When Can I PLAY It

Applicazioni e Servizi Web

Konrad Gomulka - 0900065698 {konrad.gomulka@unibo.it}

8 Ottobre 2020

Capitolo 1

Introduzione

Il mercato videoludico è parte integrante del settore dell'entertainment, con nuovi titoli che vengono rilasciati quotidianamente.
Questo progetto ha come obiettivo il fornire un'applicazione in grado di visualizzare tutte le uscite recenti, nonché di tenere traccia mediante una funzione di ricerca o tramite notifiche via mail, di quelle future.

Capitolo 2

Requisiti

L'applicazione permette a un qualsiasi utente di avere accesso alle seguenti funzionalità:

- Visualizzazione dei videogiochi, con relativa piattaforma e zona, rilasciati a una settimana dalla data corrente
- Visualizzazione dei videogiochi, con relativa piattaforma e zona, prossimi all'uscita
- Visualizzazione di un elenco con tutti i videogiochi e relative date di uscita
- Possibilità di filtrare i videogiochi per data di rilascio, genere e piattaforma
- Possibilità di ordinare la lista dei videogiochi in base a criteri vari (ordine alfabetico, ecc..)

Vi sono inoltre alcune funzionalità riservate ad utenti che intendono effettuare/hanno effettuato l'accesso:

- Possibilità di effettuare il login mediante account Google
- Possibilità di seguire una data di rilascio
- Possibilità di modificare le impostazioni di notifica a proprio piacimento

Capitolo 3

Design

Il sistema adotta un'architettura REST separando la logica del frontend, la quale reperisce le informazioni tramite richieste HTTP, da quella di backend, che mette a disposizione delle API per ricevere o modificare dati.

La parte di backend è inoltre suddivisa nei seguenti servizi:

- APIService: permette all'utente di effettuare richieste in modo da ottenere dati e/o modificare il proprio profilo
- IGDBService: si occupa di mantenere il database sempre aggiornato, reperendo i dati dal servizio pubblico IGDB (Internet Game DataBase) [1]
- MailService: si occupa di inviare mail riguardanti l'imminente rilascio di un videogioco agli utenti

3.1 Interfaccia

In fase di design sono stati effettuati dei mockup utilizzando il software Balsamiq Wireframes 4.0.47.

