ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ανάπτυξη ιστοσελίδας κριτικής παιχνιδιών και ταινιών

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

των

ΚΥΡΓΙΔΗ ΑΠΟΣΤΟΛΟΥ

4230

ΚΑΛΟΥΛΗ ΣΤΕΦΑΝΟΥ

4426

Επιβλέπων : Βέργαδος Ι. Δημήτριος

Πρόεδρος του τμήματος

Καστοριά Μήνας - Έτος (παρουσίασης της εργασίας)

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ανάπτυξη ιστοσελίδας κριτικής παιχνιδιών και ταινιών

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

των

ΚΥΡΓΙΔΗ ΑΠΟΣΤΟΛΟΥ

4230

ΚΑΛΟΥΛΗ ΣΤΕΦΑΝΟΥ

4426

Επιβλέπων : Βέργαδος Ι. Δημήτριος

Πρόεδρος του τμήματος

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την ημερομηνία εξέτασης

.....	.....	.....
Ον/μο Μέλους	Ον/μο Μέλους	Ον/μο Μέλους
Ιδίοτητα Μέλους	Ιδίοτητα Μέλους	Ιδίοτητα Μέλους

Καστοριά Μήνας - Έτος (παρουσίασης της εργασίας

Ευχαριστίες

Η παρούσα πτυχιακή εργασία ολοκληρώθηκε μετά από πολύ μεγάλη προσπάθεια και πολλές δυσκολίες. Στην προσπάθεια αυτή μας βοήθησαν ιδιαιτέρως ο επιβλέπων καθηγητής μας Δημήτριος Βέργαδος, καθώς και οι οικογένειές μας και οι κοντινοί μας φίλοι Αματέο Αλίου, Ιωάννης Πετρούσης και Σάκης Δούρβας.

Σε όλους όσους συνέβαλαν, άμεσα ή έμμεσα, στην ολοκλήρωση αυτής της προσπάθειας, εκφράζουμε τις θερμές μας ευχαριστίες.

Περίληψη

Η εργασία αυτή ασχολείται με την ανάπτυξη και σχεδίαση μιας ολοκληρωμένης web εφαρμογής που αφορά την προβολή και συγγραφή κριτικών για παιχνίδια και ταινίες που μπορεί να κάνει ένας χρήστης. Αναπτύχθηκε με σύγχρονες web τεχνολογίες τόσο front end όσο και back end και παρέχει στους χρήστες εύκολη αναζήτηση ταινιών και παιχνιδιών, δυνατότητα προσθήκης σε λίστα (Watchlist, Game list) και δυνατότητα συγγραφής κριτικών.

Λέξεις Κλειδιά: ιστοσελίδα, βιντεοπαιχνίδια, ταινίες, σειρές, κριτικές, ψυχαγωγία, διασκέδαση, άρθρα, ενημέρωση, προγραμματισμός, διακόσμηση, βάση δεδομένων, front-end, back-end, full-stack

Abstract

The following thesis is about the development and design of an integrated web application that involves viewing and writing reviews for games and movies that a user can make. It was developed using modern web technologies for both front-end and back-end and provides users with flexibility in searching movies and games, the ability to add a movie or a game to their list (Watchlist, Game list) and the ability to write reviews.

Key Words: website, video games, movies, series, reviews, entertainment, articles, up-to-date information, programming, styles, database, front-end, back-end, full-stack

Πίνακας Περιεχομένων

Περιεχόμενα

Λίστα Εικόνων

Λίστα Πινάκων

Εισαγωγή

Η πτυχιακή μας εργασία έχει ως στόχο τη δημιουργία μιας ιστοσελίδας, η οποία θα λειτουργεί ως πλατφόρμα για κριτικές και αξιολογήσεις σχετικά με παιχνίδια και ταινίες. Σκοπός είναι να παρέχουμε στους χρήστες τη δυνατότητα να ενημερώνονται, να αξιολογούν και να μοιράζονται τις απόψεις τους για διάφορους τίτλους, συνεισφέροντας ποικίλες απόψεις, που θα διευκολύνουν την επιλογή τους για ψυχαγωγικό περιεχόμενο.

Η δομή της εργασίας διαρθρώνεται ως εξής:

- Κεφάλαιο 1: Front-end τεχνολογίες

  Στο κεφάλαιο αυτό αναλύονται οι τεχνολογίες που χρησιμοποιήθηκαν για το front-end κομμάτι της εφαρμογής.

- Κεφάλαιο 2: Back-end τεχνολογίες

  Στο κεφάλαιο αυτό αναλύονται οι τεχνολογίες που χρησιμοποιήθηκαν για το back-end κομμάτι της εφαρμογής.

- Κεφάλαιο 3: GitHub

  Στο κεφάλαιο αυτό αναλύεται το GitHub και οι εντολές του για να μπορούμε να το χρησιμοποιήσουμε.

- Κεφάλαιο 4: Αρχιτεκτονική του CineGame Critic

  Στο κεφάλαιο αυτό γίνεται ανάλυση της δομής του ιστότοπου και παρουσιάζονται σε βάθος οι τεχνολογίες και τα εργαλεία που χρησιμοποιήθηκαν για την κατασκευή του

- Κεφάλαιο 4: Ανάπτυξη του CineGame-critic

  Στο κεφάλαιο αυτό παρουσιάζονται τα εργαλεία που χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής καθώς επίσης και κάποια παραδείγματα του πηγαίου κώδικα.

- Κεφάλαιο 5: Build-Deploy του CineGame-critic

  Στο κεφάλαιο αυτό αναλύουμε τη διαδικασία του build και deploy της εφαρμογής, η οποία αναπτύχθηκε με το Next.js και φιλοξενείται στην πλατφόρμα Vercel.

- Κεφάλαιο 6: After-Launch: Ρύθμιση & Βελτιστοποίηση της σελίδας

  Στο κεφάλαιο αυτό περιγράφουμε τα βήματα που ακολουθήσαμε μετά το deployment της σελίδας. Αναλύουμε τον τρόπο απόκτησης του domain

name, το google search console, καθώς επίσης και κάποιες μεθόδους παραμετροποίησης για SEO.

- Κεφάλαιο 7: Περιγραφή User Interface του CineGame-critic

  Στο κεφάλαιο αυτό παρουσιάζονται screenshots της σελίδας και η περιγραφή τους σχετικά με τα βασικά χαρακτηριστικά της εφαρμογής.

- Συμπεράσματα

  Στο κεφάλαιο αυτό γίνεται μια ανασκόπηση της εργασίας και παρουσιάζονται τόσο τα συμπεράσματα, όσο και για μελλοντικά σχέδια και περαιτέρω λειτουργίες της εφαρμογής.

- Βιβλιογραφία

  Στο κεφάλαιο αυτό φαίνεται η βιβλιογραφία που χρησιμοποιήθηκε για τη δημιουργία της πτυχιακής εργασίας μας.

1. Front-End Τεχνολογίες

Εδώ θα αναλυθούν οι τεχνολογίες που χρησιμοποιήθηκαν για το front-end κομμάτι της εφαρμογής. Δηλαδή για όσα αφορούν την μεριά του χρήστη (client-side).

1.1 React

Η React είναι μια δημοφιλής βιβλιοθήκη της JavaScript και χρησιμοποιείται για την κατασκευή διεπαφών χρήστη (User Interfaces - UIs). Αναπτύχθηκε αρχικά από το Facebook και έγινε διαθέσιμη στο ευρύ κοινό το 2013. Χρησιμοποιείται ιδιαιτέρως στην ανάπτυξη εφαρμογών μονής σελίδας (Single Page Applications - SPAs), όπου η απόκριση της εφαρμογής πρέπει να είναι άμεση και να μην πρέπει να περιμένει ο χρήστης την πλήρη ανανέωση της σελίδας για κάθε αλλαγή.

Κάποια κύρια χαρακτηριστικά της React είναι:

- **Virtual DOM:** Αρχικά το DOM είναι μια αναπαράσταση του HTML αρχείου της ιστοσελίδας ως δέντρο, δηλαδή ενημερώνει τον browser για την σωστή διάταξη του περιεχομένου. Το Virtual DOM είναι μια εικονική τεχνική, όπου αντικαθιστά την καθυστέρηση που έχει το πραγματικό DOM να φορτώσει τα δεδομένα σε κάθε αλλαγή της σελίδας. Όταν κάνει μια αλλαγή ο χρήστης, η React δημιουργεί νέο Virtual DOM με την ενημερωμένη κατάσταση και συγκρίνει την τρέχουσα με την προηγούμενη. Έτσι εντοπίζει μόνο τις αλλαγές που έχουν γίνει και ενημερώνει μόνο τα κατάλληλα σημεία στο DOM, βελτιώνοντας σημαντικά την απόδοση.
- **Components:** Τα components είναι ανεξάρτητα και επαναχρησιμοποιούμενα κομμάτια κώδικα. Έχουν τον ίδιο σκοπό με τις συναρτήσεις της JavaScript, με τη διαφορά ότι λειτουργούν απομονωμένα και επιστρέφουν περιεχόμενο html. Μια δυνατότητα της React είναι το πέρασμα των props (properties) στα components, τα οποία χρησιμοποιούνται για να ανταλλάσσουν δεδομένα μεταξύ τους.
- **React Hooks:** Τα hooks της React μας επιτρέπουν να χρησιμοποιούμε διάφορες δυνατότητες της React στα components μας. Μας παρέχεται η δυνατότητα χρήσης των ήδη έτοιμων hooks που προσφέρει η React, καθώς και η δημιουργία custom hooks. Κάποια βασικά Hooks είναι τα:
    - **State Hooks:** Ένα state (κατάσταση), είναι ένα αντικείμενο javascript που χρησιμοποιείται για να αποθηκεύσει και να διαχειριστεί δεδομένα των components. Όταν ένα state αλλάζει, η React θα κάνει επανεκτέλεση (re-render) του component, ενημερώνοντας το UI να δείχνει το καινούργιο state.

Η προσθήκη state σε ένα component γίνεται με το εξής hook:

useState: ορίζει ένα state σε μια μεταβλητή την οποία μπορούμε να ενημερώσουμε άμεσα.

- **Effect Hooks:** Τα effects επιτρέπουν σε ένα component να συνδέεται και να συγχρονίζεται με εξωτερικά συστήματα. Αυτό συμπεριλαμβάνει το δίκτυο, το DOM του browser, τα animations και οτιδήποτε δεν είναι κώδικας της React.

Η σύνδεση ενός component με κάποιο εξωτερικό σύστημα (όπως κάποιο API ή το localstorage του browser) γίνεται με το useEffect.

- **Context Hooks:** Το context επιτρέπει σε ένα component να λαμβάνει πληροφορίες από μακρινούς γονείς χωρίς να χρειάζεται να περαστούν ως props.

Η χρήση του context γίνεται με το useContext.

Λόγω της ευελιξίας και της αποδοτικότητάς της, η React έχει καθιερωθεί ως μία από τις κορυφαίες επιλογές για την ανάπτυξη frontend εφαρμογών και αποτελεί αναπόσπαστο μέρος του μοντέρνου οικοσυστήματος ανάπτυξης ιστού.

1.1.1 Client components

Ο όρος client (πελάτης) αναφέρεται στο πρόγραμμα περιήγησης σε μια συσκευή του χρήστη που στέλνει κάποιο αίτημα στον server για τις ανάγκες της εφαρμογής. Στη συνέχεια, μετατρέπει την απάντηση που λαμβάνει από τον server σε μια διεπαφή διαχειρίσιμη από τον χρήστη.

Τα Client Component είναι ένα χαρακτηριστικό της React, που αναφέρονται σε components, τα οποία εκτελούνται αποκλειστικά από την μεριά του χρήστη και λειτουργούν σε περιβάλλον του εκάστοτε προγράμματος περιήγησης. Μας επιτρέπουν να φτιάχνουμε διαδραστικά UI που προαποδίδονται πρώτα στον διακομιστή (server), εξασφαλίζοντας ταχύτητα στη φόρτωση στης σελίδας. Αφού η σελίδα φορτώσει, η JavaScript ενεργοποιείται στον browser και προσθέτει διαδραστικότητα.

Οφέλη του Client-Side Rendering

- **Διαδραστικότητα:** Με τα Client Components, ο προγραμματιστής μπορεί να χρησιμοποιήσει use states, use effects, και event listeners, που σημαίνει ότι μπορούν να έχουν απευθείας ανταπόκριση στον χρήστη.

- **APIs του προγράμματος περιήγησης:** Τα Client Components έχουν πρόσβαση σε ενσωματωμένα browser APIs, όπως είναι το geolocation ή το localstorage.

Χρήση/Δήλωση των Client Components

Για να χρησιμοποιήσει ο χρήστης τα Client Components, θα πρέπει να προσθέσει την εντολή “use client” στην αρχή του αρχείου και πριν από όλα τα imports. Το εξής παράδειγμα παρατίθεται από το documentation της Next:

```
"use client";

import { useState } from "react";

const counter = () => {
    const [count, setCount] = useState(0);
  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>Click me</button>
    </div>
  );
};

export default counter;
```

Όπου εδώ, σε περίπτωση που δεν δηλωθεί το “use client”, θα εμφανισθεί error στο terminal ότι η εντολή δεν προστέθηκε.

Απόδοση (Rendering) των Client Components

Η απόδοση διαφέρει, αναλόγως με το αν αναφερόμαστε σε πλήρη φόρτωση σελίδας, δηλαδή στην περίπτωση που ο χρήστης επισκέπτεται για πρώτη φορά τη σελίδα ή αν πλοηγείται σε κάποια υποσελίδα.

Για να βελτιστοποιήσει την αρχική φόρτωση της σελίδας, το Next.js χρησιμοποιεί τα APIs του React τόσο για τα Client Components όσο και για τα Server Components που θα αναφερθούν πιο κάτω. Αυτό σημαίνει ότι όταν ο χρήστης επισκέπτεται για πρώτη φορά την εφαρμογή σας, θα δει το περιεχόμενο της σελίδας αμέσως, χωρίς να περιμένει να κατέβει η client-side JavaScript για τον browser. Κατά τις επόμενες πλοηγήσεις ωστόσο, η απόδοση εξαρτάται πλήρως από τον client, που σημαίνει ότι η JavaScript κατεβαίνει και αναλύεται.

1.1.2 Server components

Ο όρος server (διακομιστής) αναφέρεται σε ένα πρόγραμμα υπολογιστή ή μια συσκευή που παρέχει υπηρεσίες σε άλλα προγράμματα ή συσκευές, που ονομάζονται clients (πελάτες). Στην περίπτωση μιας web εφαρμογής, ο server λαμβάνει αιτήματα από τους clients (π.χ., αιτήματα για την εμφάνιση μιας σελίδας ή για την αποστολή δεδομένων) και απαντά σε αυτά τα αιτήματα παρέχοντας τις ζητούμενες πληροφορίες ή εκτελώντας τις ζητούμενες ενέργειες. Ο server μπορεί επίσης να αποθηκεύει και να διαχειρίζεται δεδομένα, να ελέγχει την πρόσβαση σε πόρους, και να εκτελεί άλλες εργασίες για την υποστήριξη της εφαρμογής.

Τα server components, είναι ένα νέο είδος component στην React που αντί να εκτελούνται στον browser του χρήστη, εκτελούνται στον server. Αυτό σημαίνει ότι ο κώδικας τους δε συμπεριλαμβάνεται στο JavaScript που αποστέλλεται στον client. Με αυτό τον τρόπο Βελτιώνεται η απόδοση της σελίδας καθώς μειώνεται το μέγεθος του JavaScript, γίνεται ευκολότερη και καλύτερη η διαχείριση των δεδομένων μιας και δε χρειάζεται κάποιο API για να μπορούν τα components να προσπελάσουν βάσεις δεδομένων. Ακόμα, μπορούμε να έχουμε καλύτερη ασφάλεια αποθηκεύοντας πληροφορίες όπως API Keys μακριά από τον client.

Υπάρχουν 3 τρόποι για server-side rendering.

1. **Static Rendering:** Με το static rendering όλες οι διαδρομές (routes) γίνονται rendered κατά τη στιγμή δημιουργίας ή στο παρασκήνιο μετά την επικύρωση των δεδομένων. Το αποτέλεσμα αποθηκεύεται στην προσωρινή μνήμη (cache) και μπορεί να προωθηθεί σε ένα δίκτυο παράδοσης περιεχομένου (CDN-Content Delivery Network). Αυτή η βελτιστοποίηση μας επιτρέπει να μοιραζόμαστε το αποτέλεσμα της απόδοσης μεταξύ των χρηστών και τον αιτήσεων του server.
2. **Dynamic Rendering:** Με το dynamic rendering οι διαδρομές αποδίδονται στον χρήστη κατά τη στιγμή του αιτήματος. Αυτό είναι χρήσιμο όταν μια διαδρομή πρέπει να περιλαμβάνει δεδομένα που είναι εξατομικευμένα για τον χρήστη ή έχει πληροφορίες που μπορούν να γίνουν γνωστές μόνο κατά τη στιγμή της αίτησης, όπως είναι τα cookies ή οι παράμετροι αναζήτησης ενός URL.
3. **Streaming:** Το streaming επιτρέπει στο UI μας να κάνει render σταδιακά από τον διακομιστή. Η εργασία χωρίζεται σε κομμάτια και μεταδίδεται στον client μόλις είναι έτοιμη. Αυτό επιτρέπει στον χρήστη να μπορεί να δει κομμάτια της ιστοσελίδας αμέσως, χωρίς να χρειάζεται να ολοκληρωθεί το rendering ολόκληρου του περιεχομένου. Το streaming υπάρχει ήδη από προεπιλογή στον app router της Next.js, βελτιώνοντας την απόδοση της αρχικής σελίδας, καθώς

επίσης και κομμάτια του UI που εξαρτώνται από πιο αργά fetch δεδομένων που θα μπλόκαραν ολόκληρη τη διαδρομή (route).

Διαφορές Client και Server Component

Επιλέγουμε αν θέλουμε ένα component να είναι client ή server ανάλογα με τις ανάγκες που έχουμε:

What do you need to do?	Server Component	Client Component
Fetch data	✓	×
Access backend resources (directly)	✓	×
Keep sensitive information on the server (access tokens, API keys, etc)	✓	×
Keep large dependencies on the server / Reduce client-side JavaScript	✓	×
Add interactivity and event listeners (`onClick()`, `onChange()`, etc)	×	✓
Use State and Lifecycle Effects (`useState()`, `useReducer()`, `useEffect()`, etc)	×	✓
Use browser-only APIs	×	✓
Use custom hooks that depend on state, effects, or browser-only APIs	×	✓
Use React Class components ↗	×	✓

Εικόνα 1 . Τρόπος επιλογής client ή server component.

1.2. Next.js

Η Next.js είναι ένα framework της React που χρησιμοποιείται για τη δημιουργία full-stack web εφαρμογών. Χρησιμοποιούνται τα components της React για τη δημιουργία UI και η Next.js για περισσότερες δυνατότητες και βελτιστοποιήσεις.

Σαν framework εμπεριέχει αυτόματη μεταγλώττιση, κατηγοριοποίηση και άλλες λειτουργίες που ελαχιστοποιούν το χρόνο για διαμόρφωση από τη μεριά του χρήστη.

Κάποια κύρια χαρακτηριστικά της Next.js είναι τα:

Πίνακας 1 . Κύρια χαρακτηριστικά της Next.js

Routing (Δρομολόγηση)	Ένας δρομολογητής βασισμένος σε σύστημα αρχείων που έχει χτιστεί πάνω σε server components και υποστηρίζει ένθετη δρομολόγηση (nested routing), διατάξεις (layouts), χειρισμό σφαλμάτων και άλλα.
Typescript	Βελτιωμένη υποστήριξη για Typescript, με καλύτερους ελέγχους τύπων και πιο αποδοτική μεταγλώττιση (compile), καθώς και custom Typescript plugin.
Styling	Υποστήριξη CSS modules, tailwind CSS, CSS-in-js και άλλα.
Data fetching	Απλοποιημένο fetching των δεδομένων με τη χρήση async/await στα Server components.
Rendering (εκτέλεση)	Διάκριση εκτέλεσης σε πλευρά του client και πλευρά του server, με διαφορετικά components στο καθένα και ξεχωριστές δυνατότητες, έτσι ώστε να υπάρχει ποικιλία στον τρόπο διαμόρφωσης της σελίδας.
Optimizations (Βελτιστοποιήσεις)	Όπως εικόνες και βελτιστοποιήσεις των scripts για καλύτερη εμπειρία του χρήστη.

Η Next.js χρησιμοποιεί δύο διαφορετικούς δρομολογητές: Τον app router και τον pages router.

App router

Ο app router της next.js χρησιμοποιεί ένα μοντέλο το οποίο μπορεί και χρησιμοποιεί τις τελευταίες δυνατότητες της React (όπως server components, server actions) που δεν ήταν εφικτές με τον παραδοσιακό pages router.

Ο app router λειτουργεί μέσα σε έναν φάκελο που λέγεται “app”. Από προεπιλογή όσα components βρίσκονται μέσα σε αυτό τον φάκελο είναι server components που βοηθάει στην απόδοση της ιστοσελίδας. Υπάρχει επίσης δυνατότητα για client components.

Η δρομολόγηση μέσα στον app router λειτουργεί ως εξής: Οι φάκελοι χρησιμοποιούνται για να καθορίσουν διαδρομές (routes). Κάθε φάκελος αναπαριστά ένα τμήμα διαδρομής που αντιστοιχίζεται σε ένα τμήμα URL. Για να οριστεί μια διαδρομή που να είναι προσβάσιμη από τον χρήστη δημιουργούμε ένα αρχείο“page.js (ή page.tsx για typescript)” μέσα στον φάκελο που αντιστοιχεί σε αυτή τη διαδρομή.

Pages router

Ο pages router της Next αποτελεί έναν σχετικά παλιό τρόπο δημιουργίας διαδρομών μεταξύ των αρχείων. Φτιάχνει διαδρομές διαισθητικά και τις αποδίδει στα αρχεία σε αντίθεση με τον app router ο οποίος προσφέρει μια πιο μοντέρνα προσέγγιση στη δημιουργία διαδρομών.

Στατική Δρομολόγηση

Οι διαδρομές αποδίδονται με βάση τη δομή των αρχείων στον φάκελο pages/, όπου το index.(ts,tsx,js) αποτελεί το default μονοπάτι. Επομένως το αρχείο pages/index.tsx αντιστοιχεί στη διαδρομή “/”, ενώ ένα αρχείο με δομή pages/contact.tsx αντιστοιχεί στη δομή /contact. Αντίστοιχα λειτουργεί και η δρομολόγηση στα API routes.

Δυναμική Δρομολόγηση

Οι διαδρομές αποδίδονται με βάση τις αγκύλες στα ονόματα των αρχείων. Δηλαδή ένα αρχείο pages/profile/[id].js, θα αντιστοιχεί σε διαδρομές όπως profile/1 ή profile/my-details.

Συνολικά, ο pages router είναι ένας δρομολογητής που χρησιμεύει στην δημιουργία μιας απλοϊκής εφαρμογής που ο χρήστης θέλει να στήσει γρήγορα διαδρομές και δεν χρειάζεται σύνθετες λειτουργίες από την διεπαφή του χρήστη.

1.3 Typescript

Για να καταλάβει κανείς πως δομείται η Typescript, θα πρέπει πρώτα να κατανοήσει την σύνταξη της JavaScript. Αρχικά η JavaScript δημιουργήθηκε το 1995 και σκοπός της ήταν να μπορεί κάποιος να γράψει γρήγορα μικρά scripts. Ωστόσο με την έκδοση ES6 το 2015 η JavaScript απέκτησε μεγάλη δημοτικότητα με αποτέλεσμα να χρησιμοποιείται από πολλούς για όλες τις χρήσεις (ακόμα και για desktop εφαρμογές). Προστέθηκαν μεγάλες αλλαγές στη γλώσσα όπως είναι: κλάσεις, modules, arrow functions κ.α. Ωστόσο, η JavaScript παρουσιάζει ένα βασικό μειονέκτημα και είναι ότι οι έλεγχοι σε τύπους μεταβλητών δεν γίνονται πάντα σωστά.

Για παράδειγμα:

```
const obj = { width: 10, height: 15 };
// Why is this NaN? Spelling is hard!
const area = obj.width * obj.heigth;
```

Εικόνα 2 . Έλεγχοι τύπων JavaScript.

Αυτό το τυπογραφικό λάθος δε θα εμφάνιζε error στην JavaScript, κάτι το οποίο σε κώδικα με χιλιάδες γραμμές κάνει το debugging αρκετά περίπλοκο και δύσκολο. Εδώ έρχεται και εφαρμόζει η Typescript η οποία είναι μια γλώσσα προγραμματισμού ανοιχτού κώδικα που δημιουργήθηκε το 2012 και αναπτύχθηκε από τη Microsoft. Περιλαμβάνει όλες τις δυνατότητες της JavaScript και προσθέτει επιπλέον λειτουργίες όπως είναι η στατική τυποποίηση, που μπορεί και ανιχνεύει τα σφάλματα στον κώδικα χωρίς να χρειαστεί εκτέλεση.

Με τον όρο στατική τυποποίηση, εννοούμε ότι οι χρήστες μπορούν να δηλώνουν τύπους για μεταβλητές, παραμέτρους και επιστρεφόμενες τιμές συναρτήσεων όπως στα παρακάτω παραδείγματα:

```
const obj = { width: 10, height: 15 };
const area = obj.width * obj.heigth;
```

```
Property 'heigth' does not exist on type '{ width: number; height: number; }'. Did you mean 'height'?
```

Εικόνα 3 . Έλεγχοι τύπων Typescript.

ή

```
function getName(name: String) {console.log(name)}
```

Συνεπώς, βελτιώνεται η συνολική εμπειρία του χρήστη, καθώς ο προγραμματιστής δεν χρειάζεται να αφιερώνει τόσο χρόνο στο debugging και ο κώδικας έχει καλύτερη οργάνωση και αναγνωσιμότητα.

1.4 Tailwind CSS

Η Tailwind CSS είναι ένα utility-first framework της CSS που δημιουργήθηκε το 2017 από τον Adam Wathan. Σκοπός την είναι να απλοποιεί την ανάπτυξη ιστοσελίδων προσφέροντας ένα σετ προσχεδιασμένων κλάσεων χρησιμότητας (utility classes). Οι κλάσεις αυτές μας επιτρέπουν να φτιάχνουμε custom σχεδιασμούς χωρίς να χρειάζεται να φτιάξουμε custom CSS κώδικα. Για παράδειγμα αντί να χρειαστεί να γραφτεί CSS κώδικας για να αλλάξουμε το χρώμα ενός στοιχείου σε κόκκινο μπορούμε απλά να χρησιμοποιήσουμε την κλάση “text-red" της Tailwind CSS.

1.4.1 CSS

Η CSS (Cascading Style Sheets) είναι μια γλώσσα σήμανσης που δημοσιεύθηκε για πρώτη φορά το 1996 από τον Håkon Wium Lie. Καθορίζει τον τρόπο αναπαράστασης ενός εγγράφου HTML ή XML στην οθόνη, στον εκτυπωτή ή σε άλλα μέσα. Επιτρέπει στους προγραμματιστές να ελέγχουν τα χρώματα, τις γραμματοσειρές των στοιχείων και άλλα οπτικά χαρακτηριστικά, χωρίς να επηρεάσουν το περιεχόμενο.

