. Στην συνέχεια θα αναλύσουμε την εφαρμογή κατά το τεχνικό κομμάτι της και θα παρουσιαστεί η πλήρης ανάλυση της λειτουργίας της.

Abstract

The purpose of this paper is to develop a web-based social activity environment for organizing sports activities between strangers. It is an application in which a user can easily choose an activity he would like to do (predefined activities) at a certain time and another user can choose him so that they end up doing the same activity at the same time together.

The deliverable consists of a full-stack application i.e. front-end implementation (FE), back-end (BE) and a relational SQL database.

The combination of these three objects achieves the desired result, allowing the user to have a graphical representation of the application and communication of the BE with our relational database, always combined with the security of the user's data.

A basic element of the application besides the logic that has been developed is that the user can reproduce it from any device he has with the optimal display on his screen. In the following we will see the analysis of the application in its technical part and the complete analysis of its operation.

Πίνακας περιεχομένων

Κεφάλαιο 1 – Εισαγωγή

1.1 Βασική ιδέα

Η βασική ιδέα της κατασκευής αυτής της εφαρμογής προήλθε από τη σκέψη ότι πολλές φορές έχουμε 'ξεμείνει' χωρίς παρέα για ένα διάστημα λόγω κάποιων καταστάσεων που μπορεί να επικρατούν. Έτσι, με βασική έμπνευση από άλλες εφαρμογές που κάνουν αντίστοιχα πράγματα αλλά όχι με το συγκεκριμένο αντικείμενο, σχεδιάστηκε η εφαρμογή που θα παρουσιαστεί στη συνέχεια.

Το μοντέλο που θα αναλυθεί είναι σχεδιασμένο ώστε να μπορεί ο καθένας, από κάθε συσκευή, να το αναπαράγει και να το χρησιμοποιήσει. Με αυτόν τον τρόπο, όλοι, χωρίς περιορισμούς, μπορούν να έχουν άμεση πρόσβαση σε αυτή την εφαρμογή. Παρακάτω θα αναλυθεί τόσο η αρχιτεκτονική όσο και οι γλώσσες προγραμματισμού και τα εργαλεία αποθήκευσης δεδομένων που χρησιμοποιήθηκαν.

1.2 Επιλογή γλωσσών προγραμματισμού

1.2.1 Vue.js framework

Το Vue είναι ένα JavaScript open-source framework με βασική λογική το model-view-viewmodel (MVVM). Χρησιμοποιείται για την κατασκευή διεπαφών χρήστη και εφαρμογών. Κυκλοφόρησε για πρώτη φορά τον Φεβρουάριο του 2014. Το framework αυτό χαρακτηρίζεται από την απλότητα και την επεκτασιμότητά του ταυτόχρονα. Ακόμα, για να επιτύχουμε την αναπαραγωγή σε διαφορετικές διαστάσεις οθονών, είναι το πλέον κατάλληλο εργαλείο.

Βασικά χαρακτηριστικά

Το framework προωθεί την επαναχρησιμοποίηση του κώδικα μέσα στο project. Αυτό σημαίνει ότι κατά τη μεταγλώττιση μπορούν να χρησιμοποιηθούν τα ίδια components σε πολλαπλά σημεία της εφαρμογής. Ένα ακόμα βασικό χαρακτηριστικό είναι η δρομολόγηση μιας σελίδας. Αυτό μας επιτρέπει να έχουμε πλοήγηση σε πολλαπλές σελίδες χωρίς να γίνεται κάποια επαναφόρτωση της υπάρχουσας σελίδας. Για αυτό το feature είναι υπεύθυνο το Vue Router. Τέλος, η βιβλιοθήκη Vuex είναι μια βιβλιοθήκη που αναπτύχθηκε για τη διαχείριση και τον συγχρονισμό δεδομένων μεταξύ των components. Αναπτύχθηκε από τον Evan You, ιδρυτή του Vue.js.

1.2.2 C# Microsoft .Net 8 framework

Το .NET Framework αναπτύχθηκε από τη Microsoft. Είναι μια τεχνολογία που υποστηρίζει την ανάπτυξη λογισμικού τόσο για εφαρμογές σε λειτουργικό σύστημα Windows (Windows Applications) όσο και για εφαρμογές που βασίζονται στο διαδίκτυο (Web Based Applications). Οι εφαρμογές μπορούν να αναπτυχθούν μέσω διαφόρων IDE που παρέχονται στην αγορά, όπως το Visual Studio, το Visual Studio Code, το Rider κ.ά.

Η Microsoft, μέσω αυτού του framework, προωθεί ένα αντικειμενοστραφές περιβάλλον προγραμματισμού που μπορεί να εκτελείται τοπικά, τοπικά αλλά να διανέμεται στο διαδίκτυο, ή εξ αποστάσεως.

Παρέχετε ένα περιβάλλον εκτέλεσης του κώδικα που :

- Μπορεί να αποφύγει τις διάφορες συγκρούσεις κατά την ανάπτυξη και την έκδοση λογισμικού
- Προάγει την ασφαλή εκτέλεση του πηγαίου κώδικα, έχοντας ως γνώμονα ότι μπορεί να υπάρχει κώδικας που έχει δημιουργηθεί από τρίτους
- Δίνει στον προγραμματιστή την επιλογή να κατασκευάσει διάφορων τύπων εφαρμογές (web-based, windows-based , mobile apps)
- Κατασκευάζει όλη την επικοινωνία με βάση τα πρότυπα για να μπορεί να ενσωματωθεί με οποιονδήποτε άλλο κώδικα

Η αρχική έκδοση ονομάστηκε .NET Framework, με πρώτη κυκλοφορία στις 13 Φεβρουαρίου 2002. Έπειτα, στις 27 Ιουνίου 2016, έγινε η πρώτη κυκλοφορία του .NET Core, και μετά το .NET Core 3.1 προχωρήσαμε στην έκδοση .NET 5. Αισίως, εμείς αναπτύξαμε αυτήν την εφαρμογή στην έκδοση .NET 8, η οποία είναι και η τελευταία έκδοση του framework.

1.2.3 PostgreSQL Database

Η PostgreSQL είναι ένα open-source σύστημα διαχείρισης σχεσιακών βάσεων δεδομένων που επικεντρώνεται στην επεκτασιμότητα και στη συμβατότητα με SQL.

Η PostgreSQL έχει πολλές ιδιότητες εκτός από την αποθήκευση και την προβολή δεδομένων, όπως η διαχείριση ξένων κλειδιών και η αποθήκευση αυτοματοποιημένων διαδικασιών. Επιπλέον, μπορεί να υποστηριχθεί από πολλά λειτουργικά συστήματα (Linux, FreeBSD, macOS, Windows) και να διαχειριστεί μεγάλο αριθμό εργασιών στην ουρά προς εκτέλεση, είτε από μεμονωμένες μηχανές ως μηχανές αποθήκευσης δεδομένων, είτε από πολλαπλούς χρήστες σε υπηρεσίες ιστού.

1.3 Αρχιτεκτονική του έργου

Η αρχιτεκτονική του έργου υλοποίησης της εφαρμογής σχεδιάστηκε για την ομαλή και εύκολη λειτουργία της από τους χρήστες. Η λογική σχεδίασης βασίζεται στην απλοϊκή χρήση κατά την περιήγηση των χρηστών στις σελίδες.

Όταν ο χρήστης ανοίγει την εφαρμογή, ξεκινά με τη διαδικασία εγγραφής ή σύνδεσης. Μετά την επιτυχή εισαγωγή των σωστών στοιχείων του, ο χρήστης έχει δύο κύριες επιλογές για την οργάνωση των δραστηριοτήτων του. Πρώτον, μπορεί να επιλέξει και να εισάγει την ημερομηνία και τη δραστηριότητα που επιθυμεί να κάνει χωρίς να επιλέξει κάποιον συναθλητή. Αυτό είναι χρήσιμο για χρήστες που θέλουν απλώς να καταγράψουν τις προτιμήσεις τους και να αναζητήσουν διαθέσιμους συναθλητές αργότερα. Επίσης, ο χρήστης μπορεί να ακολουθήσει την ίδια διαδικασία εισαγωγής ημερομηνίας και δραστηριότητας, με την προσθήκη της επιλογής κάποιου συναθλητή, ολοκληρώνοντας έτσι την τελική κράτηση για μια συγκεκριμένη ημερομηνία και άτομο.

Το μοντέλο αυτό εξασφαλίζει ότι ο χρήστης μπορεί εύκολα και αποτελεσματικά να προγραμματίσει τις δραστηριότητές του και να βρει συναθλητές, δημιουργώντας μια αυτοματοποιημένη διαδικασία κρατήσεων. Όλη η διαδικασία είναι σχεδιασμένη ώστε να είναι διαισθητική και να προσφέρει μια ευχάριστη εμπειρία χρήστη.

1.4 Σχεδιαγράμματα λογικής

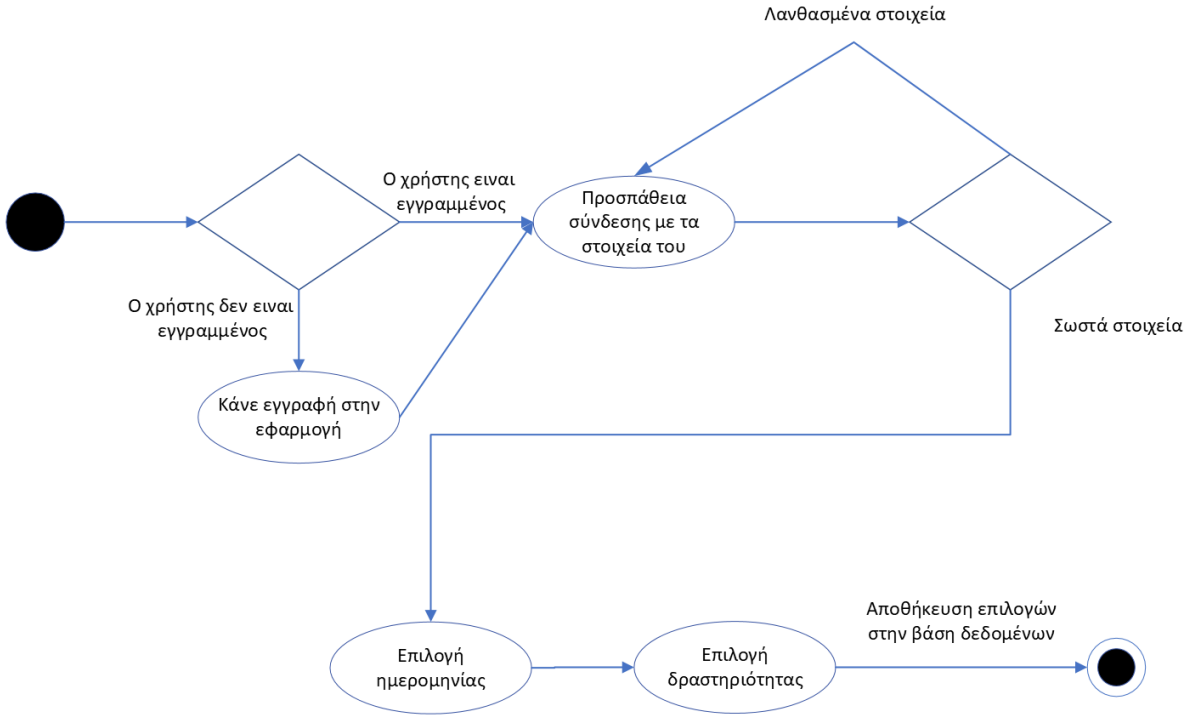
Το διάγραμμα ροής της εφαρμογής, όπως φαίνεται παρακάτω, αναπαριστά τη σχεδίαση της εφαρμογής μας βήμα-βήμα. Όπως μπορούμε να διακρίνουμε, ο χρήστης πρέπει πρώτα να εγγραφεί. Αν ο χρήστης είναι ήδη εγγεγραμμένος, τότε πρέπει να εισάγει τα στοιχεία του. Αν εισάγει λανθασμένα στοιχεία, θα πρέπει να τα διορθώσει και να εισάγει τα σωστά.

Έπειτα, ο χρήστης έχει δύο επιλογές:

1. Να επιλέξει και να εισάγει την ημερομηνία και τη δραστηριότητα χωρίς να επιλέξει κάποιον συναθλητή.
2. Να ακολουθήσει την ίδια διαδικασία με τελικό σκοπό την επιλογή κάποιου συναθλητή και την τελική κράτηση για συγκεκριμένη ημερομηνία και άτομο.

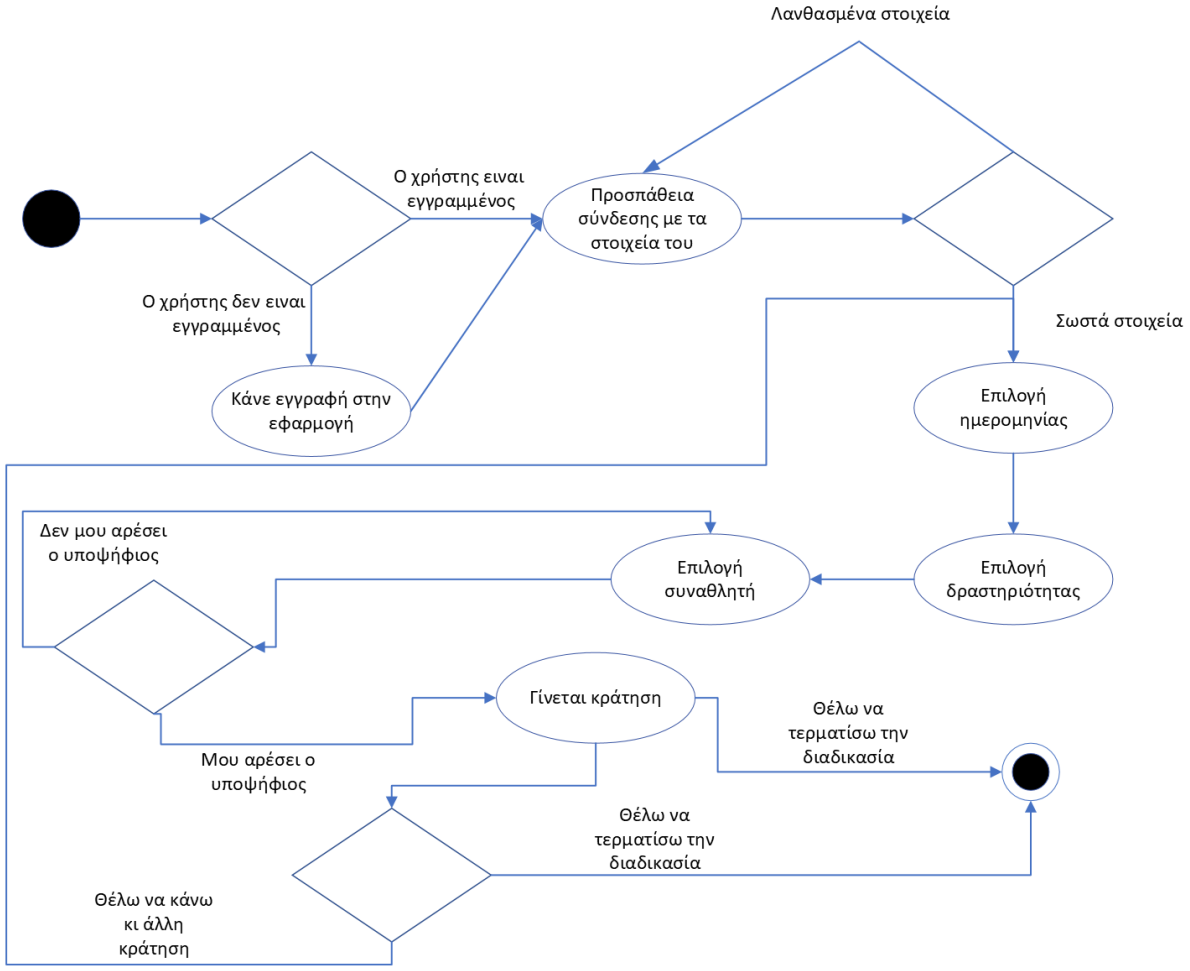
Σε κάθε περίπτωση, ο τελικός σκοπός είναι η αποθήκευση των πληροφοριών στη βάση δεδομένων της εφαρμογής.

Πρώτη περίπτωση : Ο χρήστης κάνει κράτηση ημερομηνίας, δραστηριότητας.



Εικόνα 1 Διάγραμμα ροής (1ο σενάριο χρήσης)

Δεύτερη περίπτωση : Ο χρήστης κάνει κράτηση ημερομηνίας, δραστηριότητας και συναθλητή.



Εικόνα 2 Διάγραμμα ροής (2ο σενάριο χρήσης)

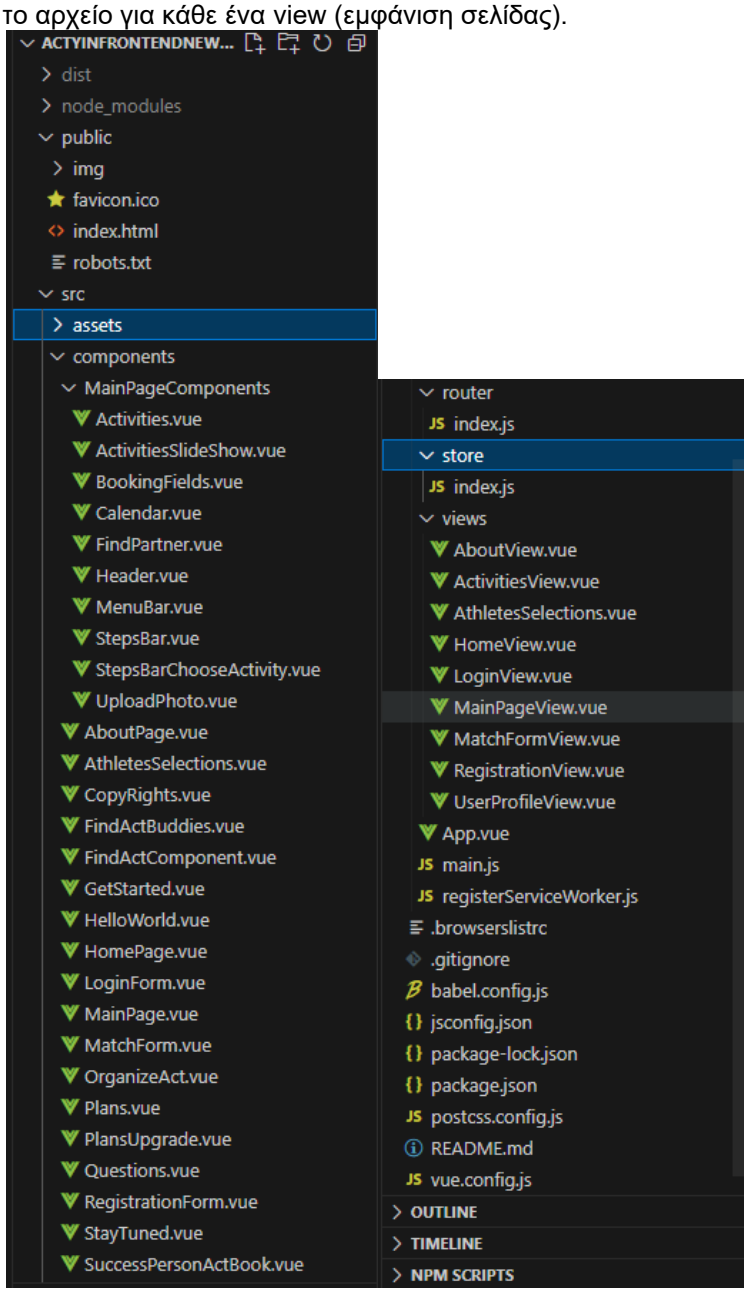
Συνοψίζοντας, σε αυτό το κεφάλαιο είδαμε τις τεχνολογίες που χρησιμοποιήθηκαν για την υλοποίηση της εφαρμογής μας, τη βασική ιδέα της εφαρμογής και πώς ο χρήστης μπορεί να περιηγηθεί και να κάνει χρήση των λειτουργιών της.