3.1.1 Home

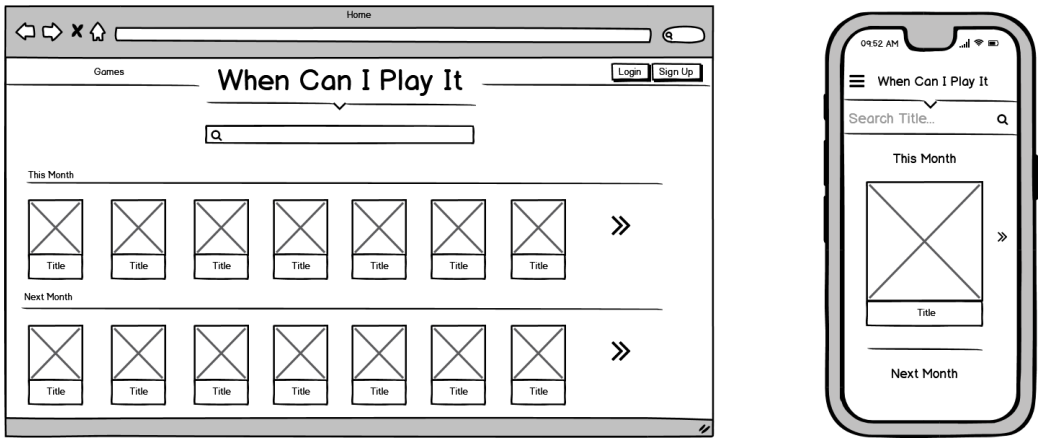

Figura 3.1: Home page in versione desktop e mobile

3.1.2 Games

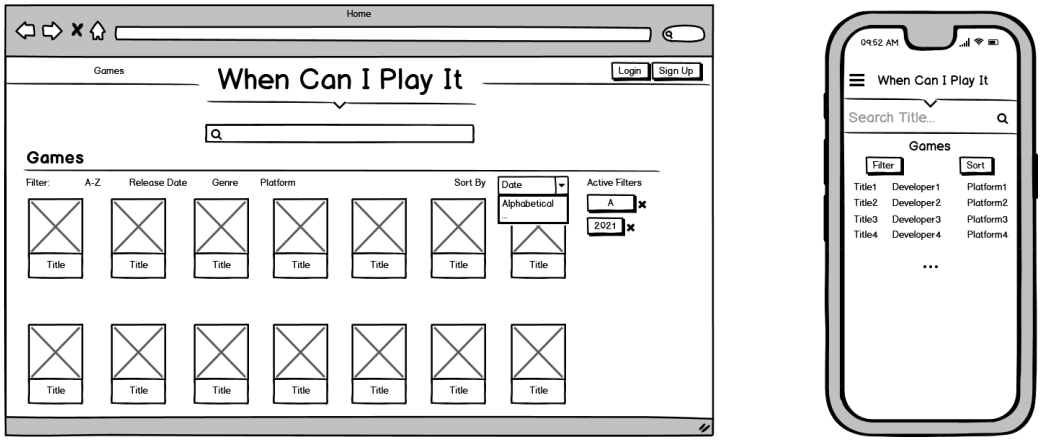

Figura 3.2: Pagina Games in versione desktop e mobile

3.1.3 Game

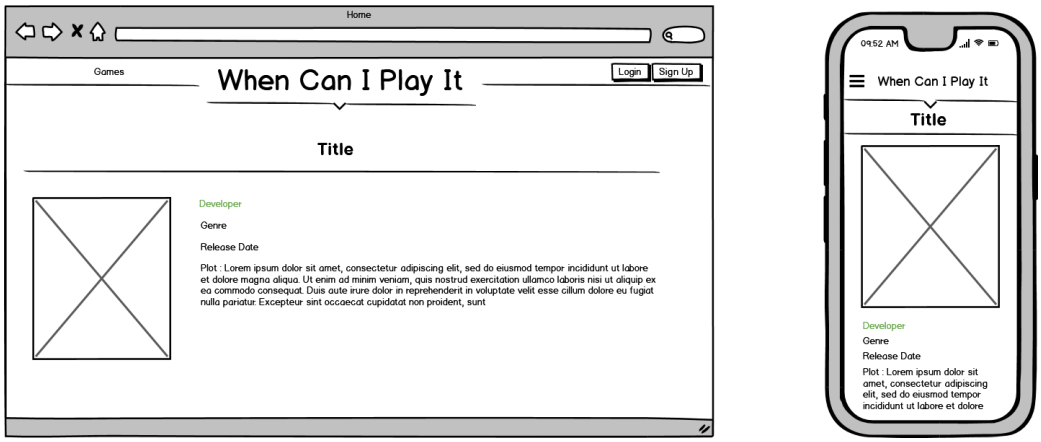

Figura 3.3: Pagina di info videogioco in versione desktop e mobile

3.1.4 User Options

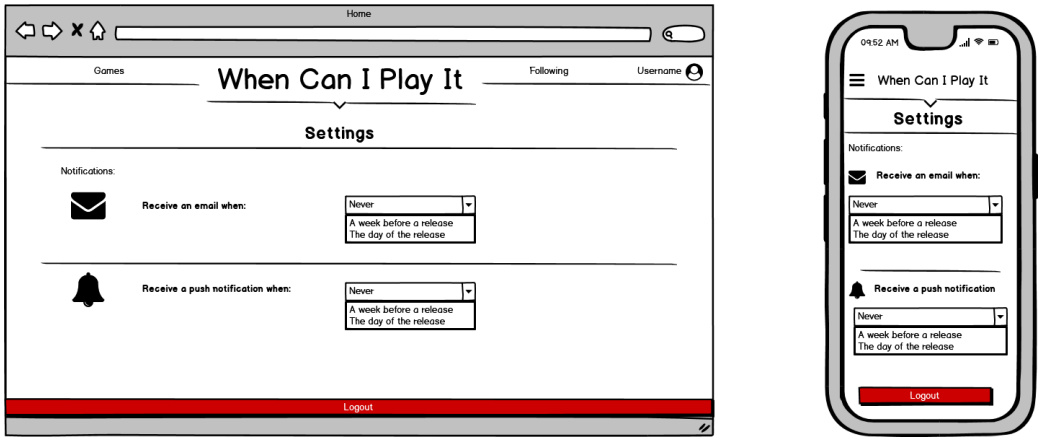

Figura 3.4: Pagina di impostazioni in versione desktop e mobile

3.2 Route

Il servizio APIService offre numerose API mediante le quali l'utente può interfacciarsi col database.

3.2.1 User Routes

Rappresentano le routes accessibili da chiunque visiti il sito.

/api/releases

GET Permette di ottenere le date di rilascio dei videogiochi.

Parametri:

- from: Timestamp Unix che rappresenta il limite inferiore per la data di rilascio
- to: Timestamp Unix che rappresenta il limite superiore per la data di rilascio
- recentlyAdded: Boolean per ottenere le date aggiunte al database negli ultimi sette giorni
- category: Valore numerico da 0 a 7 che indica il formato della data di rilascio secondo i criteri rappresentati in tabella 3.1.

0	YYYYMMMMDD
1	YYYYMMMM
2	YYYY
3	YYYYQ1
4	YYYYQ2
5	YYYYQ3
6	YYYYQ4
7	TBD

Tabella 3.1: Formati date di rilascio

- platform: Array di stringhe che rappresentano le piattaforme desiderate
- region: Stringa per specificare il continente di uscita desiderato
- sort: Stringa per ordinare in base al campo indicato
- limit: Valore numerico per indicare il numero massimo di risultati desiderati

/api/games

GET Permette di ottenere tutti i dati riguardanti i videogiochi.