1.4.2 Πλεονεκτήματα Tailwind CSS

- **Ελαχιστοποιημένη CSS:** Η Tailwind CSS μειώνει σημαντικά τα πολύ μεγάλα custom CSS αρχεία, βελτιώνοντας την οργάνωση και την αναγνωσιμότητα των αρχείων κώδικα μας.
- **Τοπικές μορφοποιήσεις:** Οι κλάσεις της Tailwind μπορούν να εφαρμοστούν τοπικά σε κάθε στοιχείο, χωρίς να υπάρχει ανάγκη για global καθορισμό των ιδιοτήτων, όπως στην παραδοσιακή CSS.
- **Δεν χρειάζεται να δώσουμε περίπλοκα ονόματα κλάσεων:** Μπορούμε να στοχεύσουμε απευθείας το κάθε στοιχείο χωρίς να χρειάζεται να δώσουμε κλάσεις και IDs.
- **Εύκολο customization:** Η tailwind CSS μας επιτρέπει εύκολη προσαρμογή σχεδίων χωρίς να χρειάζεται επιπρόσθετη CSS, κάτι που μας βοηθάει στο να δημιουργούμε επαναχρησιμοποιούμενα components.
- **Built-in Responsiveness:** Η tailwind CSS έχει έτοιμες κλάσεις που είναι βελτιστοποιημένες για responsiveness, που επιτρέπουν στους προγραμματιστές να φτιάχνουν εύκολα και γρήγορα φιλικές προς τα κινητά διατάξεις. Για παράδειγμα: Μπορούμε να γράψουμε sm:ml-4 σε ένα div, που σε κανονική CSS μεταφράζεται σε:

```
@media (min-width: 640px) {
    div {
        margin-left: 1rem;
    }
}
```

1.5 TMDB, RAWG APIs

1.5.1 TMDB

Το TMDB API (The Movie Database API) είναι ένα API που μας δίνει πρόσβαση σε μια πολύ μεγάλη βάση δεδομένων που περιλαμβάνει πολλές πληροφορίες για ταινίες, σειρές, βαθμολογίες και πολλά άλλα. Οι προγραμματιστές μπορούν εύκολα να τα χρησιμοποιήσουν τα δεδομένα αυτά για τη δημιουργία εφαρμογών που εμφανίζουν πληροφορίες σχετικά με τον κινηματογράφο.

Μερικά πλεονεκτήματα που μας παρέχει το TMDB είναι τα:

- **Ευκολία χρήσης:** Το συγκεκριμένο API είναι σχεδιασμένο να είναι εύκολο στη χρήση του και να γίνεται με απλά requests.
- **Πρόσβαση σε πάρα πολλά δεδομένα:** Το API αυτό μας προσφέρει πάρα πολλές πληροφορίες σχετικά με τον κινηματογράφο. Αυτό συμπεριλαμβάνει: τίτλους ταινιών, πλοκή ταινιών, cast, ομάδα παραγωγής, ηθοποιοί, μέση βαθμολογία, Εικόνα ταινίας, Προτάσεις για ταινίες που μοιάζουν με αυτή που βλέπει ο χρήστης, είδος ταινίας που μας βοηθάει να φτιάξουμε εύκολα φίλτρα, αναζήτηση, .
- **Ενεργή κοινότητα:** Υπάρχει μεγάλη και ενεργή κοινότητα χρηστών του TMDB API, κάτι που το καθιστά εύκολο να βρεθεί βοήθεια και υποστήριξη σε ό,τι χρειαστεί

Η χρήση του TMDB API είναι δωρεάν και η λήψη API key γίνεται από την επίσημη σελίδα τους. Όλα τα δεδομένα επιστρέφονται σε μορφή JSON και είναι εύκολο να πάρει κάποιος τα δεδομένα αυτά.

Παρακάτω δίνεται παράδειγμα κώδικα σε Next.js:

```
const getMovieData = async (page: string) => {
  const res = await
fetch(`${baseUrl}discover/movie?include_adult=false&page=${page}&${process.
env.MOVIE_API_KEY}`);
  const data = await res.json();
  return data;
};
```

Αυτός ο κώδικας ανακτά δεδομένα για ταινίες από το TMDB API, χρησιμοποιώντας τον αριθμό σελίδας (page) ως παράμετρο. Το baseUrl είναι ένα URL που μπορούμε να βρούμε εύκολα στην ιστοσελίδα του TMDB και το API key είναι αυτό που μας δίνεται από την σελίδα.

1.5.2 RAWG

Το RAWG API αποτελεί μια εκτενή βάση δεδομένων για βιντεοπαιχνίδια. Περιλαμβάνει αρκετούς τίτλους, καθώς και αρκετές πληροφορίες για αυτούς. Ένας προγραμματιστής που θα επιλέξει το API θα μπορεί να έχει πρόσβαση σε τίτλους, περιγραφές, βαθμολογίες, στιγμιότυπα των παιχνιδιών καθώς και πολύ υλικό ακόμη.

Εκτός όμως από την δυνατότητα λήψης της πληροφορίας, που βασίζεται στο ξεχωριστό “id” πεδίο κάθε παιχνιδιού, υπάρχει και η δυνατότητα φιλτραρίσματος. Οι χρήστες μπορούν να επιλέξουν τα δεδομένα που λαμβάνουν με βάση διάφορα κριτήρια, όπως είναι η ημερομηνία κυκλοφορίας του, το είδος του παιχνιδιού, τους εκδότες του παιχνιδιού ή και μαζικά κριτήρια όπως τα πιο δημοφιλή παιχνίδια για την τωρινή περίοδο.

Η χρήση του API είναι δωρεάν, αλλά απαιτείται εγγραφή και λήψη ενός API key για την αλληλεπίδραση με τις διάφορες υπηρεσίες που προσφέρει. Ωστόσο, να σημειωθεί ότι ο κάθε εγγεγραμμένος χρήστης έχει περιορισμένα requests (φορές που καλεί δεδομένα από το API) και αυτά ανανεώνονται μηνιαίως.

Όλα τα δεδομένα επιστρέφονται σε μορφή JSON και επομένως είναι ευανάγνωστα και απλοϊκά στη μορφή τους. Ένα παράδειγμα κλήσης του API από τα αρχεία είναι το εξής:

```
const getGameData = async (url: string, page: number) => {
  try {
    const res = await fetch(`${url}&page=${page}`);
    if (!res.ok) {
      throw new Error(`HTTP error! status: ${res.status}`);
    }
    const data = await res.json();
    if (!data || !data.results) {
      throw new Error("Invalid data structure");
    }
    return data.results as PostResult[];
```

```
  } catch (error) {
    console.error("Error fetching game data:", error);
    throw error;
  }
};
```

2. Back-End Τεχνολογίες

Εδώ θα αναλυθούν οι τεχνολογίες που χρησιμοποιήθηκαν για το back-end κομμάτι της εφαρμογής. Δηλαδή για όσα αφορούν την μεριά του διακομιστή (server-side).

2.1 MongoDB

Το MongoDB είναι η πιο δημοφιλής και πιο διαδεδομένη μη σχεσιακή βάση δεδομένων. Χρησιμοποιεί κείμενα JSON ή BSON για την αποθήκευση των δεδομένων. Η ανάπτυξη αυτού του συστήματος ξεκίνησε το 2007, όταν η εταιρεία 10gen (γνωστή σήμερα ως MongoDB Inc.) άρχισε να χρησιμοποιεί αυτήν τη βάση δεδομένων ως υπηρεσία για τους πελάτες της. Αυτή η υπηρεσία αποτελούνταν από μια εφαρμογή διακομιστή και μια βάση δεδομένων που θα μπορούσε να κλιμακωθεί ανάλογα με τις ανάγκες. Στους προγραμματιστές δεν άρεσε αυτή η υπηρεσία, επειδή χρειαζόταν να δώσουν πλήρη έλεγχο των δεδομένων τους σε μια εφαρμογή. Γι' αυτό το λόγο, η εταιρεία αποφάσισε να δημοσιεύσει το σύστημα βάσης δεδομένων της ως ανοιχτού κώδικα το 2009.

Ένα σχεσιακό σύστημα βάσης MySQL χρησίμευσε ως βάση, το οποίο τροποποιήθηκε σε μια βάση μη σχεσιακού τύπου με καταγραφή σε έγγραφα. Με αυτό τον τρόπο, το MongoDB απέκτησε κάποια από τα πλεονεκτήματα μιας σχεσιακής βάσης δεδομένων. Αυτό το σύστημα βάσης δεδομένων είναι σχετικά καινούργιο, επομένως εξακολουθεί να αναπτύσσεται. Τα κύρια χαρακτηριστικά του MongoDB είναι:

Ευελιξία: η δυνατότητα αποθήκευσης και επέκτασης εγγραφών χωρίς την ανάγκη καθορισμού ενός σταθερού σχήματος.

Απόδοση: Υποστηρίζει πολλά στοιχεία από συστήματα σχεσιακών βάσεων δεδομένων, όπως δυναμικές ερωτήσεις, ταξινομήσεις και κλειδώματα.

Ταχύτητα / Κλιμάκωση: Η βάση δεδομένων μπορεί να διανεμηθεί σε πολλούς διακομιστές, ώστε να μπορεί να ενισχύσει την απόδοση μιας λειτουργίας. Υποστηρίζει επίσης εύκολη και αυτόματη κλιμάκωση με την προσθήκη επιπλέον κόμβων. Η χωρητικότητα μπορεί να ενισχυθεί επίσης όταν η βάση δεδομένων λειτουργεί.

Εύκολη χρήση: Η εγκατάσταση είναι πολύ εύκολη και το MongoDB προσπαθεί το καλύτερο δυνατό για να διαμορφώσει τη βάση δεδομένων έτσι ώστε ο προγραμματιστής να έχει ελάχιστη δουλειά με τη ρύθμιση της βάσης δεδομένων.

Το MongoDB έχει πολλές επίσημες και ανεπίσημες βιβλιοθήκες για τη σύνδεση βάσης δεδομένων με διάφορες γλώσσες προγραμματισμού (PHP, Java, C, Python, κλπ.). Γι' αυτό τον λόγο, η ανάπτυξη εφαρμογών χρησιμοποιώντας το MongoDB είναι σχετικά εύκολη.

2.1.1 Δημιουργία Δεικτών

Ο σκοπός της δημιουργίας δεικτών είναι η αποτελεσματική εκτέλεση ερωτημάτων στη βάση δεδομένων. Αν δεν υπήρχαν δείκτες στη βάση δεδομένων, κάθε ερώτημα θα χρειαζόταν να εκτελεί σάρωση κάθε εγγράφου σε μια συλλογή, πράγμα που είναι υψηλά ανεπιθύμητο. Γι' αυτό τον λόγο, το MongoDB επιτρέπει τη δημιουργία δεικτών (64 για μια συλλογή). Ο δείκτης είναι μια ειδική δομή δεδομένων που αποθηκεύει ένα μέρος των δεδομένων από μια συλλογή που χρησιμοποιούνται ως συνθήκες ή χρησιμοποιούνται σε λειτουργίες με ερωτήματα. Από τη βάση δεδομένων διαβάζονται μόνο οι επιλεγμένες εγγραφές που πληρούν τις απαιτήσεις βάσει των δεικτών. Οι τιμές στους δείκτες είναι ταξινομημένες και είναι εύκολο να δουλέψει κανείς με αυτές. Το MongoDB υποστηρίζει επίσης τη δημιουργία δεικτών για έγγραφα που περιλαμβάνουν άλλα έγγραφα (nested documents).

2.1.2 Ασφάλεια

Η ασφάλεια των ευαίσθητων δεδομένων είναι ένα απαραίτητο μέτρο, επομένως, κατά την εκτέλεση ενός διακομιστή MongoDB, είναι απαραίτητο να διασφαλιστεί ότι ένας χρήστης και μια εφαρμογή έχουν πρόσβαση μόνο στα δεδομένα που χρειάζονται. Το MongoDB υποστηρίζει αρκετά χαρακτηριστικά που βοηθούν τον διαχειριστή να περιορίσει την πρόσβαση των μη εξουσιοδοτημένων χρηστών, όπως είναι:

- Ρύθμιση της εφαρμογής πριν από την πρώτη σύνδεση του πρώτου χρήστη.
- Δημιουργία λογαριασμών διαχειριστή και χρηστών με περιορισμένη πρόσβαση στη βάση δεδομένων.
- Κρυπτογραφημένη εισερχόμενη και εξερχόμενη επικοινωνία στο σύστημα.
- Περιορισμένη ακρόαση σε συγκεκριμένη θύρα.
- Επιλογή ενεργοποίησης καταγραφής λειτουργίας στη βάση δεδομένων.

2.1.3 Αντιγραφή

Το MongoDB υποστηρίζει την δημιουργία αντιγράφων εγγραφών μεταξύ των κόμβων της βάσης δεδομένων, βελτιώνοντας έτσι τη διαθεσιμότητα των δεδομένων στη βάση δεδομένων. Αυτό αυξάνει τον αριθμό των περιττών δεδομένων, αλλά αποτελεί πλεονέκτημα σε περίπτωση αποτυχίας ενός κόμβου, επειδή τα δεδομένα είναι προσβάσιμα από έναν άλλο κόμβο. Μπορεί επίσης να χρησιμοποιηθεί για εφαρμογές που απαιτούν υψηλή ταχύτητα ανάγνωσης, επειδή τα δεδομένα μπορούν να διαβαστούν από πολλούς κόμβους. Το σύνολο αντιγράφων αποτελείται από έναν πρωτεύον διακομιστή και περισσότερους δευτερεύοντες διακομιστές. Ο πρωτεύον διακομιστής λαμβάνει τόσο αιτήματα ανάγνωσης όσο και εγγραφής, αλλά ο δευτερεύων διακομιστής λαμβάνει μόνο αιτήματα ανάγνωσης. Ο πρωτεύον διακομιστής στέλνει επίσης ενημερώσεις ανά τακτά χρονικά διαστήματα στους δευτερεύοντες διακομιστές. Εάν ο πρωτεύον διακομιστής δεν είναι προσβάσιμος, θα ξεκινήσει η διαδικασία εκλογής για νέο πρωτεύον διακομιστή. Αυτός θα δημιουργηθεί από έναν δευτερεύον διακομιστή μετά τη διαδικασία. Εάν τα δεδομένα διαβάζονται μόνο από τον πρωτεύον διακομιστή, θα είναι συνεπή, αντιθέτως αν διαβάζονται και από τους δευτερεύοντες διακομιστές, ενδέχεται να μην είναι πλήρως συνεπείς λόγω των καθυστερημένων ενημερώσεων. Εκτός αυτού, τα αντίγραφα μπορούν να έχουν ειδικές εργασίες:

Κρυφό αντίγραφο: Αυτό το αντίγραφο είναι απομονωμένο από το σύστημα και δημιουργείται για λόγους ανάλυσης.

Αντίγραφο με καθυστέρηση: Οι εγγραφές σε αυτό το αντίγραφο προορίζονται για αντίγραφο ασφαλείας ή για εμφάνιση της κατάστασης με καθυστέρηση και σύγκριση με τα πραγματικά δεδομένα.

2.1.4 Αυτόματη Κατανομή

Το MongoDB υποστηρίζει την οριζόντια κλιμάκωση για τη διανομή της απόδοσης μεταξύ αρκετών διακομιστών για την απαλοιφή ενός ορίου απόδοσης σε έναν μόνο διακομιστή. Το MongoDB διανέμει αυτόματα τις συλλογές μεταξύ των διακομιστών αμέσως μετά την προσθήκη ενός νέου κόμβου στο σύστημα. Το σύστημα θα προσπαθήσει αυτόματα να ισορροπήσει τα δεδομένα μεταξύ των βάσεων δεδομένων. Αυτό θα απλοποιήσει το έργο του διαχειριστή(host).

Ο διαχειριστής μπορεί να ρυθμίσει την πολιτική ενός κατανεμημένου συστήματος: Κατανομή εύρους: τα έγγραφα διανέμονται ανάλογα με την τιμή του κλειδιού shard. Τα έγγραφα με παρόμοια τιμή θα τοποθετηθούν πιθανώς στον ίδιο διακομιστή. Κατανομή κατακερματισμού: τα έγγραφα διαιρούνται ανάλογα με την τιμή του hash MD5 που

δημιουργείται από το κλειδί shard. Τα έγγραφα με παρόμοια τιμή θα τοποθετηθούν επίσης, πιθανώς στον ίδιο διακομιστή.

2.2 Node.js

Το Node.js αποτελεί ένα περιβάλλον εκτέλεσης ανοιχτού κώδικα για JavaScript. Δημιουργήθηκε το 2009 από τον Ryan Dahl και πλέον η χρήση του είναι πολύ διαδεδομένη καθώς επιτρέπει στους προγραμματιστές να εκτελέσουν JavaScript Το node.js μπορεί να τρέξει κώδικα JavaScript εκτός του browser. Αυτό του επιτρέπει να είναι πολύ αποδοτικό.

2.2.1 Node js και ασύγχρονος προγραμματισμός

Η Node.js μπορεί να εξυπηρετεί πολλά αιτήματα ταυτόχρονα. Αυτό συμβαίνει χάρη στον ασύγχρονο προγραμματισμό και τη χρήση μη μπλοκαριστικών λειτουργιών εισόδου/εξόδου (non-blocking I/O operations).

Κύρια χαρακτηριστικά της Node.js:

- **Μονή διεργασία:** Η Node.js εκτελεί μια εφαρμογή σε μια μόνο διεργασία χωρίς να χρειάζεται να δημιουργεί ένα καινούργιο νήμα (thread) για κάθε αίτημα που λαμβάνει.
- **Ασύγχρονες λειτουργίες (I/O):** Η Node.js παρέχει ένα σύνολο από ασύγχρονες λειτουργίες I/O στη βιβλιοθήκη της. Αυτό επιτρέπει στην Node.js να τρέξει μια εργασία I/O και να συνεχίσει η εκτέλεση άλλου κώδικα χωρίς να περιμένει να ολοκληρωθεί η I/O εργασία. Ένα παράδειγμα αυτού είναι η ανάγνωση ενός αρχείου.
- **Μη μπλοκαριστικές βιβλιοθήκες:** Οι περισσότερες βιβλιοθήκες της Node.js είναι μη μπλοκαριστικές. Αυτό σημαίνει ότι δεν εμποδίζουν την εκτέλεση άλλου κώδικα όσο περιμένουν να ολοκληρωθεί μια εργασία.

Παράδειγμα εκτέλεσης Node.js κώδικα:

```
const fs = require('fs')
fs.readFile('myfile.txt', (err, data) => {
    if(err) throw err;
    console.log(data.toString());
});

console.log("Hello world");
//Hello world will be printed first
```

Αυτός ο κώδικας διαβάζει ένα αρχείο ασύγχρονα. Θα εκτυπωθεί πρώτα το “Hello world” και έπειτα το περιεχόμενο του αρχείου, αφού ολοκληρωθεί η ανάγνωση του.

2.2.2 Πλεονεκτήματα της Node.js

- Οι προγραμματιστές μπορούν να γράφουν κώδικα από την μεριά του πελάτη (client-side), αλλά μπορούν πλέον με ευκολία να γράψουν κώδικα και στη μεριά του server (server-side), καθώς η Node.js ενσωματώνει δικό της server. Με τη χρήση μιας βιβλιοθήκης HTTP που περιλαμβάνεται, μας επιτρέπει να δημιουργήσουμε έναν HTTP server απευθείας στον κώδικά μας.

παράδειγμα δημιουργίας του server με το Node.js:

```
const http = require('http');

const server = http.createServer((req, res) => {
  res.writeHead(200); // HTTP status code 200
  res.setHeader('Content-Type', 'text/plain'); // Headers of the response
  res.end('Hello, World!'); // response is the message "hello world"
});

//server listens to port 3000
server.listen(3000, () => {
  console.log('Server running at http://localhost:3000/');
});
```

- Τα νέα ECMAScript standards διευκολύνουν ακόμα περισσότερο τη χρήση της Node.js. Επειδή η Node.js εκτελείται στον server, οι προγραμματιστές μπορούν να χρησιμοποιούν άμεσα αυτές τις νέες λειτουργίες, χωρίς να περιορίζονται από τον κύκλο ενημέρωσης των προγραμμάτων περιήγησης. Η ασύγχρονη διαδικασία αυτή, βελτιώνει σημαντικά την απόδοση της εφαρμογής.
- Επιτρέπει κλιμάκωση. Δηλαδή ανταποκρίνεται άμεσα ακόμα και σε αυξημένες απαιτήσεις χρήσης και φόρτου της εφαρμογής. Η κλιμάκωση διακρίνεται στην κάθετη και στην οριζόντια, όπου και οι δύο περιπτώσεις έχουν μόνο να προσφέρουν στην εφαρμογή. Παρακάτω εξηγούνται λίγα περισσότερα για αυτές:

Κάθετη Κλιμάκωση

Αναφέρεται στην αναβάθμιση των πόρων του ίδιου του server της εκάστοτε εφαρμογής, όπως αύξηση της ισχύς της CPU καθώς και της μνήμης RAM.

Οριζόντια Κλιμάκωση

Αναφέρεται στην προσθήκη περισσότερων κόμβων της εφαρμογής για την αποτελεσματικότερη διαχείριση του αυξημένου φόρτου. Πρακτικά αυτό σημαίνει ότι ο χρήστης μπορεί να εκτελέσει πολλαπλά αντίγραφα της Node.js σε διαφορετικούς servers και να κατανομήσει τα αιτήματα που δημιουργούνται με πιο αποδοτικό τρόπο.

2.2.3 NPM

Το npm είναι ένας διαχειριστής πακέτων (packet manager) για Node.js εφαρμογές, που είναι διαθέσιμο για δημόσια χρήση. Όλα τα project που είναι διαθέσιμα στο μητρώο του npm ονομάζονται “πακέτα”.

Το npm μας επιτρέπει να χρησιμοποιήσουμε κώδικα γραμμένο από άλλους προγραμματιστές πολύ εύκολα, χωρίς να χρειάζεται να τον αναπτύξουμε μόνοι μας.

Package.json:

Το αρχείο package.json μας βοηθάει στο να παρακολουθούμε όλα τα κατεβασμένα npm πακέτα σε ένα project. Είναι σημαντικό να δημιουργείται αυτό το αρχείο, μιας και περιλαμβάνει πληροφορίες για το project όπως είναι το όνομα του , το version, τα scripts, τα dependencies κ.α.

Για τη δημιουργία του package.json χρησιμοποιούμε την εντολή:

```
npm init
```

Μια ακόμα σημαντική χρησιμότητα του package.json είναι όταν γίνεται clone από το GitHub. Όταν κάποιος κάνει push το project τους στο GitHub ο φάκελος αφήνει εκτός τον φάκελο node_modules ο οποίος περιλαμβάνει τα πακέτα και τα dependencies της εφαρμογής.

Για να μπορέσουμε να φτιάξουμε τον δικό μας φάκελο node_modules μετά από κάποιο clone, χρησιμοποιούμε την εντολή:

```
npm install
```

Αυτό επιτρέπει στο npm να ανατρέξει ολόκληρο το package.json του repository και να εγκαταστήσει όλα τα απαραίτητα πακέτα.

NPM εντολές για τα πακέτα:

Ο τρόπος εγκατάστασης των npm πακέτων γίνεται με τον εξής τρόπο:

- **Globally:** Υπάρχει δυνατότητα λήψης npm πακέτων globally, δηλαδή μπορούμε να έχουμε πρόσβαση σε αυτά τα πακέτα από οποιοδήποτε φάκελο του υπολογιστή μας.

  ```
  Npm i -g [packageName]
  ```

  Όπου i είναι για το install, -g είναι για το global και packagename το όνομα του πακέτου που θέλουμε (πχ typescript, tailwindcss κλπ.)
- **Τοπικά:** Το ίδιο μπορούμε να κάνουμε και για να κάνουμε εγκατάσταση τοπικά, δηλαδή θα έχουμε πρόσβαση μόνο στον φάκελο που έχει κατεβεί το πακέτο

  ```
  npm i typescript
  ```

Ο τρόπος απεγκατάστασης των npm πακέτων γίνεται με τον εξής τρόπο (τοπικά):

```
npm uninstall [packageName]
```

Και σε περίπτωση όπου ο χρήστης επιθυμεί να ενημερώσει τα αρχεία (τοπικά):

npm update [packageName]

Πλεονεκτήματα χρήσης του NPM

1. Ενέχει μια ποικιλία στα εργαλεία που μπορεί να χρησιμοποιήσει ο προγραμματιστής χωρίς επιπλέον κόστος.
2. Συμβάλλει στην αύξηση της απόδοσης της εφαρμογής στην φάση της ανάπτυξής της, αφού χρησιμοποιεί προκατασκευασμένες εξαρτήσεις.
3. Επιτρέπει στον χρήστη να εγκαταστήσει βιβλιοθήκες, frameworks, και άλλα προγραμματιστικά εργαλεία, που επομένως περιορίζουν το debugging του κώδικα.
4. Οι εντολές του npm δεν είναι δυσνόητες και μπορούν να χρησιμοποιηθούν από το ευρύ κοινό.

2.3 Next.js Route handlers

Στην Next.js 14 με τον app router, οι Route Handlers είναι μια σημαντική ιδιότητα καθώς προσφέρουν συνολικά πιο εύκολο χειρισμό των routes και των API. Επιτρέπουν τη δημιουργία των API endpoints (σημεία τέλους API), τα οποία είναι μονοπάτια υπεύθυνα για τη διάδραση με τον server μέσω αιτημάτων HTTP καθώς και δυναμική δρομολόγηση. Αυτό πραγματοποιείται για αρχεία που βρίσκονται στον φάκελο app χωρίς την ανάγκη να χρησιμοποιούνται τα παραδοσιακά αρχεία api όπως στον pages router, που απαιτεί τη δομή pages/api, αλλά συνήθως τα χρησιμοποιούμε για δική μας ευκολία.

Με τον όρο Route Handlers πρακτικά αναφερόμαστε σε αρχεία JavaScript ή TypeScript που έχουν τη δυνατότητα να χειριστούν HTTP αιτήματα, απευθείας μέσα από το σύστημα αρχείων. Αυτά τα αρχεία επιτρέπουν στον προγραμματιστή να δημιουργήσει προσαρμοσμένα API endpoints, ή να επεξεργαστεί δεδομένα σε συγκεκριμένα routes.

Προσδιορίζονται σε αρχείο με όνομα route.ts ή .js και το αρχείο αυτό μπορεί να βρίσκεται μέσα στον φάκελο app σε ιεραρχία παρόμοια με το layout.tsx, αλλά δεν μπορεί να υπάρχει στο ίδιο επίπεδο διαδρομής με το page.tsx αρχείο.

Response/Request

Τα Request και Response αντικείμενα είναι τα βασικά εργαλεία για τη διαχείριση HTTP αιτημάτων και αποκρίσεων. Χρησιμοποιούνται κυρίως στον pages router του Next.js, αλλά η χρήση τους επεκτείνεται και στον app router.

Το αντικείμενο Request αναπαριστά το αίτημα που στέλνεται στον server από τον client και μπορεί να περιέχει πληροφορίες όπως:

- **method:** Η HTTP μέθοδος (π.χ., GET, POST).
- **headers:** Τα headers του αιτήματος.
- **url:** Η διεύθυνση (URL) του αιτήματος.
- **body:** Τα δεδομένα που στέλνει ο client (για αιτήματα POST, PUT, κλπ.).

Αντίθετα το αντικείμενο Response αναπαριστά την απόκριση του server προς τον client και περιέχει μεθόδους για:

- **json:** Αποστολή δεδομένων σε μορφή JSON.
- **status:** Ορισμός του κωδικού κατάστασης της απόκρισης (π.χ., 200 για επιτυχία, 404 για μη εύρεση).
- **send:** Αποστολή plain text ή άλλου τύπου δεδομένων.

Τα NextRequest και NextResponse αποτελούν επεκτάσεις των Request/Response αντικειμένων, και είναι ειδικά κατασκευασμένες για το Next.js, καθώς προσφέρουν κάποιες επιπλέον μεθόδους που είναι χρήσιμες σε server-side rendering.

Υποστηριζόμενες μέθοδοι HTTP

Ένα αρχείο διαδρομής επιτρέπει στον χρήστη να δημιουργεί προσαρμοσμένους χειριστές αιτημάτων για μια δεδομένη διαδρομή.

Οι μέθοδοι που υποστηρίζονται είναι: GET,POST,PUT,PATCH,DELETE,HEAD και OPTIONS

Μερικά παραδείγματα των μεθόδων αυτών είναι:

GET:

```
export async function GET(req: NextRequest, { params }: { params: { userid:
string }}) {
  const userid = params.userid;
  try {
    const result = await findUserById(userid);
    return Response.json({ success: true, data: result });
  } catch (err) {
    return Response.json({ success: false, message: "No data found" });
  }
}
```

Το παράδειγμα αυτό δείχνει έναν route handler που χειρίζεται αιτήματα GET στη διαδρομή api/users/[userid]. Λαμβάνει το userid ως παράμετρο και προσπαθεί να βρει τον χρήστη με το συγκεκριμένο id. Αν βρεθεί επιστρέφει τα δεδομένα του, διαφορετικά επιστρέφει ένα μήνυμα σφάλματος.