Κεφάλαιο 2 – Τεχνικά Χαρακτηριστικά

2.1 Δομή κώδικα

2.1.1 Front-End

Παρακάτω παρουσιάζονται δύο φωτογραφίες που απεικονίζουν τη δομή της υλοποίησης του Front-End κομματιού, όπως περιγράφηκε προηγουμένως. Η υλοποίηση της εφαρμογής έχει γίνει σε Vue.js. Αρχικά, δημιουργήθηκε ένας φάκελος 'components', ο οποίος περιλαμβάνει στατικά στοιχεία που χρησιμοποιήθηκαν για την υλοποίηση. Έπειτα, δημιουργήθηκαν οι σελίδες που εμφανίζονται, και τέλος υλοποιήθηκε το αρχείο για κάθε ένα view (εμφάνιση σελίδας).



Εικόνα 3 Δομή Front-end

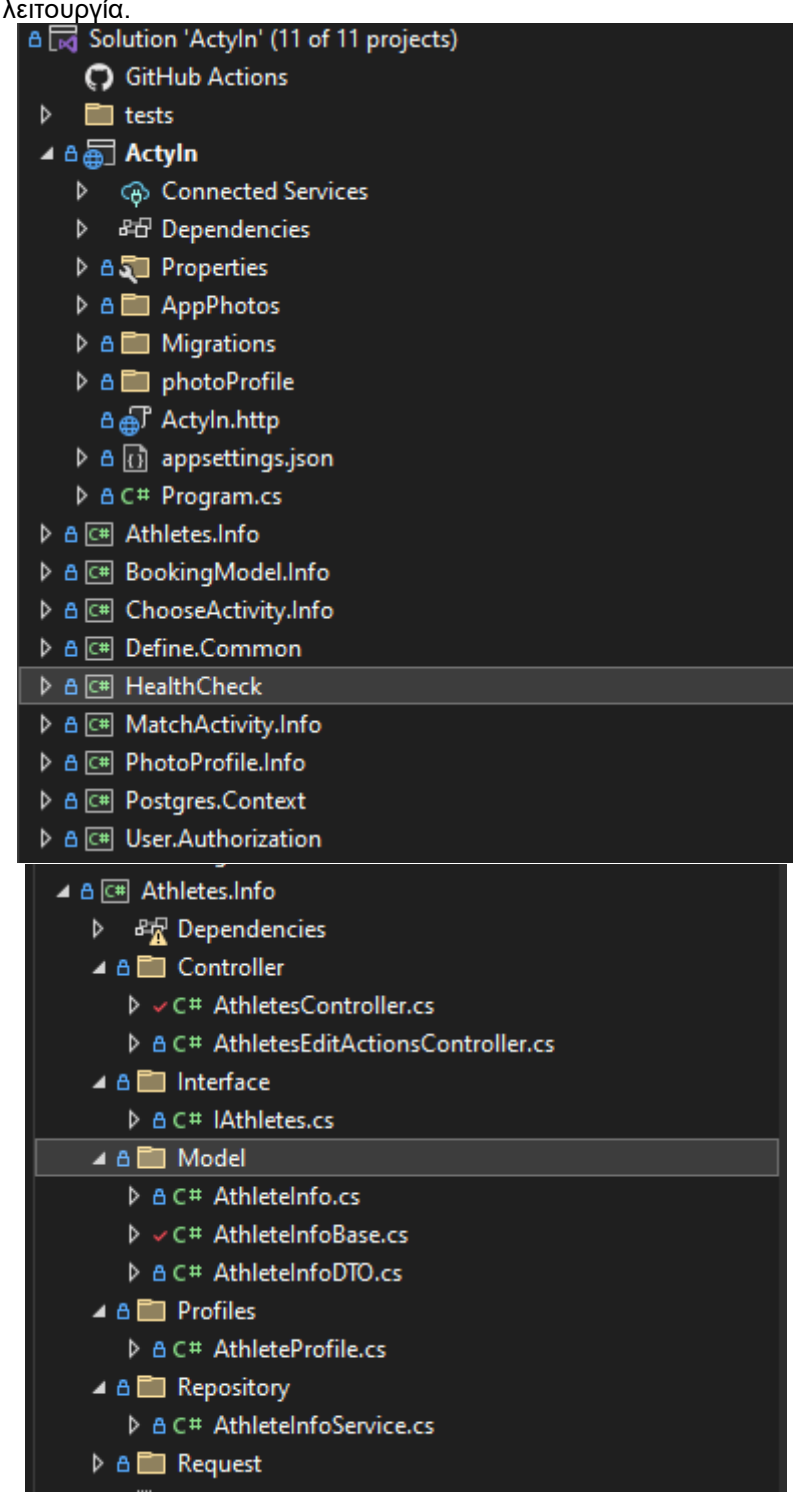
Ακόμα είναι πολύ σημαντικό να σχολιάσουμε το αρχείο στον φάκελο router, όπου μας δίνει τη δυνατότητα να αρχικοποιήσουμε τις σελίδες που έχουν υλοποιηθεί σε συγκεκριμένους συνδέσμους.

```
import { createRouter, createWebHashHistory } from 'vue-router'
import HomeView from '../views/HomeView.vue'
import LoginView from '../views/LoginView.vue'
import RegistrationView from '../views/RegistrationView.vue'
import UserProfileView from '../views/UserProfileView.vue'
import MainPageView from '../views/MainPageView.vue'
import AthletesSelectionsView from '../views/AthletesSelections.vue'
const routes = [
  {
    path: '/',
    name: 'home',
    component: HomeView
  },
  {
    path: '/about',
    name: 'about',
    component: () => import('../views/AboutView.vue')
  },
  {
    path: '/login',
    name: 'login',
    component: LoginView
  },
```

Εικόνα 4 Router file

2.1.2 Back-End

Η υλοποίηση του Back-End έχει πραγματοποιηθεί σε .Net 8, όπως αναφέρεται παραπάνω, και όπως βλέπουμε η συνολική υλοποίηση έχει χωριστεί σε πολλά class libraries, με την κάθε βιβλιοθήκη να έχει την δική της λειτουργία.



Εικόνα 5 Δομή Back-End

2.1.3 Database

2.2 Ασφάλεια εφαρμογής

2.2.1 Επιλογή εργαλείου ασφάλειας

Κατά την ανάπτυξη μιας εφαρμογής, είναι συχνά απαραίτητο να ασφαλίσουμε τα API που έχουμε δημιουργήσει, έτσι ώστε μόνο πιστοποιημένοι χρήστες να έχουν πρόσβαση σε αυτά. Ο δημοφιλέστερος τρόπος για να προσθέσουμε ασφάλεια στην εφαρμογή μας είναι με τη χρήση του JSON Web Token (JWT).

2.2.2 Τι είναι το JSON Web Token;

Το JSON Web Token (JWT) είναι ένα βιομηχανικό πρότυπο για την ασφαλή μετάδοση δεδομένων μεταξύ συστημάτων σε μορφή JSON. Επιλέχθηκε αυτή η μέθοδος λόγω της συνοπτικότητας και του μικρού μεγέθους του. Επιπλέον, είναι πολύ ασφαλές να μοιράζεστε ένα μυστικό κλειδί μεταξύ του παραλήπτη (χρήστη) και του εκδότη (εφαρμογής). Σχεδόν όλες οι γλώσσες προγραμματισμού έχουν αναπτύξει κάποια μέθοδο για την ανάγνωση JSON δεδομένων, επομένως είναι πολύ εύκολο στη χρήση τους αυτά τα κλειδιά.

Τα JSON web tokens αποτελούνται από τρία μέρη:

- Header (Τύπος, αλγόριθμος κρυπτογράφησης)
- Payload (Στοιχεία που κρυπτογραφούνται)
- Signature (Σύνθεση header, payload και ενός κλειδιού)

2.2.3 Εγκατάσταση και διαμόρφωση JWT Authentication & Authorization

Σε αυτό το βήμα θα δούμε τη διαδικασία εγκατάστασης και χρήσης του JWT τόσο στο Back-End όσο και πώς να το χρησιμοποιήσουμε από το Front-End.

Ξεκινάμε με το Back-End. Αρχικά, πρέπει να εγκαταστήσουμε το NuGet πακέτο Microsoft.AspNetCore.Authentication.JwtBearer. Αυτό το πακέτο παρέχει όλα τα απαραίτητα εργαλεία για να χρησιμοποιήσουμε το JWT authentication στην εφαρμογή μας.

Έπειτα, πρέπει να τροποποιήσουμε το αρχείο Program.cs προσθέτοντας τη διαμόρφωση για το Jwt_authentication, όπως αναφέρεται παρακάτω.

```
// Security Config
builder.Services.AddAuthentication(x =>
    {
        x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
        x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
        x.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
    })
    .AddJwtBearer(options =>
    {
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuerSigningKey = true,
            ValidateAudience = false,
            ValidateIssuer = false,
            ValidateLifetime = true,
            IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(config["AppSettings:SecretForKey"]))
        };
    });
```

Εικόνα 6 JWT authentication

Ο παραπάνω κώδικας παρέχει όλες τις απαραίτητες πληροφορίες για τη διαμόρφωση του JWT authentication κατά την εκτέλεση του API μας. Όπως βλέπουμε στη διαμόρφωση του Bearer, η ιδιότητα TokenValidationParameters χρησιμοποιείται για τη δημιουργία του.

Μπορούμε να προσθέσουμε και άλλα στοιχεία προτού παράγουμε το τελικό μας token. Για παράδειγμα, το κλειδί του εκδότη είναι αποθηκευμένο στο αρχείο applicationSettings.json το οποίο βρίσκεται στο solution.

```
"AppSettings": {
  "SecretForKey": "My nice secret key is very secret",
  "Issuer": "Karachristos",
  "Audience": "The world"
},
```

Εικόνα 7 Κλειδί JWT Token

Για να γίνει ενεργοποίηση αυτής την διαμόρφωσης τότε πρέπει να πάμε πάλι στο Program.cs και να προσθέσουμε ακόμη δυο γραμμές κώδικα όπως φαίνεται παρακάτω.

```
144
145     app.UseAuthentication();
146
147     app.UseAuthorization();
148
149     app.MapControllers();
150
151     app.UseCors();
```

Εικόνα 8 Ενεργοποίηση Authorization/Authentication

Με αυτόν τον τρόπο, μπορούμε να απαιτούμε πιστοποίηση κλειδιών από τους χρήστες για τη χρήση της εφαρμογής μας. Αρκεί να προσθέσουμε το αντίστοιχο attribute σε κάθε endpoint για το οποίο δεν επιθυμούμε τη χρήση από το μη πιστοποιημένο κοινό, όπως θα δούμε παρακάτω. Έτσι, δεν επιτρέπεται σε οποιονδήποτε να δει όλες τις κρατήσεις όλων των χρηστών.

```
[Authorize]
[ProducesResponseType(typeof(BookingEntity), StatusCodes.Status200OK)]
[HttpGet(ActionNames.GetAllBookings)]
0 references | Evangelos Karachristos, 137 days ago | 1 author, 3 changes
public async Task<ActionResult<IEnumerable<BookingEntity>>> GetAllBookingsAsync()
{
    var bookings = await _bookingInfo.GetAllBookingsAsync();
    if (bookings == null)
    {
        _logger.LogInformation(BookingServiceMessages.EmptyBookingsList);
        return NoContent();
    }
    return Ok(bookings);
}
```

Εικόνα 9 Authorization Attribute

Αντίθετα από τα endpoint που χρησιμοποιούνται για ενέργειες όπως η εμφάνιση κρατήσεων, στα οποία απαιτείται πιστοποίηση, ορισμένα endpoints δεν απαιτούν ταυτοποίηση. Για παράδειγμα, όταν ένας χρήστης επιθυμεί να εγγραφεί ή να συνδεθεί στην εφαρμογή, δεν χρειάζεται προηγούμενη ταυτοποίηση, καθώς αυτή η διαδικασία ξεκινά την παραγωγή ενός μοναδικού κλειδιού ταυτοποίησης (JWT). Στην πρώτη εικόνα παρατηρούμε το endpoint που χρησιμοποιείται κατά τη διαδικασία εγγραφής χρήστη, ενώ αμέσως μετά απεικονίζεται η διαδικασία δημιουργίας ενός JSON Web Token.

```
[HttpPost(ActionNames.RegisterUser)]
[ProducesResponseType(typeof(TokenForRegister), StatusCodes.Status200OK)]
0 references | Evangelos Karachristos, 156 days ago | 1 author, 2 changes
public ActionResult<TokenForRegister> RegisterUser([FromBody] AthleteRegisterRequest registerRequest)
{
    var register = _mapper.Map<AthletesEntity>(registerRequest);
    var entity = _athletesInfo.RegisterAthlete(register);

    if (entity is BadRequestObjectResult badRequest)
    {
        return BadRequest(new { Error = badRequest.Value });
    }

    // Generate JWT token for the registered athlete
    var token = _authorization.GenerateTokenForRegister(register);

    return Ok(new { Token = token });
}
```

Εικόνα 10 Register endpoint

```
2 references | - changes | -authors, -changes
public TokenForRegister GenerateTokenForRegister(AthletesEntity registeredAthlete)
{
    var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_configuration["AppSettings:SecretForKey"]));
    var credentials = new SigningCredentials(securityKey, SecurityAlgorithms.HmacSha256);

    var claims = new List<Claim>
{
    new Claim(ClaimTypes.Name, registeredAthlete.Username),
    new Claim(ClaimTypes.Email, registeredAthlete.Email),
    // Add more claims as needed
};

    var token = new JwtSecurityToken(
        issuer: _configuration["AppSettings:Issuer"],
        audience: _configuration["AppSettings:Audience"],
        claims: claims,
        expires: DateTime.UtcNow.AddMinutes(90),
        signingCredentials: credentials
    );
    var newEntity = ToRegisterAthlete(registeredAthlete);
    newEntity.Token = new JwtSecurityTokenHandler().WriteToken(token);

    return newEntity;
}
```

Εικόνα 11 Δημιουργία Json Web Token

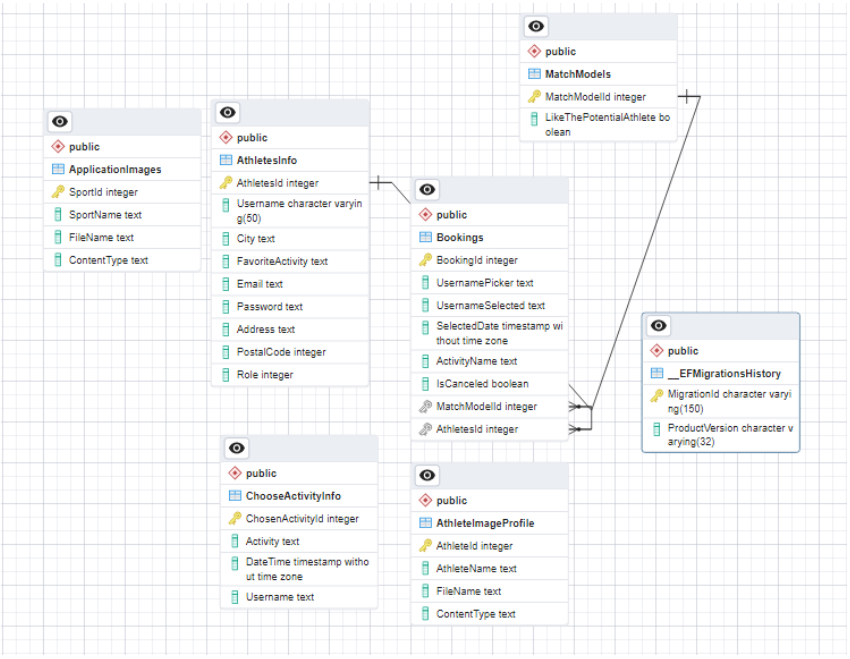
Μετά την παραγωγή του κλειδιού ταυτοποίησης, αυτό επιστρέφεται στο Front-End, όπου μπορεί να χρησιμοποιηθεί σε κλήσεις που απαιτούν authentication και authorization για την εκτέλεσή τους.

```
// Set the token in Axios defaults for future requests
this.$axios.defaults.headers.common['Authorization'] = `Bearer ${accessToken.token}`;
```

Εικόνα 12 Διαμόρφωση JWT για μελλοντική χρήση

2.3 Τεχνική ανάλυση τρόπου αποθήκευσης δεδομένων

Όπως αναφέρθηκε παραπάνω, για την αποθήκευση των δεδομένων της εφαρμογής μας χρησιμοποιήσαμε την PostgreSQL. Η PostgreSQL είναι ένα ισχυρό, ανοικτού κώδικα σύστημα διαχείρισης σχεσιακών βάσεων δεδομένων που επικεντρώνεται στην επεκτασιμότητα και τη συμβατότητα με το πρότυπο SQL. Η επιλογή της PostgreSQL έγινε λόγω της αξιοπιστίας, της ευελιξίας και των πλούσιων χαρακτηριστικών της.



Εικόνα 13 ERD διάγραμμα σχέσεων της βάσης δεδομένων.

Η βάση δεδομένων έχει δημιουργηθεί με σκοπό να επεκταθεί σε μια πραγματικού χρόνου βάση δεδομένων στο μέλλον. Αυτό σημαίνει ότι η υπάρχουσα υποδομή έχει σχεδιαστεί και υλοποιηθεί με τη δυνατότητα να υποστηρίξει αυξημένη απόδοση και δυνατότητες πραγματικού χρόνου.

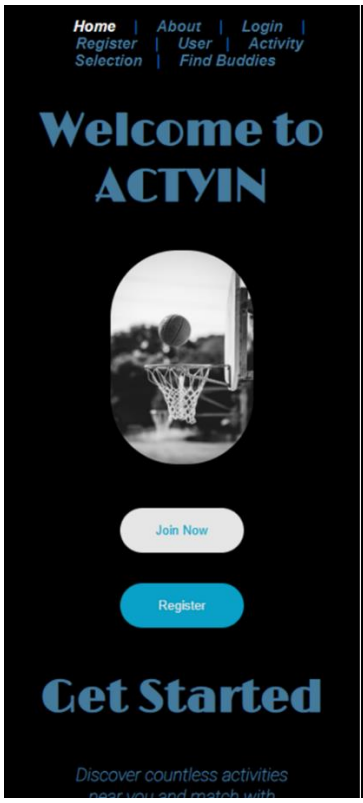
2.4 Αναλύσεις οθόνης

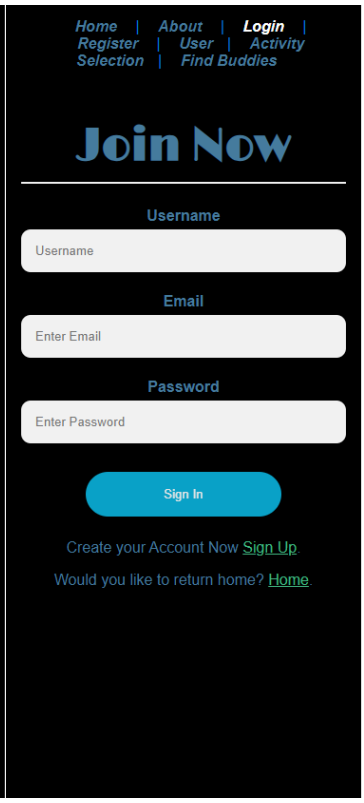
Η εφαρμογή έχει την ιδιότητα να προσαρμόζεται σε διαφορετικές αναλύσεις οθόνης. Αυτό μας δίνει την δυνατότητα ο χρήστης να μπορεί να χειριστεί από διαφορετικές συσκευές την εφαρμογή χωρίς να έχει κάποιο ιδιαίτερο πρόβλημα.