Parametri:

- options: Parametro necessario per il plugin mongoose-paginate-v2 descritto in seguito [**?**]
- released: Stringa per indicare se il gioco è rilasciato, non rilasciato o con data ancora da annunciare ("true" per giochi rilasciati, "false" ancora da rilasciare e "TBA" per giochi senza date)
- followed: Se impostato indica l'id dell'utente, in base al quale vengono filtrati soltanto i giochi seguiti
- minrating: Valore numerico che rappresenta il limite inferiore per il voto del pubblico (in %)
- maxrating: Valore numerico che rappresenta il limite superiore per il voto del pubblico (in %)
- category: Stringa che rappresenta la tipologia del videogioco ("main_game", "dlc_addon", "expansion", "bundle", "standalone_expansion", "mod", "episode" o "season")
- genres: Array di stringhe rappresentano i generi videoludici desiderati
- name: Stringa in basa alla quale filtrare i titoli dei videogiochi (mediante una funzione regex)
- platforms: Array di stringhe che rappresentano le piattaforme desiderate
- mintobeat: Valore numerico che rappresenta il limite inferiore per il tempo medio di completamento del gioco (espresso in minuti)
- mintobeat: Valore numerico che rappresenta il limite superiore per il tempo medio di completamento del gioco (espresso in minuti)

/api/game/:id

GET Permette di ottenere tutti i dati riguardanti un videogioco in particolare.

Parametri:

- id: ObjectID che rappresenta il gioco desiderato

/api/platforms

GET Permette di ottenere i nomi di una serie di piattaforme in base ai loro ObjectID.