PUT:

```
export async function PUT(req: NextRequest, { params }: { params: { userid:
string }}) {
const userid = params.userid;
const { username, email, password } = await req.json();
try {
// Fetch the current user data
const currentUser = await findUserById(userid);
```

```
// Prepare the updated fields
const updatedFields: any = {
username: username || currentUser?.username, // Default to current username
if not provided
email: email || currentUser?.email, // Default to current email if not
provided
};

// Only update password if it's provided and non-empty
if (password && password.trim()) {
updatedFields.password = password;
}

// Update user in the database
const result = await updateUserById(userid, updatedFields);

return Response.json({ success: true, data: result });
} catch (err) {
return Response.json({ success: false, message: "No data found" });
}
}
```

Το παράδειγμα αυτό δείχνει έναν route handler που χειρίζεται αιτήματα PUT στη διαδρομή api/users/[userid]. Λαμβάνει το userid ως παράμετρο και προσπαθεί να βρει τον χρήστη με το συγκεκριμένο id. Αν βρεθεί ενημερώνει δεδομένα του, διαφορετικά επιστρέφει ένα μήνυμα σφάλματος.

POST:

```
export async function POST(req: NextRequest) {
  try {
    const { username, email, password } = await req.json();
    const result = await addUser({
      username,
      email,
      password,
    });
    return Response.json({
      message:
        "User added successfully, please check your email for
verification.",
      result,
    });
```

```
  } catch (error: any) {
    console.error("Failed to add user:", error);
    // Determine the appropriate status code and message
    let status = 400;
    let message = error.message || "Failed to add user";

    return new Response(JSON.stringify({ message }), { status });
  }
}
```

Το παράδειγμα αυτό δείχνει έναν route handler που χειρίζεται αιτήματα POST στη διαδρομή api/users. Χρησιμοποιείται για να εγγράψει έναν νέο χρήστη στη βάση δεδομένων.

DELETE:

```
export async function DELETE(req: NextRequest) {
  const { userid } = await req.json();

  try {
    // Find the user by ID and delete them
    const deletedUser = await deleteUserById(userid);

    if (!deletedUser) {
      return Response.json({ success: false, message: "User not found" });
    }

    return Response.json({
      success: true,
      message: "User deleted successfully",
    });
  } catch (error) {
    console.error("Failed to delete user:", error);
    return Response.json({ success: false, message: "Failed to delete user"
});
  }
}
```

Το παράδειγμα αυτό δείχνει έναν route handler που χειρίζεται αιτήματα DELETE στη διαδρομή api/users/[userid]. Χρησιμοποιείται για να διαγράψει τον χρήστη με το συγκεκριμένο id από τη βάση δεδομένων.

2.4 NextAuth.js: Απλοποιημένο authentication για Next.js

Η NextAuth.js είναι μια βιβλιοθήκη ανοιχτού κώδικα που απλοποιεί σημαντικά την προσθήκη αυθεντικοποίησης (authentication) σε Next.js εφαρμογές.

Με τη χρήση της NextAuth.js μπορούν να ενσωματωθούν εύκολα αρκετοί πάροχοι αυθεντικοποίησης όπως είναι η Google, Facebook, Github ή ακόμα και με Credentials (email/password).

Μερικές βασικές έννοιες της NextAuth.js είναι οι εξής:

2.4.1 Providers

Οι providers επιτρέπουν στους χρήστες να μπορούν να συνδεθούν στην εφαρμογή χρησιμοποιώντας ήδη υπάρχοντες λογαριασμούς από διάφορες πλατφόρμες. Οι providers αυτοί χρησιμοποιούν το πρωτόκολλο oAuth έτσι ώστε να μπορούν οι χρήστες να συνδέονται στην εφαρμογή μας με ασφάλεια.

Ο τρόπος με τον οποίο μπορούμε να ορίσουμε providers στην εφαρμογή μας, γίνεται με το να προστεθούν τα στοιχεία στο αρχείο […nextAuth]/route.ts. Για παράδειγμα για να ρυθμίσουμε τον Google provider πρέπει αρχικά να πάρουμε τα Google_id και Google_secret από τη σελίδα της Google και έπειτα να τα τοποθετήσουμε μέσα στο […nextAuth]/route.ts με τον εξής τρόπο (υποθέτουμε ότι βρίσκονται σε .env αρχείο):

```
export const authOptions: NextAuthOptions = {
  providers: [
    GoogleProvider({
      clientId: process.env.GOOGLE_CLIENT_ID as string,
      clientSecret: process.env.GOOGLE_CLIENT_SECRET as string,
    })
  ],
}
```

Είναι επίσης εφικτή και η σύνδεση του χρήστη με δικά του credentials και αυτό επιτυγχάνεται με αυτό τον τρόπο:

```
CredentialsProvider({
      name: "Credentials",
      credentials: {
        email: { label: "Email", type: "email" },
        password: { label: "Password", type: "password" },
      },
```

```
      async authorize(credentials) {
        if (!credentials?.email || !credentials?.password) {
          console.log("No such credentials");
          return null; // Return null if no credentials
        }

        const user = await logUser(credentials.email,
credentials.password);

        if (user.status === 200 && user._id) {
          console.log("passwords matched");
          return {
            id: user._id.toString(),
            email: user.email,
            name: user.username,
          }; // Return user object with id as string
        }
        console.log("something went wrong with authorize");
        return null; // Return null if invalid
      },
    }),
```

2.4.2 Session

Τα sessions είναι ένας μηχανισμός για να αποθηκεύονται πληροφορίες του συνδεδεμένου χρήστη.

Όταν ένας χρήστης συνδέεται στην εφαρμογή μας, η NextAuth δημιουργεί ένα session object που περιέχει τα στοιχεία id, name, email του χρήστη. Αυτό το object αποθηκεύεται σαν Cookie στον browser του χρήστη, με αποτέλεσμα όταν ο χρήστης κάνει ένα αίτημα στον διακομιστή, η nextAuth ελέγχει το cookie για να δει αν ο χρήστης είναι συνδεδεμένος.

Ένα παράδειγμα για το πως ορίζουμε το session είναι το εξής

```
       callbacks: {

    async session({ session, token }) {
      if (token.user) {
        session.user = token.user as User;
      }
      return session;
    },
```

Ο κώδικας αυτός ελέγχει αν υπάρχει το token του user. Αν υπάρχει τότε το αποθηκεύει στο session.user

Το session ορίζεται στο ίδιο αρχείο με τους providers με το όνομα callbacks.

2.4.3 Callbacks

Τα Callbacks είναι συναρτήσεις που μας επιτρέπουν να ορίσουμε τη συμπεριφορά της NextAuth σε διάφορα στάδια της αυθεντικοποίησης. Η NextAuth παρέχει προεπιλεγμένες συμπεριφορές για Callbacks αλλά υπάρχει δυνατότητα να ορίσουμε και δικούς μας τρόπους.

Για παράδειγμα για το signIn callback, που χρησιμοποιείται για έλεγχο αν μπορεί ένας χρήστης να συνδεθεί στην εφαρμογή, μπορεί να τροποποιηθεί, έτσι ώστε να ελέγχει τα credentials του χρήστη (όπως το email και το password) πριν επιτρέψει την πρόσαβαση.

```
 async signIn({ user, account }) {
      if (user && account) {
        const email = user.email;
        const name = user.name;

        // Find user by email to check existence
        const existingUser = await findUserByEmail(email || "");

        if (existingUser) {
          // Check if the existing user was created via credentials or
another provider
          let provider = existingUser.provider || "credentials"; // Default
to "credentials" if undefined

          // Check if the user is trying to sign in with an OAuth provider
          if (["google", "github", "facebook"].includes(account.provider))
{
            // If the account provider is OAuth but the user was created
with credentials
            if (provider === "credentials") {
              return "/Signin?error=EmailInUse"; // If trying to sign up,
redirect to Signup page with error
            }
            // Allow sign-in if it's an OAuth provider and matches the
existing user provider
            return true;
          }
          // If the user is signing in with credentials and the account was
also created with credentials
          if (
            account.provider === "credentials" &&
            provider === "credentials"
          ) {
```

```
            // Allow sign-in with credentials
            return true;
          }
        }

        // Create a new user if it does not exist
        await addUserOath({
          email: email ?? "",
          name: name ?? "",
          profilePicture: user.image || "", // Set a default or fetch from
provider
          isVerified: true, // OAuth users are considered verified
          provider: account.provider, // Track the provider (google,
github, credentials, etc.)
        });

        return true; // Allow sign-in
      }

      return false; // Deny sign-in if no user or account is provided
    },
```

Αυτός ο κώδικας ελέγχει αν ο χρήστης που προσπαθεί να συνδεθεί υπάρχει ήδη στη βάση δεδομένων. Αν δεν υπάρχει, δημιουργεί έναν νέο λογαριασμό. Επιπλέον, ελέγχει αν ο χρήστης προσπαθεί να συνδεθεί με έναν OAuth provider (όπως Google ή GitHub) και αν ναι, ελέγχει αν ο λογαριασμός του έχει δημιουργηθεί με τον ίδιο provider.

2.4.4 Εγκατάσταση πακέτου

- Η εγκατάσταση του next-auth γίνεται απλοϊκά με την εντολή:

```
npm install next-auth
```

- Στη συνέχεια ο χρήστης θα πρέπει να δημιουργήσει ένα αρχείο στη διαδρομή app/api/auth/[...nextauth]/ με όνομα route.ts ή .js και στη συνέχεια αφού ενσωματώσει το πακέτο από την βιβλιοθήκη next-auth θα πρέπει να γίνει η ενσωμάτωση του επιθυμητού παρόχου με τον τρόπο που αναδείχθηκε παραπάνω.
- Επόμενο βήμα είναι η προσθήκη των id, secret μεταβλητών σε περίπτωση που αναφερόμαστε σε εξωτερικό προμηθευτή, όπως για παράδειγμα στη google με τα GOOGLE_CLIENT_ID, GOOGLE_CLIENT_SECRET, τα οποία προσαρτούμε σε ένα .env ή σε ένα .local.env αρχείο.

  Στην περίπτωση των Credentials ως προμηθευτή, εισάγουμε τη δομή που αναφέρθηκε παραπάνω που βασίζεται στα email, password τα οποία είτε

προστίθενται είτε ως μεταβλητές που δίνει ο χρήστης είτε καλούνται από κάποια βάση δεδομένων μέσω της χρήσης συναρτήσεων.

Ένα παράδειγμα σελίδας για τη σύνδεση του χρήστη με Google είναι το εξής:

```
  <FaGoogle
            size={40}
            onClick={() => signIn("google", { callbackUrl: "/" })}
            className="cursor-pointer hover:scale-125 transition-all
duration-200 ease-in-out"
          />
```

Ένα παράδειγμα αποσύνδεσης είναι το εξής:

```
"use client";
import { signOut } from "next-auth/react";

const Logout = () => {
  return (
    <button onClick={() => signOut({ callbackUrl: "/" })}>Sign out</button>
  );
};

export default Logout;
```

3. GitHub

3.1 Εισαγωγή

Το GitHub είναι μια διαδικτυακή πλατφόρμα που χρησιμοποιεί το Git, ένα σύστημα ελέγχου εκδόσεων (version control system) που βοηθά τους προγραμματιστές να διαχειρίζονται και να επιβλέπουν τον κώδικά τους με ευκολία. Επίσης, επιτρέπει στους χρήστες να συνεργάζονται και να μοιράζουν των κώδικά τους από οπουδήποτε κι αν βρίσκονται.

3.2 Κύρια χαρακτηριστικά του GitHub

- **Version control system:**

Το σύστημα ελέγχου εκδόσεων (Version Control System - VCS) είναι ένα εργαλείο που διαχειρίζεται τις αλλαγές σε αρχεία και φακέλους κατά τη διάρκεια της ανάπτυξης μιας εφαρμογής. Η βασική του λειτουργία είναι να παρακολουθεί τις αλλαγές που γίνονται στον κώδικα ή σε άλλα αρχεία αλλά

ταυτόχρονα διευκολύνει αρκετά τη συνεργασία μεταξύ πολλαπλών χρηστών. Επίσης, είναι σημαντικό να αναφερθεί ότι διατηρεί το ιστορικό όλων των αλλαγών, κάτι που είναι κρίσιμο για την ανάπτυξη μιας εφαρμογής.

- **Repositories:**

Ένα αποθετήριο (ή repository) είναι ένα κεντρικό μέρος στο οποίο αποθηκεύονται όλα τα αρχεία μιας εφαρμογής. Κάθε ένα από αυτά, μπορεί να περιέχει πολλαπλά αρχεία και φακέλους και παρακολουθεί το ιστορικό κάθε αλλαγής που γίνεται. Τα αποθετήρια μπορεί να είναι είτε δημόσια είτε ιδιωτικά.

- **Branches:**

Οι κλάδοι (branches) είναι μια σημαντική δυνατότητα του GitHub που επιτρέπουν την παράλληλη ανάπτυξη και την καλύτερη ταξινόμηση. Ένας χρήστης μπορεί να δημιουργήσει ένα κλάδο και να εργαστεί σε κάτι διαφορετικό ή να διορθώσει ένα σφάλμα χωρίς να επηρεάσει τον κύριο κώδικα (main branch). Έπειτα, μόλις οι αλλαγές ολοκληρώσουν, μπορεί να συγχωνεύσει τον κλάδο που δημιούργησε πίσω στον κύριο κλάδο.

- **Pull requests:**

Τα Pull requests είναι μια μέθοδος για να προτείνουμε αλλαγές σε ένα repository. Όταν υποβάλουμε ένα pull request, ζητάμε ουσιαστικά από τους διαχειριστές συντηρητές του project να αναθεωρήσουν και να συγχωνεύσουν αλλαγές στην κύρια βάση κώδικα. Αυτό το χαρακτηριστικό προωθεί την συνεργασία και διασφαλίζει την ποιότητα του κώδικα, αφού αξιολογείται από άτομα του ίδιου επίπεδου γνώσης.

- **Issues and project management:**

Οι ενέργειες του GitHub μας επιτρέπουν να αυτοματοποιούμε ροές εργασίας, όπως είναι για παράδειγμα η εκτέλεση test ή το deployment κώδικα απευθείας μέσα από το repository μας. Αυτό το χαρακτηριστικό ενισχύει την παραγωγικότητα και διασφαλίζει την συνοχή σε όλες τις διαδικασίες ανάπτυξης.

- **Ενέργειες και αυτοματισμός:**

Οι ενέργειες του GitHub μας επιτρέπουν να αυτοματοποιούμε ροές εργασίας, όπως είναι για παράδειγμα η εκτέλεση test ή το deployment κώδικα απευθείας μέσα από το repository μας. Αυτό το χαρακτηριστικό ενισχύει την παραγωγικότητα και διασφαλίζει την συνοχή σε όλες τις διαδικασίες ανάπτυξης.

3.3 Διαδικασία δημιουργίας Repository

Για να δημιουργήσουμε ένα repository από έναν τοπικό φάκελο που περιέχει τον κώδικα που θέλουμε να ανεβάσουμε στο GitHub ακολουθούμε μια σειρά από βήματα.

1) Αρχικά βεβαιωνόμαστε βρισκόμαστε στον σωστό φάκελο του project που θέλουμε να ανεβάσουμε στο GitHub. Στη συνέχεια, αρχικοποιούμε τον τοπικό φάκελο ως ένα git repository με την εντολή:

```
git init
```

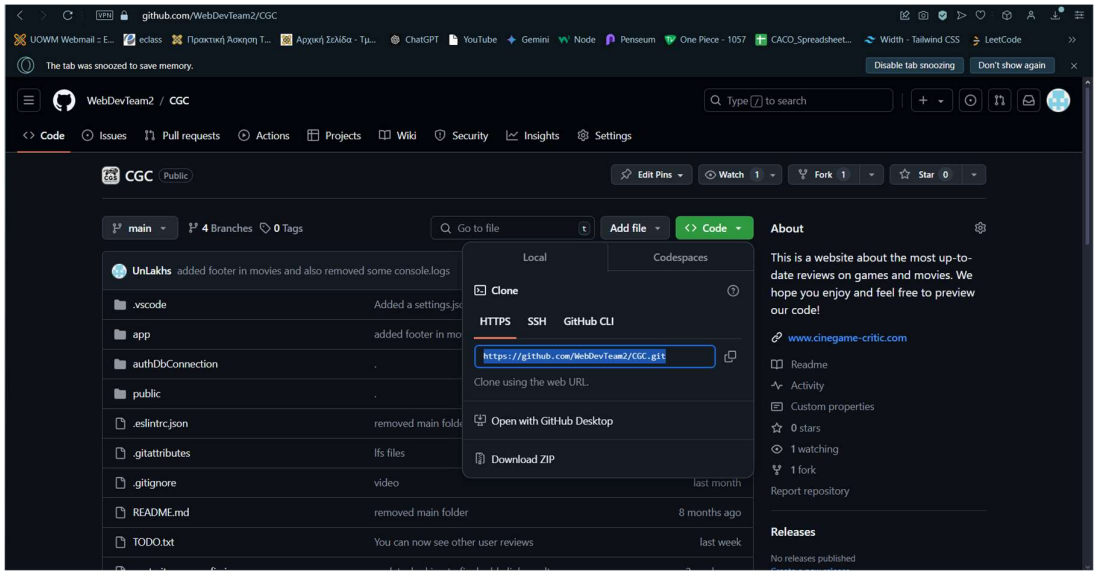
2) Αφού εκτελεστεί αυτή η εντολή, ένας νέος .git φάκελος δημιουργείται στο project μας, ο οποίος είναι κρυφός από προεπιλογή. Προσθέτουμε τα αρχεία μας στο καινούργιο μας local repository με την εντολή

```
git add .
```

3) Αφού γίνουν αυτά τα βήματα προχωράμε στο κομμάτι του commit των αλλαγών αυτών με την εντολή:

```
Git commit -m “First commit”
```

4) Στη συνέχεια στη σελίδα του GitHub μας, μπορούμε να δούμε στο quick setup του repository μας το remote repository URL, το οποίο το κάνουμε αντιγραφή.



5) Έπειτα στο command prompt μας προσθέτουμε το remote repository URL στο οποίο θέλουμε να κάνουμε push το τοπικό μας repository:

```
Git remote add origin remote repository URL
```

Αφού το προσθέσουμε το κάνουμε verify με την εντολή:

```
Git remote -v
```

6) Τέλος κάνουμε push τις αλλαγές του local repository μας στο GitHub με την εντολή:

```
Git push origin master
```

3.4 Εντολές GitHub

- **git clone http_url**
  Με αυτή την εντολή αντιγράφουμε ένα υπάρχων repository σε ένα καινούργιο φάκελο στον υπολογιστή μας.

- **git commit –m “commit message”**
  Με αυτή την εντολή φτιάχνουμε ένα σημείο στο repository όπου μπορούμε να το επαναφέρουμε ανά πάσα στιγμή και επίσης είναι η εντολή που ακολουθείται μετά την git add και πριν την git push. Χωρίς την εντολή αυτή το push δεν είναι εφικτό.

- **git push**
  Ότι αλλαγές κάνουμε με τις εντολές git add και git commit, βρίσκονται τοπικά μέχρι να τις στείλουμε στο απομακρυσμένο αποθετήριο (remote repository). Με την εντολή git push οι αλλαγές στέλνονται στο remote repository.

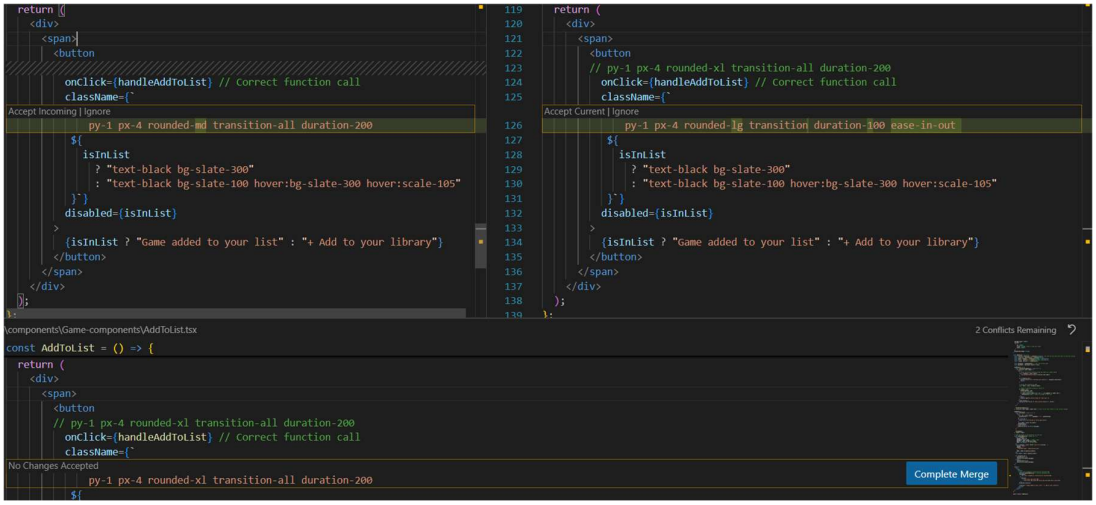
- **Git fetch:**
  Όταν εκτελούμε την εντολή git fetch, το git επικοινωνεί με το απομακρυσμένο repository και κατεβάζει όλα τα νέα commits, branches και tags από το τελευταίο fetch ή pull. Οι αλλαγές αυτές αποθηκεύονται στο τοπικό μας repository, χωρίς όμως να επηρεάζει τα αρχεία μας ή τα τοπικά branches.

- **git pull**

Με αυτή την εντολή τραβάμε τοπικά ό,τι αλλαγή έχει γίνει από άλλους και έχει πάει στο remote repository.

*Υποσημείωση για Conflicts (συγκρούσεις)

Όταν αλλάξουμε ένα αρχείο και αυτό το αρχείο έχει επίσης αλλάξει από κάποιον άλλον και έχει γίνει push στο remote repository, όταν κάνουμε push μπορεί να δημιουργηθεί σύγκρουση (γιατί το αρχείο έχει αλλάξει από κάποιον άλλον και το έχουμε αλλάξει και εμείς με διαφορετικό τρόπο). Σε αυτή την περίπτωση το git κρατά και τις 2 versions και πρέπει να κάνουμε manually resolve το conflict (δηλαδή να ενώσουμε τις αλλαγές, να σβήσουμε ότι δε χρειάζεται κλπ.). Ένα παράδειγμα merge conflict είναι το παρακάτω:



Εικόνα 4 . Merge conflict στο VS Code.

Στο παράδειγμα αυτό βλέπουμε ένα αρχείο στο οποίο έχει γίνει push από 2 άτομα. Στα αριστερά βλέπουμε την εισερχόμενη αλλαγή, δηλαδή την αλλαγή του αρχείου από τον άλλο χρήστη και στα δεξιά βλέπουμε την δικιά μας αλλαγή. Το VS Code μας δίνει την δυνατότητα να κάνουμε resolve αυτό το merge conflict είτε αποδέχοντας την αλλαγή του άλλου χρήστη ή τη δικιά μας. Μόλις ολοκληρωθούν οι αλλαγές που θέλουμε να κάνουμε, κάνουμε κλικ στο “complete merge” και κάνουμε git commit, git push.

- **git status**
  Αφού αλλάξουμε το περιεχόμενο κάποιου αρχείου, γράφοντας αυτή την εντολή μας εμφανίζει αναλυτικά ποια αρχεία έχουν αλλάξει.

```
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   app/account/[userid]/movies/page.tsx

no changes added to commit (use "git add" and/or "git commit -a")
```

Εικόνα 5 . Εκτέλεση εντολής git status.

Στο παράδειγμα αυτό βλέπουμε ότι μετά την εκτέλεση του git status εμφανίζεται το αρχείο app/account/[userid]/movies/page.tsx με το μήνυμα ότι έχει γίνει κάποια αλλαγή στο περιεχόμενο του αρχείου και δεν έχει γίνει ακόμα stage για commit.

- **git reset --hard**
  Με αυτή την εντολή σβήνουμε ότι αλλαγή έχουμε κάνει τοπικά σε οποιοδήποτε αρχείο και δεν έχει γίνει ακόμα commit ή add (δηλαδή δεν είναι σε διαδικασία stage). Να χρησιμοποιείται με προσοχή, για ότι προστεθεί στο repository με τη διαδικασία add και commit μπορεί να γίνει επαναφορά του, αν γίνει όμως πριν από αυτή τη διαδικασία, οι αλλαγές μπορούν να χαθούν με αυτή την εκτελώντας την εντολή.

4. Αρχιτεκτονική του CineGame-Critic

4.1 Εισαγωγή

Εφόσον αναλύθηκαν οι τεχνολογίες που χρησιμοποιήθηκαν για την ανάπτυξη του CineGame Critic, τώρα θα αναλύσουμε την αρχιτεκτονική σχεδίαση της εφαρμογής. Θα περιγραφεί η δομή της ιστοσελίδας, η οργάνωση της, και η αλληλεπίδραση μεταξύ τους. Επιπλέον, θα αναλυθούν κάποια κρίσιμα σημεία της αρχιτεκτονικής, όπως είναι η επικοινωνία client-server, η δομή του client-side και του server-side κώδικα, καθώς επίσης και η χρήση της βάσης δεδομένων και εξωτερικών υπηρεσιών.

Το κεφάλαιο αυτό έχει στόχο να δώσει μια ολοκληρωμένη εικόνα της αρχιτεκτονικής του CineGame-Critic, έτσι ώστε να μπορεί να γίνει κατανοητός ο τρόπος λειτουργίας του και να διευκολυνθεί η μελλοντική συντήρηση και επέκτασή του.

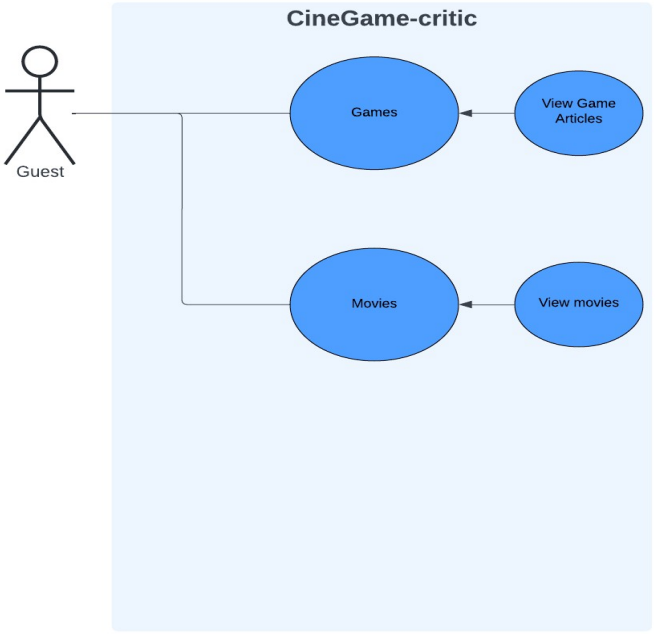
4.2 Περιπτώσεις Χρήσης (UML Case Diagrams)

Οι περιπτώσεις χρήσης περιγράφουν λεπτομερώς τη διαδραστικότητα που μπορεί να έχει ο χρήστης με την εφαρμογή. Μέσω του UML έχουμε την οπτική απεικόνιση της

αλληλεπίδρασης αυτής, κάτι που βοηθάει τον χρήστη στο να κατανοήσει καλύτερα πως λειτουργεί το σύστημα.

Στόχος είναι να κατανοήσουμε τις βασικές λειτουργίες της πλατφόρμας και τις σχέσεις μεταξύ των διαφορετικών τύπων χρηστών, όπως οι επισκέπτες και οι εγγεγραμμένοι χρήστες.