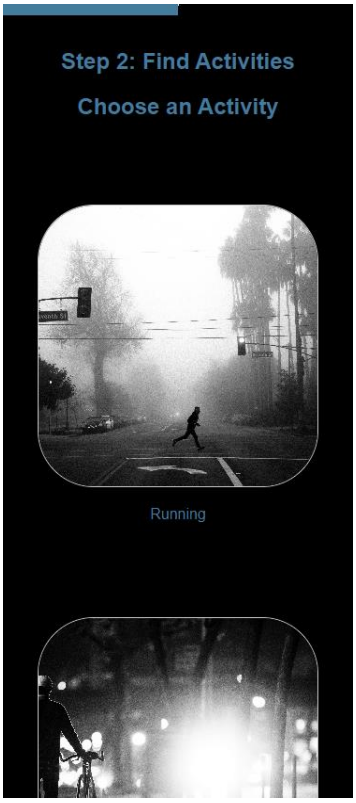
Παρακάτω θα δούμε πώς φαίνεται η εφαρμογή μας σε διαφορετικές οθόνες, παρουσιάζοντας ενδεικτικά μερικές σελίδες της. Η εφαρμογή έχει σχεδιαστεί να λειτουργεί κυρίως στον ιστό, με κύρια έμφαση στη χρήση από κινητές συσκευές.

2.4.1 Κινητά τηλέφωνα

Η πρώτη ανάλυση που θα δούμε είναι αυτή του κινητού τηλεφώνου στα 390x844 pixel.
Οι σελίδες είναι το Home Page όπου ο χρήστης μπορεί να κάνει μια μικρή ξενάγηση του τι πρόκειται να ακολουθήσει κατά την χρήση της εφαρμογής, το Login Page όπου ο χρήστης βάζει τα στοιχεία του για να μπορέσει να κάνει είσοδο στην εφαρμογή και τέλος βλέπουμε την σελίδα όπου ξεκινάει η διαδικασία επιλογής δραστηριότητας.







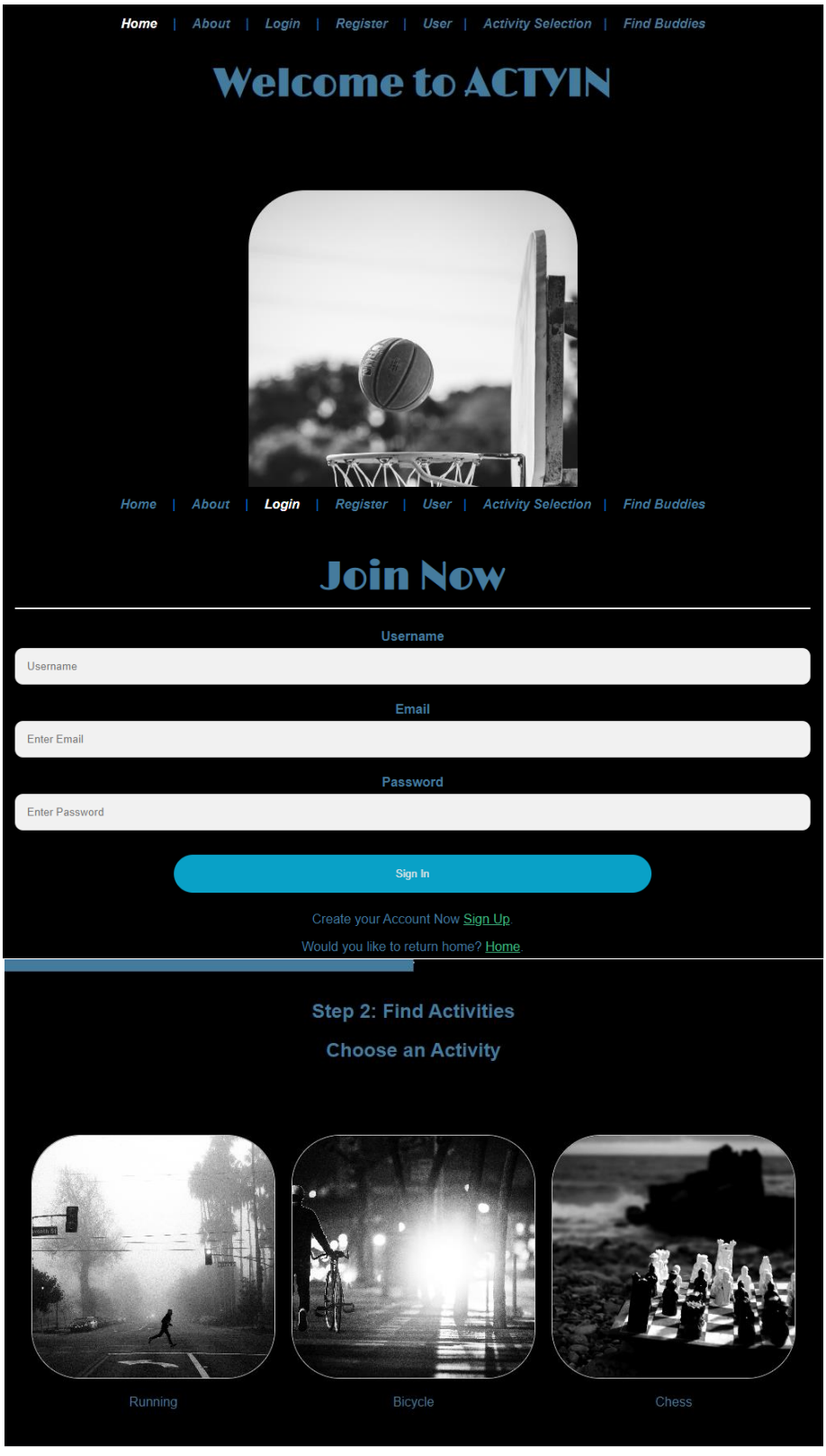
Εικόνα 14 Home Page

Εικόνα 15 Login Page

Εικόνα 16 Find Activities Page

2.4.2 Tablet

Η δεύτερη ανάλυση που θα δούμε είναι αυτή του στα 1024x600 pixel. Ακριβώς όπως και παραπάνω βλέπουμε τις ιδίες σελίδες και σε tablet. Αυτήν την φορά προσαρμοσμένες στην ανάλυση της οθόνης που χρησιμοποιείται.



2.4.3 Σταθερός υπολογιστής/ Φορητός υπολογιστής

Η τρίτη και τελευταία ανάλυση που θα δούμε είναι αυτή του στα 1920x1080 pixel. Ακριβώς όπως και παραπάνω βλέπουμε τις ιδίες σελίδες και σε υπολογιστή είτε σταθερό είτε φορητό. Αυτήν την φορά προσαρμοσμένες στην ανάλυση της οθόνης που χρησιμοποιείται.

2.5 Έλεγχος υγείας εφαρμογής

Το ASP.NET Core προσφέρει Health Checks Middleware και βιβλιοθήκες για την αναφορά της υγείας των στοιχείων υποδομής εφαρμογών.

Οι έλεγχοι υγείας εκτίθενται από μια εφαρμογή ως endpoint στην δική μας περίπτωση (“https:localhost:7254/health”).
Αυτά τα endpoints μπορούν να διαμορφωθούν για διάφορα σενάρια παρακολούθησης σε πραγματικό χρόνο:

- Η χρήση της μνήμης, του δίσκου και άλλων φυσικών πόρων του διακομιστή μπορεί να παρακολουθείται για την υγιή κατάσταση της εφαρμογής
- Οι έλεγχοι αυτοί μπορούν να ελέγξουν τις εξαρτήσεις μιας εφαρμογής, όπως οι βάσεις δεδομένων και τα τελικά σημεία εξωτερικών υπηρεσιών, για να επιβεβαιώσουμε τη διαθεσιμότητα και την κανονική λειτουργία της εφαρμογής

Οι έλεγχοι υγείας χρησιμοποιούνται συνήθως με μια εξωτερική υπηρεσία παρακολούθησης για τον έλεγχο της κατάστασης μιας εφαρμογής. Πριν την υλοποίηση του ελέγχου υγείας της εφαρμογής μας, αποφασίζουμε ποιο σύστημα παρακολούθησης θα χρησιμοποιήσουμε. Το σύστημα παρακολούθησης υπαγορεύει ποιοι τύποι ελέγχων υγείας πρέπει να δημιουργηθούν και πώς να διαμορφωθούν τα endpoints.

Η αρχικοποίηση του Health Check στην υλοποίηση μας γίνεται με την παρακάτω γραμμή κώδικα στο αρχείο Program.cs.

```
// Configure health checks
builder.Services.AddHealthChecks().AddCheck<HealthChecks>("CustomCheck");
```

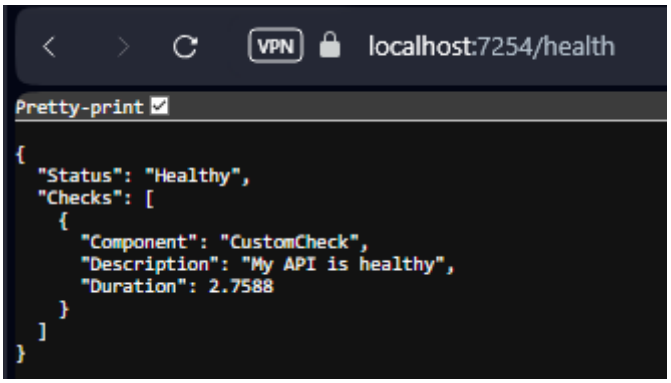
Κατά την αρχικοποίηση αφού γίνει build η εφαρμογή μας πρέπει να διαμορφώσουμε τα στοιχεία τα οποία θέλουμε το endpoint που θα χρησιμοποιηθεί να μας αναφέρει.

```
app.MapHealthChecks("/health", new HealthCheckOptions
{
    ResultStatusCodes =
    {
        [HealthStatus.Healthy] = StatusCodes.Status200OK,
        [HealthStatus.Degraded] = StatusCodes.Status200OK,
        [HealthStatus.Unhealthy] = StatusCodes.Status503ServiceUnavailable,
    },
    ResponseWriter = async (context, report) =>
    {
        var result = new
        {
            Status = report.Status.ToString(),
            Checks = report.Entries.Select(entry => new
            {
                Component = entry.Key,
                Description = entry.Value.Description,
                Duration = entry.Value.Duration.TotalMilliseconds
            }),
        };

        context.Response.ContentType = "application/json";
        await context.Response.WriteAsync(JsonConvert.SerializeObject(result));
    }

});
```

Έπειτα έχουμε φτιάξει ένα αρχείο που γίνεται αρχικοποίηση του περιεχομένου που θέλουμε να εμφανίζουμε κατά την επιτυχή ή αποτυχημένη εκτέλεση του endpoint.

```
public async Task<HealthCheckResult> CheckHealthAsync(HealthCheckContext context, CancellationToken cancellationToken = default)
{
    string descriptionHealthy = "My API is healthy";
    string descriptionNotHealthy = "My API is not healthy";
    try
    {
        return HealthCheckResult.Healthy(descriptionHealthy);

    }
    catch (Exception ex)
    {
        return HealthCheckResult.Unhealthy(descriptionNotHealthy);
    }
}
```

Τέλος, η διαδικασία ανάκτησης της πληροφορίας αυτής γίνεται όπως είπαμε από την σελίδα στο σύνδεσμο https:localhost:7254/health το endpoint όπως παρακάτω παρατηρούμε τα αποτελέσματα για την υγειά της εφαρμογής μας στο back-end.



2.6 Unit Testing

Το unit testing είναι ένας τρόπος επιβεβαίωσης της σωστής λειτουργίας μικρών μονάδων κώδικα του λογισμικού που έχει αναπτυχθεί. Με αυτόν τον τρόπο μπορούμε να δημιουργήσουμε σενάρια που καλύπτουν συγκεκριμένες λειτουργίες και μονάδες του πηγαίου κώδικα.

Παρακάτω θα δούμε μια ανάλυση ενός στοιχείου, όπου αναπτύξαμε ένα σενάριο για την καλή λειτουργία των λειτουργιών αποθήκευσης και ανάκτησης των πεδίων του. Για να μπορέσουμε να αποκτήσουμε πρόσβαση σε όλα τα απαραίτητα στοιχεία, είναι αναγκαίο να δημιουργήσουμε δεδομένα δοκιμής που αντιστοιχούν στα πραγματικά δεδομένα της εφαρμογής. Όπως φαίνεται παρακάτω, τα δεδομένα που συνθέτουμε είναι συναφή με αυτά που χρησιμοποιεί η εφαρμογή κατά την εκτέλεση των λειτουργιών της.

```
0 references | Evangelos Karachristos, 5 minutes ago | 1 author, 7 changes
public class AthletesControllerTests
{
    AthletesController _controller;
    IAthletes _athletesInfo;
    ILogger<AthletesController> _logger;
    IMapper _mapper;
    AthleteInfoService _athleteInfoService;
    NpgsqlContext _context;

    [SetUp]
    0 references | Evangelos Karachristos, 7 minutes ago | 1 author, 4 changes
    public void Setup()
    {
        _athletesInfo = A.Fake<IAthletes>();
        _logger = A.Fake<ILogger<AthletesController>>();
        _mapper = A.Fake<IMapper>();

        var options = new DbContextOptionsBuilder<NpgsqlContext>()
                          .UseInMemoryDatabase(databaseName: "TestDatabase")
                          .Options;
        _context = new NpgsqlContext(options);

        _athleteInfoService = new AthleteInfoService(_context);
        _controller = new AthletesController(_athletesInfo, _logger, _mapper, _athleteInfoService);
    }
```

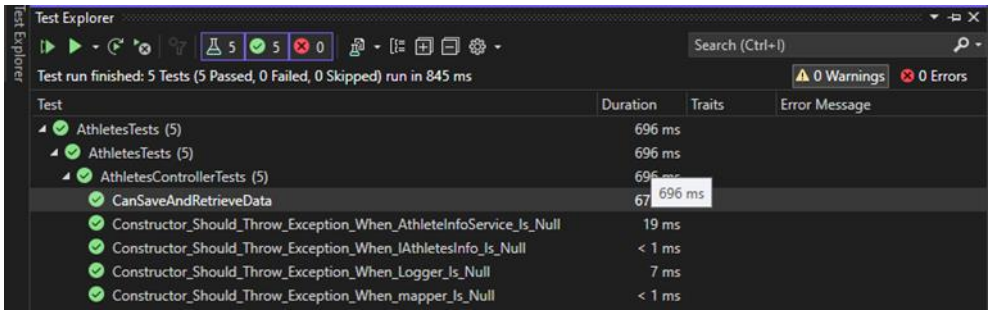
Εικόνα 17 Αρχικοποίηση στοιχείων για την δοκιμή των λειτουργιών της εφαρμογής

Έχει αναπτυχθεί ένα σενάριο που αντιπροσωπεύει την εκτέλεση μιας λειτουργίας στην εφαρμογή μας. Συγκεκριμένα, ένας χρήστης εγγράφεται στην εφαρμογή, και τα στοιχεία του αποθηκεύονται στη βάση δεδομένων. Όπως μπορούμε να δούμε, γίνεται αρχικοποίηση και αποθήκευση των στοιχείων ενός τυχαίου χρήστη. Ακολούθως, τα δεδομένα απορροφώνται από τη βάση δεδομένων, και μπορούμε να συγκρίνουμε τα αποθηκευμένα δεδομένα με αυτά που έχουν καταχωρηθεί.

```
[Test]
0 references | 0 changes | 0 authors, 0 changes
public void SaveAndRetrieve_Data_From_Database()
{
    // Arrange
    var athlete = new AthletesEntity
    {
        Username = "user1",
        Email = "user1@example.com",
        Password = "password1",
        Address = "123 Main St",
        City = "City1",
        PostalCode = 12345,
        FavoriteActivity = "Running",
        Role = Roles.User
    };

    // Act
    var dbContext = _context;
    dbContext.AthletesInfo.Add(athlete);
    dbContext.SaveChanges();

    // Assert
    var retrievedAthlete = dbContext.AthletesInfo.FirstOrDefault(a => a.AthletesId == 1);
    retrievedAthlete?.Username.Should().Be(athlete.Username);
    retrievedAthlete?.Email.Should().Be(athlete.Email);
    retrievedAthlete?.Address.Should().Be(athlete.Address);
    retrievedAthlete?.PostalCode.Should().Be(athlete.PostalCode);
    retrievedAthlete?.FavoriteActivity.Should().Be(athlete.FavoriteActivity);
    retrievedAthlete.Should().NotBeNull();
}
```

Εικόνα 18 Έλεγχος καλής λειτουργίας στοιχείου της εφαρμογής

Επιπλέον, έχουν δημιουργηθεί δοκιμές για τους constructors οι οποίοι αρχικοποιούν τα στοιχεία της εφαρμογής. Με αυτόν τον τρόπο μπορούμε να ελέγξουμε εάν όλοι οι constructors παρέχουν όλα τα απαραίτητα στοιχεία για τη δημιουργία και τη λειτουργία όλων των λειτουργιών.



Εικόνα 19 Εκτέλεση όλων των δοκίμων των λειτουργιών

2.7 DevOps

Το DevOps είναι ένα σύνολο καλών πρακτικών και εργαλείων που μας βοηθούν στην αυτοματοποίηση κάποιον διαδικασιών μεταξύ των ομάδων ανάπτυξης λογισμικού. Για την ασφάλεια της εφαρμογής κατά την διαδικασία της ανάπτυξης και της συνεχής παρακολούθησής της, αναπτύχθηκαν και παραμετροποιήθηκαν στοιχεία γι' αυτόν τον σκοπό.

Τα εργαλεία που χρησιμοποιήθηκαν για να την επιτυχή εκτέλεση και ανάπτυξη του DevOps ελέγχου είναι :

- Docker
- Git
- GitHub
- Jenkins
- SonarQube
- Ngrok
- LocalTunnel

2.7.1 DevOps Pipeline

Ένα DevOps Pipeline είναι ένα σύνολο αυτοματοποιημένων διαδικασιών και εργαλείων που επιτρέπουν στους προγραμματιστές και στους εργαζομένους στο τμήμα των λειτουργιών (operations) να συνεργάζονται για τη συνεχή κατασκευή και ανάπτυξη κώδικα σε ένα περιβάλλον παραγωγής. Οι έλεγχοι που γίνονται στα pipelines κατά την ανάπτυξή τους διαφέρουν ανάλογα με τις απαιτήσεις της κάθε εφαρμογής και του κάθε οργανισμού. Επίσης, είναι δυνατόν να προστεθούν χειροκίνητες εγκρίσεις κατά την εκτέλεσή τους για να προσαρμόσουν τον ανθρώπινο παράγοντα στην ολοκλήρωση ή τη συνέχιση των διαδικασιών.