Parametri:

- platforms: Array di ObjectID che rappresentano le piattaforme desiderate

/api/genres

GET Permette di ottenere i nomi di una serie di generi in base ai loro ObjectID.

Parametri:

- genres: Array di ObjectID che rappresentano i generi desiderati

3.2.2 Auth Routes

Rappresentano le routes utilizzate per effettuare il login sull'applicazione.

/api/auth/google

GET Permette di effettuare la richiesta di login tramite account Google.

/api/auth/google/callback

GET Route utilizzata dai servizi Google per confermare il login. Redireziona alla Home page impostando un parametro per confermare o annullare la richiesta di login.

/api/auth/logout

GET Permette di effettuare il logout dall'applicazione.

/api/auth/user

GET Permette di ottenere tutte le informazioni riguardanti un utente. L'ObjectID richiesto deve coincidere con l'ObjectID contenuto nell'header della richiesta.

3.2.3 Follow Routes

Routes utilizzate da un utente registrato per seguire le date di rilascio.

/api/follow/date

POST Permette di inserire una data nell'array di date seguite dall'utente.

Parametri:

- date: ObjectID rappresentante la data da seguire

PUT Permette di eliminare una data dall'array di date seguite dall'utente.

Parametri:

- date: ObjectID rappresentante la data da eliminare

3.2.4 Notification Routes

Routes utilizzate da un utente registrato per gestire le proprie impostazioni di notifica.

/api/notifications/mail

PUT Permette di cambiare le impostazioni di notifica mail dall'utente.

Parametri:

- value: Boolean rappresentante la preferenza dell'utente nella ricezione di notifiche mail

Capitolo 4

Tecnologie

L'applicazione è stata sviluppata seguendo lo stack MEVN per via della sua flessibilità nonché semplicità nell'utilizzo.

4.0.1 Frontend

Il frontend è stato sviluppando utilizzando Vue, sfruttando la sua ampia libreria di packages. Per la stilizzazione delle pagine è stato utilizzato il precompilatore SASS, in grado di aggiungere varie funzionalità utili alla realizzazione del CSS. Una lista completa dei packages utilizzati è riportata in seguito.

- @fortawesome/fontawesome-free: package contenente vari stili di icone svg
- axios: utilizzato per la comunicazione col backend mediante richieste HTTP
- bootstrap: package contentente vari css compilati per la stilizzazione delle pagine
- moment: utilizzato per la gestione delle date
- popper.js: dipendenza utilizzata da vue-multiselect
- sass: utilizzato per la compilazione dei file scss
- sass-loader: utilizzato per la compilazione dei file scss
- vue: utilizzato per la realizzazione del frontend
- vue-burger-menu: utilizzato per l'implementazione di una sidebar
- vue-carousel: utilizzato per l'implementazione di vari carousel contenenti slide
- vue-infinite-loading: utilizzato per la realizzazione di liste infinite in grado di ampliarsi dinamicamente

- vue-js-modal: utilizzato per la stilizzazione dei modal
- vue-js-toggle-button: utilizzato per la stilizzazione dei bottoni
- vue-meta: utilizzato per gestire le varie informazioni META delle pagine
- vue-multiselect: utilizzato per la realizzazione di combobox a scelta singola o multipla
- vue-router: utilizzato per la gestione del routing tra pagine
- vuex: package utilizzato per la gestione di stati condivisi dell'applicazione

4.0.2 Backend

Il backend è stato suddiviso come già presentato nel Capitolo 3.

APIService

Il servizio APIService gestisce tutte le interazioni tra frontend e database, locato sulla piattaforma online gratuita messa a disposizione da MongoDB. Tale servizio include anche tutte le funzioni necessarie per autenticare un utente sulla piattaforma.

Una lista completa dei packages utilizzati è riportata in seguito.