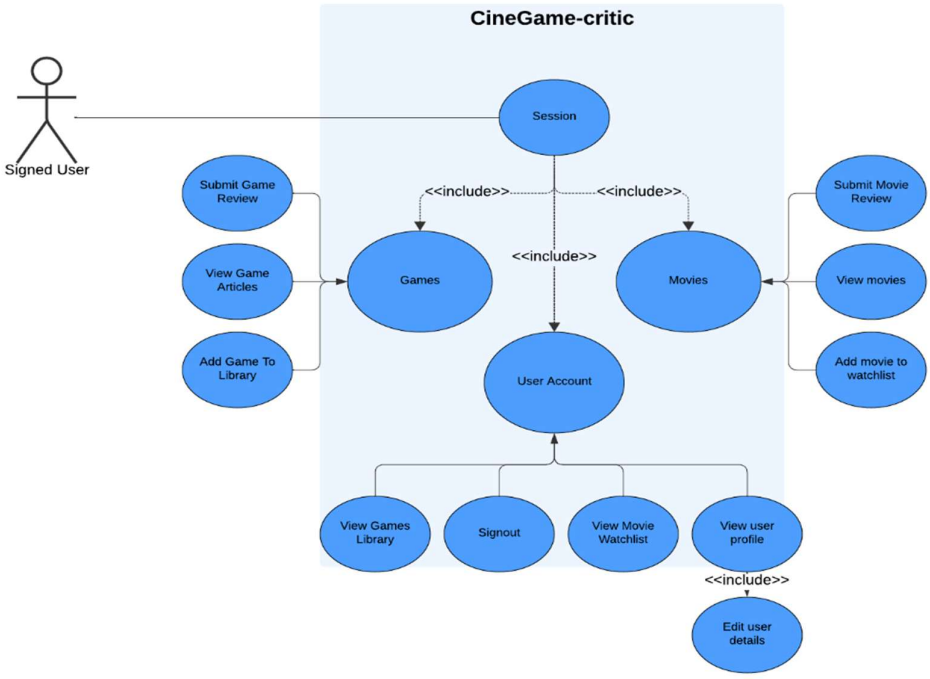
- Το σχεδιάγραμμα για τους επισκέπτες (guests) της σελίδας είναι το εξής:



Εικόνα 6 . Use case για έναν χρήστη που δεν είναι συνδεδεμένος.

Εδώ είναι το παράδειγμα ενός επισκέπτης της σελίδας που δεν είναι συνδεδεμένος με λογαριασμό. Οι δυνατότητες του είναι περιορισμένες, καθώς μπορεί μόνο να συνδεθεί στις δύο σελίδες: Games και Movies αντίστοιχα και να δει τα παιχνίδια που προβάλλονται από τη βάση δεδομένων καθώς και γενικές πληροφορίες για αυτά.

- Το σχεδιάγραμμα για τους εγγεγραμμένους χρήστες (signed users) της σελίδας είναι το εξής:



Εικόνα 7 . Use Case για έναν χρήστη που είναι συνδεδεμένος.

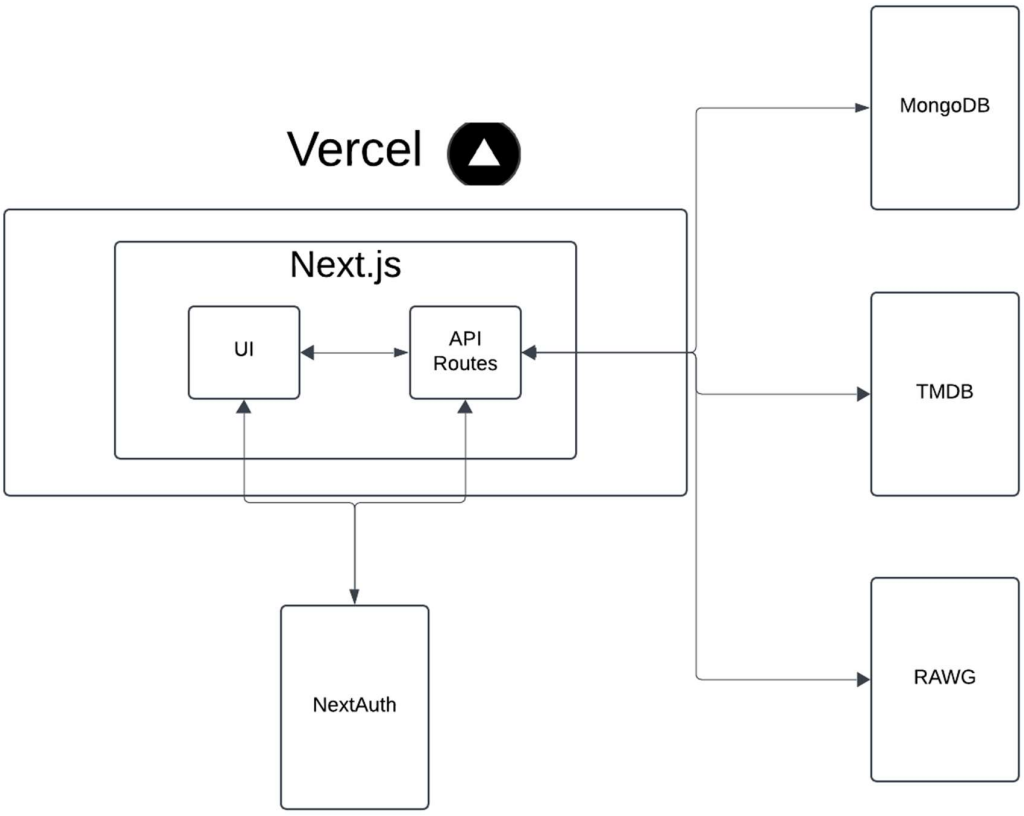
Εδώ είναι το παράδειγμα ενός χρήστη της σελίδας που είναι συνδεδεμένος με λογαριασμό. Αρχικά ο κόμβος session αναπαριστά τη συνεδρία του χρήστη, όπου εάν αυτή είναι επιτυχής, μπορεί να έχει πρόσβαση σε διάφορες λειτουργίες. Εκτός από την προβολή των μέσων στις δύο μεγάλες κατηγορίες και την άντληση πληροφοριών τους, ο χρήστης μπορεί να κάνει τη δική του κριτική και ακόμη και να προσθέσει το επιλεγμένο παιχνίδι/ταινία στην προσωπική του βιβλιοθήκη.

Επιπλέον, υπάρχει μια γενική διαδρομή (url path) όπου ο συνδεδεμένος χρήστης μπορεί να δει πληροφορίες για τον εαυτό του, να τις διαχειριστεί (μόνο σε περίπτωση που δεν είναι συνδεδεμένος με εξωτερικό προμηθευτή, όπως GOOGLE) και να διαγράψει τον λογαριασμό του ανεξάρτητα από τον προμηθευτή. Το path αυτό είναι διαφορετικό για κάθε χρήστη και συντάσσεται ως εξής: https://www.cinegame-critic.com/account/[userid]/info, με το id να καλείται από τη βάση κάθε φορά.

Επίσης, υπάρχει δυνατότητα για τον χρήστη, μέσω του λογαριασμού του, να ανατρέξει τα αποθηκευμένα μέσα που υπάρχουν στη προσωπική του βιβλιοθήκη και τέλος να αποσυνδεθεί.

4.3 Αρχιτεκτονική σχεδίαση

Στο κεφάλαιο αυτό θα αναλυθεί η αρχιτεκτονική σχεδίαση που ακολουθήσαμε κατά την ανάπτυξη της εφαρμογής μας, εστιάζοντας στο τρόπο με τον οποίο οργανώνονται τα διάφορα μέρη της και πως συνδέονται μεταξύ τους.



Εικόνα 8 . Αρχιτεκτονική σχεδίαση του CineGame-critic.

Αυτό το διάγραμμα, Αναπαριστά την αρχιτεκτονική της εφαρμογής μας, δίνοντας έμφαση στα κύρια μέρη της και τις σχέσεις μεταξύ τους. Μας δείχνει πως αξιοποιούμε διάφορες τεχνολογίες για να προσφέρουμε μια αποτελεσματική εμπειρία χρήστη (user experience - UX).

Components

- **Vercel:** Η Vercel είναι η πλατφόρμα που χρησιμοποιήσαμε για να κάνουμε deploy την εφαρμογή μας.
- **Next.js:** Η Next.js είναι ο πυρήνας της εφαρμογής μας. Μας προσφέρει τη δυνατότητα να φτιάχνουμε γρήγορες και αποδοτικές εφαρμογές.

- **UI:** Το component αυτό αναπαριστά το front-end κομμάτι της εφαρμογής μας, που είναι υπεύθυνο για τη διαδραστικότητα με τον χρήστη, καθώς και την παρουσίαση της σελίδας.

- **API Routes:** Τα API Routes διαχειρίζονται την server-side λογική της εφαρμογής. Είναι υπεύθυνα δηλαδή για παραλαβή δεδομένων (data fetching) και την επικοινωνία με τα external APIs TMDB και RAWG.

- **NextAuth.js:** Η NextAuth.js είναι υπεύθυνη για την αυθεντικοποίηση των χρηστών και διαχειρίζεται τις συνδέσεις τους.

- **MongoDB:** Η MongoDB είναι η βάση δεδομένων μας, που είναι υπεύθυνη για να αποθηκεύει στοιχεία για τον κάθε χρήστη και στοιχεία για ταινίες.

- **TMDB & RAWG:** Αυτά είναι τα external APIs που χρησιμοποιήσαμε για το περιεχόμενο των σελίδων μας.

Ροή διαγράμματος:

Αρχικά οι χρήστες αλληλεπιδρούν με το UI της σελίδας, το οποίο επικοινωνεί με τα API Routes για δεδομένα και ενέργειες, καθώς επίσης και με την NextAuth.js μέσω κάποιου form ή button. Στη συνέχεια Τα API Routes επικοινωνούν με το MongoDB, για να αποθηκεύσουν και να λάβουν δεδομένα, με την NextAuth.js για τη διαδικασία της αυθεντικοποίησης και επίσης με τα external APIs, για να εμπλουτίσουν το περιεχόμενο της σελίδας.

4.3.1 Modules

Το CineGame-critic χωρίζεται σε διάφορα modules για καλύτερη οργάνωση και συντήρηση.

Βασικά modules της εφαρμογής είναι τα:

- **App:** Είναι υπεύθυνο για την δομή των roots, περιέχει όλα τα υπόλοιπα modules που αφορούν άμεσα την σελίδα και μας επιτρέπει να χρησιμοποιήσουμε τον App router της Next.js
- **Node_modules:** Τα Node modules αποτελούν το βασικότερο κομμάτι της εφαρμογής, καθώς είναι χτισμένη με βάση αυτά. Περιλαμβάνουν όλες τις εξαρτήσεις και τα πακέτα που χρησιμοποιούνται για τη δημιουργία και τη λειτουργία της εφαρμογής.
- **Games:** Καλύπτει όλες τις λειτουργίες με τις οποίες αλληλοεπιδρά ο χρήστης για τα βιντεοπαιχνίδια. Εκτός της προβολής των παιχνιδιών και των αναλυτικών λεπτομερειών τους, υπάρχει η δυνατότητα συγγραφής

κριτικών, η προσθήκη ενός παιχνιδιού στη προσωπική βιβλιοθήκη του χρήστη.

- **Movies:** Διαχειρίζεται όλες τις λειτουργίες που αφορούν το κομμάτι των ταινιών. Παράδειγμα αυτού, είναι η εμφάνιση πληροφοριών, προσθήκη στο watchlist, προσθήκη κριτικών.
- **User Collection:** Το Collection module είναι αυτό που ενσωματώνει τη σύνδεση της βάσης με την εφαρμογή και αναλυτικότερα την κλήση της συλλογής users από αυτήν. Εδώ βρίσκονται όλες οι συναρτήσεις που αφορούν τους χρήστες, όπως η προσθήκη τους στη βάση κατά την επιτυχή εγγραφή τους και η διαγραφή τους από τη βάση μέσω της διαχείρισης λογαριασμού στη σελίδα, αλλά και η προσθήκη κριτικών ή η αποθήκευση των μέσων στην προσωπική βιβλιοθήκη του χρήστη.
- **Game Collection:** Περιέχει τη σύνδεση της βάσης με την εφαρμογή και συγκεκριμένα καλείται η συλλογή games από αυτή. Ακόμη, εδώ βρίσκονται όλες οι συναρτήσεις που αφορούν τα παιχνίδια που προβάλλονται στη σελίδα, όπως την κλήση τους από το RAWG API, την προσθήκη τους στη βάση, τα είδη τους, κάποιες συναρτήσεις φιλτραρίσματος κ.α.
- **Components:** Περιέχει επαναχρησιμοποιούμενα components που χρησιμοποιούνται σε διάφορα σημεία της εφαρμογής. Χωρίζεται σε Games components και Movie components.
- **Api:** Περιέχει όλα τα αιτήματα για την βάση δεδομένων (όπως GET, POST, PUT, DELETE).
- **Account:** Περιέχει τις λειτουργίες που αφορούν τον λογαριασμό του χρήστη, όπως είναι η προβολή των δεδομένων του και η επεξεργασία τους, προβολή του watchlist του και game library του.
- **authDbConnection:** Περιέχει βασικές τεχνικές λειτουργίες όπως είναι οι ρυθμίσεις της NextAuth και τη σύνδεση με την mongodb.
- **Constants:** Περιέχει διάφορες επαναχρησιμοποιούμενες συναρτήσεις και interfaces που παραμένουν ίδια σε διάφορα αρχεία κώδικα.
- **Authentication:** Περιέχει τις σελίδες που αφορούν το Sign up και Sign in των χρηστών στην εφαρμογή.
- **Public:** Το public module είναι σημαντικό για τη προσθήκη και διαχείριση στατικών αρχείων της εφαρμογής, όπως εικόνες ή βίντεο τα οποία δεν απαιτούν επεξεργασία από τον server. Επίσης, όλα τα αρχεία που τοποθετούνται μέσα σε αυτόν τον φάκελο είναι προσβάσιμα από το root path της εφαρμογής.

4.3.2 Παραδείγματα συνδέσεων μεταξύ modules

Τα modules του CineGame Critic έχουν την δυνατότητα να αλληλοεπιδρούν μεταξύ τους.

Το **Game module** εξαρτάται από το **Game collection module**, για να ανακτήσει τα παιχνίδια από το **API** και από το **User collection module** για να ανακτήσει και να προβάλλει πληροφορίες σχετικές με τον χρήστη, όπως τις κριτικές του ή το game list του.

Ακόμη, πιο βαθιά στη δομή των Games συναντάμε τα Components -> Game Components, τα οποία είναι επαναχρησιμοποιούμενα αρχεία κώδικα όπως το Navbar, που ενσωματώνονται στη σελίδα που θέλουμε με τη σύνταξη <Navbar/> ή πχ <Genres/>. Επιπλέον, χρησιμοποιείται το **Api module** που περιέχει αρχεία με αιτήματα http, όπως GET, POST που είναι χρήσιμα για την προσθήκη αλληλεπίδρασης με τη βάση, όπως είναι η προσθήκη παιχνιδιών και η ανάκτηση τους.

Το **Movie module** εξαρτάται άμεσα από το **component module** αφού χρησιμοποιεί πολλά κομμάτια από εκεί, όπως είναι το navbar για την εμφάνιση της μπάρας πλοήγησης και το filter για το φιλτράρισμα ταινιών. Συνδέεται επίσης άμεσα με το **Account module** αφού γίνονται έλεγχοι σε διάφορα σημεία του κώδικα για το αν ένας χρήστης είναι συνδεδεμένος ή όχι. Αυτό είναι απαραίτητο καθώς μόνο οι συνδεδεμένοι χρήστες έχουν τη δυνατότητα να υποβάλουν κριτικές και να προσθέσουν κάποια ταινία στο watchlist τους. Επιπλέον το **Movies module** συνδέεται με το **Api module** καθώς και με το **User collection module** αφού components όπως το AddToWatchlist καλούν μεθόδους από το **API module** για να ενημερώσουν τη βάση δεδομένων. Οι μέθοδοι αυτές με τη σειρά τους περιέχουν άλλες μεθόδους που ορίζονται στο **user collection module** και αφορούν τη συμπεριφορά τους στη βάση και τη σύνδεση της εφαρμογής στο συγκεκριμένο MongoDB collection.

4.3.3 Επικοινωνία με εξωτερικά APIs

Στην εφαρμογή μας, η επικοινωνία με τα εξωτερικά APIs(RAWG, TMDB) γίνεται μέσα στις σελίδες. Η προσέγγιση αυτή έγινε με σκοπό να απλοποιηθεί η δομή του κώδικα και να γίνεται εύκολα fetch από διάφορες υπηρεσίες των API σύμφωνα με την ανάγκη της κάθε σελίδας, χωρίς πολύ σύνθετους τρόπους.

Η διαδικασία της σύνδεσης με τις ανοιχτές βιβλιοθήκες API RAWG και TMDB είναι σχετικά απλή και γίνεται με τον ίδιο τρόπο, καθώς ο χρήστης πρέπει να ακολουθήσει τα εξής βήματα:

- Τη δημιουργία λογαριασμού στις σελίδες https://rawg.io/ και https://www.themoviedb.org/ αντίστοιχα

- Την παραγωγή κλειδιού API από τις ρυθμίσεις λογαριασμού
- Ασφαλή αποθήκευση του κλειδιού (συνήθως σε .env αρχείο)
- Χρήση του κλειδιού στην εφαρμογή

Κάθε API έχει ένα βασικό URL από το οποίο ξεκινάνε όλα τα αιτήματα. Σε αυτά τα API τα base URLs είναι:

TMDB: https://api.themoviedb.org/3/movie, RAWG: https://api.rawg.io/api/games

Για παράδειγμα στο μονοπάτι /Movies/[id] χρησιμοποιείται η συνάρτηση fetch για να ανακτηθούν δεδομένα για την ταινία με το συγκεκριμένο id από το TMDB API. Η συνάρτηση παίρνει ως όρισμα το id και επιστρέφει τα δεδομένα της ταινίας σε μορφή json.

```
  const getMovieDetails = async (id: string) => {
    const res = await fetch(
      `${baseUrl}/movie/${id}?${process.env.MOVIE_API_KEY}`,
      options
    );
    const data = await res.json();
    return data;
  };
```

Αντίστοιχα, στο μονοπάτι /Games/[name] χρησιμοποιείται η συνάρτηση fetch για να ανακτηθούν δεδομένα για το παιχνίδι με το συγκεκριμένο name από το RAWG API. Η συνάρτηση παίρνει ως όρισμα το name και επιστρέφει τα δεδομένα του παιχνιδιού σε μορφή json.

```
  export const getGameDets = async (name: string) => {
    const res = await fetch(basePosterUrl + "/" + name + "?" +
apiPosterKey);
    const data = await res.json();
    return data;
  };
```

4.3.4 Βάση δεδομένων

Η βάση που χρησιμοποιήσαμε είναι το MongoDB και συγκεκριμένα το MongoDB Atlas, ένα multi-cloud database το οποίο απλοποιεί την διαχείριση των βάσεων καθώς και το deployment. Multi-cloud πρακτικά σημαίνει ότι χρησιμοποιεί ταυτόχρονα υπηρεσίες cloud από διάφορους παρόχους, το οποίο προσφέρει στη συνολική ευελιξία της εφαρμογής. Επιπλέον, η επιλογή του Atlas έγινε γιατί μας επέτρεπε να στήσουμε τη βάση μας χωρίς αυτή να χρειάζεται να τρέχει τοπικά.

Ο λόγος που επιλέξαμε το MongoDB είναι επειδή μας προσφέρει γρήγορη εισαγωγή δεδομένων στην βάση, είναι εύκολα κλιμακωμένη, δεν υπήρχε ανάγκη σύνθετων queries με joins κλπ. και η σύνδεση με την Next.js γίνεται με αρκετά απλό τρόπο.

Η βάση δεδομένων μας ονομάζεται **connectdb**, βρίσκεται μέσα σε ένα cluster που ονομάζεται **cluster0** και μέσα έχει 2 συλλογές (collections) οι οποίες είναι οι:

- Games: αποθηκεύει πληροφορίες για τα παιχνίδια και κάθε έγγραφο έχει τα πεδία: id, added, added_by_status, background_image, clip, dominant_color, esrb_rating, genres, metacritic, name, parent_platforms, platforms, playtime, rating, rating_top, ratings, ratings_count, released, reviews_count, reviews_text_count, saturated_color, score, short_screenshots, slug, stores, suggestions_count, tags, tba, updated, user_game.
- Users: Αποθηκεύει τα δεδομένα των χρηστών, και κάθε έγγραφο έχει τα πεδία: username, password, email, watchlist, provider, library, user_reviews, user_movie_reviews.

Ο λόγος που αποφασίσαμε να συμπεριλάβουμε τα games μέσα στην βάση και να μην λαμβάνουμε όλα τα δεδομένα απευθείας από το RAWG, είναι επειδή η δωρεάν χρήση του RAWG μας περιορίζει στα 20000 αιτήματα ,τα οποία δε μας επαρκούν. Επομένως, καταλήξαμε να κάνουμε fetch τα games από το API σε περίπτωση που η βάση είναι κενή ή σε περίπτωση που ανανεώνεται η αρχική του RAWG με κάποιο παιχνίδι. Σε κάθε άλλη περίπτωση, τα δεδομένα λαμβάνονται από τη βάση.

Η σύνδεση της βάσης με την εφαρμογή γίνεται με τη χρήση του MongoDB driver. Η σύνδεση με την βάση γίνεται σε ένα ξεχωριστό αρχείο, στο **authDbConnection module**. Σκοπός είναι να μας επιστραφεί το clientPromise, για να μπορέσουμε στη συνέχεια να το χρησιμοποιήσουμε στο **user collection module** και στο **Game collection module**, όπου γίνεται η σύνδεση με τα collections της βάσης. Η σύνταξη είναι η εξής:

```
import { MongoClient, MongoClientOptions } from "mongodb";

const URI = process.env.MONGODB_URI as string;
const options: MongoClientOptions = {};

if (!URI) throw new Error("Please add your Mongo URI to .env.local file");

let client: MongoClient = new MongoClient(URI, options);
let clientPromise: Promise<MongoClient>;

// Allow access to the global object in TypeScript
declare global {
  var _mongoClientPromise: Promise<MongoClient> | undefined;
}
```

```
/* If the environment is not production, the code stores the
MongoDB client promise (clientPromise) in a global variable
(global._mongoClientPromise). This prevents creating a new
MongoDB connection every time the code is executed
(e.g., when your application reloads during development).
*/
if (process.env.NODE_ENV !== "production") {
  if (!global._mongoClientPromise) {
    global._mongoClientPromise = client.connect();
  }

  clientPromise = global._mongoClientPromise;
} else {
  clientPromise = client.connect();
}

export default clientPromise;
```

Στον κώδικα αυτό επιτυγχάνεται η σύνδεση στην βάση δεδομένων με το mongodb driver. Αρχικά δημιουργείται ένας mongoClient ο οποίος είναι ένα object που χρησιμοποιείται για την αλληλεπίδραση με τη βάση δεδομένων και λαμβάνει σαν ορίσματα το URI που μας δίνεται από την mongodb και μερικά options.

Η διαδικασία εύρεσης του URI από το Atlas είναι η εξής:

- Αφού επιλέξουμε το project μας, επιλέγουμε το connect.
- Έπειτα επιλέγουμε τον τρόπο που θέλουμε να συνδεθούμε.
- Στις οδηγίες που εμφανίζονται, βλέπουμε μια επιλογή σαν αυτό το παράδειγμα:
  mongodb+srv://<username>:<password>@cluster0.mongodb.net/<dbname>?retryWrites=true&w=majority
- Αφού αντικαταστήσουμε τις παραμέτρους με τα στοιχεία μας, το κάνουμε επικόλληση στην εφαρμογή μας στο .env αρχείο με μορφή:
  MONGODB_URI=
  mongodb+srv://<username>:<password>@cluster0.mongodb.net/<dbname>?retryWrites=true&w=majority και η σύνδεση είναι επιτυχής.

Χρησιμοποιείται επίσης ένα global variable το _mongoClientPromise για να αποθηκεύσει το promise της σύνδεσης. Αυτό μας βοηθάει στο να μη γίνεται κάθε φορά καινούργια σύνδεση όταν εκτελείται ο κώδικας.

Τέλος μας επιστρέφεται το clientPromise, το οποίο μπορεί να χρησιμοποιηθεί και σε άλλα σημεία της εφαρμογής και πραγματοποιείται η σύνδεση.

Μερικά παραδείγματα ακόμα με εντολές της MongoDB από την εφαρμογή μας (για το collection των users):

```
async function init(): Promise<void> {
  if (db) return;
  try {
    client = await clientPromise;
    db = await client.db();
    users = await db.collection("users");
  } catch (error) {
    console.error("Error during initialization:", error);
    throw new Error("Failed to establish connection to database");
  }
}
```

Σε αυτό το κομμάτι κώδικα δημιουργούμε μια συνάρτηση η οποία χρησιμοποιεί το clientPromise που αναφέρθηκε προηγουμένως και μας συνδέει με το collection των users.

```
export const checkUserExists = async (username: string, email: string) =>
{
  if (!users) throw new Error("Users collection is not initialized");
  try {
    const usernameExists = await users.findOne({ username });
    const emailExists = await users.findOne({ email });

    return {
      usernameExists: !!usernameExists,
      emailExists: !!emailExists,
    };
  } catch (error) {
    console.error("Failed to check user existence:", error);
    throw new Error("Failed to check user existence");
  }
};
```

Σε αυτό το κομμάτι κώδικα δημιουργούμε μια συνάρτηση η οποία χρησιμοποιείται στο κομμάτι του sign up για να ελέγξει αν υπάρχει ήδη κάποιος χρήστης στο collection των users. Χρησιμοποιείται η συνάρτηση findOne της mongodb για να κάνει αυτό.

Ο τελεστής !! χρησιμοποιείται για να μετατρέψουμε την τιμή που θα επιστρέψει το findOne σε Boolean. Αν η findOne μας επιστρέψει ένα έγγραφο (δηλαδή αν βρεθεί ο χρήστης) τότε το !!usernameExists γίνεται true διαφορετικά γίνεται false.

4.3.5 Βιβλιοθήκες και εργαλεία

Οι βιβλιοθήκες και τα εργαλεία που επιλέξαμε για την εφαρμογή μας είναι τα παρακάτω:

- **Nodemailer:** Το nodemailer είναι μια βιβλιοθήκη που χρησιμοποιούμε για να στέλνουμε verification message στο email που έδωσε ο χρήστης κατά τη διάρκεια της εγγραφής του. Αυτό το κάνουμε για να επαληθεύσει ο χρήστης το email του.
- **UploadThatThing:** Το uploadThatThing είναι μια βιβλιοθήκη που χρησιμοποιείται με σκοπό να μπορούν οι χρήστες να ανεβάσουν δικές τους φωτογραφίες στην εφαρμογή.
- **Swiper:** Το swiper είναι ένα npm πακέτο το οποίο μας επιτρέπει να δημιουργούμε εύκολα καρουζέλ εικόνων. Χρησιμοποιείται για να εμφανίσουμε τις εικόνες με έναν οπτικά όμορφο τρόπο.
- **bcrypt:** Το bcrypt χρησιμοποιείται για να μπορούμε να αποθηκεύουμε τον κωδικό ενός χρήστη με hashing μέσα στη βάση. Αυτό εγγυάται στον χρήστη ότι τα δεδομένα του είναι ασφαλή μέσω της κρυπτογράφησης.
- **crypto:** Το crypto είναι ένα πακέτο της Node που παρέχει λειτουργίες όπως κρυπτογράφηση και αποκρυπτογράφηση, έτσι ώστε να υπάρχει ασφαλή διαχείριση των δεδομένων. Στην εφαρμογή μας, το χρησιμοποιούμε για να προσθέσουμε ένα τυχαίο verification token (δεκαεξαδικό) στην εφαρμογή μας, από την στιγμή που ο χρήστης κάνει την εγγραφή του μέχρι να κάνει verify από το email του, όπου τότε καταργείται.
- **Sitemap:** Το sitemap ή sitemap.xml όπως το έχουμε στην εφαρμογή, είναι ένα αρχείο το οποίο βοηθά την Google να χαρτογραφήσει την σελίδα αναλύοντας τη δομή της. Αυτό βοηθά και τα αποτελέσματα αναζήτησης (SEO). Το αρχείο αυτό τοποθετείται στον public φάκελο. Στην εφαρμογή μας χρησιμοποιούμε ένα πακέτο της Node που κάνει αυτοματοποιημένα αυτή τη διαδικασία και λέγεται next-sitemap.
- **react-icons:** Τα react-icons είναι ένα πακέτο της Node που χρησιμοποιεί σετ εικονιδίων που είναι σχεδιασμένα ειδικά για τη React και επομένως καθιστά τη χρήση τους πιο εύκολη στον κώδικά μας.
- **sharp:** Το sharp είναι ένα πακέτο της Node.js που χρησιμοποιείται για επεξεργασία εικόνων. Το χρησιμοποιούμε στην σελίδα μας γιατί συνεισφέρει στη βελτιστοποίηση των εικόνων που κάνουμε render και στην συνολική αύξηση της απόδοσης.