Όπως αναφέρθηκε, ο σχεδιασμός και η υλοποίηση κάθε pipeline είναι μοναδικοί και εξαρτώνται από τις ανάγκες του κάθε οργανισμού, την εμπειρία του DevOps μηχανικού, τις χρησιμοποιούμενες τεχνολογίες και πολλούς άλλους παράγοντες. Ο μηχανικός που υλοποιεί το pipeline θα πρέπει να έχει γνώσεις ανάπτυξης και λειτουργιών, καθώς και διαχείρισης υποδομών και εργαλείων DevOps, καθώς η πολυπλοκότητα συνδέεται άμεσα με την τεχνολογική στοίβα του κάθε οργανισμού.

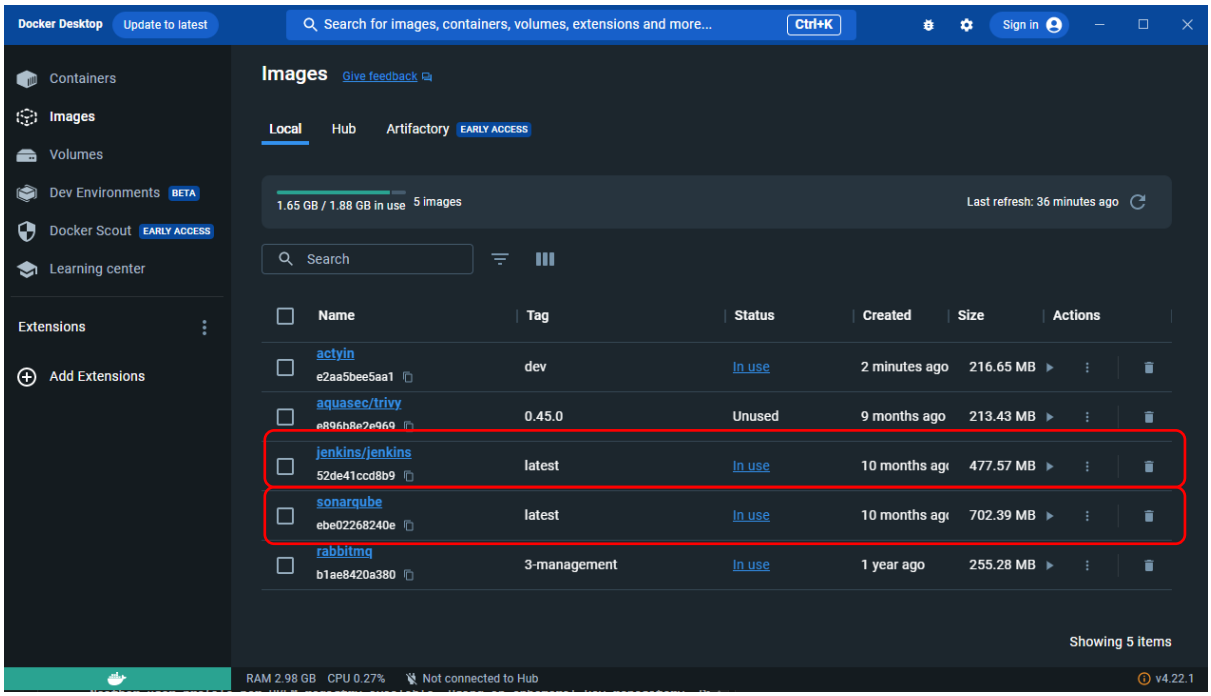
Ακόμα και αν η μοναδικότητα κάθε pipeline είναι αυτονόητη, έχουν οριστεί ορισμένοι θεμελιώδεις κανόνες. Κάθε βήμα ενός pipeline αξιολογείται βάσει της επιτυχούς εκτέλεσής του. Σε περίπτωση αποτυχίας, επανεξετάζεται και αξιολογείται από τον υπεύθυνο μηχανικό.

2.7.2 Docker

Το Docker είναι μια ανοικτή πλατφόρμα που μας βοηθάει στην ανάπτυξη, την αποστολή και την εκτέλεση εφαρμογών. Επίσης επιτρέπει να διαχωριστούν οι εφαρμογές από την υποδομή, έτσι ώστε να μπορεί να παραδίδετε το λογισμικό γρήγορα. Το Docker, μπορεί να διαχειριστεί την υποδομή με τους ίδιους τρόπους που διαχειρίζονται οι εφαρμογές. Αξιοποιώντας τις μεθοδολογίες του Docker για την αποστολή, τον έλεγχο και την ανακατασκευή κώδικα, μπορεί να μειωθεί σημαντικά η καθυστέρηση μεταξύ της συγγραφής του κώδικα και της εκτέλεσής του στην παραγωγή.

Παρακάτω βλέπουμε το περιβάλλον του Docker Desktop και τις εφαρμογές που έχουμε προσθέσει στο περιβάλλον αυτό:

- Jenkins
- SonarQube



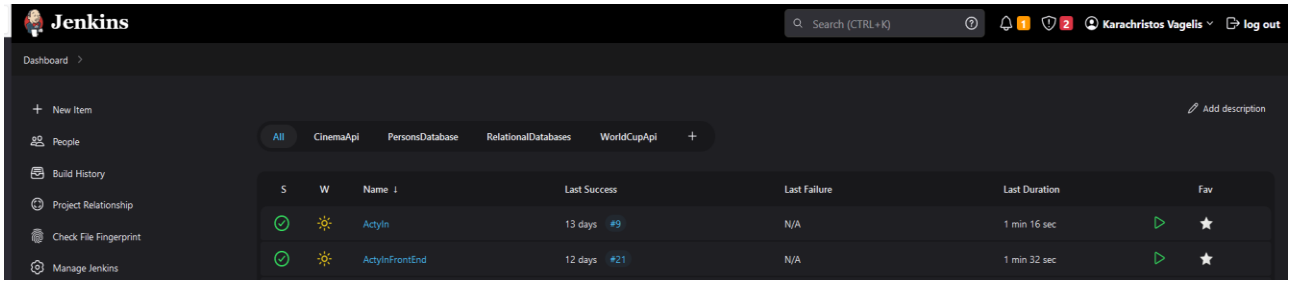
Εικόνα 20 Περιβάλλον Docker Desktop

2.7.3 Jenkins

Το Jenkins είναι ένας ανοικτού κώδικα server αυτοματισμού. Βοηθά στον αυτοματισμό τμημάτων της ανάπτυξης λογισμικού που σχετίζονται με τη δημιουργία, τον έλεγχο και την αναπτυξιακή διαδικασία, διευκολύνοντας την συνεχή ολοκλήρωση και τη συνεχή παράδοση.

Είναι ένα σύστημα που βασίζεται σε διακομιστές και λειτουργεί σε servlet containers όπως το Apache Tomcat. Υποστηρίζει εργαλεία ελέγχου εκδόσεων, συμπεριλαμβανομένων των AccuRev, CVS, Subversion, Git, Mercurial, Perforce, ClearCase και RTC, και μπορεί να εκτελεί έργα βασισμένα σε Apache Ant, Apache Maven, και sbt, καθώς και αυθαίρετα shell scripts και εντολές επιτρόπου Windows.

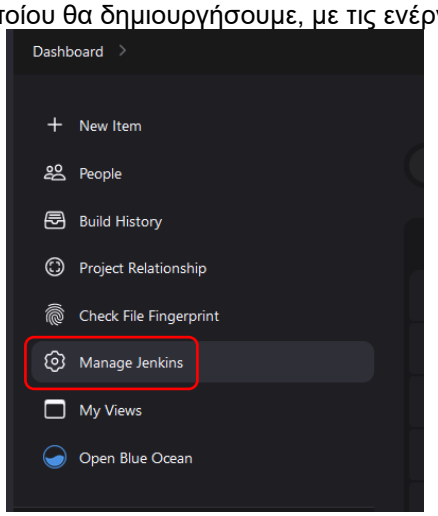
Το περιβάλλον του Jenkins είναι πολύ κατανοητό στην χρήση όσο και στην παραμετροποίηση των στοιχείων, των οποίων θέλουμε χρησιμοποιήσουμε.



Εικόνα 21 Περιβάλλον Jenkins

2.7.4 Αρχικοποίηση απαραίτητων στοιχείων

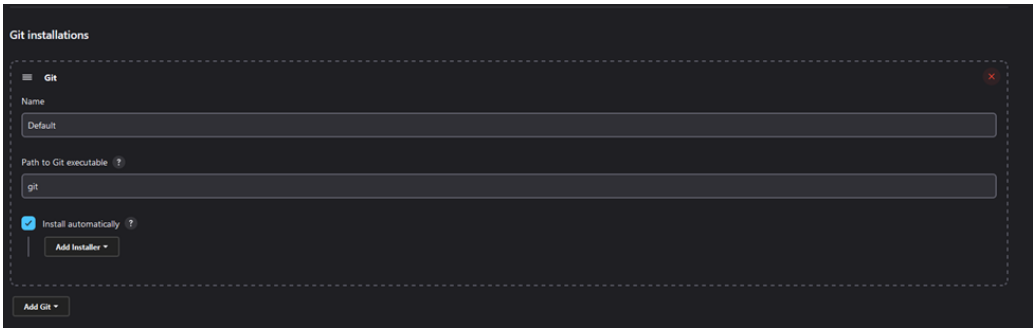
Στο μονοπάτι Manage Jenkins -> Tools γίνεται η παραμετροποίηση των απαραίτητων στοιχείων για την εκτέλεση του pipeline, του οποίου θα δημιουργήσουμε, με τις ενέργειες όπου θέλουμε.



Εικόνα 22 Παραμετροποίηση στοιχείων χρήσης

Τα στοιχεία που θα παραμετροποιήσουμε για την επιτυχή εκτέλεση του pipeline είναι τα εξής:

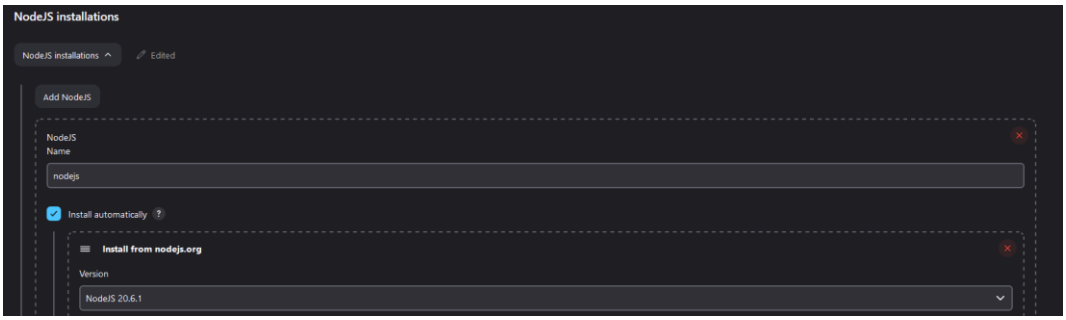
- Git
- SonarQube
- Owasp Dependency Check
- Maven
- NodeJS



Εικόνα 23 Αρχικοποίηση Git

Εικόνα 24 Αρχικοποίηση Maven

Εικόνα 25 Αρχικοποίηση Owasp Dependency Check

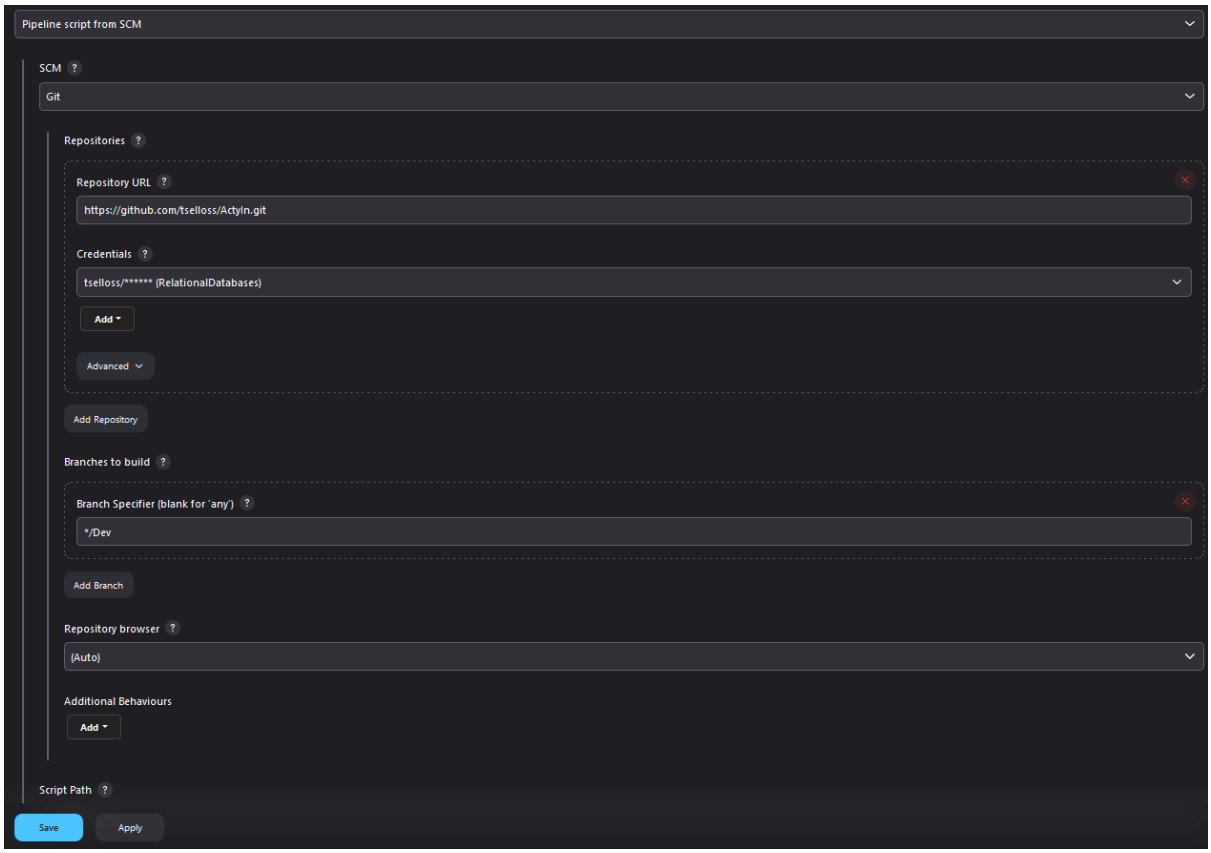


Εικόνα 26 Αρχικοποίηση NodeJS

2.7.5 Αρχικοποίηση Pipelines

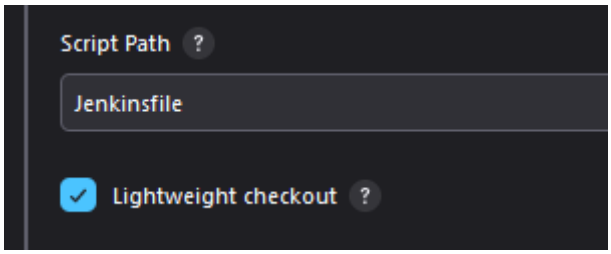
Για την εκτέλεση των αυτοματοποιημένων διαδικασιών είναι αναγκαίο να γίνει η σύνθεση τους, αυτό επιτυγχάνεται με την αρχικοποίηση ενός Pipeline. Είναι ουσιαστικά το μέσο το οποίο μας επιτρέπει να πάμε βήμα-βήμα τις απαραίτητες ενέργειες που έχει τεθεί ως στόχος. Δηλαδή την επιτυχή εκτέλεσή τους.

Παρακάτω θα αναλύσουμε τα στοιχεία που έχουμε υλοποιήσει για τις λύσεις Back-End και Front-End. Η αρχικοποίηση των pipeline είναι ακριβώς με τον ίδιο τρόπο. Αλλάζει ουσιαστικά το repository URL το οποίο βρίσκεται η κάθε λύση.σ



Εικόνα 27 Αρχικοποίηση Pipeline

Για να μπορέσει το Jenkins να ακολουθήσει τα βήματα του Pipeline έχει αρχικοποιήσεις ότι θα πρέπει να ακολουθήσει το αρχείο Jenkinsfile όπως φαίνεται παρακάτω :



Εικόνα 28 Μονοπάτι Pipeline στην λύση

```
1.  pipeline {
    agent any
    tools{
        jdk  'jdk17'
        maven  'maven3'
    }
    environment
    {
        SCANNER_HOME= tool 'sonar-scanner'
        SONARQUBE_IMAGE_NAME = 'sonarqube:latest'
        JENKINS_IMAGE_NAME = 'jenkins/jenkins'
    }
    stages {
        stage('Git Checkout') {
            steps {
                git branch: 'Dev', credentialsId: 'RelationalDatabases', url:
'https://github.com/tselloss/ActyIn'
            }
        }

        stage('OWASP Dependency Check') {
            steps {
                dependencyCheck additionalArguments: ' --scan ./ ', odcInstal-
lation: 'DC'
                    dependencyCheckPublisher pattern: '**/dependency-check-re-
port.xml'
            }
        }

        stage('File System Scan') {
            steps {
                sh "/var/jenkins_home/workspace/trivy fs ."
            }
        }

        stage('Sonarqube Image Scan') {
            steps {
                 sh "/var/jenkins_home/workspace/trivy repo
https://github.com/SonarSource/docker-sonarqube.git"
            }
        }

        stage('Jenkins Image Scan') {
            steps {
                sh "/var/jenkins_home/workspace/trivy image ${JENKINS_IM-
AGE_NAME}"
            }
        }
```

```
        stage('Sonarqube Analysis') {
            steps {
                withSonarQubeEnv('sonar'){
                    sh ''' $SCANNER_HOME/bin/sonar-scanner -Dsonar.project-
Name=ActyIn \
                    -Dsonar.projectKey=ActyIn '''
                }
            }
        }
    }
}
```

Κώδικας : Pipeline Back-End Solution

Παρακατω παραθετεται το Pipeline του Front-End:

```
pipeline {
    agent any
    tools {
        jdk 'jdk17'
        maven 'maven3'
        nodejs 'nodejs'
    }
    environment {
        SCANNER_HOME = tool 'sonar-scanner'
        SONARQUBE_IMAGE_NAME = 'sonarqube:latest'
        JENKINS_IMAGE_NAME = 'jenkins/jenkins'
    }

    stages {
        stage('Git Checkout') {
            steps {
                git branch: 'VueNewVersion', credentialsId: 'RelationalData-
bases', url: 'https://github.com/tselloss/ActyInFrontEnd.git'
            }
        }

        stage('Install Dependencies') {
            steps {
                sh 'npm install'
            }
        }

        stage('OWASP Dependency Check') {
            steps {
                dependencyCheck additionalArguments: '--scan ./', odcInstalla-
tion: 'DC'
                dependencyCheckPublisher pattern: '**/dependency-check-re-
port.xml'
            }
```

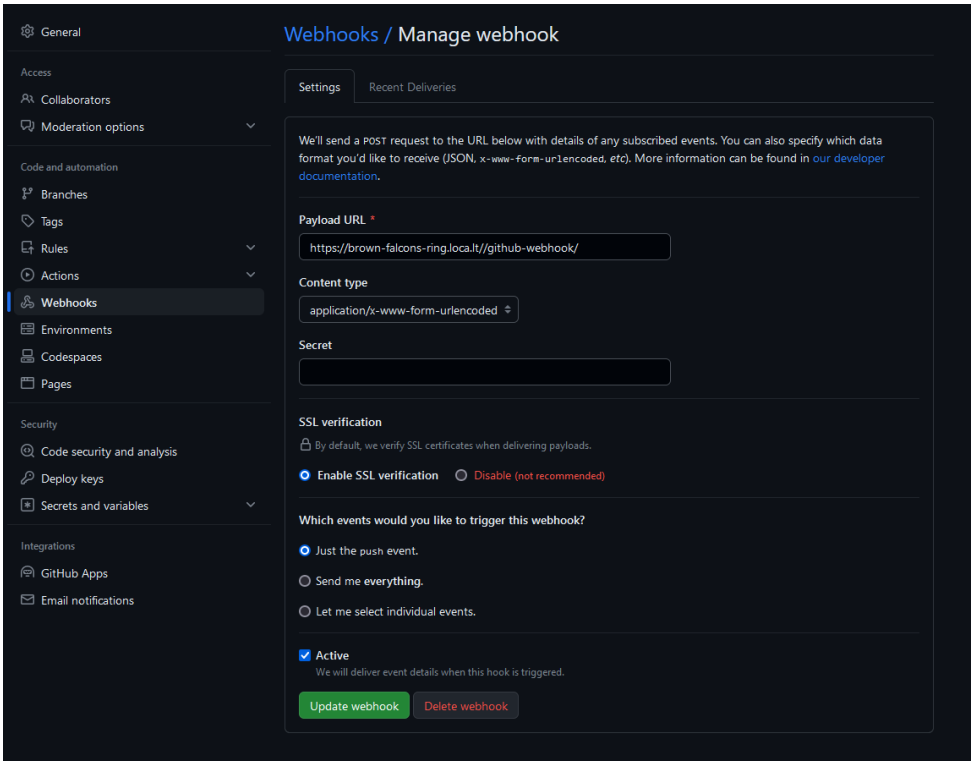
```
        }

       stage('File System Scan') {
            steps {
                sh "/var/jenkins_home/workspace/trivy fs ."
            }
        }

        stage('Sonarqube Image Scan') {
            steps {
                 sh "/var/jenkins_home/workspace/trivy repo
https://github.com/SonarSource/docker-sonarqube.git"
            }
        }

        stage('Jenkins Image Scan') {
            steps {
                sh "/var/jenkins_home/workspace/trivy image ${JENKINS_IM-
AGE_NAME}"
            }
        }

        stage('Sonarqube Analysis') {
            steps {
                withSonarQubeEnv('sonar') {
                    sh '''
                        ${SCANNER_HOME}/bin/sonar-scanner \
                        -Dsonar.projectName=ActyInFrontEnd \
                        -Dsonar.projectKey=ActyInFrontEnd
                    '''
                }
            }
        }
    }
}
```

Κώδικας : Pipeline Front-End Solution

2.7.4 GitHub Webhooks

Η GitHub, Inc. είναι μια πλατφόρμα και υπηρεσία βασισμένη στο cloud για την ανάπτυξη λογισμικού και τον έλεγχο εκδόσεων χρησιμοποιώντας το Git. Επιτρέπει στους προγραμματιστές να αποθηκεύουν, να διαχειρίζονται και να παρακολουθούν τις αλλαγές στον κώδικά τους. Μέσω του GitHub, οι προγραμματιστές μπορούν να συνεργάζονται σε έργα από απόσταση, να παρακολουθούν την πρόοδο και να εντοπίζουν σφάλματα. Επιπλέον, προσφέρει δυνατότητες όπως ζητήματα (issues), αιτήματα αλλαγών (pull requests) και συνεχή ενσωμάτωση (continuous integration), διευκολύνοντας την αποτελεσματική ανάπτυξη και διαχείριση λογισμικού.

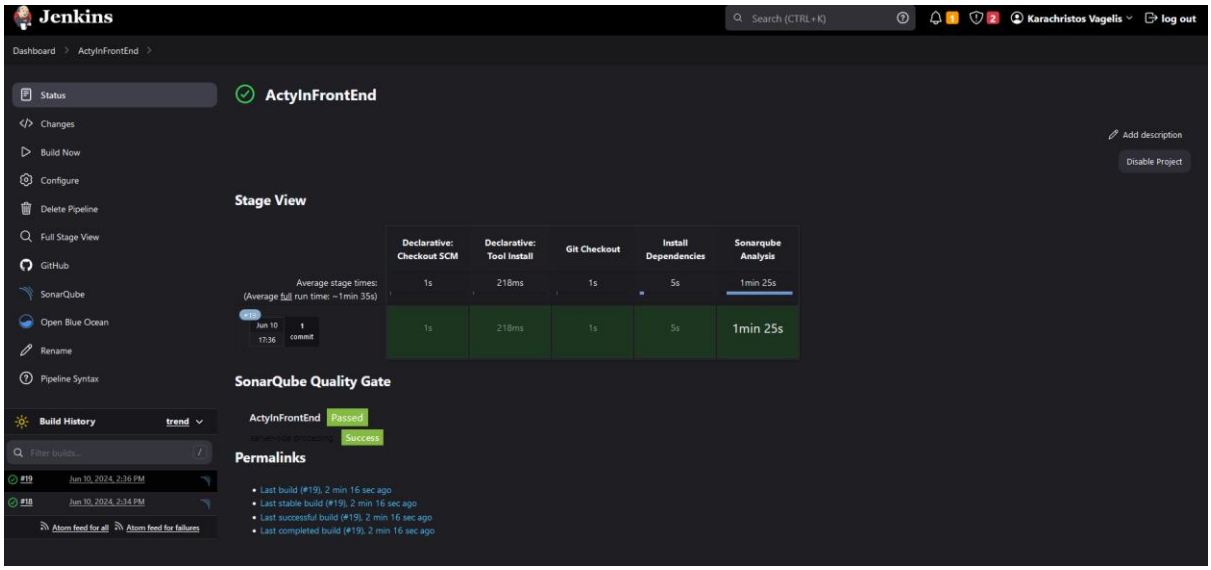
Ένα webhook στην ανάπτυξη ιστοσελίδων είναι ένας τρόπος επέκτασης ή τροποποίησης της συμπεριφοράς μιας ιστοσελίδας ή μιας εφαρμογής μέσω καλουπιών (callbacks). Αυτά τα callbacks μπορεί να διατηρούνται, να τροποποιούνται και να διαχειρίζονται από τρίτους χρήστες και προγραμματιστές που ενδεχομένως να μην έχουν άμεση σχέση με την ιστοσελίδα ή την εφαρμογή προέλευσης. Τα webhooks ενεργοποιούνται όταν συμβαίνει ένα συγκεκριμένο γεγονός, όπως μια ενημέρωση στον κώδικα ή μια νέα εγγραφή χρήστη. Αυτή η δυνατότητα επιτρέπει την αυτοματοποίηση και την ευελιξία, καθιστώντας την ανάπτυξη και τη συντήρηση εφαρμογών πιο αποδοτική και προσαρμόσιμη στις ανάγκες των χρηστών.



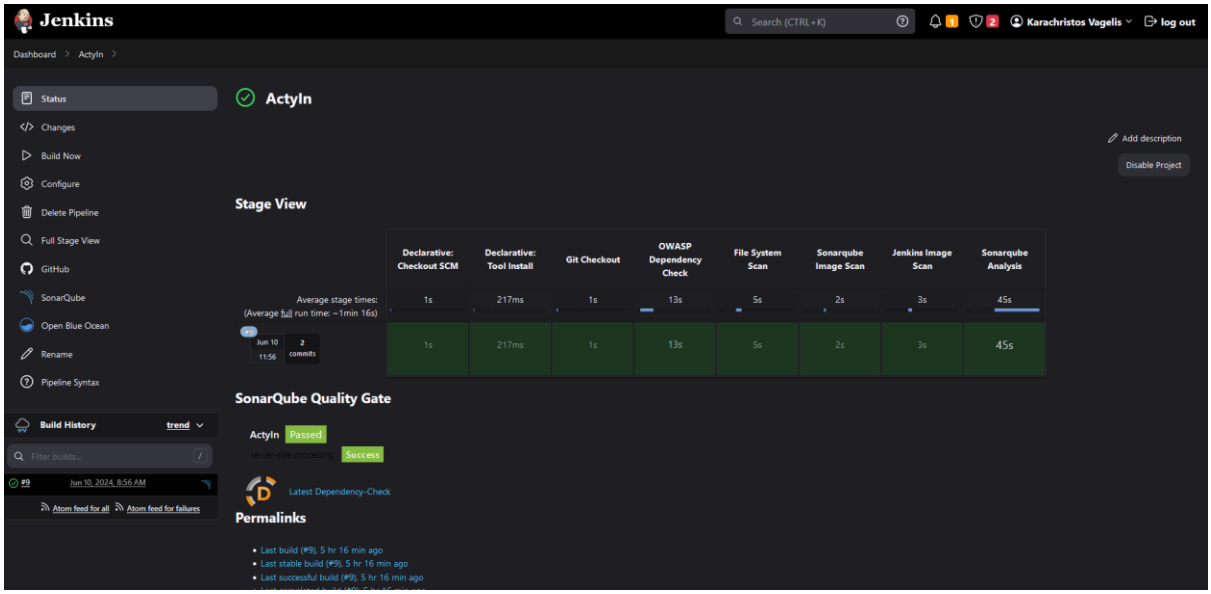
Εικόνα 29 GitHub Webhook Setup Page

Τα webhooks εκτελούνται κάθε φορά που ένας προγραμματιστής κάνει push νέο τμήμα κώδικα. Αυτές οι αυτόματες ειδοποιήσεις επιτρέπουν στους προγραμματιστές να παρακολουθούν και να αντιδρούν σε αλλαγές στον κώδικα σε πραγματικό χρόνο. Για παράδειγμα, όταν γίνεται ένα push, το webhook θα ενεργοποιήσει την διαδικασία συνεχούς ενσωμάτωσης (CI), θα τρέξει τα pipeline τα οποία έχουμε συγγράψει. Με αυτόν τον τρόπο, μπορεί να διασφαλιστεί ότι οι αλλαγές ενσωματώνονται και ελέγχονται άμεσα, βελτιώνοντας την αποδοτικότητα και την ποιότητα του λογισμικού.

Κατά την εκτέλεση ενός push, θα δούμε το Jenkins, το οποίο έχει ενεργοποιηθεί από το webhook, να ξεκινά να τρέχει τα pipelines.



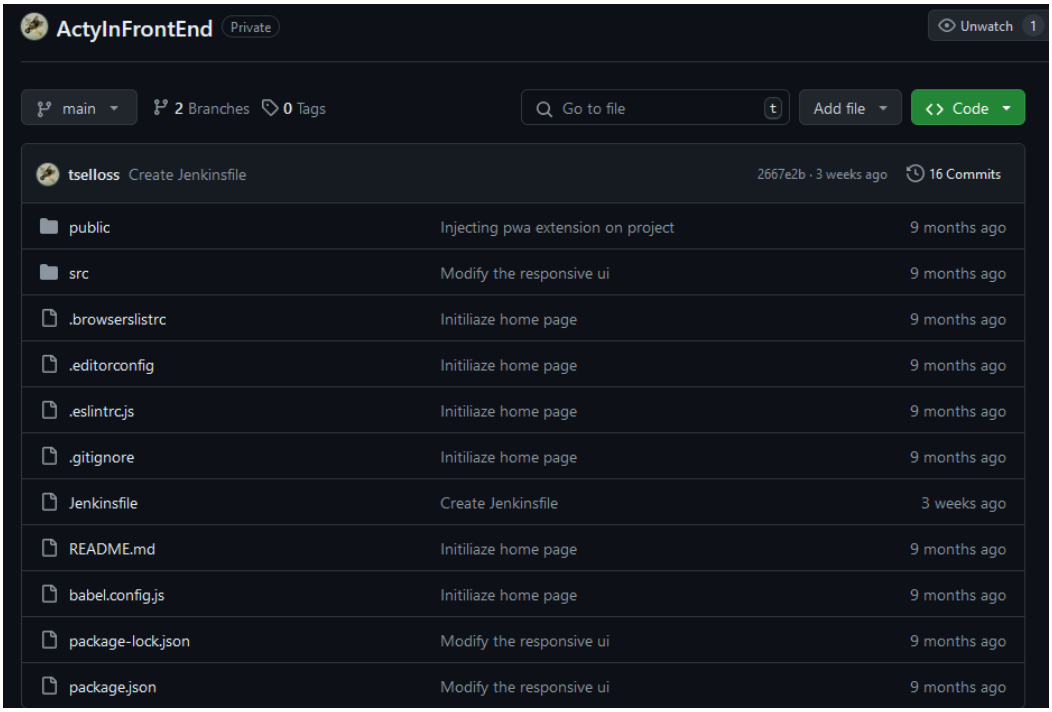
Εικόνα 30 Front-End Pipeline



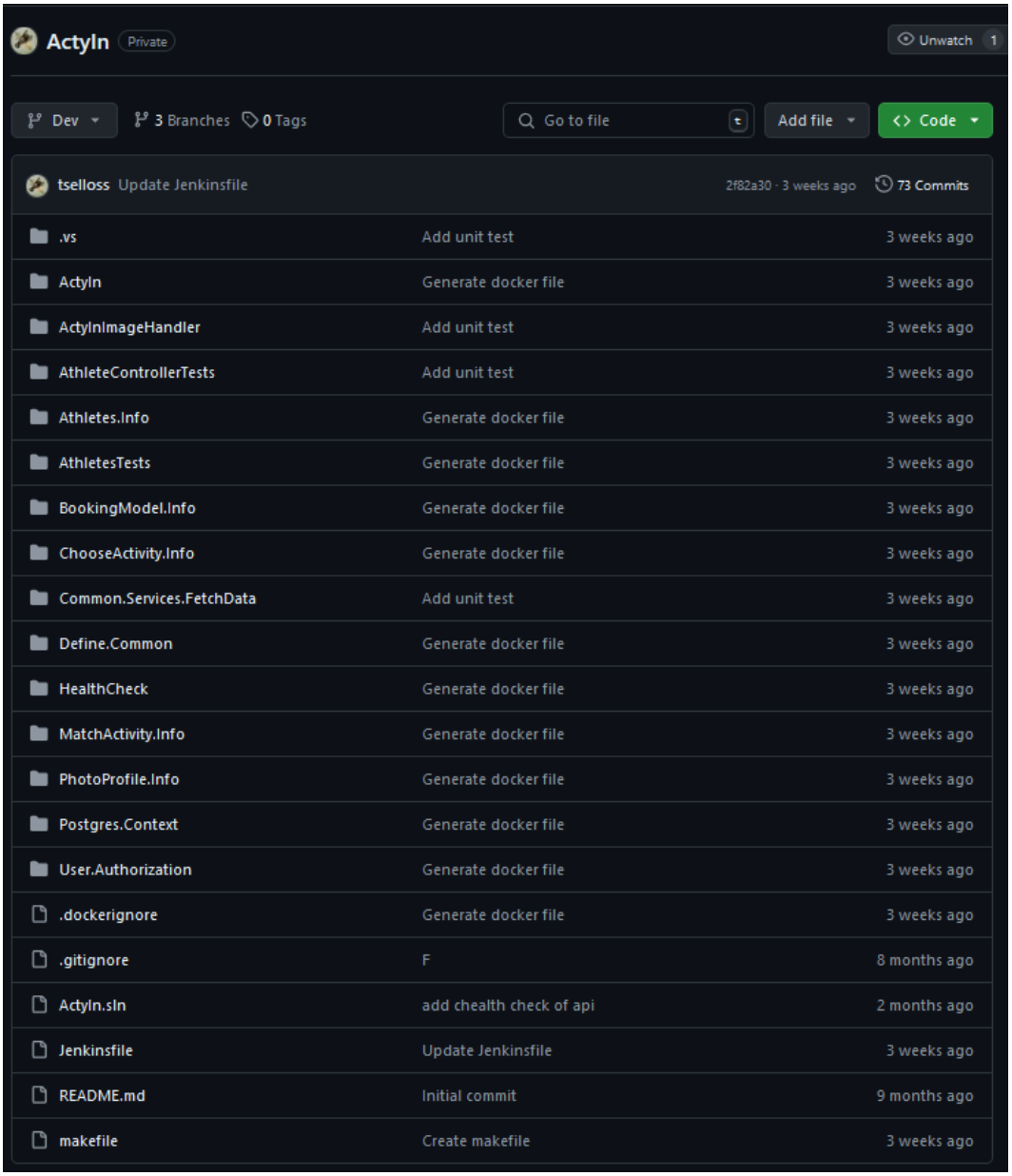
Εικόνα 31 Back-End Pipeline

Με την ολοκλήρωση, όπως θα δούμε παρακάτω στην ενότητα 2.7.5, τα αποτελέσματα δημοσιεύονται στο SonarQube. Το SonarQube αναλύει τα αποτελέσματα και παρέχει μια λεπτομερή αναφορά της ποιότητας του κώδικα.

2.7.4.1 GitHub Projects



Εικόνα 32 Front-End GitHub Project

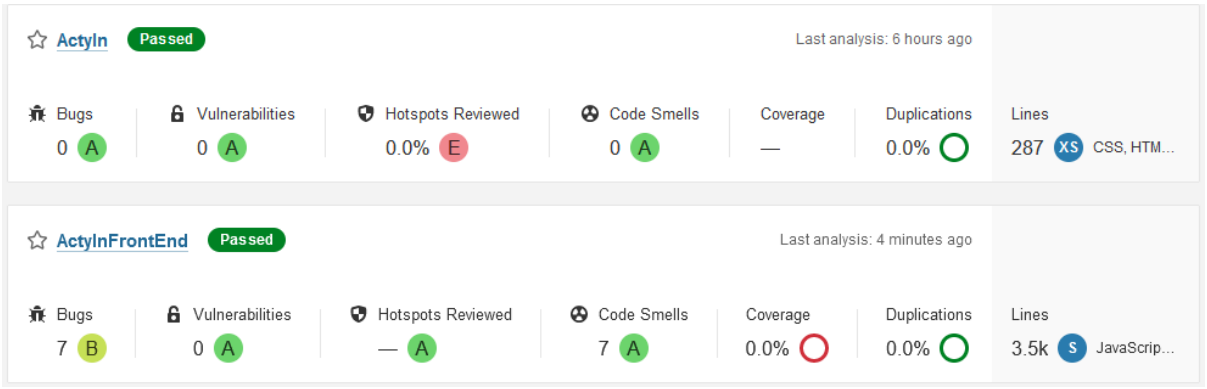


Εικόνα 33 Back-End GitHub Project

2.7.5 SonarQube

Το SonarQube είναι μια πλατφόρμα ανοικτού κώδικα που αναπτύχθηκε από τη SonarSource για τον συνεχή έλεγχο της ποιότητας του κώδικα, προσφέροντας αυτόματες αναθεωρήσεις με στατική ανάλυση του κώδικα για τον εντοπισμό σφαλμάτων και ανωμαλιών κώδικα σε 30 γλώσσες προγραμματισμού. Το SonarQube παρέχει αναφορές για διπλότυπο κώδικα, πρότυπα κωδικοποίησης, μονάδες ελέγχου, κάλυψη κώδικα, πολυπλοκότητα κώδικα, σχόλια, σφάλματα και συστάσεις ασφαλείας.

Όπως θα δούμε παρακάτω, το γραφικό περιβάλλον του SonarQube αρχικά μας δείχνει μια γενική εικόνα των έργων που έχουν αναλυθεί. Έπειτα, μπορούμε να δούμε κάθε έργο με περισσότερες λεπτομέρειες.