Χρησιμοποιήθηκαν επίσης τα:

RAWG API, TMDB API, Tailwind CSS, Typescript, MongoDB, GitHub στα οποία έχει ήδη γίνει αναφορά.

4.3.6 Παραδείγματα server components

- Cards/TVShowCards components στο κομμάτι των ταινιών

Το component αυτό χρησιμοποιείται για να εμφανιστούν οι πληροφορίες που κάνουμε fetch από το TMDB API σε μορφή καρτών.

```
import {
  getVotecolor,
  imageURL,
  Movie,
  MovieProps,
} from "@/app/Constants/constants";
import { FaStar } from "react-icons/fa";
import Link from "next/link";
import AddToWatchlist from "./AddToWatchlist";

const Cards = ({ movieData, upcoming, movieResultData }: MovieProps ) => {
  const currentDate = new Date().toISOString().split("T")[0];
  return upcoming && movieResultData ? (
    <>
      {movieResultData.map((item) => (
        <Link
          key={item.id}
          href={`/Movies/${item.id}`}
          className="hover:scale-110 hover:z-9 md:hover:border
lg:hover:border md:hover:shadow-2xl lg:hover:shadow-2xl card-link
lg:hover:shadow-gray-600 md:hover:shadow-gray-600 transition w-[90%] md:w-
full lg:w-full h-1/2 md:h-full lg:h-full md:mb-0 lg:mb-0 duration-500 ease-
in-out"
        >
          {/* Container for the image and content */}
          <div className="flex flex-col items-center">
            {/* Image container */}
            <div className="relative w-full h-56 sm:h-56 lg:h-96">
              <img
                src={`${imageURL}${item.poster_path}`}
                alt={item.title}
                className="absolute w-full h-full object-cover"
              />
            </div>
```

```
            {/* Text container */}
            <div className="bg-[#4c545b] w-full h-44 gap-4 p-4 card-text-
container">
              <div className="flex justify-between text-white">
                <h2 className="lg:text-xl">{item.title}</h2>
              </div>
              <p className="mt-4 text-white">{item.overview.slice(0,
40)}...</p>
            </div>
          </div>
        </Link>
      ))}
    </>
  ) : ( …
```

Εδώ παρατηρούμε ένα κομμάτι κώδικα του Cards.tsx component. Στο component αυτό περνάμε ως props το movieData που είναι οι πληροφορίες για κάποια ταινία που τις παίρνουμε μέσω του TMDB API, το upcoming είναι μια Boolean μεταβλητή που αλλάζει το περιεχόμενο ανάλογα με το αν υπάρχει ή όχι και το movieResultData είναι μια μεταβλητή που περιέχει τα αποτελέσματα του TMDB API σε μορφή πίνακα για συγκεκριμένη χρήση.

- GameList.tsx στην κατηγορία των Games

Το GameList είναι ένα server component που αφορά την εμφάνιση των παιχνιδιών σε μορφή λίστας καρτελών με κάθετη διάταξη. Είναι επαναχρησιμοποιούμενο σε υποσελίδες της σελίδας που αφορούν την εμφάνιση των games, με κριτήρια όπως κονσόλες, είδη παιχνιδιών κτλ.

Το παράρτημα κώδικα είναι το παρακάτω:

```
const GameList: React.FC<GameListProps> = ({ paginatedGames }) => {
  return (
    <>
      <ul className="relative pointer-events-none flex mt-6 mb-12 w-
         full flex-col items-center justify-center xl:gap-12 gap-16">
        {paginatedGames.map(
          (item, index) =>
            item.description_raw && (
              <li
                key={`${item._id}-${index}`}
                className="text-slate-200 pointer-events-auto text-
               balance text-lg hover:scale-105 xl:w-3/5 md:w-4/5 w-4/5
               transition-all duration-500 ease-in-out"
              >
                <Link
                  href={`/Games/${item.slug}`}
```

```
                    className="relative flex group border-2 md:h-60 h-
                   [32rem] max-[550px]:h-[25rem] border-white rounded-lg
                    transition-all duration-300"
                  >
                   <div className="bg-black overflow-hidden rounded-lg
                    bg-opacity-[.7] relative flex flex-col md:flex-row
                    md:gap-0 gap-0 transition-all duration-400">
                     <div className="relative md:w-[25rem] md:h-[15rem]
                       w-full h-[20rem] max-[550px]:h-[15rem] max-
                       [416px]:h-[10rem] flex-shrink-0 flex-grow-0">
                       <img
                         src={item.background_image}
                         alt={item.name}
                         className="w-full h-full md:border-r-4 object-
                              cover border-none rounded-l-lg border-white
                              transition duration-500 ease-in-out"
                       />
                     </div>
                     {/* Item name on hover */}
                     <div
                       className="
                     h-10 opacity-100
                     md:h-0 md:opacity-0 md:group-hover:opacity-100
                     md:group-hover:h-10 md:max-w-80 min-[550px]:max-w-
                     60 max-w-48
                     absolute flex items-center border border-black bg-
                       black md:rounded-b-xl rounded-br-xl text-md max-
                       [440px]:text-sm md:ml-3 ml-0 p-1
                     transition-all duration-500 ease-in-out
                   "
                       >
                     <span className="text-white truncate">{item.name}
                   </span>
```

4.3.7 Παραδείγματα client components:

- Nav.tsx (Movies): Αυτό το component υλοποιεί την μπάρα πλοήγησης στο κομμάτι των ταινιών. Ανήκει στα client components γιατί απαιτεί αλληλεπίδραση με τον χρήστη καθώς και στο API του browser (πχ το window).

```
"use client";
//components
```

```
import HomePage from "@/app/Components/Movie-components/Nav-
items/HomePage";
import Random from "@/app/Components/Movie-components/Nav-items/Random";
import Trending from "@/app/Components/Movie-components/Nav-
items/Trending";
import MovieHomePage from "@/app/Components/Movie-components/Nav-
items/MovieHomePage";
import ChangeTheme from "./Nav-items/ChangeTheme";
import Search from "./Nav-items/Search";
import TVShowHomePage from "@/app/Components/Movie-components/Nav-
items/TVShowsHomePage";
import UpComing from "@/app/Components/Movie-components/Nav-
items/UpComing";

//utils and icons
import React, { useEffect, useState } from "react";
import Link from "next/link";
import { FaMagnifyingGlass } from "react-icons/fa6";
import Login from "./Nav-items/Login";
import { useSession } from "next-auth/react";

export default function Nav() {
  const { data: session } = useSession();
  const [searchVisible, setSearchVisible] = useState(false);
  const [userId, setUserId] = useState<string>();
  const [imageUrl, setImageUrl] = useState<string>("");
  const [prevScrollPos, setPrevScrollPos] = useState(0); //Variable that
initializes the user's scroll to 0

  //Used for the scrolling in nav
  useEffect(() => {
    const handleScroll = () => {
      const currentScrollPos = window.scrollY; //CurrentScrollPos is the
position of the user's scroll
      const isSmallScreen = window.innerWidth < 1024; //Check if the screen
is smaller than 1024px

      //If the user has scrolled down and the screen is smaller than 1024px
then we hide the nav using a custom css class
      if (isSmallScreen)
        currentScrollPos > prevScrollPos
          ? document.getElementById("scroll-nav")?.classList.add("hide-
nav")
          : document.getElementById("scroll-nav")?.classList.remove("hide-
nav");

      setPrevScrollPos(currentScrollPos); //We update the currentScroll
position
```

```
    };

    window.addEventListener("scroll", handleScroll);

    return () => {
      window.removeEventListener("scroll", handleScroll);
    };
  }, [prevScrollPos]);
```

```
  return (
    <div className="nav-for-hover dummy-class">
      <nav
        id="scroll-nav"
        className="lg:w-20 lg:h-full sm:w-full overflow-auto overflow-x-
auto fixed bg-[#23232e] sm:z-10 sm:bottom-0 lg:hover:w-56 group duration-
700 ease-in-out not-search navbar"
      >
        <ul className="flex sm:flex-row lg:flex-col items-center p-0 m-0 h-
full not-search">
          <li className="text-[#b6b6b6] text-l w-full transition duration-
500 ease-in-out not-search last:mt-auto last:hover:none">
            <HomePage />
          </li>
          <li className="text-[#b6b6b6] text-l w-full [&:not(:last-
child)]:hover:bg-[#6B6B6B] transition duration-500 ease-in-out not-search
last:mt-auto last:hover:none">
            <MovieHomePage />
          </li>
          <li className="text-[#b6b6b6] text-l w-full [&:not(:last-
child)]:hover:bg-[#6B6B6B] transition duration-500 ease-in-out not-search
last:mt-auto last:hover:none">
            <TVShowHomePage />
          </li>
          <li className="text-[#b6b6b6] text-l w-full [&:not(:last-
child)]:hover:bg-[#6B6B6B] transition duration-500 ease-in-out not-search
last:mt-auto last:hover:none">
            <Trending />
          </li>
```

Εδώ παρατηρούμε ένα εισαγωγικό κομμάτι κώδικα από το αρχείο Nav.tsx των Movies, όπου μπορούμε να διακρίνουμε το καθοριστικό “use client” στην αρχή του αρχείου, αλλά και όλα τα απαραίτητα imports, όπως το Search, react-icons, κάποιες αρχικοποιήσεις με hooks της react κ.ά. Χρησιμοποιείται επίσης και το useEffect hook για να καθορίσουμε διάφορες λειτουργίες του navbar, όπως είναι αυτό στο παράδειγμα που διαχειρίζεται τη συμπεριφορά του navbar σε μικρότερες οθόνες.

Βλέπουμε επίσης και το html που μας επιστρέφει αυτό το component που αποτελείται στην ουσία από διάφορα components τα οποία περιέχουν Links για διάφορα σημεία της σελίδας.

- SearchBar.tsx (Games)

```
{search.map((result, index) => (
            <Link
              key={index}
              href={`/Games/${result.slug}`}
              className="flex items-center flex-row transition-all
               duration-300 ease-in-out hover:scale-105 pl-3
               hover:text-stone-400"
            >
              <div className="relative overflow-hidden sm:w-44 sm:h-32
               min-[420px]:w-40 min-[420px]:h-28 w-36 h-24 flex-shrink-
               0 flex-grow-0">
                <img
                  src={result.background_image}
                  alt={result.name}
                  className="w-full h-full object-contain"
                />
              </div>
              <div
                onClick={() => handleAutoComplete(result.name)}
                className="search-result py-1.5 cursor-pointer flex
               flex-col text-md pl-6 pr-4"
              >
                {result.name}
              </div>
            </Link>
          ))}
```

Εδώ παρατίθεται το σημαντικότερο κομμάτι από το αρχείο SearchBar.tsx των Games, όπου γίνεται η εμφάνιση των αποτελεσμάτων αναζήτησης σύμφωνα με το τι πληκτρολογεί ο χρήστης σε αντιστοιχία με τα παιχνίδια που υπάρχουν στη βάση. Τα παιχνίδια περνιούνται σε ένα state hook της React με όνομα search όπως φαίνεται παραπάνω.

5. Build-Deploy του CineGame-Critic

Για το build και το deploy της εφαρμογής χρησιμοποιήσαμε την Vercel, που είναι εξειδικευμένη στις Next.js εφαρμογές. Η Vercel είναι μια cloud πλατφόρμα, δομημένη με serverless αρχιτεκτονική, που κάνει τη διαδικασία deploy για front-end εφαρμογές αρκετά εύκολη. Ωστόσο, δεν μένει εκεί. Με τις συνεχείς ενημερώσεις της, η πλατφόρμα πλέον υποστηρίζει χαρακτηριστικά όπως, την άμεση αλληλεπίδραση του GitHub repository με τα αρχεία της σελίδας, καθώς και την δυνατότητα ζωντανής επίβλεψης της απόδοσης και της διαδραστικότητας της ιστοσελίδας. Αποτελεί καινούργιο προϊόν στην αγορά καθώς ιδρύθηκε τον Νοέμβριο του 2015 και από τότε εξελίσσεται συνεχώς.

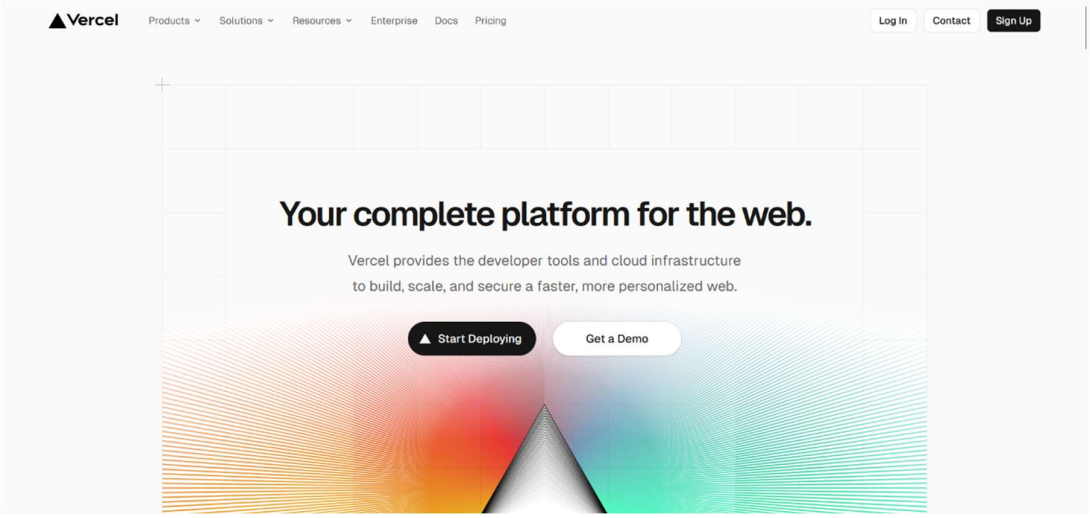
5.1 Πλεονεκτήματα Vercel

- **Serverless functions:** Η Vercel μας επιτρέπει να χρησιμοποιούμε κώδικα back-end χωρίς να χρειάζεται να διαχειριζόμαστε έναν διακομιστή. Ένα παράδειγμα αυτού, είναι το σύστημα του authentication που χρησιμοποιούμε.
- **Αυτόματο Build:** Η Vercel κάνει κάθε φορά την εφαρμογή build αυτόματα όταν γίνεται κάποιο push στο repository.
- **CDN (Content Delivery Network):** Η Vercel χρησιμοποιεί ένα CDN για να διανέμει την εφαρμογή σε ολόκληρο τον κόσμο.
- **Εύκολη ρύθμιση:** Είναι πολύ εύκολο να γίνει η διαδικασία του build και του deploy, καθώς η Vercel ανιχνεύει και ρυθμίζει αυτόματα τα Next.js projects.

5.2 Διαδικασία Build και Deploy

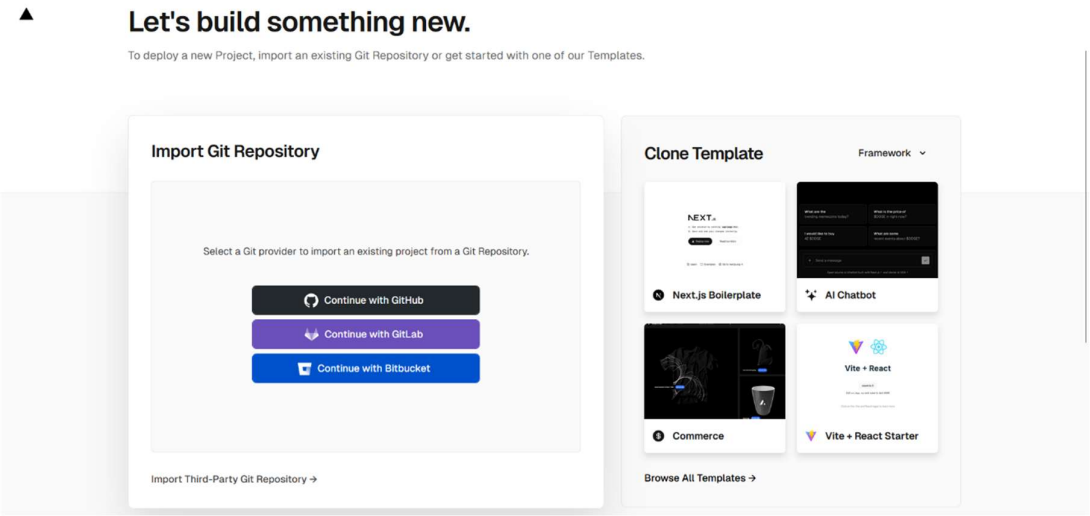
Η διαδικασία του Build και του deploy είναι πολύ απλή στη Vercel. Αφού συνδέσουμε τον λογαριασμό του GitHub μας στην Vercel, επιλέγουμε το Repository που έχουμε τον κώδικα της σελίδας. Η Vercel ανιχνεύει αυτόματα ότι πρόκειται για μια Next.js εφαρμογή και κάνει τις κατάλληλες ρυθμίσεις. Στη συνέχεια το deployment γίνεται απλά με ένα click και η Vercel κάνει build την εφαρμογή και την ανεβάζει στο cloud.

Αφού μπούμε στη σελίδα της Vercel κάνουμε κλικ στο start deploying.



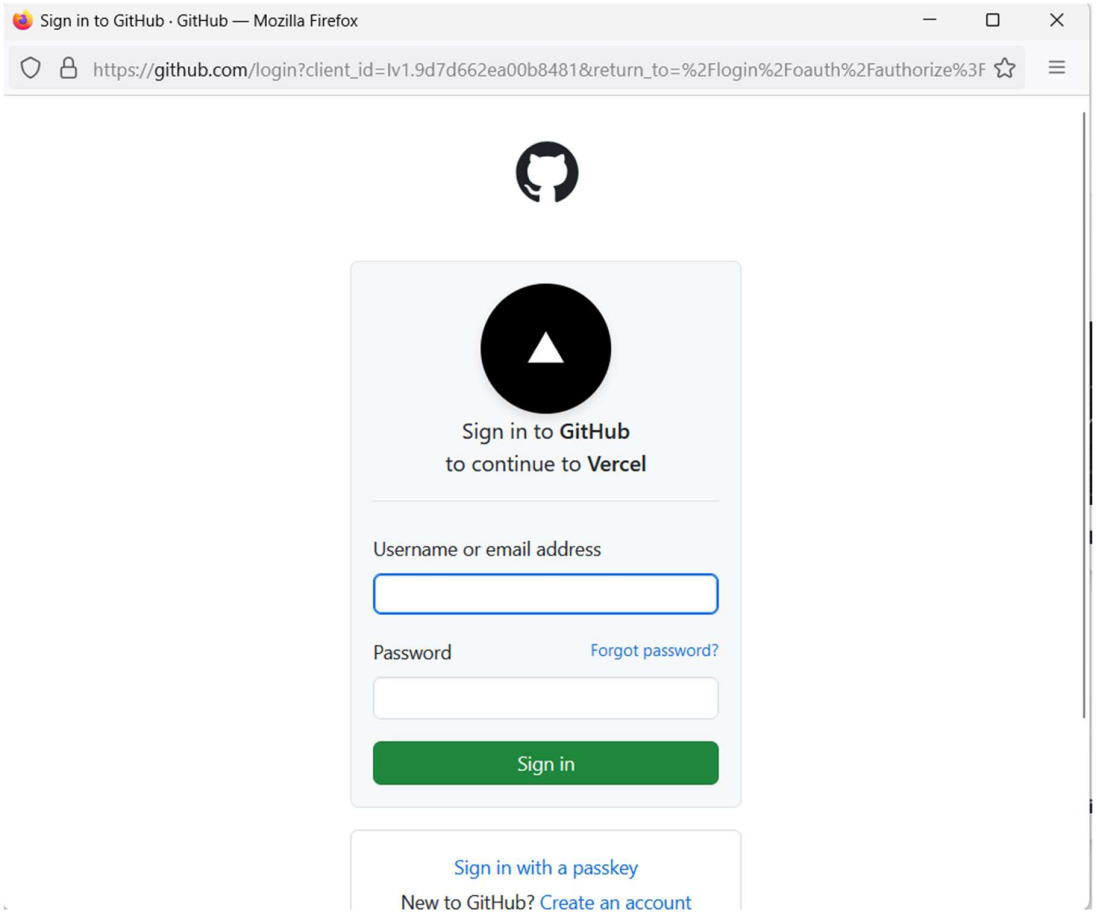
Εικόνα 9 . Αρχική σελίδα της Vercel.

Στη συνέχεια η Vercel μας παραπέμπει στο να συνδεθούμε με έναν από τους τρεις τρόπους. GitHub, GitLab, Butbucket.



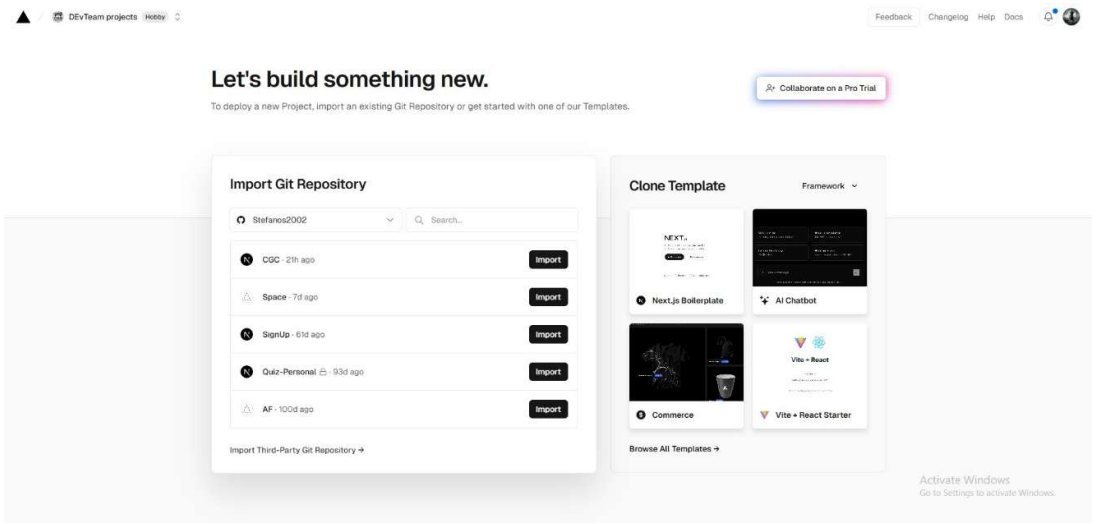
Εικόνα 10 . Σύνδεση στη σελίδα της Vercel.

Στη δικιά μας περίπτωση επιλέξαμε να κάνουμε την σύνδεση μέσα από το GitHub μιας και ήταν αυτό στο οποίο φτιάξαμε το repository της εφαρμογής.



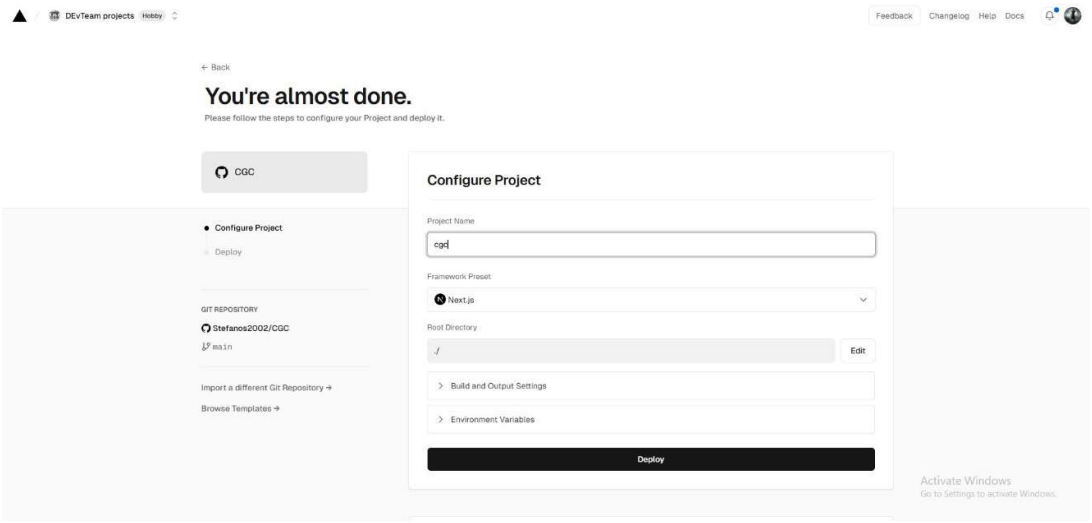
Εικόνα 11 . Σύνδεση στον GitHub λογαριασμό μας.

Αφού συνδεθούμε, η Vercel μας βάζει να διαλέξουμε με ποιο GitHub repository θέλουμε να κάνουμε import.



Εικόνα 12 . Επιλογή repository στην Vercel.

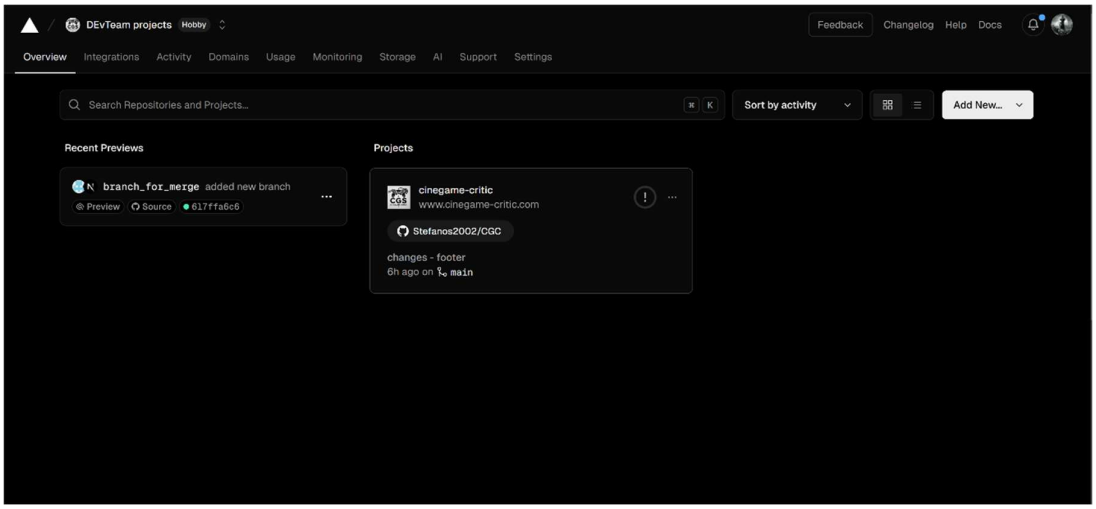
Μόλις το επιλέξουμε μας βγάζει στη σελίδα του configure. Σε αυτό το σημείο μπορούμε να δώσουμε περισσότερες ρυθμίσεις όσον αφορά το build καθώς επίσης να του περάσουμε τις μεταβλητές που έχουμε ορίσει στο .env αρχείο.



Εικόνα 13 . Project configuration στη σελίδα της Vercel.