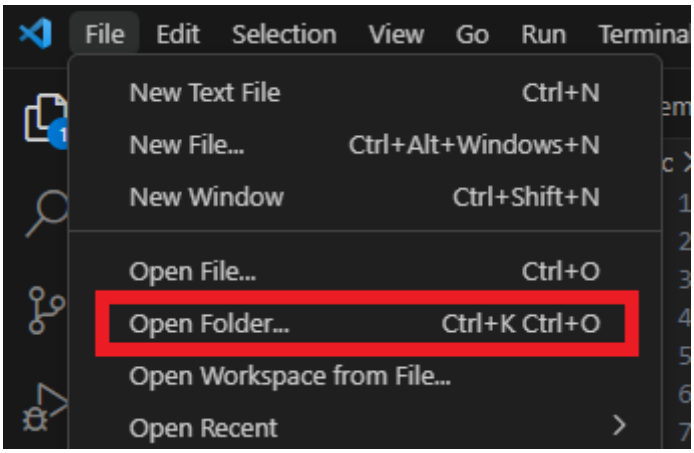
Εικόνα 34 Περιβάλλον SonarQube, συνοπτική απεικόνιση έργων ενδιαφέροντος

2.8 Οδηγίες τοπικής εκτέλεσης και χρήσης εφαρμογής

2.8.1 Εκτέλεση Front-End

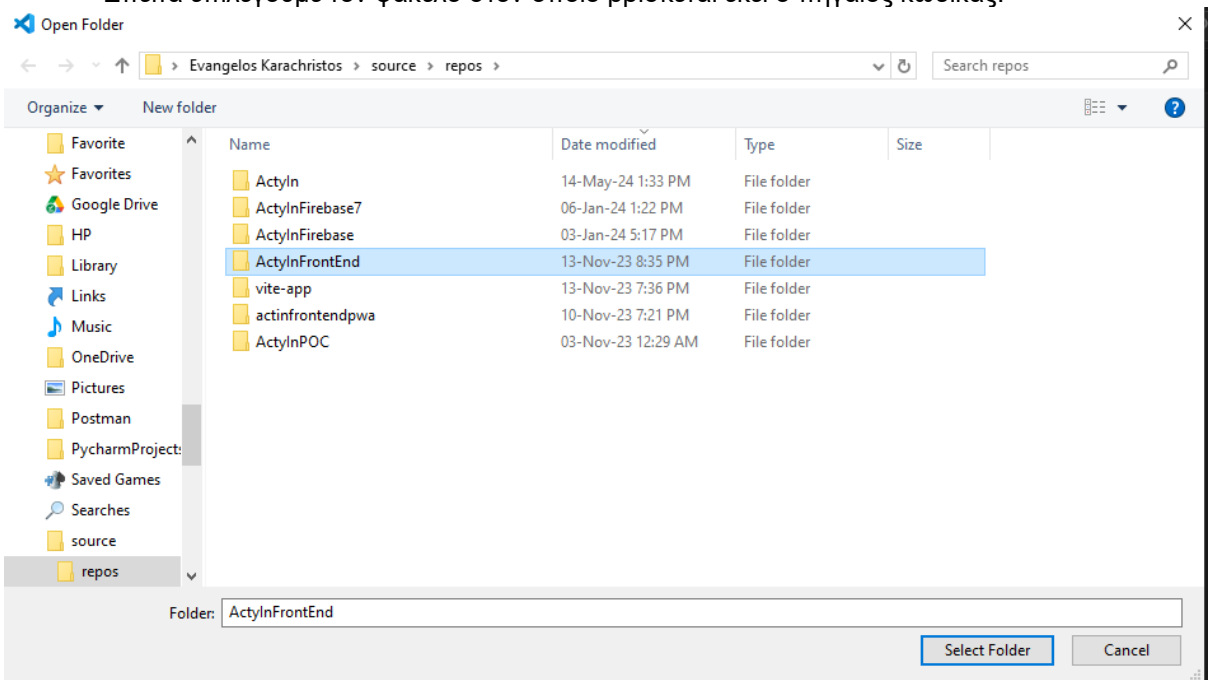
Για την εκτέλεση του front-end θα πρέπει να ανοίξουμε μέσω Visual studio Code τον φάκελο με τον πηγαίο κώδικα. Στην δική μας περίπτωση δεν είναι μέσα στον ίδιο φάκελο με τον πηγαίο κώδικα που έχουμε από την μεριά του back-end, συνήθως είναι κάτω από τον ίδιο φάκελο για λογούς ευκολίας συντήρησης.

Το πρώτο βήμα που πρέπει να κάνουμε είναι να εκτελέσουμε το IDE Visual studio Code, και να πάμε στην διαδρομή File -> Open Folder όπως βλέπουμε και παρακάτω.



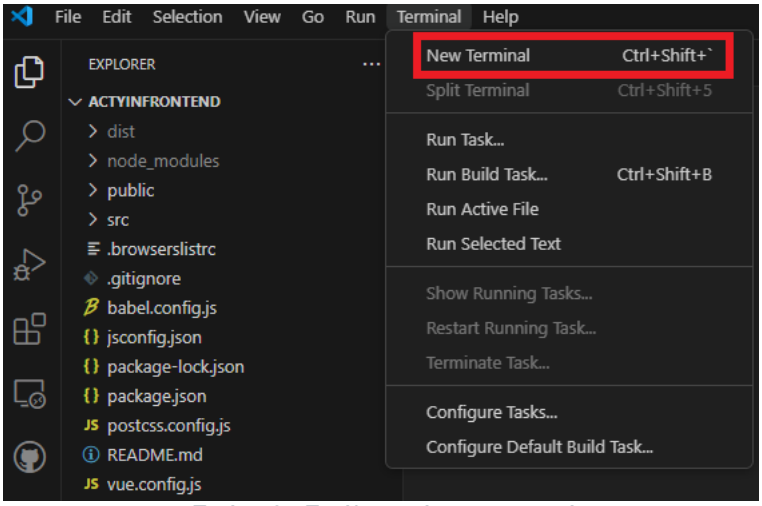
Εικόνα 35 Αναζήτηση φακέλου πηγαίου κώδικα FE

Έπειτα επιλέγουμε τον φάκελο στον οποίο βρίσκεται εκεί ο πηγαίος κώδικας.



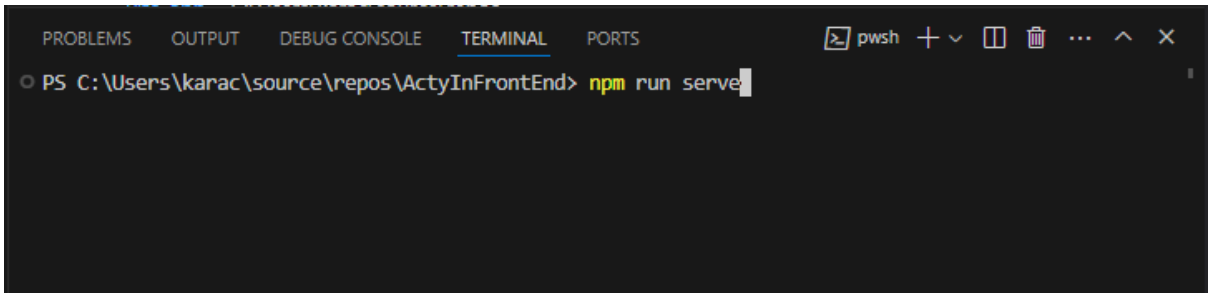
Εικόνα 36 Επιλογή φακέλου πηγαίου κώδικα

Το πρώτο πράγμα που βλέπουμε είναι η κεντρική οθόνη του Visual Studio Code με αριστερά να έχει παραχθεί ο πηγαίος κώδικας του Front-End. Μόλις γίνει αυτό θα πρέπει να πάμε στην μπάρα εργαλείων πάλι και να αναζητούμε το μονοπάτι Terminal -> New Terminal όπως θα δούμε στην παρακάτω εικόνα.



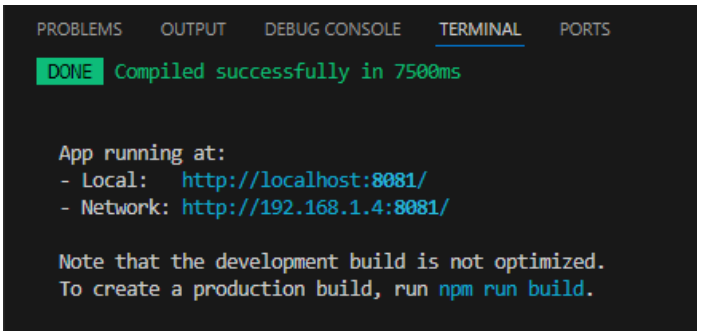
Εικόνα 37 Εκτέλεση νέου τερματικού

Όταν γίνουν αυτά τα βήματα θα έχουμε ανοίξει ένα νέο τερματικό ώστε να μπορούμε να εκτελέσουμε την εφαρμογή μας με την εντολή **npm run serve.** Όπως βλέπουμε παρακάτω το μονοπάτι που εκτελείται η εντολή εκκίνησης της εφαρμογής είναι ο φάκελος που περιέχει τον πηγαίο κώδικα της vue.js εφαρμογής μας.



Εικόνα 38 Εντολή εκτέλεσης FE πηγαίου κώδικα

Κατά την εκτέλεση αυτής της εντολής θα δούμε, το ¨χτίσιμο¨ της εφαρμογής. Αν όλα πάνε καλά και δεν έχουμε πρόβλημα ώστε να μπορεί να γίνει η σύσταση της εφαρμογής μας τότε θα διακρίνουμε την διευθυνσιοδότηση της εφαρμογής και θα μπορούμε να περιηγηθούμε στις σελίδες της.

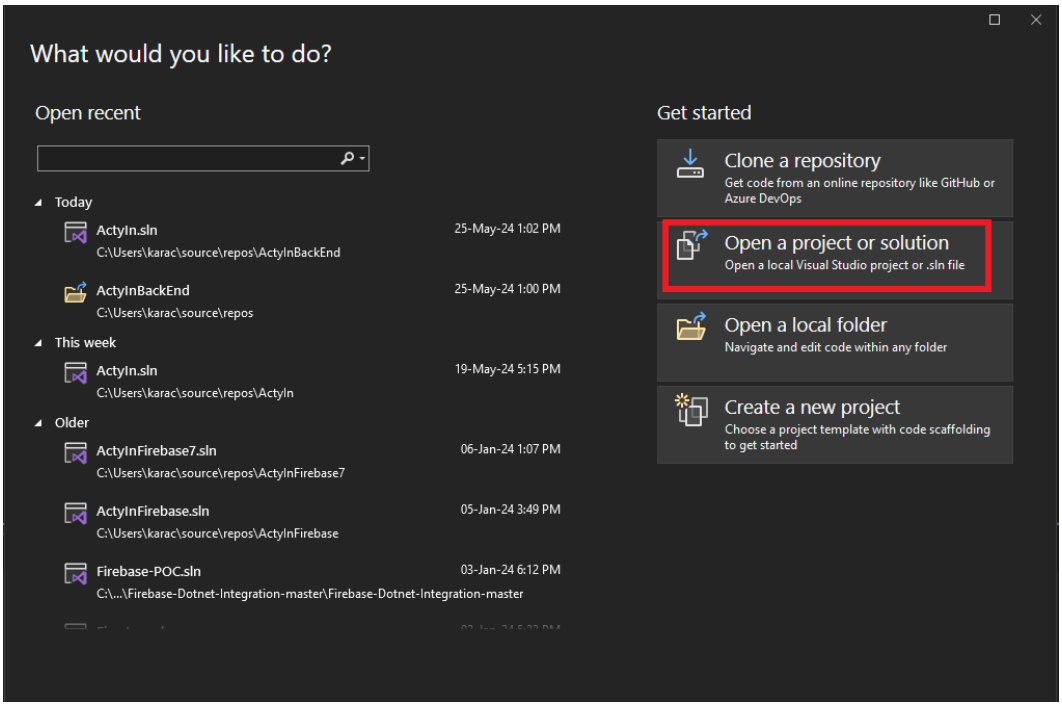


Εικόνα 39 Επιτυχής σύσταση εφαρμογής

2.8.2 Εκτέλεση Back-End

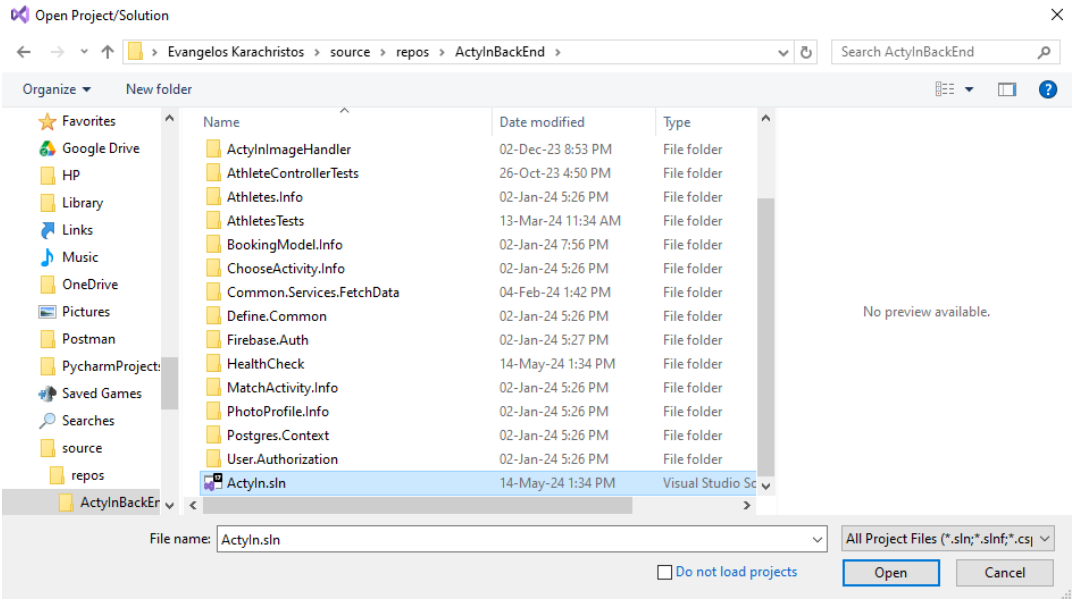
Μια περίπου παρόμοια διαδικασία για την εκτέλεση του Back-End πηγαίου κώδικα άλλα αυτήν την φορά δεν θα ανοίξουμε τον φάκελο μόνο αλλά θα μπούμε μέσα σε αυτόν και θα επιλέξουμε το αρχείο του project. Τον πηγαίο κώδικα αυτό θα τον εκτελέσουμε με το IDE Visual Studio. Όπως και στην παραπάνω περίπτωση του Front-End θα ανοίξουμε τον compiler μας και θα ψάξουμε τον φάκελο με τον πηγαίο κώδικα στο λειτουργικό μας σύστημα.

Κατά την εκτέλεση του Visual Studio μας παραπέμπει στην διαδικασία εύρεσης φακέλου του project μας. Θα κάνουμε ακριβώς τα ίδια βήματα όπως κάναμε και παραπάνω για την εκτέλεση του Back-End κώδικα.



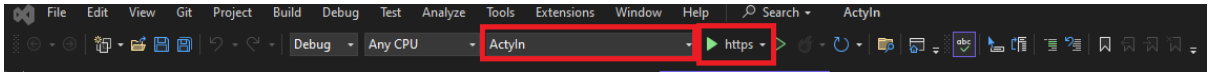
Εικόνα 40 Αναζήτηση αρχείου του project (.sln)

Όπως βλέπουμε στην εικόνα 22 θα επιλέξουμε τον σωστό φάκελο που περιέχει το αρχείο τύπου .sln που είναι και το αρχείο του project.



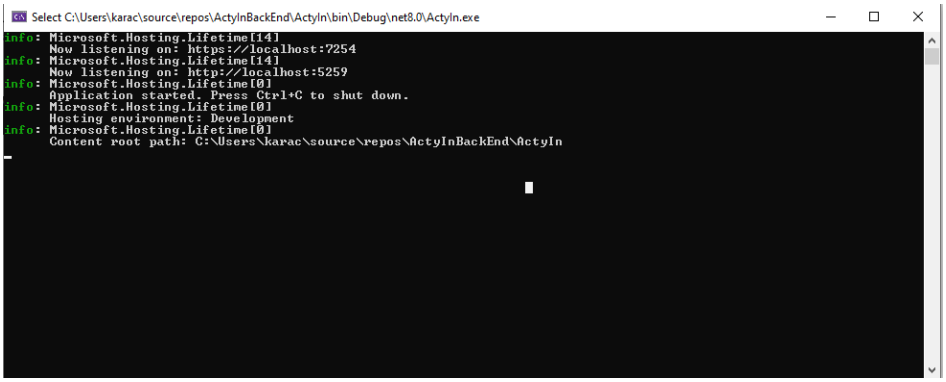
Εικόνα 41 Επιλογή αρχείου project (.sln)

Έχοντας ανοίξει το project τότε πρέπει να πάμε στην μπάρα εργαλείων του Visual Studio και να το εκτελέσουμε. Με την ενέργεια αυτή θα γίνει εξίσου ένας έλεγχος αν μπορεί να γίνει σύσταση της εφαρμογής μας και έτσι να μπορέσει να εκτελεστεί σωστά. Έχοντας επιλέξει το εκτελέσιμο που θέλουμε να γίνει σύσταση πρέπει να πατήσουμε το κουμπί που αναγράφει το σχέδιο play με την αναγραφή https όπως φαίνεται στην Εικόνα 23.