Αφού γίνουν όλα αυτά τότε η σελίδα έχει πλέον γίνει deployed και μας δίνονται και κάποιες ακόμα επιλογές όπως είναι το να δώσουμε ένα custom domain name που μπορεί να έχουμε αγοράσει από κάποιο hosting provider.

Τέλος βλέπουμε τα projects που έχουμε σε ένα dashboard.



Εικόνα 14 . Project Dashboard στη σελίδα της Vercel.

6. After-Launch: Ρύθμιση & Βελτιστοποίηση Σελίδας

Σε αυτό το κεφάλαιο αναλύουμε όλα τα βήματα που ακολουθήσαμε μετά το αρχικό deployment της σελίδας, έτσι ώστε να κάνουμε την εφαρμογή κατάλληλη για δημόσια χρήση. Αυτό περιλαμβάνει την αγορά domain name, την χρήση εργαλείων της google για να παρακολουθούμε και να αναλύουμε την επισκεψιμότητα στην σελίδα καθώς επίσης και να αναδειχθεί στα SEO ranks, με κατάλληλη παραμετροποίηση του κώδικα.

6.1 Domain

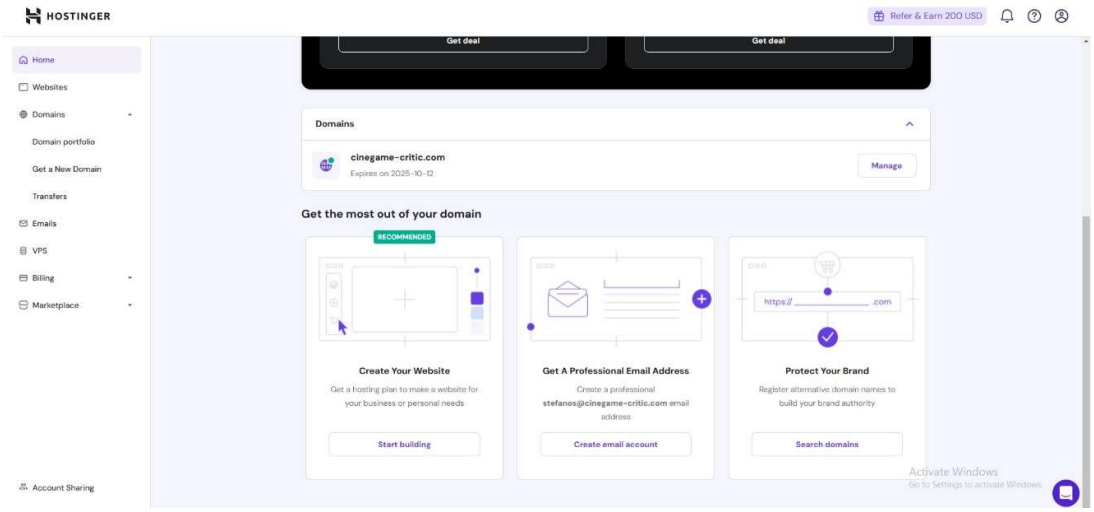
Για τη χρήση domain name, επιλέξαμε την πλατφόρμα Hostinger, από την οποία βρήκαμε διαθέσιμο το .com όνομα, που ταιριάζει στην επωνυμία της εφαρμογής μας. Αφού ολοκληρώσαμε την αγορά, προχωρήσαμε στα επόμενα βήματα διασύνδεσής του με το Vercel. Να σημειωθεί ότι το Hostinger παρέχει τη δυνατότητα φιλοξενίας της εφαρμογής μας στην πλατφόρμα του. Επομένως, θα μπορούσαμε να επιλέξουμε το Hostinger για να κάνουμε host τη σελίδα μας, αντί για το Vercel.

6.1.1 DNS records/settings

Τα DNS (Domain Name System) records είναι εγγραφές που κατευθύνουν το domain προς τον κατάλληλο Server φιλοξενίας, στην περίπτωσή μας το Vercel. Μέσω αυτών των ρυθμίσεων, εξασφαλίζουμε ότι οι χρήστες που πληκτρολογούν το όνομα του domain μας θα οδηγούνται στη σελίδα μας.

Σε αυτήν την ενότητα, θα δούμε τη διαδικασία προσθήκης των απαραίτητων εγγραφών, όπως το **A record** και το **CNAME record**, που απαιτούνται για τη σύνδεση του domain με το Vercel. Στη συνέχεια, παρατίθενται τα screenshots της διαδικασίας αυτής.

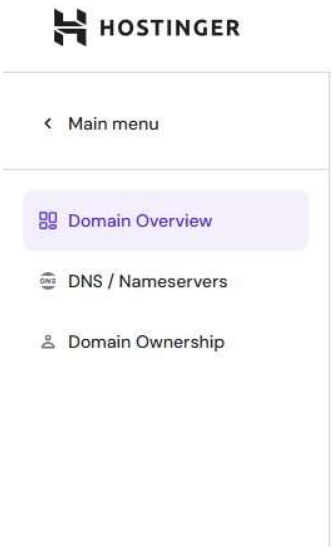
Αρχικά, πηγαίνοντας στην Αρχική του Vercel και έπειτα Project --> Settings --> Domains, θα δούμε την επιλογή για add, όπου προσθέτουμε το custom domain μας (π.χ mysitename.com ή www.mysiteame.com). Στη συνέχεια κατευθυνόμαστε στο Hostinger.



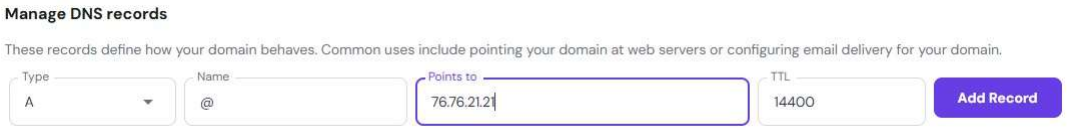
Εικόνα 15 . Αρχική σελίδα Hostinger.

Μόλις ο χρήστης βρεθεί στην αρχική σελίδα του Hostinger, στο πλαίσιο που εμφανίζονται τα Domains του κάνει κλικ στην επιλογή Manage στο Domain που θέλει να ρυθμίσει.

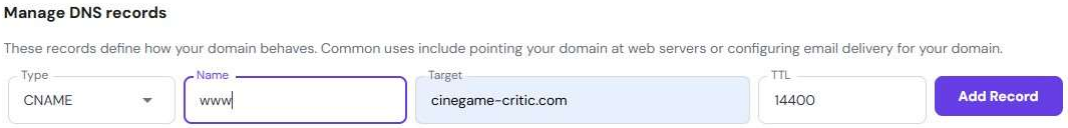
Στη συνέχεια ο χρήστης θα μεταφερθεί σε μια σελίδα, όπου στο αριστερο side-menu που του εμφανίζεται, επιλέγει στο DNS / Nameservers. Έπειτα, θα βρεθεί σε ένα πλαίσιο που θα περιλαμβάνει όλα τα DNS records, όπου θα πρέπει να προσθέσει τα κατάλληλα DNS records, σύμφωνα με τις προδιαγραφές του Vercel (είναι αυτά που φαίνονται στις εικόνες). Μόλις ο χρήστης προσθέσει τις εγγραφές, η ιστοσελίδα θα γίνει προσβάσιμη στο διαδίκτυο.



Εικόνα 16 . Side-menu του manage domain στο Hostinger.



Εικόνα 17 . Επεξεργασία A Record στην σελίδα του Hostinger.



Εικόνα 18 . Επεξεργασία CNAME Record στη σελίδα του Hostinger.

6.2 Google Search Console (GSC)

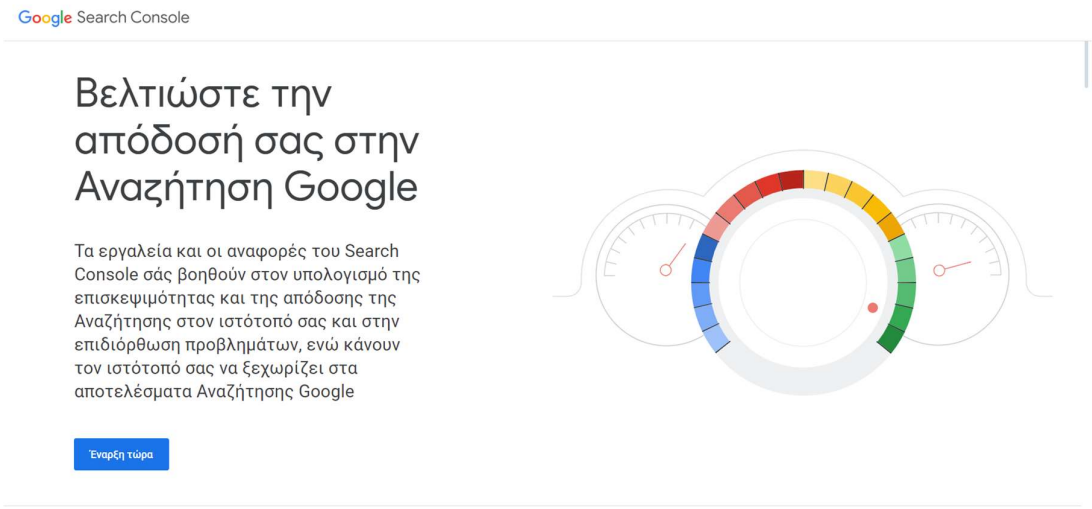
Το επόμενο βήμα που ακολουθήσαμε, ήταν να κάνουμε επαλήθευση της ιστοσελίδας, με τη βοήθεια του Google Search Console. Το Google Search Console αποτελεί ένα δωρεάν εργαλείο της Google που βοηθά τον χρήστη να παρακολουθεί και να διαχειρίζεται την εμφάνιση της ιστοσελίδας στα αποτελέσματα της Google.

Συγκεκριμένα προσφέρει τις εξής παροχές:

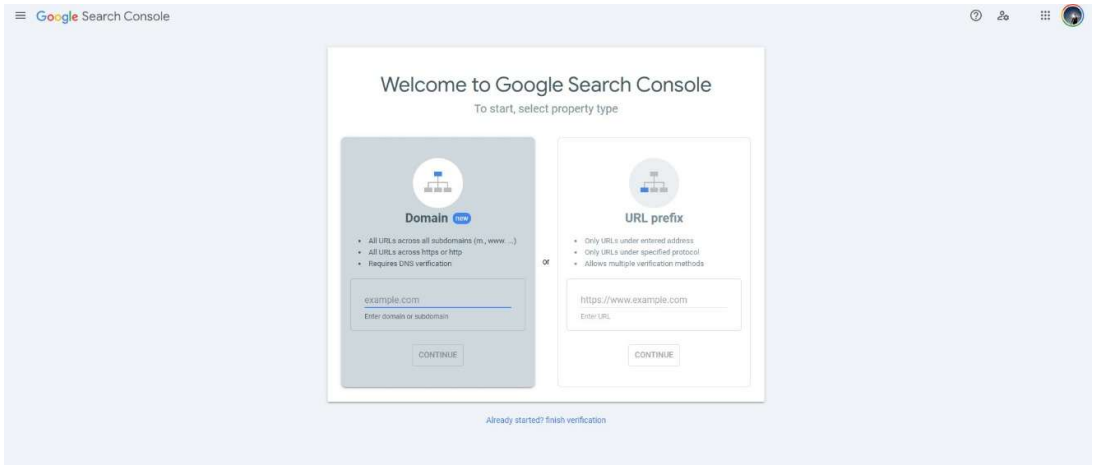
- Επιβεβαιώνει ότι η Google μπορεί να ανιχνεύσει την ιστοσελίδα
- Λαμβάνει ειδοποιήσεις όταν η Google συναντάει spam mails ή άλλα προβλήματα
- Σου δείχνει ποιές ιστοσελίδες συνδέονται με την δική σου
- Παρακολουθεί το απόδοση(traffic) των αποτελεσμάτων αναζήτησης, δηλαδή πόσο συχνά εμφανίζεται η ιστοσελίδα στους χρήστες ή πόσες φορές έχει πατηθεί το link.
- Διορθώνει κάποια θέματα χαρτογράφησης της σελίδας και παρέχει τη δυνατότητα αίτησης επαναχαρτογράφησης για ανανέωση του περιεχομένου.

6.2.1 Διαδικασία Validation/GSC

Από την αρχική σελίδα του Google Search Console επιλέγουμε "Έναρξη τώρα".

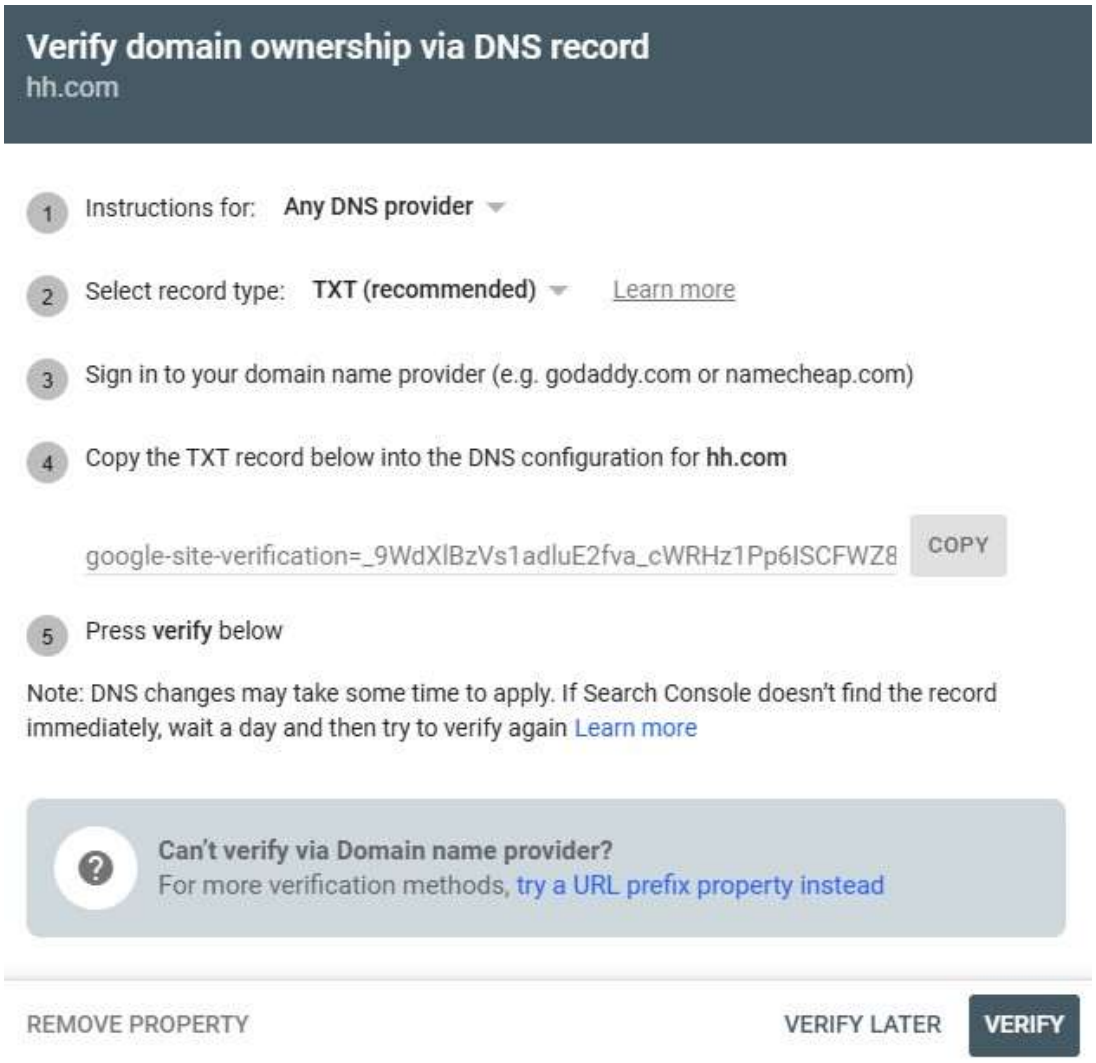


Εικόνα 19 . Αρχική σελίδα Google Search Console.



Εικόνα 20 . Επιλογή Domain ή URL prefix στο Google Search Console.

Από τις δύο επιλογές που εμφανίζονται, ο χρήστης πρέπει να επιλέξει αν θέλει να κάνει επαλήθευση μόνο για μια συγκεκριμένη διεύθυνση (URL prefix), ή για όλο το Domain, που περιλαμβάνει όλες τις διευθύνσεις κάτω από αυτό, συμπεριλαμβανόμενων των http και https πρωτοκόλων, καθώς και αποκλειστικότητα επαλήθευσης με DNS record. Εμείς συνεχίσαμε με την επιλογή Domain στην εφαρμογή μας.



Εικόνα 21 . Επαλήθευση κατοχής Domain στο Google Search Console.

Εδώ φαίνεται ένα ενδεικτικό παράδειγμα για το επόμενο βήμα της επιλογής Domain, όπου ο χρήστης πρέπει να προσθέσει το google-site-verification σαν txt record στα DNS settings της Hostinger ή οποιουδήποτε άλλου παρόχου χρησιμοποιεί.

Στη συνέχεια, θα έχει ολοκληρωθεί η διαδικασία επαλήθευσης.

6.3 Παραμετροποίηση Κώδικα για SEO

Εκτός των παραπάνω ρυθμίσεων, χρειάστηκε να κάνουμε κάποιες αλλαγές στο κομμάτι του κώδικα, για να βοηθήσουμε ουσιαστικά την διαδικασία που ανέλαβε το Google Search Console να αναζητήσει και να χαρτογραφήσει την ιστοσελίδα μας. Για τις αλλαγές εδώ υπάρχει μια πληθώρα επιλογών, με διάφορους τρόπους που

μπορεί ένας χρήστης να βελτιστοποιήσει την απόδοση της ιστοσελίδας στο SEO. Σε αυτή την ενότητα, θα αναφέρουμε τους βασικούς που χρησιμοποιήσαμε εμείς.

6.3.1 Προσαρμογή του layout.tsx

```
export const metadata: Metadata = {
  title: "CineGame Critic - Reviews of Movies & Games",
  description:
    "CineGame Critic provides the latest reviews and ratings for movies
and video games. Stay updated with our curated reviews from critics.",
  keywords:
    "movies, games, reviews, articles, information, entertainment,
ratings, film critique, video game critique",
  authors: [
    {
      name: "Stefanos Kaloulis",
      url:     "www.linkedin.com/in/stefanos-kaloulis-b4ba792b6",     //
Optional: link to author profile
    },
    {
      name: "Apostolos Kyrgidhs",
      url: "www.linkedin.com/in/apostolos-kyrgidis/", // Optional: link
to author profile
    },
  ],
  robots: "index, follow",
};
```

Εδώ φαίνεται η δήλωση του object metadata μέσα στο layout αρχείο και το interface για την δήλωση των τύπων πιο πάνω. Ουσιαστικά μέσα από εδώ, μπορούμε να αλλάξουμε σημαντικές πληροφορίες που αφορούν την εμφάνιση στα αποτελέσματα αναζήτησης. Κάποια από αυτά είναι ο τίτλος της σελίδας, η περιγραφή για το τι αφορά, κάποια keywords που προωθούν την αναγνωρισιμότητα κ.ά.

Σημαντικό είναι να υπογραμμίσουμε την τελευταία μεταβλητή robots: “index,follow”, η οποία αποτελεί οδηγία για τα bots της εκάστοτε μηχανής αναζήτησης, για το πως να διαχειριστούν την σελίδα.

Αναλυτικότερα, η εντολή index είναι υπεύθυνη για να εμφανίζεται επιτυχώς η σελίδα στα αποτελέσματα αναζήτησης. Η εντολή follow από την άλλη, λέει στις μηχανές αναζήτησης να ακολουθούν σωστά και να λαμβάνουν υπόψη τα link στη σελίδα. Επομένως ο συνδυασμός τους συμβάλλει στην καλύτερη προώθηση της σελίδας.

```
<title>
  {typeof metadata.title === "string"
    ? metadata.title
    : "Default Title"}
</title>
<meta
  name="description"
  content={
    typeof metadata.description === "string"
      ? metadata.description
      : "Default description goes here"
  }
/>
<link
  rel="icon"
  href="/assets/images/site-logo-cropped.png"
  sizes="32x32"
  type="image/png"
/>
<link
  rel="icon"
  href="/assets/images/site-logo-cropped.png"
  sizes="48x48"
  type="image/png"
/>
<link
  rel="icon"
  href="/assets/images/site-logo-cropped.png"
  sizes="672x672"
  type="image/png"
/>
```

Το παραπάνω παράρτημα κώδικα βρίσκεται στη συνέχεια του αρχείου layout.tsx και συγκεκριμένα μέσα στο head. Εδώ βλέπουμε τη χρήση των meta στοιχείων που αναφέρθηκαν πιο πάνω. Το icon που έχει εισαχθεί πάνω από μία φορά, είναι το logo της σελίδας και αποσκοπεί στην αύξηση των πιθανοτήτων ένας browser να το επιλέξει και να φανεί στα αποτελέσματα αναζήτησης.

6.3.2 Προσθήκη του next-sitemap

Το next-sitemap είναι ένα πακέτο για τις εφαρμογές της Next, το οποίο μας βοηθά να δημιουργήσουμε με αυτοματοποιημένο τρόπο sitemap.xml αρχείο καθώς και robots.txt (προαιρετικά). Αυτά τα αρχεία είναι κρίσιμα για το SEO, καθώς βοηθούν τις μηχανές αναζήτησης να κάνουν indexing (ευρετηριάσουν) τις σελίδες μιας ιστοσελίδας πιο αποδοτικά. Η αυτόματη διαχείριση που παρέχει το πακέτο είναι πιο αποτελεσματική σε μια ιστοσελίδα με πολλές δυναμικές σελίδες και για αυτόν τον λόγο το επιλέξαμε.

Το sitemap.xml που δημιουργείται έχει τον ρόλο ενημέρωσης στις μηχανές αναζήτησης για τις σελίδες που υπάρχουν σε μια ιστοσελίδα, πότε ενημερώθηκαν τελευταία και αν υπάρχουν παραλλαγές γλώσσας ή περιοχής. Έτσι, η ιστοσελίδα γίνεται πιο εύκολα ανιχνεύσιμη.

6.3.2.1 Διαδικασία εγκατάστασης του πακέτου και ρυθμίσεις

Το πακέτο το εγκαθιστούμε όπως και όλες τις άλλες εντολές της Node, με npm install next-sitemap. Στη συνέχεια δημιουργούμε ένα αρχείο next.sitemap.config.js, στο path του root, όπου γίνονται η κατάλληλες προσαρμογές πριν γίνει η εκτέλεση με την εντολή `npx next-sitemap.` Ένα παράδειγμα περιεχομένου του αρχείου είναι το παρακάτω:

```
   /** @type {import('next-sitemap').IConfig} */
   module.exports = {
     siteUrl: "https://www.cinegame-critic.com", // Set your site URL
     generateRobotsTxt: true, // Generate a robots.txt file
     exclude: [
       "/Movies/TVShows/*", // Exclude dynamic subpages under /TVShows, if
not necessary for SEO
       "/Signin", // Exclude login, signup, and other utility pages
       "/Signup",
       "/Verified",
     ],
     changefreq: "daily", // Set default change frequency
     priority: 0.7, // Set default priority for pages
     sitemapSize: 5000, // Max number of URLs per sitemap file
     additionalPaths: async (config) => [
       { loc: "/Movies/Movies-trending", changefreq: "daily", priority: 0.9
},
     ],
   };
```

Μέσα στο module.exports, μπορούμε να ρυθμίσουμε πληροφορίες όπως το siteUrl, εάν θέλουμε η εντολή να παράγει robots.txt αρχείο ή ακόμη να διευκρινίσουμε σελίδες που θέλουμε να εξαιρέσουμε από την εκτέλεση μετέπειτα.

Αναφορικά με το robots.txt αρχείο είναι σημαντικό για τη σωστή καθοδήγηση των μηχανών αναζήτησης (crawlers, bots). Με τη βοήθειά τους, εξακριβώνουμε σε ποια αρχεία (σελίδες) θέλουμε οι crawlers να έχουν πρόσβαση και σε ποια όχι. Ένα παράδειγμα της σύνταξής του είναι το εξής:

User-agent: *

Allow: /

Host: https://www.cinegame-critic.com

Sitemap: https://www.cinegame-critic.com/sitemap.xml

- **User-agent: ***: Η εντολή αυτή απευθύνεται σε όλα τα bots.
- **Allow:** Επιτρέπει την ανίχνευση των σελίδων στον φάκελο `/public/`.
- **Host:** Παρέχει τη διεύθυνση της σελίδας.
- **Sitemap:** Παρέχει τη διεύθυνση του sitemap, για να διευκολύνει τα bots να εντοπίσουν όλες τις σελίδες που πρέπει να ανιχνεύσουν.

*Σημείωση
Σε περίπτωση που δεν το επιλέξουμε ως true στο στοιχείο generateRobotsTxt, υπάρχει η δυνατότητα να το προσθέσουμε μετέπειτα στο path /public/. Το ίδιο ισχύει και για το sitemap.xml αρχείο.

7. Περιγραφή User Interface του CineGame-Critic.

7.1 Αρχική σελίδα

Εικόνα 22 . Αρχική σελίδα του CineGame-critic.

Εδώ βλέπουμε την αρχική σελίδα του CineGame-Critic η οποία αποτελείται από 2 grayscale εικόνες που όταν τις κάνουμε hover αποκαλύπτουν το όνομα της σελίδας που θέλουμε να πάμε και χρωματίζονται. Το δεξί μισό είναι το κομμάτι των ταινιών και το αριστερό των παιχνιδιών, όπως φαίνεται παρακάτω.

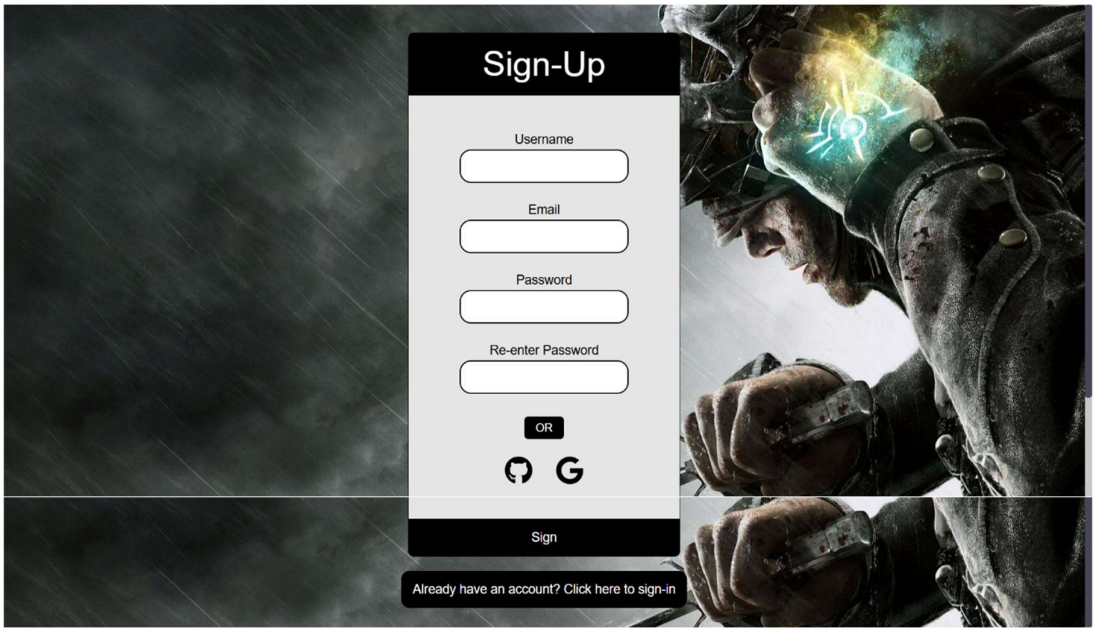
Εικόνα 23 . Hover στα Movies στην αρχική σελίδα.

Εικόνα 24 . Hover στα Games στην αρχική σελίδα.