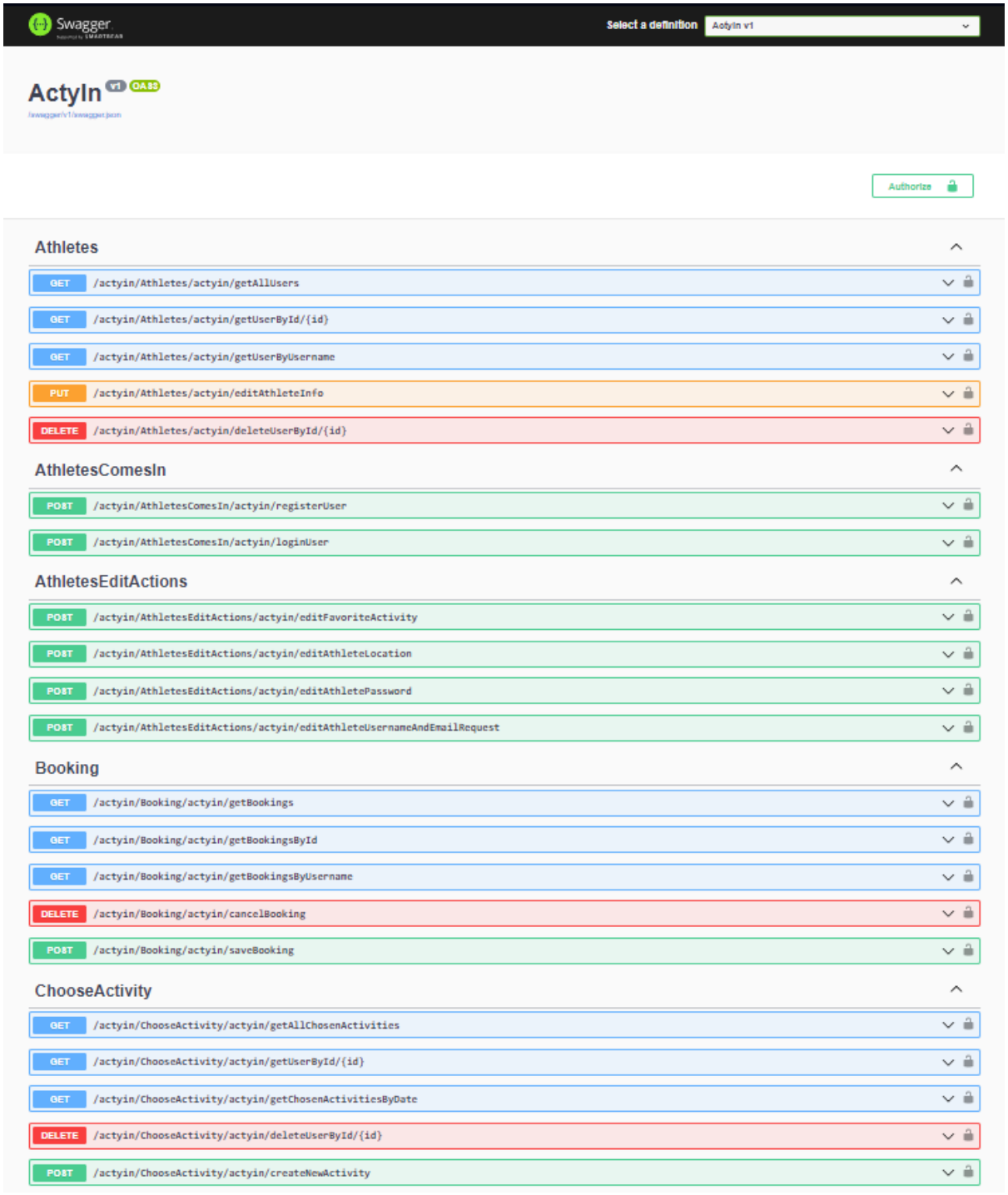
Εικόνα 42 Επιλογή και εκτέλεση επιλέξιμου αρχείου project

Κατά το πάτημα του κουμπιού εκτέλεσης του project θα δούμε στο τερματικό του Visual Studio την προσπάθεια σύστασης της εφαρμογής μας. Έχοντας γίνει επιτυχής εκτέλεση, δυο είναι τα πράγματα που πρέπει να εμφανιστούν στην οθόνη μας. Το πρώτο είναι ένα τερματικό που να μας δίνει όλη την πληροφορία για την εφαρμογή μας. Σχεδόν παράλληλα θα να μας ανοίξει ένας υπερ. σύνδεσμος με την τοπική διεύθυνση της εφαρμογής μας https://localhost:7254/swagger/index.html όπως αναγράφεται με τον πλήρη οδηγό διαθέσιμων endpoint της εφαρμογής.



Εικόνα 43 Διευθυνσιοδότηση Back-End πηγαίου κώδικα

Στην παρακάτω εικόνα θα δούμε τον πλήρη οδηγό για την εφαρμογή μας ονομαζόμενο Swagger. Μέσω αυτού του εργαλείου μπορούμε να έχουμε διαχωρισμένο το Back-End με το Front-End. Αυτό μας δίνει την δυνατότητα αν ένα project αποτελείται από πολλά άτομα, αυτά τα δυο μέρη να μπορούν να λειτουργήσουν ξεχωριστά χωρίς την ανάγκη το ένα από το άλλο.



Εικόνα 44 Εγχειρίδιο Back-End

Κεφάλαιο 3 – Οδηγίες χρήσης εφαρμογής

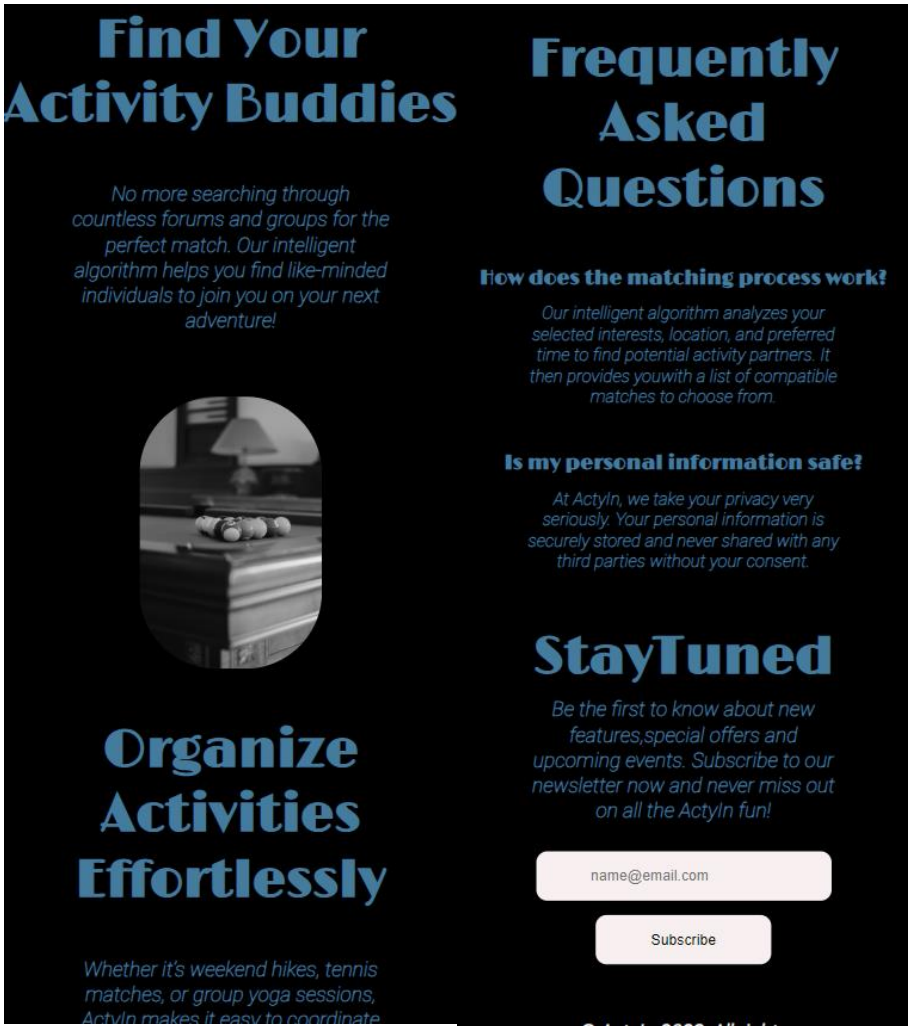
3.1 Σελίδα Έναρξης

Στην αρχική σελίδα της εφαρμογής μας θα βρούμε κυρίως πληροφορίες σχετικά με την εφαρμογή, κουμπιά εγγραφής και εισόδου στην εφαρμογή για ήδη εγγεγραμμένους χρήστες. Ακόμα θα βρούμε φόρμα εγγραφής για την ενημέρωση των ενδιαφερομένων για τυχόν ενημερώσεις της εφαρμογής. Η εφαρμογή στην κεντρική της σελίδα είναι ουσιαστικά ένας οδηγός, όπως ακόμα , να παρέχει και μερικές ακόμη ενημερώσεις για τους υποψήφιους χρήστες. Όπως για παράδειγμα συχνές ερωτήσεις που έχουν γίνει για την ασφάλεια των δεδομένων των χρηστών της εφαρμογής.

Ας συνεχίσουμε αναλυτικότερα με εικόνες, και κατά την παρουσίαση των εικόνων θα γίνει και σχολιασμός τους, ώστε να υπάρξει πλήρη κατανόηση της χρήσης της εφαρμογής.

Εικόνα 45 Ανάλυση κουμπιών Join Now & Register στην κεντρική σελίδα

Στην συνέχεια αφού περιηγηθούμε στην κεντρική μας σελίδα μπορούμε να βρούμε πληροφορίες για τον σκοπό αυτής της εφαρμογής όπως ακόμα και ποια είναι τα οφέλη της. Έπειτα θα συναντήσουμε μερικές γενικές ερωτήσεις για το πώς γίνεται η αναζήτηση χρηστών, όπως ακόμα και το αν είναι ασφαλής η εφαρμογή ώστε να βάλουμε τα στοιχεία μας. Τέλος όπως βλέπουμε υπάρχει φόρμα εγγραφής συνδρομητή για ενημέρωση των νέων της εφαρμογής.



Εικόνα 46 Ενημέρωση χρηστών πριν την εγγραφή τους για τον σκοπό της εφαρμογής

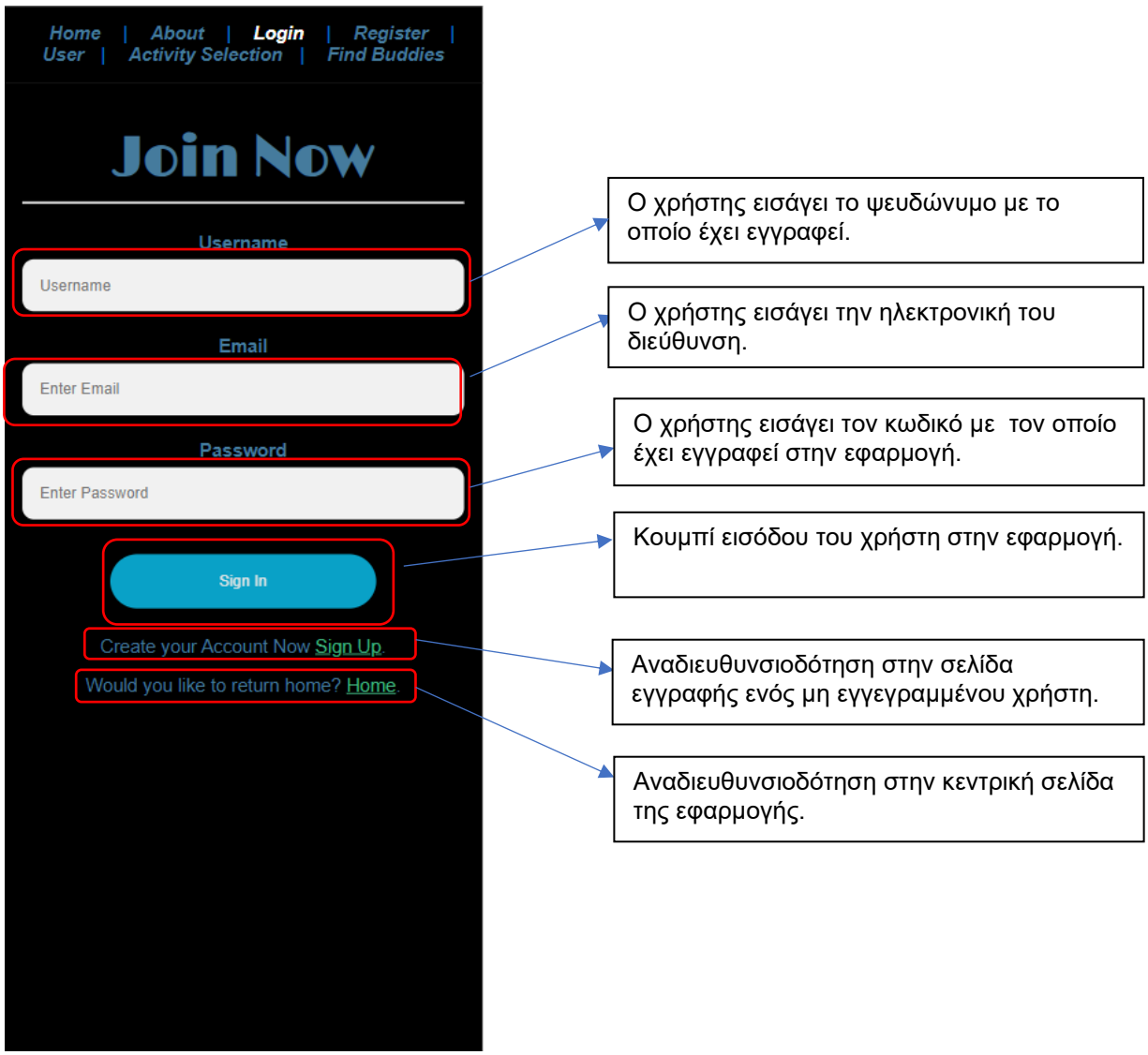
Εικόνα 47 Εγγραφή χρήστη ή μη εγγεγραμμένου χρήστη ως συνδρομητή

3.2 Σελίδα εγγραφής νέου χρήστη

Αν ο χρήστης κάνει είσοδο ή εγγραφή τότε θα μεταφερθεί στην σελίδα που θα αρχίσει τα βήματα για την αναζήτηση ημερομηνίας, δραστηριότητας και διαθέσιμων ατόμων ώστε να πραγματοποιηθεί μια κράτηση. Ας ξεκινήσουμε όμως πρώτα να δούμε τις σελίδες εγγραφής και εισόδου μέσα από εικόνες και σχολιασμό.

Εικόνα 48 Ανάλυση και σχολιασμός της σελίδας εγγραφής νέου χρήστη

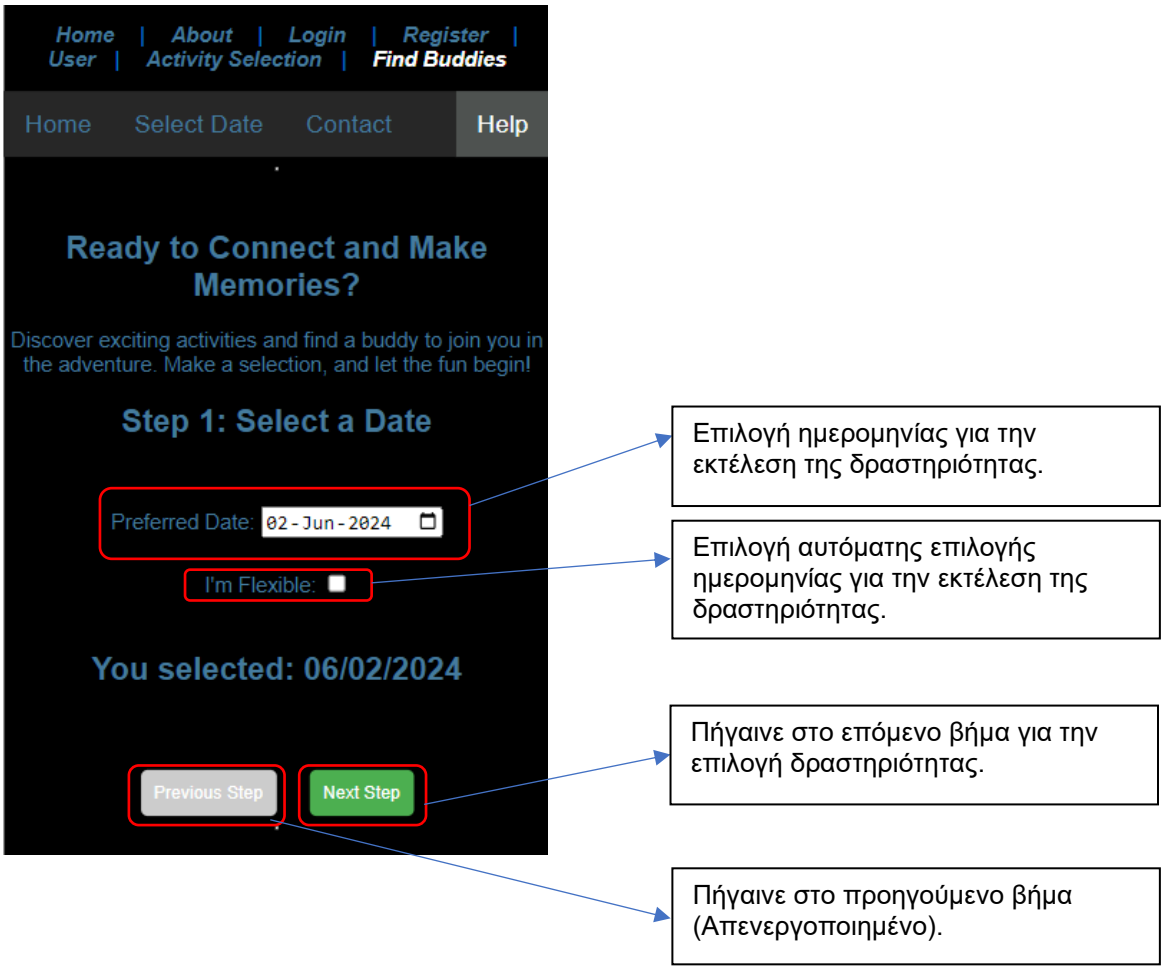
3.3 Σελίδα εισόδου εγγεγραμμένου χρήστη



Εικόνα 49 Ανάλυση και σχολιασμός της σελίδας εισόδου στην εφαρμογή ενός εγγεγραμμένου χρήστη

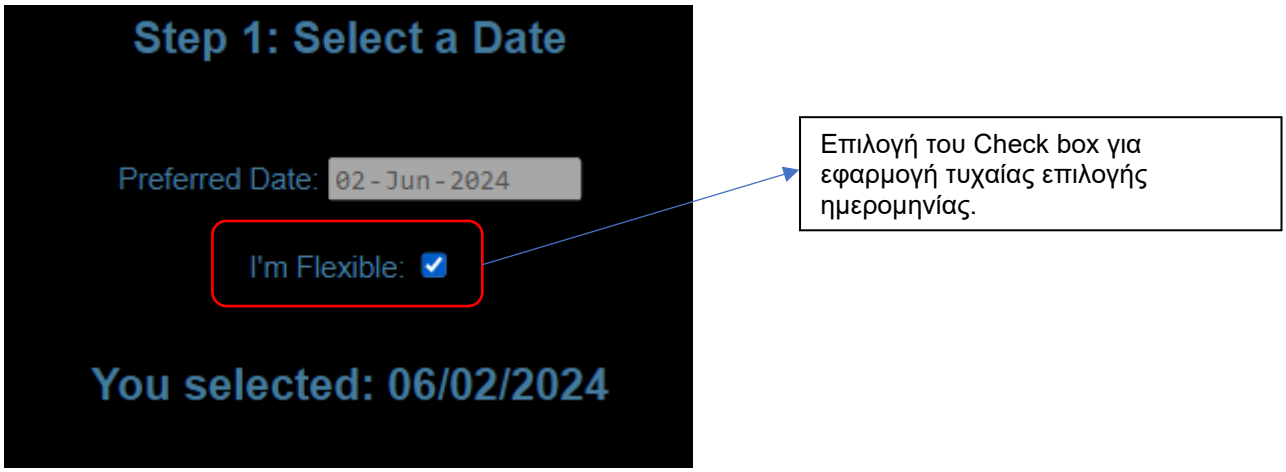
3.4 Σελίδα επιλογής ημερομηνίας δραστηριότητας

Κατά την επιτυχή εγγραφή ή είσοδο ενός χρήστη στην εφαρμογή, γίνεται η διευθυνσιοδότηση του στην σελίδα εύρεσης δραστηριότητας. Έπειτα ο χρήστης έχει τρία βήματα για να ολοκληρώσει αυτήν την διαδικασία. Αρχικά πρέπει να διαλέξει ημερομηνία που τον ενδιαφέρει να γίνει η δραστηριότητα. Έπειτα να επιλέξει αναμεσά από τις διαθέσιμες μέχρι τώρα δραστηριότητες που ικανοποιεί η εφαρμογή. Τέλος να επιλέξει το άτομο μέσα από μια λίστα διαθέσιμων ατόμων που βρίσκονται άμεσα ενδιαφερόμενοι για την ίδια δραστηριότητα και την ίδια ημερομηνία. Ας δούμε την διαδικασία μέσα από εικόνες και πλήρη σχολιασμό αυτών.



Εικόνα 50 Απεικόνιση και σχολιασμός σελίδας επιλογής ημερομηνίας δραστηριότητας (Βήμα 1ο)

Κάθε χρήστης έχει την δυνατότητα να επιλέξει μια ημερομηνία ή να επιλέξει αν γίνει τυχαία επιλογή αυτής.