7.2 Authentication

Παρακάτω θα δούμε τις σελίδες του Authentication της σελίδας μας (Sign up, Sign in).

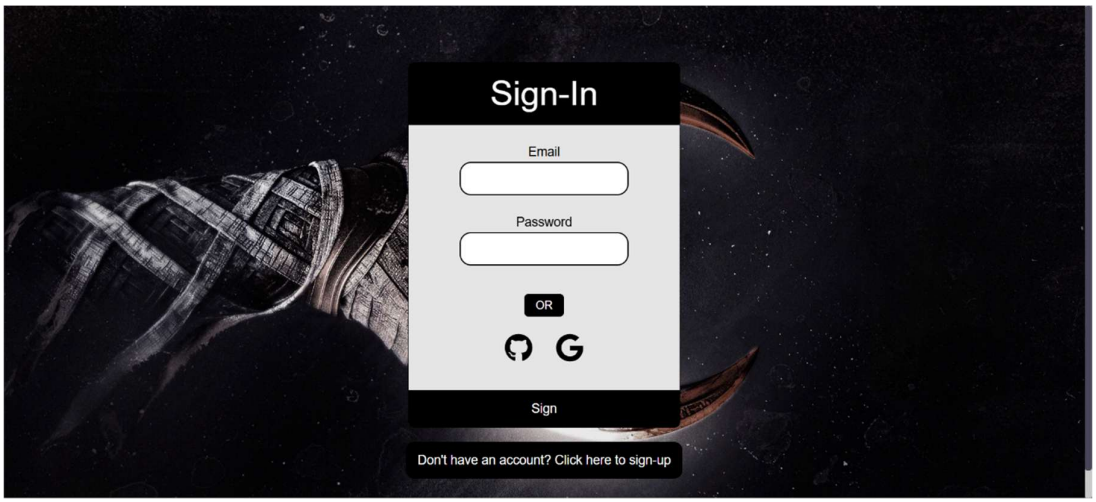
7.2.1 Sign Up



Εικόνα 25 . Σελίδα του Sign-up.

Εδώ βλέπουμε τη σελίδα που ο χρήστης μπορεί να κάνει signup. Ο χρήστης μπορεί να συνδεθεί με 3 τρόπους. Μέσω GitHub, μέσω Google και με τη δημιουργία λογαριασμού με δικό του email, username και password(credentials). Στη σελίδα αυτή υπάρχει επίσης η επιλογή να πάει κάποιος στο Sign in.

7.2.2 Sign In



Εικόνα 26 . Σελίδα του Sign-in.

Εδώ βλέπουμε τη σελίδα του Sign in. Στη σελίδα αυτή ένας χρήστης μπορεί να συνδεθεί στον υπάρχοντα λογαριασμό του με την εισαγωγή email και password(credentials). Μπορεί επίσης να συνδεθεί μέσω GitHub ή Google accounts. Υπάρχει επίσης και η δυνατότητα να ανακατευθυνθεί ο χρήστης στη σελίδα του Sign Up.

7.3 Games

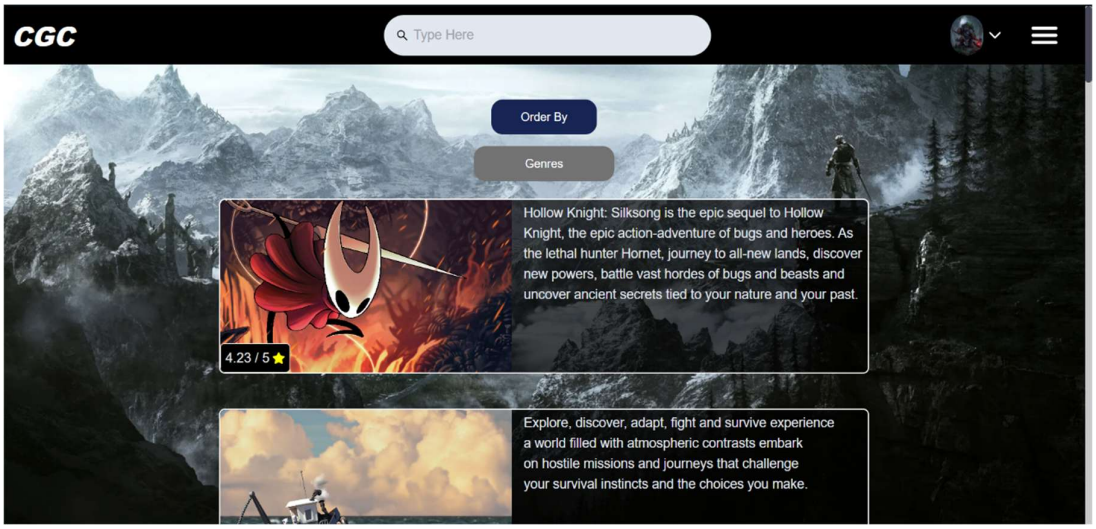
7.3.1 Navbar

Εικόνα 27 . Navbar των Games.

Εδώ βλέπουμε την μπάρα πλοήγησης ενός συνδεδεμένου χρήστη. Στη μπάρα αυτή ο χρήστης έχει αρκετές επιλογές για να κατευθυνθεί σε διάφορα σημεία της σελίδας. Αρχικά βλέπουμε το CGC, το οποίο μας ανακατευθύνει στην αρχική σελίδα της εφαρμογής. Έπειτα βλέπουμε την μπάρα αναζήτησης που είναι υπεύθυνη για την διευκόλυνση των χρηστών να βρουν ένα παιχνίδι που αναζητούν. Τέλος βλέπουμε την εικόνα προφίλ του συνδεδεμένου χρήστη και ένα μενού που περιέχει εικόνες από τις πιο γνωστές επωνυμίες κονσόλων και φιλτράρουν τα αποτελέσματα με βάση αυτού. Παρακάτω βλέπουμε το μενού αυτό αναπτυγμένο. Αν ο χρήστης δεν είναι συνδεδεμένος, τότε υπάρχουν και 2 ακόμα επιλογές, το Sign Up και το Sign In.

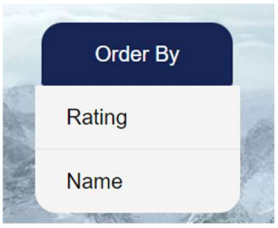
Εικόνα 28 . Μενού επιλογής κονσόλων των Games.

7.3.2 Αρχική σελίδα των Games

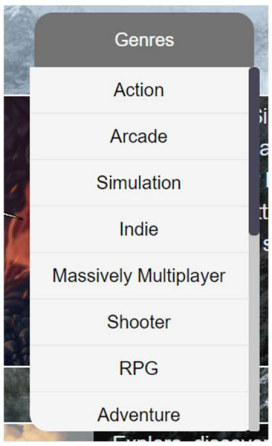


Εικόνα 29 . Αρχική σελίδα των Games.

Εδώ βλέπουμε την αρχική σελίδα των Games. Παρατηρούμε αρχικά την μπάρα πλοήγησης ενός συνδεδεμένου χρήστη, έπειτα βλέπουμε μια πληθώρα παιχνιδιών όλων των κατηγοριών. Παρατηρούμε επίσης το κουμπί “Order By”, το οποίο όταν πατηθεί εμφανίζει 2 επιλογές το “rating” που φιλτράρει το περιεχόμενο με βάση των αξιολογήσεων σε φθίνουσα σειρά και το “name” που τα φιλτράρει με αλφαβητική σειρά. Κάτω από το “Order By” παρατηρούμε και το κουμπί “Genres” που φιλτράρει το περιεχόμενο με βάση το είδος των παιχνιδιών. Παρακάτω φαίνονται τα 2 κουμπιά όταν έχουν πατηθεί.

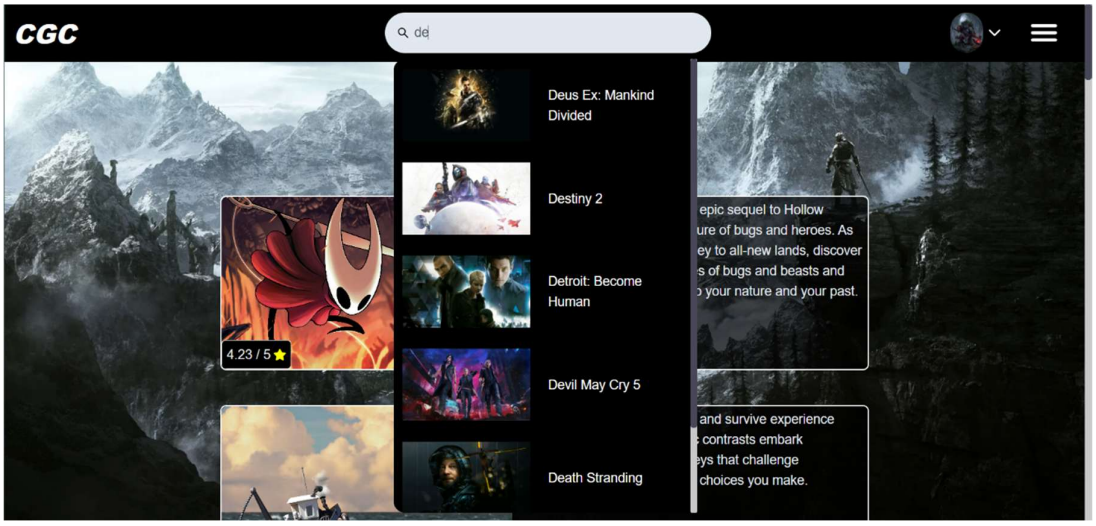


Εικόνα 30 .Επιλογές του Order By στα Games.



Εικόνα 31 . Επιλογές των Genres στα Games.

7.3.3 Μπάρα αναζήτησης

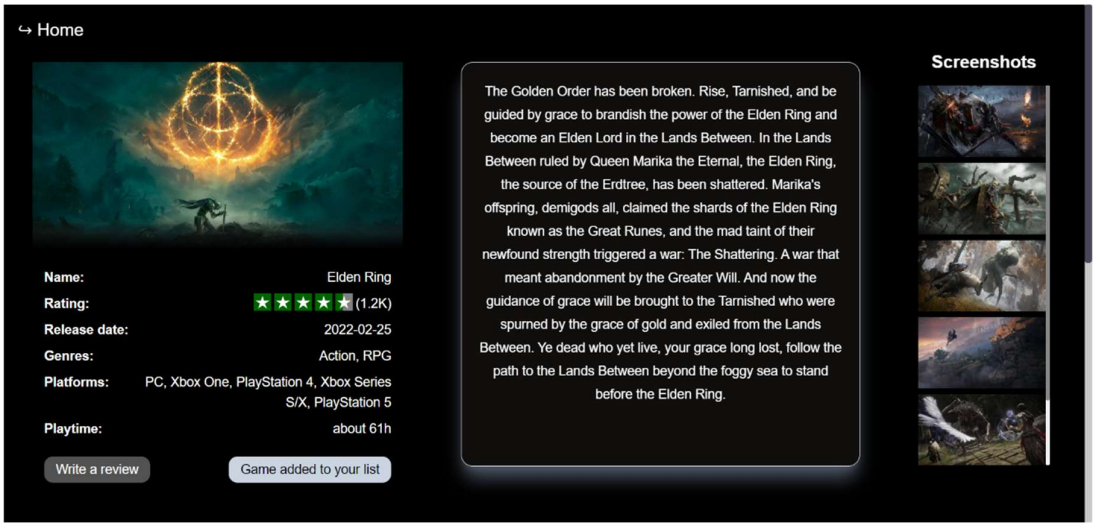


Εικόνα 32 . Searchbar στα Games.

Εδώ μπορούμε να αναζητήσουμε το παιχνίδι που επιθυμούμε. Όσο πληκτρολογούμε, εμφανίζονται αποτελέσματα από τα παιχνίδια που ταιριάζουν στην αναζήτηση μας. Στο παράδειγμα αυτό βλέπουμε με το input “de” εμφανίζονται διάφορα αποτελέσματα τα

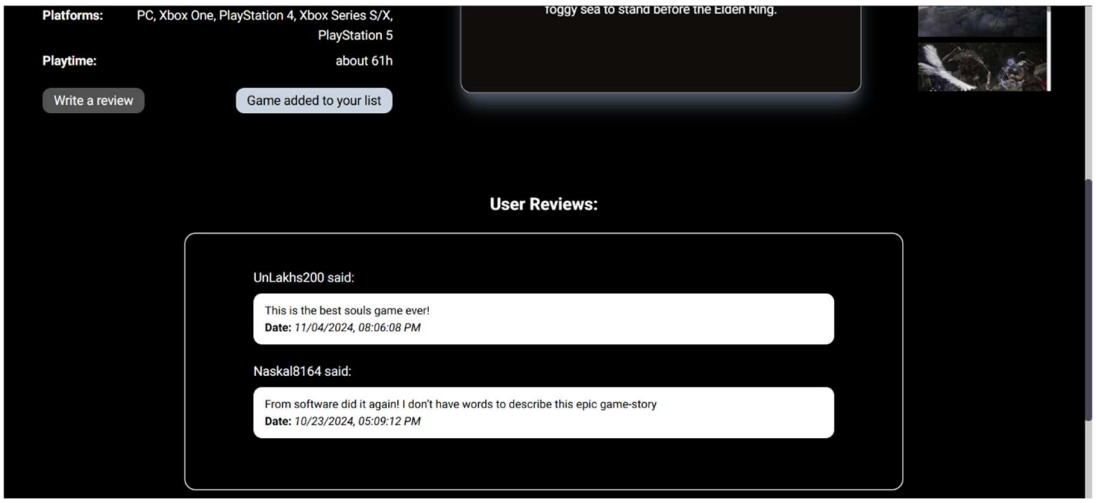
οποία μπορούμε να επιλέξουμε και να ανακατευθυνθούμε στη σελίδα του αντίστοιχου παιχνιδιού.

7.3.4 Σελίδα του κάθε παιχνιδιού



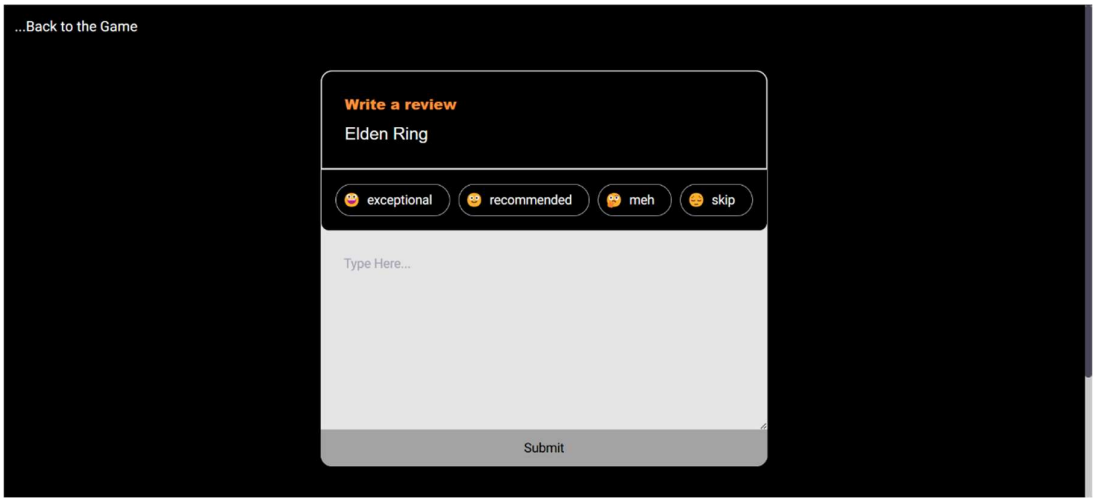
Εικόνα 33 . Σελίδα του κάθε game.

Εδώ βλέπουμε την σελίδα ενός επιλεγμένου παιχνιδιού. Στα αριστερά της σελίδας βλέπουμε αρχικά το κουμπί που μας επιστρέφει στην αρχική σελίδα των παιχνιδιών και κάτω από αυτό βλέπουμε τις πληροφορίες του παιχνιδιού καθώς επίσης και τα κουμπιά “Write a review” που μας ανακατευθύνει στη σελίδα που ο χρήστης γράφει την κριτική του για το επιλεγμένο παιχνίδι και το κουμπί του “add to list” που προσθέτει το παιχνίδι στην λίστα του χρήστη. Στη συνέχεια βλέπουμε δίπλα από τις λεπτομέρειες την περιγραφή του παιχνιδιού και δίπλα από αυτό μια λίστα από Screenshots του παιχνιδιού αυτού. Κάτω από όλα αυτά τα δεδομένα υπάρχει η περιοχή που μπορούμε να δούμε κριτικές από άλλους χρήστες.



Εικόνα 34 . User Reviews στην σελίδα του κάθε game.

7.3.5 Reviews



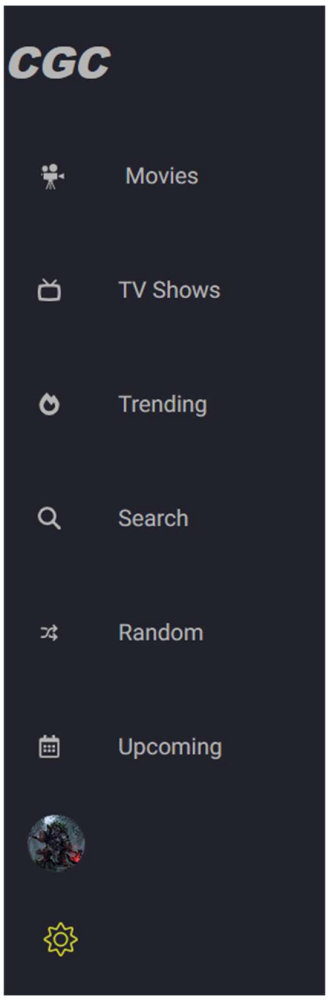
Εικόνα 35 . Σελίδα για καταχώρηση κριτικής στα Games.

Εδώ βλέπουμε την σελίδα που ανακατευθύνονται οι χρήστες για να αφήσουν μια κριτική. Αρχικά βλέπουμε ότι υπάρχει η επιλογή να επιστρέψουμε πίσω στην σελίδα με τις λεπτομέρειες του παιχνιδιού και στη συνέχεια την περιοχή που γράφουμε την κριτική μας. Ο χρήστης εδώ βλέπει το όνομα του παιχνιδιού που επέλεξε και του δίνεται η επιλογή να κάνει μια αντίδραση (exceptional, recommended, meh, skip). Τέλος γράφοντας την κριτική του μπορεί να κάνει submit και να ανακατευθυνθεί πίσω στη σελίδα των λεπτομερειών του παιχνιδιού.

7.4 Movies

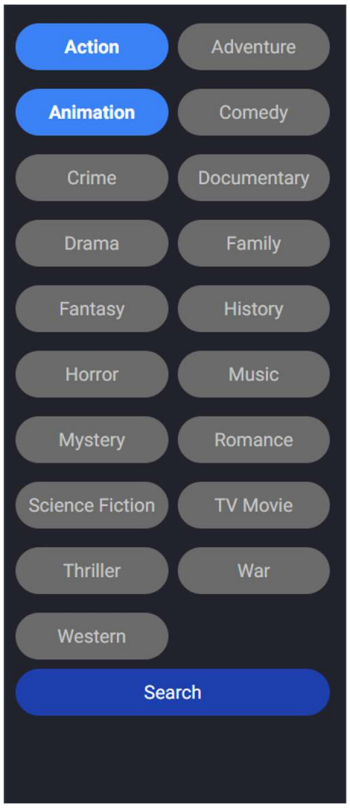
Στη συνέχεια θα δούμε το UI της κατηγορίας των ταινιών που βλέπει ένας συνδεδεμένος χρήστης. Θα αναφέρουμε μόνο το κομμάτι των ταινιών και όχι των σειρών μιας και η δομή τους είναι ίδια. Η δομή του trending είναι ίδια με την αρχική σελίδα των ταινιών οπότε δε θα αναφερθεί.

7.4.1 Components που βρίσκονται σε όλα τα κομμάτια των σελίδων



Εικόνα 36 . Αναπτυγμένο navbar στα Movies.

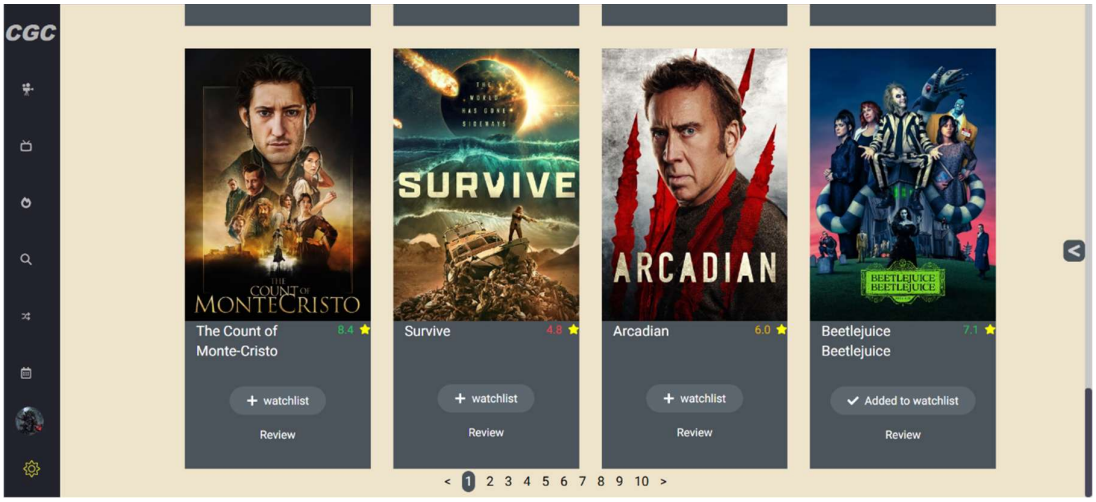
Εδώ παρατηρούμε την μπάρα πλοήγησης ενός συνδεδεμένου χρήστη όταν την κάνουμε hover. Στη μπάρα αυτή ο χρήστης έχει την δυνατότητα να πλοηγηθεί σε διάφορα σημεία της σελίδας. Το CGC μας ανακατευθύνει στην αρχική σελίδα της εφαρμογής, το Movies στην αρχική των Movies, το TV Shows στη σελίδα που βλέπουμε τις σειρές που υπάρχουν, Το Trending στις trending ταινίες, Το Search στην μπάρα αναζήτησης, το Random σε μία τυχαία ταινία, το upcoming σε ανερχόμενες ταινίες, η εικόνα του χρήστη (στη θέση της υπάρχει η επιλογή του Sign Up αν δεν είναι συνδεδεμένος ο χρήστης) που οδηγεί στις πληροφορίες του λογαριασμού του χρήστη, τέλος υπάρχει η δυνατότητα αλλαγής σε Dark Theme.



Εικόνα 37 . Φίλτρα στα Movies.

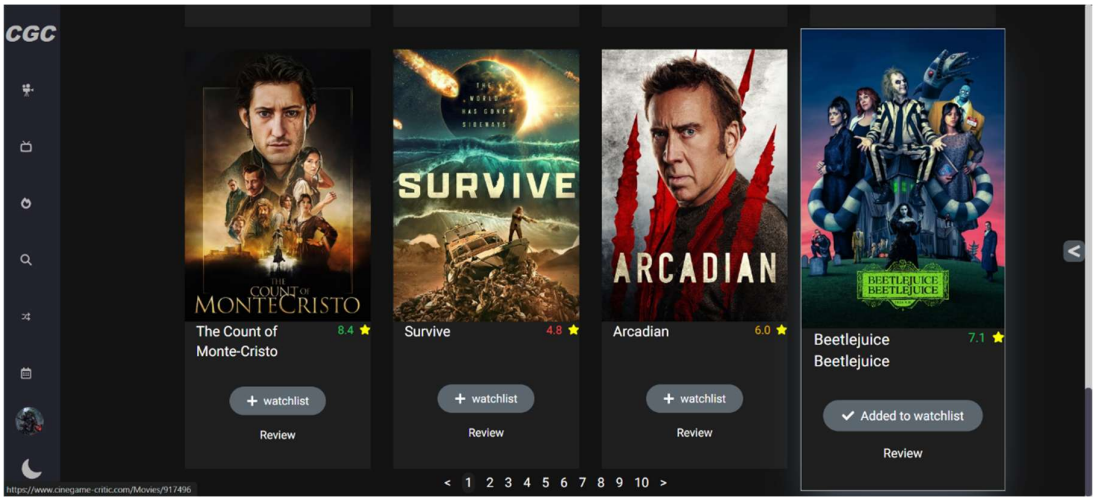
Εδώ παρατηρούμε τα διάφορα είδη ταινιών που υπάρχουν και φιλτράρουν τα αποτελέσματα των ταινιών. Μας δίνεται η δυνατότητα να επιλέξουμε περισσότερα από ένα είδος όπως φαίνεται και στην εικόνα. Κάνοντας κλικ στο Search ανακατευθυνόμαστε στην σελίδα με τα ανάλογα είδη ταινιών. Υπάρχει το ίδιο στυλ μενού στην κατηγορία των σειρών, αλλά με διαφορετικά ονόματα.

7.4.2 Αρχική σελίδα των Movies



Εικόνα 38 . Αρχική σελίδα των Movies.

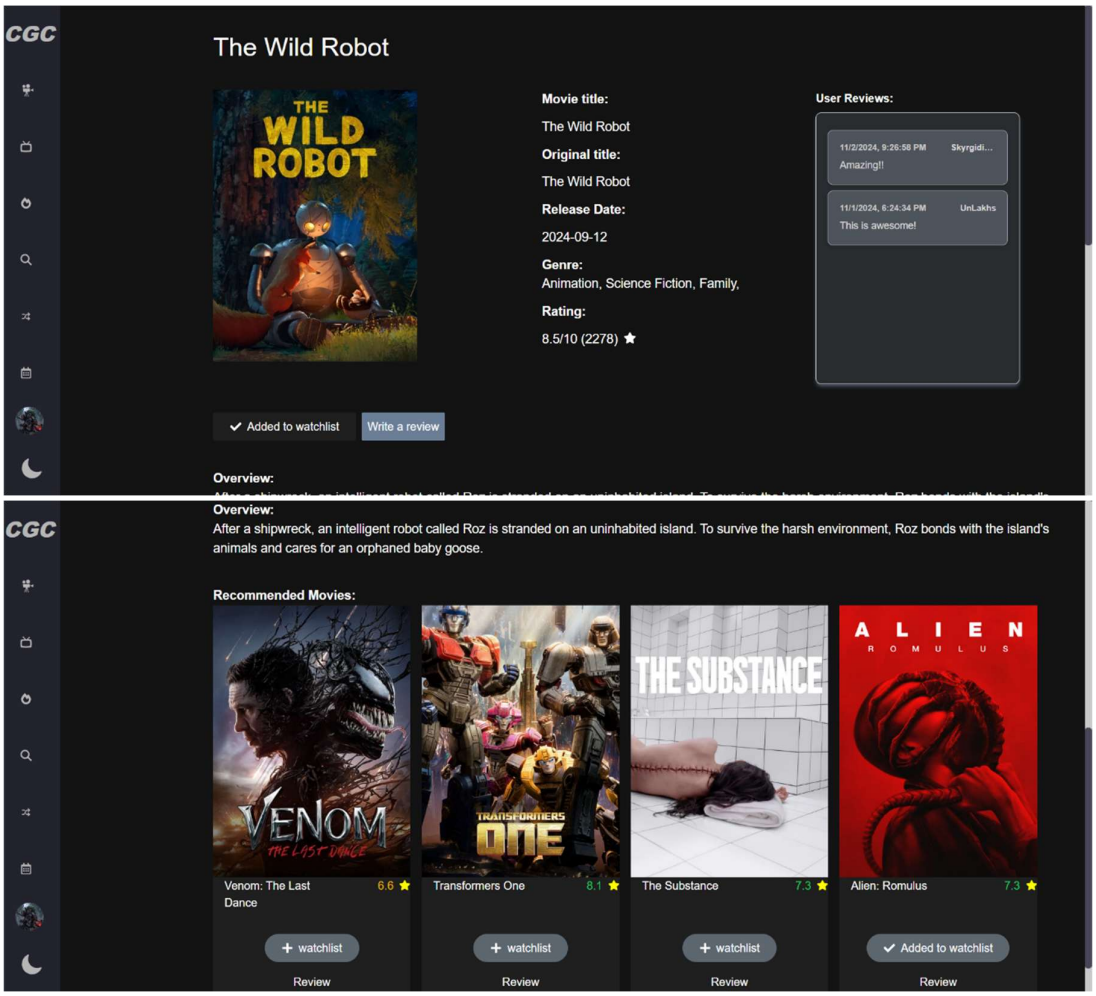
Στη σελίδα αυτή ο χρήστης μπορεί να δει μια πληθώρα ταινιών από όλες τις κατηγορίες. Στα αριστερά παρατηρούμε τη μπάρα πλοήγησης και δεξιά το κουμπί που αποκαλύπτει τα είδη των ταινιών. Ο χρήστης έχει την δυνατότητα να προσθέσει ταινίες στο watchlist του καθώς επίσης και να γράψει μια κριτική για την ταινία που επιθυμεί. Στην εικόνα αυτή βρισκόμαστε στο τέλος της σελίδας, όπου εδώ βλέπουμε και τον αριθμό των σελίδων, καθώς επίσης και τη σελίδα στην οποία βρισκόμαστε. Με το κλικ σε κάποιον άλλο αριθμό σελίδων το περιεχόμενο των ταινιών ανανεώνεται.



Εικόνα 39 . Αρχική σελίδα των movies σε dark theme.

Εδώ παρατηρούμε την ίδια σελίδα αλλά αυτή τη φορά σε dark theme. Βλέπουμε ότι στην μπάρα πλοήγησης η τελευταία εικόνα έχει αλλάξει και έχει επίσης προστεθεί ένα hover effect σκίασης σε κάθε Card.

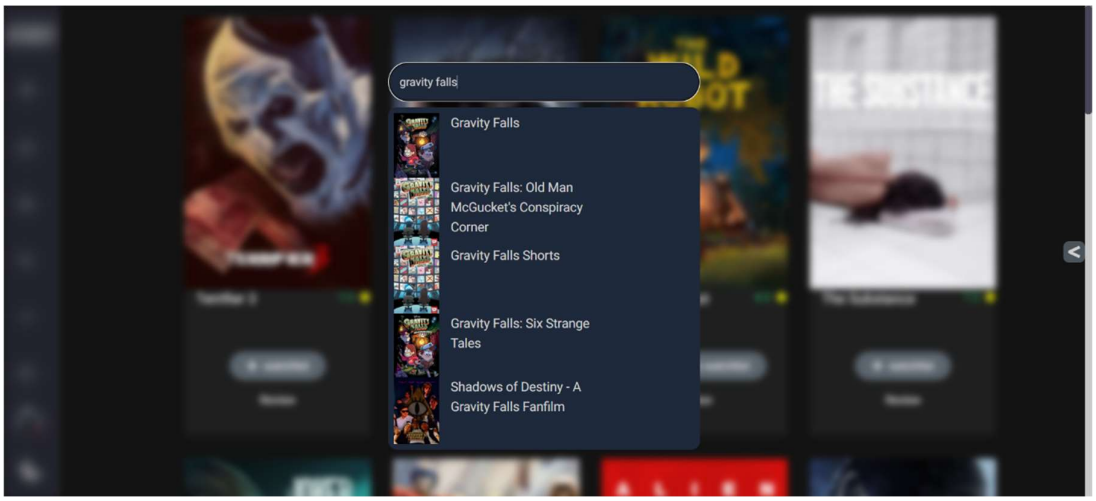
7.4.3 Σελίδα της κάθε ταινίας



Εικόνα 40 . Σελίδα της κάθε ταινίας σε dark theme των Movies.

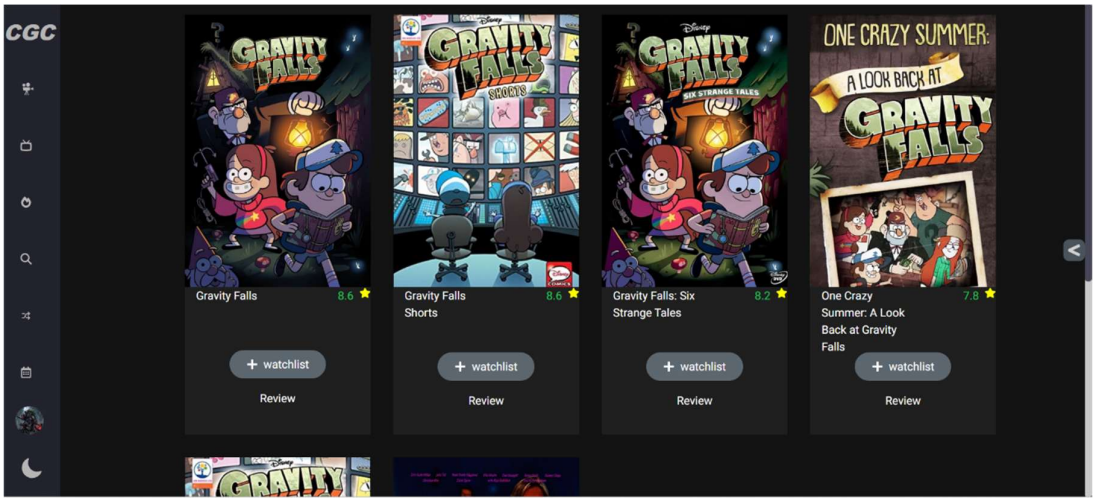
Στις εικόνες αυτές βλέπουμε το περιεχόμενο της ταινίας που επιλέξαμε να δούμε το περιεχόμενο της. Βλέπουμε το poster και τον τίτλο της ταινίας στο αριστερό μέρος, καθώς επίσης και τις λεπτομέρειες της ακριβώς από δίπλα. Στο δεξιά μέρος βλέπουμε όλα τα reviews που έχουν γίνει από εγγεγραμμένους χρήστες για τη συγκεκριμένη ταινία. Βλέπουμε επίσης και τις επιλογές Add to watchlist(ο χρήστης έχει ήδη κάνει add to watchlist οπότε είναι διαφορετικό το κείμενο του κουμπιού) και write a review που έχουν την ίδια λειτουργικότητα με τα αντίστοιχα κουμπιά στην αρχική σελίδα. Κάτω από αυτά, βρίσκεται η περιγραφή της ταινίας και πιο κάτω από αυτά βρίσκεται το κομμάτι των προτάσεων ταινιών που είναι κάπως παρόμοιες με την ταινία αυτή. Υπάρχει και ένα κουμπί που δε φαίνεται στην εικόνα, το οποίο προσθέτει 4 ταινίες ακόμα σε άθροισμα με τις υπόλοιπες προτεινόμενες ταινίες.

7.4.4 Μπάρα αναζήτησης



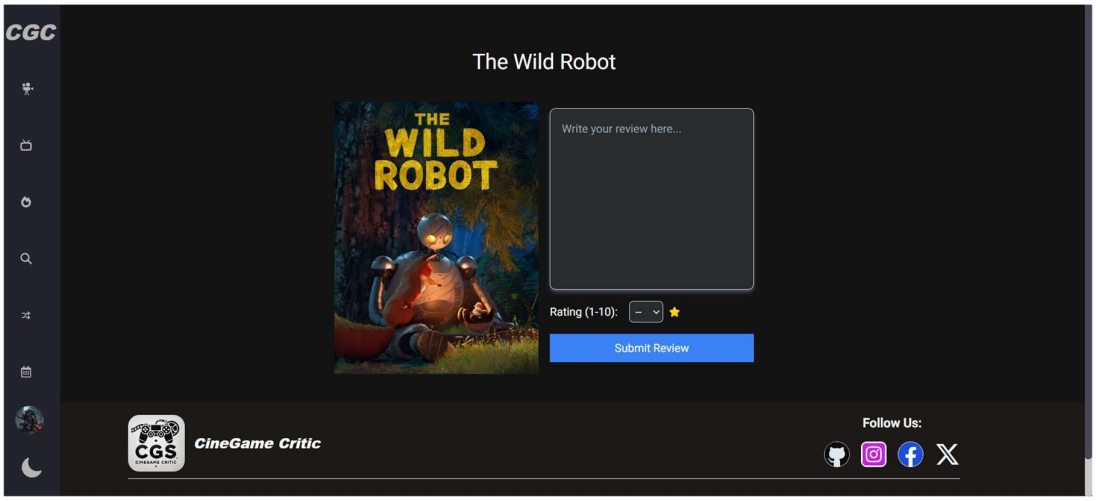
Εικόνα 41 . Searchbar των Movies.

Εδώ βλέπουμε την μπάρα αναζήτησης των Movies. Όταν γίνεται κλικ όλα τα υπόλοιπα κομμάτια της σελίδας γίνονται θολά και μπορούμε να γράψουμε τι θέλουμε να αναζητήσουμε. Στο παράδειγμα αυτό βλέπουμε ότι με τις λέξεις “gravity falls” εμφανίζονται 5 αποτελέσματα τα οποία μπορούμε να διαλέξουμε. Αν δε θέλουμε να επιλέξουμε ένα από αυτά μπορούμε να πατήσουμε το enter και θα μας ανακατευθύνει σε μια σελίδα με όσα αποτελέσματα μπορούν να βρεθούν με βάση του search query που θα του δώσουμε.



Εικόνα 42 . Σελίδα αποτελεσμάτων του search σε dark theme στα Movies.

7.4.5 Reviews



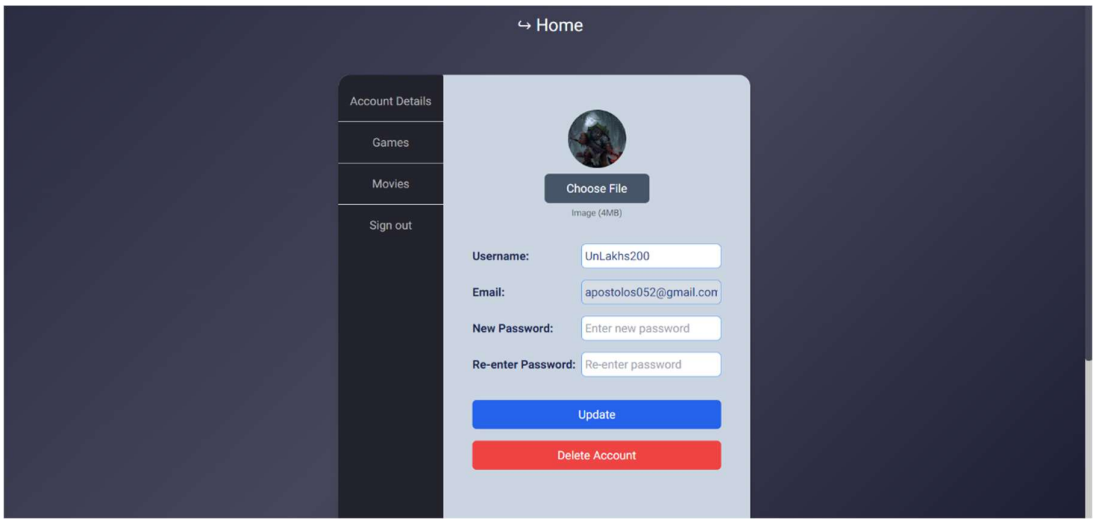
Εικόνα 43 . Σελίδα υποβολής κριτικής σε dark theme στα Movies.

Εδώ βλέπουμε την σελίδα που οδηγούμαστε με το κλικ στο κουμπί “write a review” στην ταινία “The Wild Robot”. Βλέπουμε ότι ο χρήστης έχει την δυνατότητα να γράψει ένα text από τις εντυπώσεις του για την ταινία και επίσης να την βαθμολογήσει κάνοντας κλικ στο dropdown που εμφανίζεται όταν γίνει κλικ. Όταν πατηθεί το “Submit Review”, η βαθμολογία που θα του δώσει ανεβαίνει στο TMDB και προστίθεται στον μέσο όρο της ταινίας (η βαθμολογία προστίθεται όταν κάνει update το TMDB). Επίσης το σχόλιο του χρήστη θα προστεθεί στο κομμάτι των σχολίων της ταινίας.

7.5 Account

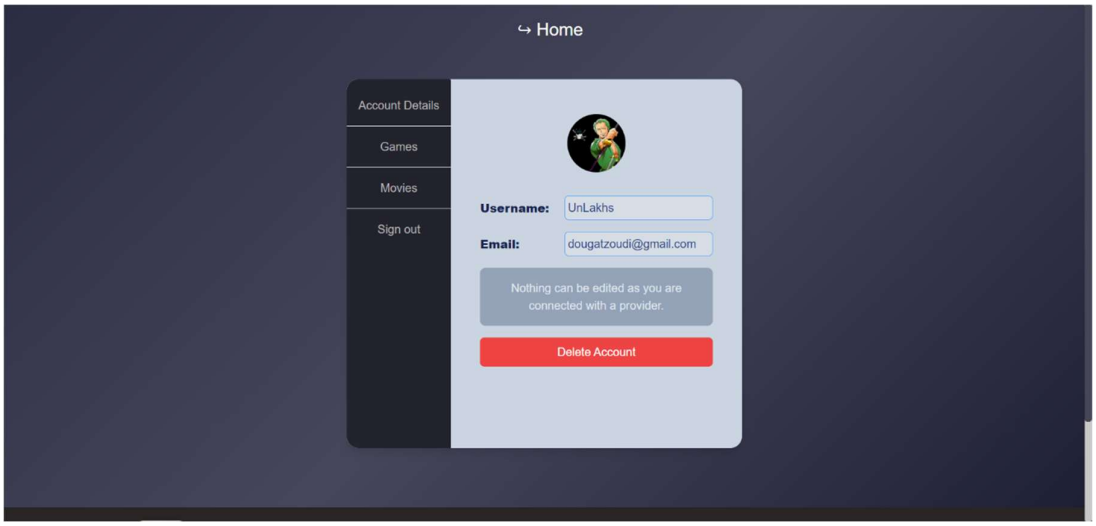
Το account είναι η σελίδα που ανακατευθύνεται ο χρήστης μέσα και από τις 2 κατηγορίες για να μπορεί κάποιος να δει πληροφορίες για τον λογαριασμό του, να δει το game list και watchlist του και επίσης να αποσυνδεθεί από τον τρέχων λογαριασμό.

7.5.1 Account Details



Εικόνα 44 . Σελίδα των λεπτομερειών λογαριασμού ενός χρήστη που έχει συνδεθεί με credentials.

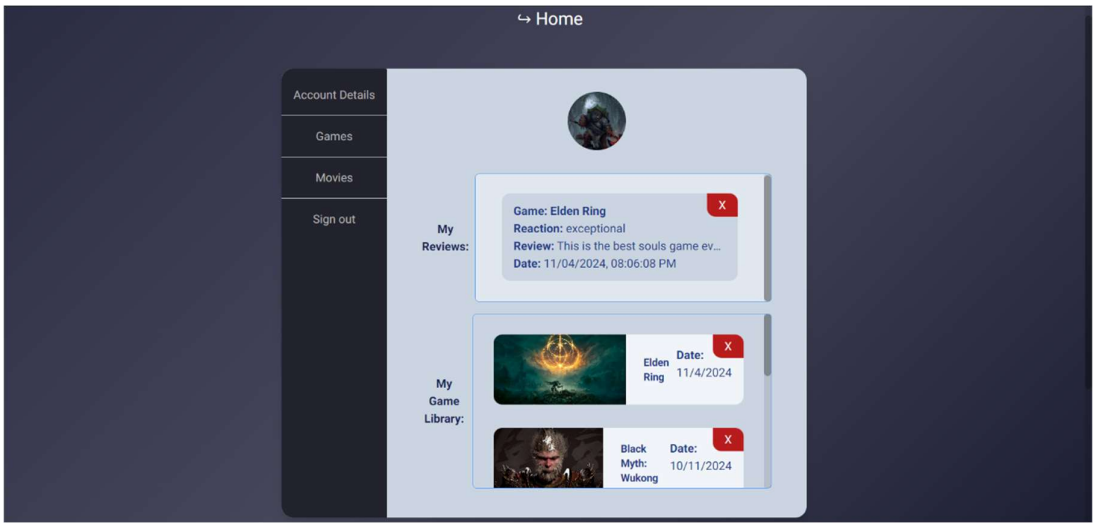
Εδώ βλέπουμε την σελίδα που βλέπει ένας χρήστης που έχει συνδεθεί με credentials. Η ανακατεύθυνση από οποιαδήποτε σελίδα μας οδηγεί σε αυτό το κομμάτι που αφορά τις πληροφορίες λογαριασμού του χρήστη. Εδώ ένας χρήστης μπορεί να αλλάξει το username του και τον κωδικό του και να δει το email που έχει χρησιμοποιήσει για την εγγραφή του στη σελίδα. Υπάρχει επίσης η δυνατότητα να αλλάξει την εικόνα προφίλ του, ανεβάζοντας μια εικόνα. Με το κουμπί “Update“γίνεται ενημέρωση των καινούργιων στοιχείων, εφόσον δοθούν. Με το κουμπί “Delete Account” ο χρήστης μπορεί να διαγράψει τον λογαριασμό του αν το επιθυμεί.



Εικόνα 45 . Σελίδα των λεπτομερειών λογαριασμού ενός χρήστη που έχει συνδεθεί με Provider.

Εδώ βλέπουμε την ίδια σελίδα, αλλά αυτή τη φορά με κάποιον χρήστη που έχει συνδεθεί μέσω provider (Google ή GitHub). Παρατηρούμε ότι ο χρήστης μπορεί μόνο να διαγράψει τον λογαριασμό του και κανένα άλλο στοιχείο δεν μπορεί να επεξεργαστεί.

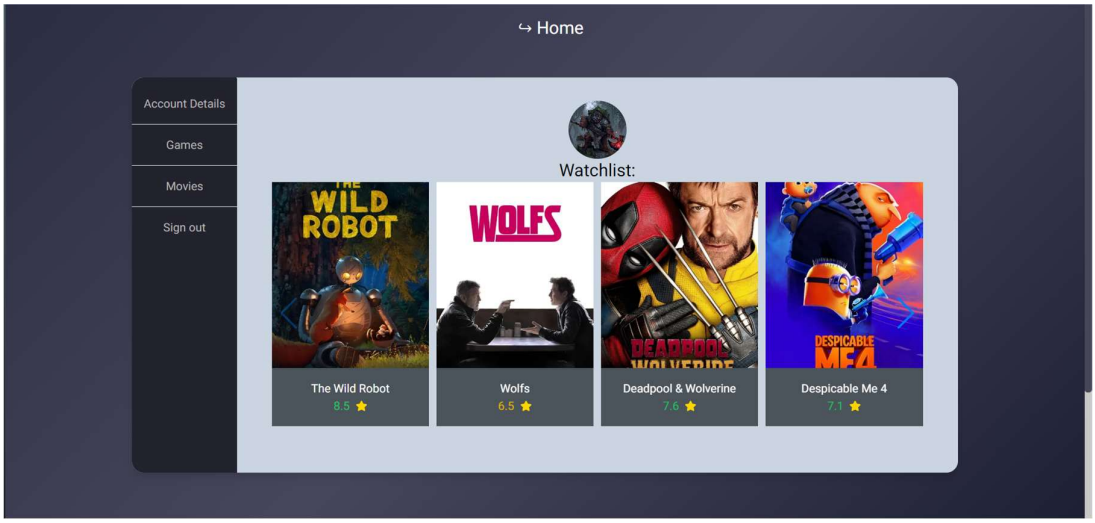
7.5.2 Games



Εικόνα 46 . Σελίδα προβολής game list ενός χρήστη.

Εδώ βλέπουμε την κατηγορία “Games” μέσα στο προφίλ του χρήστη. Σε αυτό το σημείο υπάρχουν όλα τα Reviews που έχει κάνει ο χρήστης και επίσης όλα τα παιχνίδια που έχει προσθέσει τη λίστα του.

7.5.3 Movies



Εικόνα 47 . Σελίδα προβολής watchlist ενός χρήστη.

Εδώ βλέπουμε την κατηγορία Movies μέσα στο προφίλ του χρήστη. Εδώ βρίσκεται ένα καρουζέλ με όλες τις ταινίες που έχει προσθέσει ο χρήστης στον λογαριασμό του. Ο χρήστης μπορεί να προχωρήσει το καρουζέλ, κάνοντας κλικ στα βέλη ή με scroll.

Συμπεράσματα

Κατά την ανάπτυξη της εφαρμογής, συναντήσαμε τόσο προκλήσεις όσο και ευκαιρίες να αναπτύξουμε τις τεχνικές μας γνώσεις. Χρησιμοποιώντας σύγχρονες τεχνολογίες και θέτοντας ως βάση τη Next.js, καταφέραμε να δημιουργήσουμε ένα αποδοτικό προϊόν πάνω σε ένα αρκετά δημοφιλές θέμα της εποχής, που έχει να κάνει με μέσα ψυχαγωγίας.

Από τεχνική άποψη καταλήξαμε στο συμπέρασμα ότι για μια εφαρμογή που χρειάζεται να έχει πολλές σελίδες η Next.js ήταν μια πολύ καλή λύση, καθώς μας προσφέρει server-side rendering και δυναμικό routing που ενισχύει πολύ την απόδοση της εφαρμογής. Σε συνδυασμό με την typescript, η συγγραφή του κώδικα έγινε με περισσότερη οργάνωση, καθώς ήταν πολύ πιο εύκολη η εύρεση σφαλμάτων μέσα στα αρχεία.

Η πλατφόρμα παρέχει τη δυνατότητα στους εγγεγραμμένους χρήστες να συγγράψουν τη δική τους κριτική, κάτι που ενισχύει το κλίμα της κοινότητας, στην οποία ενθαρρύνεται η έκφραση διαφορετικών απόψεων και η ανταλλαγή γνώσεων. Ακόμη, οι χρήστες μπορούν να αποθηκεύσουν την ταινία/παιχνίδι της επιλογής τους σε μια λίστα, που ανά πάσα στιγμή μπορούν να ανατρέξουν στο προφίλ τους, γεγονός που μπορεί να συμβάλλει στη μείωση του χρόνου περιήγησης μέσα στη σελίδα.

<u>Μελλοντικές επεκτάσεις:</u>

Η πρώτη επέκταση που θα θέλαμε να υλοποιήσουμε είναι να διαφημίσουμε την σελίδα μας, έτσι ώστε να έχουμε αλληλεπίδραση με περισσότερους χρήστες και να παρέχουμε μια πιο συνολική εικόνα, τόσο για τη σελίδα μας όσο και για τα παιχνίδια/ταινίες που προβάλλουμε. Στη συμβολή του σκοπού αυτού, στα αρχικά μας πλάνα είναι η δημιουργία ενός forum, όπου ο χρήστης θα μπορεί να ανακατευθύνεται μέσα από τη σελίδα και να βαθμολογεί τη σελίδα μας.

Μετέπειτα θα θέλαμε να βελτιώσουμε τη ταχύτητα rendering στα Games, καθώς τα παιχνίδια φορτώνονται από το Mongo Database και τείνουν να φορτώνουν με μια καθυστέρηση. Για αυτόν τον λόγο, προσωρινά περιορίζονται στα 160.

Στο κομμάτι της απόδοσης, θα θέλαμε να εφαρμόσουμε έναν καλύτερο τρόπο να δείχνουμε τα user reviews μιας και αυτή τη στιγμή κάθε review εμφανίζεται από το collection των users.

Θα θέλαμε επίσης να εφαρμόσουμε κάποια μέθοδο έτσι ώστε να ελέγχεται το λεξιλόγιο στις κριτικές και να αποφεύγεται βωμολοχία.

Στο κομμάτι των ταινιών θα θέλαμε να προσθέσουμε περισσότερες επιλογές φιλτραρίσματος (όπως το order by των games) και επίσης να εξελίξουμε το Watchlist του κάθε χρήστη έτσι ώστε να μπορεί να χωρίζεται σε “completed movies” και “to-watch movies”.

Συνολικά, Η ανάπτυξη της εφαρμογής αυτής αποτέλεσε μια πολύτιμη εμπειρία εκμάθησης σύγχρονων τεχνολογιών που χρησιμοποιούνται στην αγορά εργασίας και μας βοήθησε σημαντικά στο να εξελίξουμε τις γνώσεις μας στον προγραμματισμό και στην επίλυση προβλημάτων.

Βιβλιογραφία

[1]	React official documentation 2024, "*Built-in React Hooks*", accessed 18 October 2024, <https://react.dev/reference/react/hooks>
[2]	Legacy React official documentation 2024, "*Virtual DOM and internals*", accessed 18 October 2024, <https://legacy.reactjs.org/docs/faq-internals.html>
[3]	Next.js, "*An Introduction to Next.js*", accessed on 19 October 2024, <https://nextjs.org/docs>
[4]	Typescript official documentation 2024, "*TypeScript for the new programmer*", accessed 20 October 2024, <https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html>
[5]	Geeksforgeeks 2024*, "Introduction To Tailwind CSS", accessed 20 October 2024,* <https://www.geeksforgeeks.org/introduction-to-tailwind-css/>
[6]	Developer Mozilla official documentation 2024, "*CSS: Cascading Style Sheets*", accessed 20 October 2024, <https://developer.mozilla.org/en-US/docs/Web/CSS>
[7]	RAWG API official documentation 2024, "*RAWG video Games Database API (v1.0)*", accessed 21 October 2024, <https://api.rawg.io/docs/>
[8]	Node.js official documentation 2024, "*Introduction to Node.js*", accessed 22 October 2024, <https://nodejs.org/en/learn/getting-started/introduction-to-nodejs>
[9]	Geeksforgeeks 15 October 2024, "*Node.js Introduction*", accessed 22 October 2024, <https://www.geeksforgeeks.org/node-js-introduction/>
[10]	NPM, *"About NPM",* accessed 23 October 2024*,* <https://docs.npmjs.com/about-npm/>
[11]	Kinsta.com 2024, *What Is npm? An Introduction to Node's Package Manager,* accessed 23 October 2024, <https://kinsta.com/knowledgebase/what-is-npm/>
[12]	Next.js, "Route Handlers", accessed 24 October 2024, <https://nextjs.org/docs/app/building-your-application/routing/route-handlers>
[13]	Next-Auth, *"Introduction About NextAuth.js",* accessed 24 October 2024, <https://next-auth.js.org/getting-started/introduction>

[14]	NathanKr 4 September 2024, "*next.js-nextauth-github-playground*", accessed 24 October 2024, <https://github.com/NathanKr/next.js-nextauth-github-playground>
[15]	MongoDB, "*What is MongoDB Atlas?*", accessed 25 October 2024, <https://www.mongodb.com/docs/atlas/>
[16]	Next.js, *"Chapter 10 Server and Client Components"*, accessed 25 October 2024, <https://nextjs.org/learn/react-foundations/server-and-client-components>
[17]	Medium 11 February 2024, "*Demystifying Vercel: How it works and How to create Vercel*", accessed 27 October 2024, <https://medium.com/@mayank2803sharma/demystifying-vercel-how-it-works-and-how-to-create-vercel-763586070478>
[18]	Geeksforgeeks 2024, "*introduction to github*", accessed 30 October 2024, <https://www.geeksforgeeks.org/introduction-to-github/>
[19]	John Deliyiannis 2015, "*Εντολές για τη χρήση του git",* accessed 30 October 2024, <https://johndel.gr/blog/git-basics-reference>
[20]	Google support 2024, "*About Search Console*", accessed 31 October 2024, <https://support.google.com/webmasters/answer/9128668?hl=en>
[21]	Carolyn Holzman, Boris B., James Frazier 2024, *"How do you leverage sitemaps and robots.txt files for crawling and indexing?"* accessed 3 November 2024, <https://www.linkedin.com/advice/3/how-do-you-leverage-sitemaps-robotstxt-files-crawling>

Παράρτημα Κώδικα

Ο πηγαίος κώδικας είναι ανεβασμένος στο GitHub μας, στη σελίδα:
<https://github.com/WebDevTeam2/CGC>