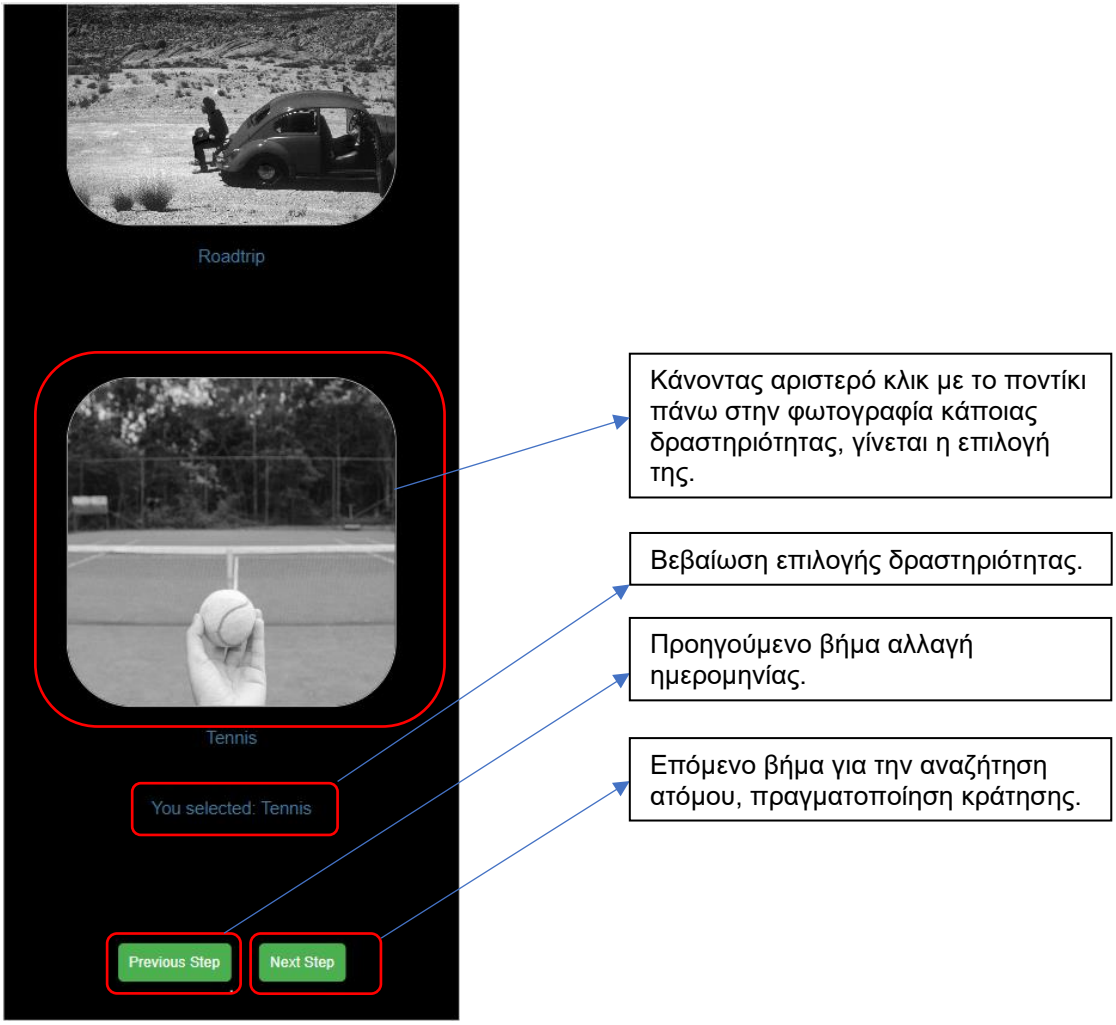
Εικόνα 51 Απεικόνιση και σχολιασμός τυχαίας επιλογής ημερομηνίας (Βήμα 1ο)

3.5 Σελίδα επιλογής δραστηριότητας

Ο χρήστης έχοντας επιλέξει την ημερομηνία που επιθυμεί να πραγματοποιήσει την δραστηριότητα τότε μπορεί να συνεχίσει στην επιλογή της. Θα ακολουθήσει η σελίδα επιλογής δραστηριότητας οπού περιέχει τις διαθέσιμες μέχρι τώρα δραστηριότητες οπού υποστηρίζει η εφαρμογή.

Οι διαθέσιμες δραστηριότητες κατά σειρά όπως στην εφαρμογή είναι :

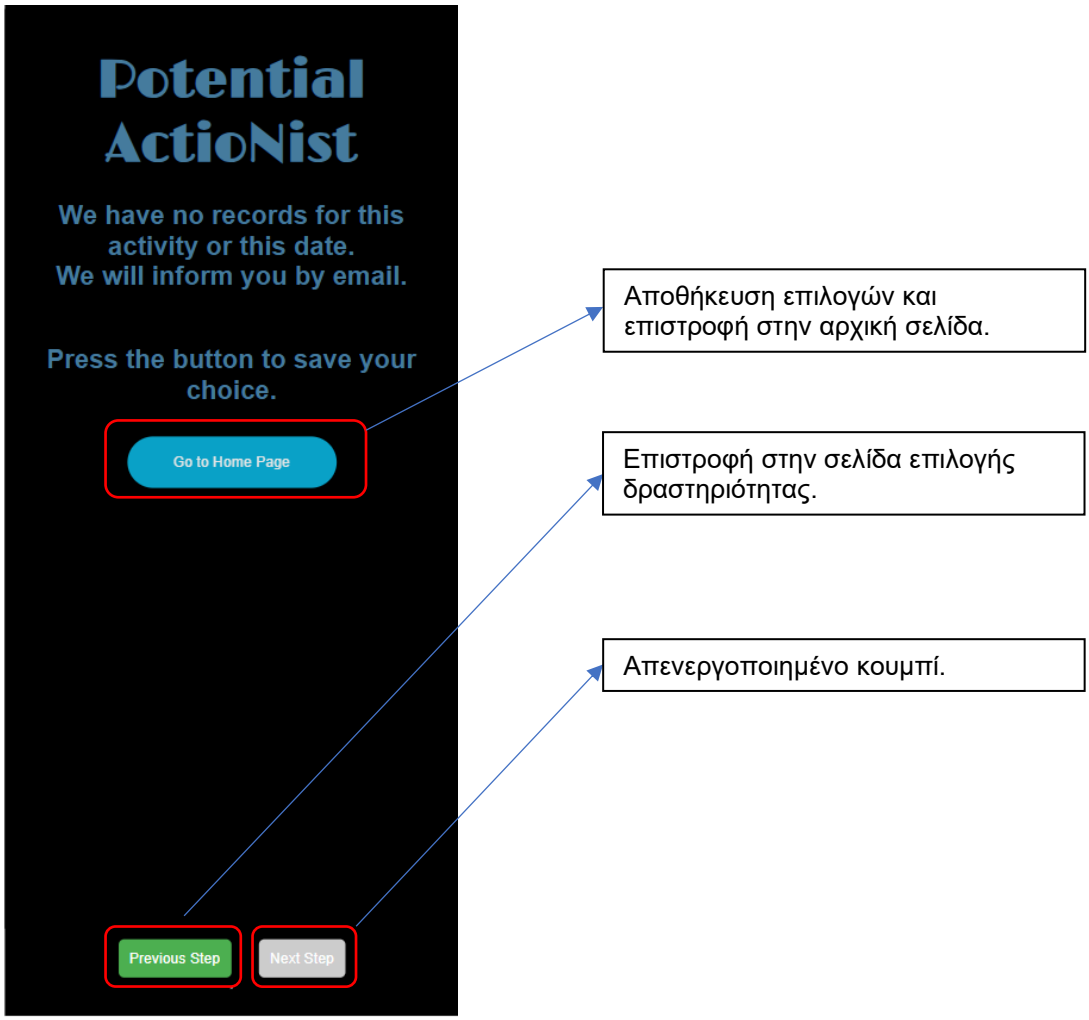
- Τρέξιμο (Running)
- Ποδηλασία (Cycling)
- Σκάκι (Chess)
- Πεζοπορία (Hiking)
- Καλαθοσφαίριση (Basketball)
- Μπιλιάρδο (Billiards)
- Αμαξάδα (Road Trip)
- Αντισφαίριση (Tennis)



Εικόνα 52 Απεικόνιση και σχολιασμός επιλογής δραστηριότητας (Βήμα 2ο)

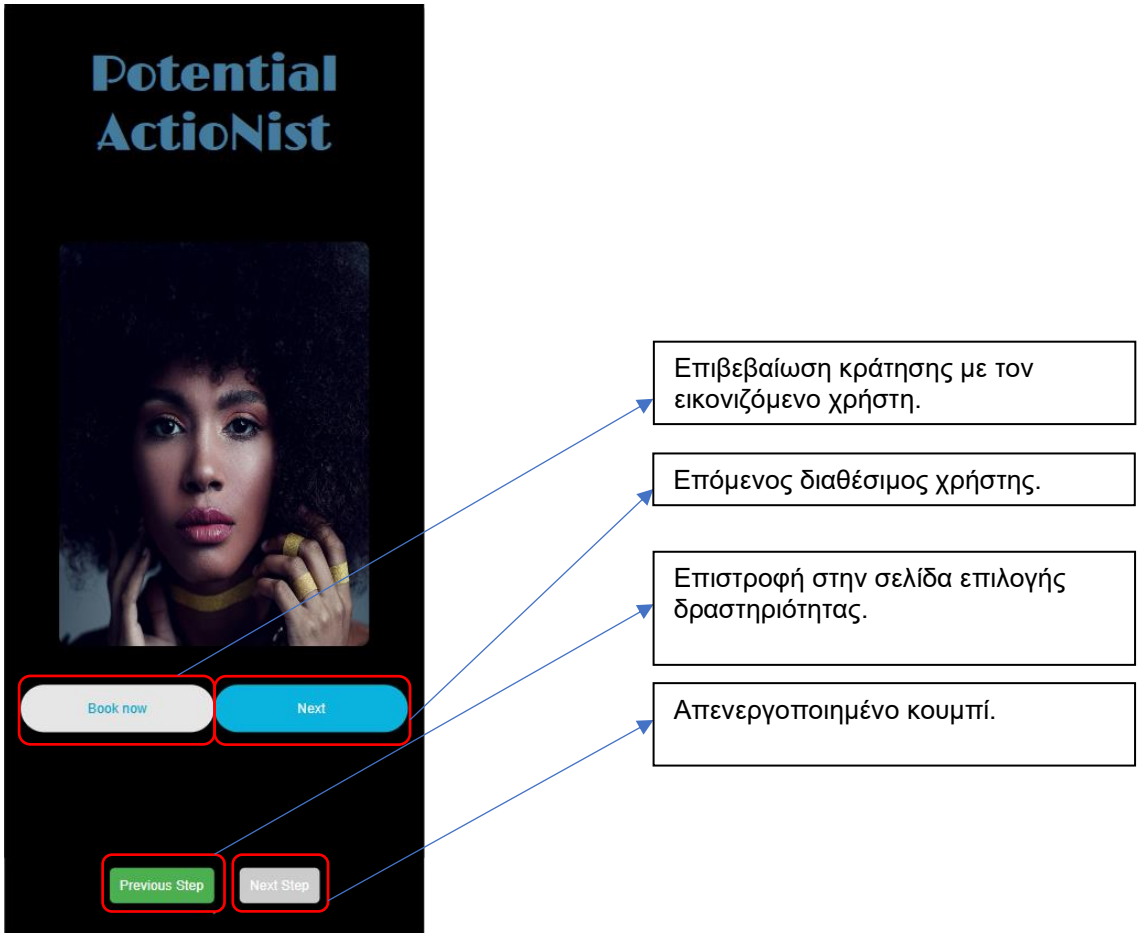
3.6 Σελίδα επιλογής συναθλητή

Έχοντας κάνει την επιλογή της ημερομηνίας και της δραστηριότητας τότε υπάρχουν δυο σενάρια που ακολουθούνται από την εφαρμογή. Αν δεν υπάρχουν διαθέσιμοι χρήστες όπου να έχουν επιλέξει τις ιδίες επιλογές τότε εμφανίζεται η σελίδα ειδοποίησης. Ας αναλύσουμε την περίπτωση πρώτα που δεν υπάρχουν προς το παρόν διαθέσιμοι χρήστες για τις επιθυμητές επιλογές του χρήστη.



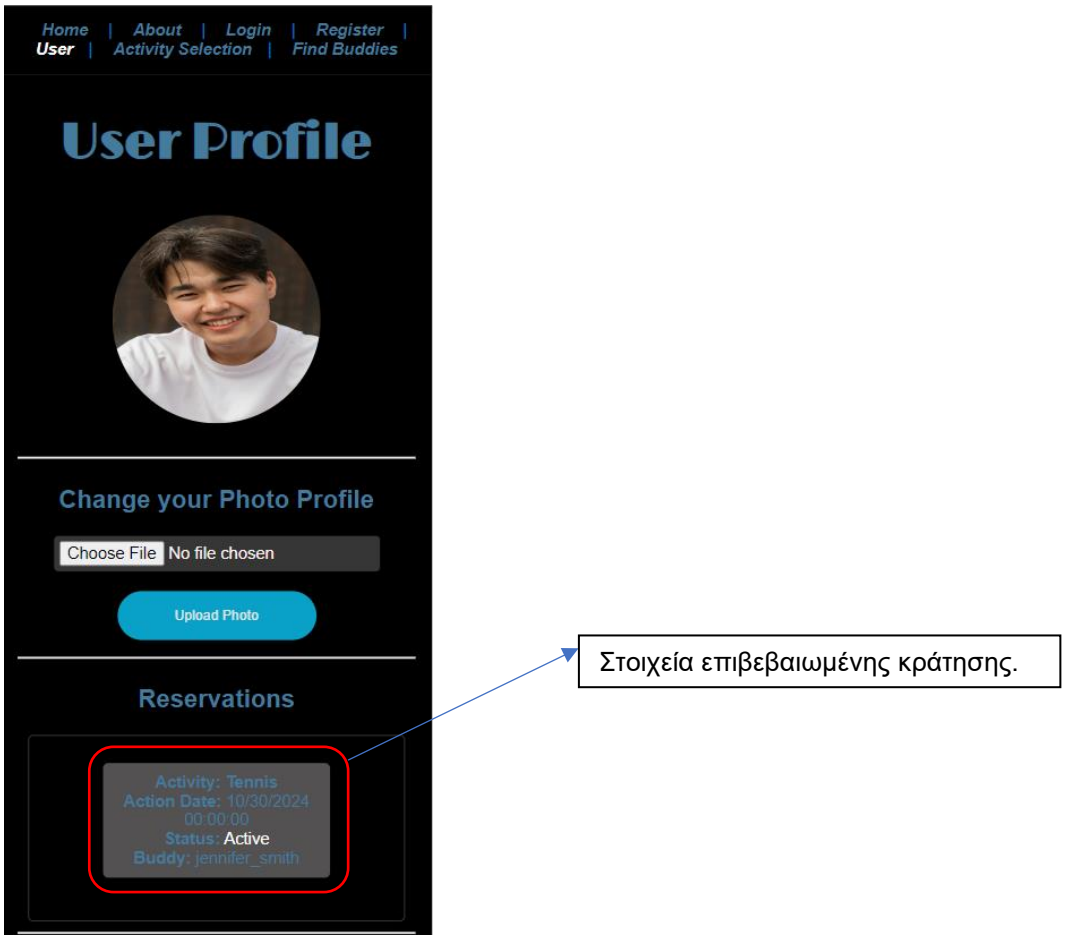
Εικόνα 53Απεικόνιση και σχολιασμός σελίδας μη διαθέσιμου συναθλητή (Βήμα 3ο)

Στην περίπτωση όμως που υπάρχουν διαθέσιμοι χρήστες για την δραστηριότητα και την ημερομηνία που ο χρήστης έχει επιλέξει τότε η σελίδα διαμορφώνεται όπως παρακάτω. Εμφανίζεται η εικόνα του προφίλ του χρήστη (αν υπάρχει) και οι επιλογές να τον διαλέξει και να γίνει η κράτηση ή να συνεχίσει με τον επόμενο. Επίσης εδώ υπάρχουν δυο σενάρια στην εφαρμογή, είτε γίνεται η κράτηση είτε πάει στον επόμενο υποψήφιο, αν οι διαθέσιμοι χρήστες εξαντληθούν θα εμφανιστεί πάλι η Εικόνα 37.



Εικόνα 54 Απεικόνιση και σχολιασμός σελίδας διαθέσιμου συναθλητή (Βήμα 3ο)

Αν ο χρηστης επιλεξει την επιλογη κρατησης της δραστηριοτητας τοτε θα δει στο προφιλ του να εμφανιζεται η κρατηση η οποια εχει γινει. Ετσι θα μπορει να βλεπει κάθε στιγμη τις επιβεβαιωμενες κρατησεις του.



Εικόνα 55 Απεικόνιση και σχολιασμός σελίδας προφίλ χρήστη

Επεκτασιμότητα

Τέλος θα ήθελα να αναφέρω, ότι κατά την διάρκεια της υλοποίησης της εφαρμογής, έχουν ληφθεί αποφάσεις που βασίζονται στις γνώσεις και τις δεξιότητες που απέκτησα κατά τη διάρκεια του μεταπτυχιακού προγράμματος. Αυτές οι αποφάσεις επιδιώκουν την επιτυχή υλοποίηση και βελτίωση της εφαρμογής, λαμβάνοντας υπόψη τις καλύτερες πρακτικές και τις σύγχρονες τεχνολογικές απαιτήσεις.

Ακόμα έχουν γίνει σκέψεις για πιθανές βελτιώσεις της εφαρμογής όπως οι παρακάτω:

- **Real-Time Ενημερώσεις**: Εφαρμογή real-time ενημερώσεις των χρηστών.
- **Ενσωμάτωση Αναλυτικών**: Χρήση εργαλείων αναλυτικών δεδομένων για την παρακολούθηση της απόδοσης της εφαρμογής και την εξαγωγή ερευνητικών στατιστικών.
- **Προσαρμογή Χρήστη**: Εφαρμογή λειτουργιών προσαρμογής για τους χρήστες, όπως προτάσεις για ημερομηνίες ή δραστηριότητες με βάση την τοποθεσία τους.
- **Ανάπτυξη Κοινότητας**: Δημιουργία chat μεταξύ των χρηστών, κατά την χρήση σε πραγματικό χρόνο

Βιβλιογραφία

[1] “Vue.js”[Online]. Available:
https://en.wikipedia.org/wiki/Vue.js

[2] “Vue.js - Το προηγμένο πλαίσιο JavaScript για σύγχρονες διαδικτυακές εφαρμογές” [Online]. Available:
https://www.mprofi.gr/tekhnologies/vuejs

[3] “Overview of .NET Framework” [Online]. Available:
https://learn.microsoft.com/en-us/dotnet/framework/get-started/overview

[4] “.NET” [Online]. Available:
https://en.wikipedia.org/wiki/.NET

[5] “The easy mocking library for .NET” [Online]. Available:
https://fakeiteasy.github.io/

[6] “Postgres SQL” [Online]. Available:
https://en.wikipedia.org/wiki/PostgreSQL

[7] “DevOps Pipeline” [Online]. Available:
https://www.atlassian.com/devops/devops-tools/devops-pipeline

[8] “Jenkins Handbook” [Online]. Available:
https://www.jenkins.io/doc/book/

[9] “Webhooks documentation” [Online]. Available:
https://docs.github.com/en/webhooks