- axios: utilizzato per la comunicazione col backend mediante richieste HTTP
- body-parser: utilizzato per ricavare parametri o dati dal body di una funzione
- cors: package per la gestione dei parametri di Cross-Origin Resource Sharing
- dotenv: utilizzato per la gestione delle variabili globali
- connect-history-api-fallback: utilizzato per esporre l'applicazione mediante express e per non perdere lo stato dell'applicativo in seguito a refresh
- express: utilizzato per la realizzazione del servizio
- express-session: utilizzato per la gestione delle sessioni degli utenti
- moment: utilizzato per la gestione delle date
- mongoose: utilizzato per l'interazione col database MongoDB
- mongoose-paginate-v2: utilizzato per la gestione della paginazione nelle richieste al database
- passport: utilizzato per implementare i servizi di login

- passport-google-oauth20: estensione di passport per effettuare il login mediante account Google
- path: utilizzato per la gestione delle path locali

IGDBService

Il servizio gestisce il reperimento dei dati dal database IGDB. Mediante uno scheduler, ogni giorno a mezzanotte viene inviata una richiesta per reperire tutti i giochi e le date non presenti nel database. Tale servizio si occupa anche del salvataggio dei dati nel database, evitando uno sovraccarico di APIService nel caso di moli elevate di dati.
Vi sono inoltre state complicazioni dovute alla mancanza della configurazione degli header CORS da parte di IGDB, ciò ha richiesto la necessità di configurazione di un proxy per il reperimento dei dati.

Una lista completa dei packages utilizzati è riportata in seguito.

- axios: utilizzato per la comunicazione col backend mediante richieste HTTP
- cors-anywhere: utilizzato per la configurazione del proxy con Cross-Origin Resource Sharing
- dotenv: utilizzato per la gestione delle variabili globali
- express: utilizzato per la realizzazione del servizio
- fs-path: utilizzato per la memorizzazione delle immagini reperite da IGDB
- moment: utilizzato per la gestione delle date
- mongoose: utilizzato per l'interazione col database MongoDB
- node-cron: utilizzato per la realizzazione di task scheduled
- node-fetch: utilizzato per il reperimento delle immagini da IGDB
- path: utilizzato per la gestione delle path locali
- puppeteer: utilizzato per utilizzare in modalità headless un'istanza di Chromium, permettendo di reperire grandi quantità di dati in maniera efficiente
- querystring: utilizzato per effettuare encoding e parsing di query

MailService

Il servizio gestisce l'invio delle mail agli utenti registrati. Mediante uno scheduler, ogni giorno a mezzanotte viene mandata una mail agli utenti che seguono un gioco rilasciato in tale giornata.

Una lista completa dei packages utilizzati è riportata in seguito.

- axios: utilizzato per la comunicazione col backend mediante richieste HTTP
- dotenv: utilizzato per la gestione delle variabili globali
- moment: utilizzato per la gestione delle date
- mongoose: utilizzato per il reperimento delle informazioni utente dal database MongoDB
- node-cron: utilizzato per la realizzazione di task scheduled
- nodemailer: utilizzato per l'invio delle mail agli utenti
- nodemailer-express-handlebar: utilizzato per la stilizzazione delle mail inviate mediante nodemailer

Capitolo 5

Codice

5.1 Frontend

5.1.1 Componente Jumbotron

Alcune pagine del sito implementano il componente Jumbotron contenente una barra di ricerca e il logo testuale del sito.
Dato che il componente non è condiviso tra tutte le pagine del sito ma possiede delle proprietà condivise tra le pagine che lo implementano (il contenuto testuale digitato dall'utente), è stato realizzato un sistema per permettere all'utente di passare dalla pagina Home alla pagina Games mantenendo il contenuto digitato e permettendo di continuare a scrivere filtrando la lista dei giochi come segue.

Ogni input emette un evento text-changed:

```
<input ref="search" v-model="text" @input="$emit('text-changed', text);">
```

Se l'evento viene rilevato e l'utente si trova nella pagina Home, verrà reindirizzato alla pagina Games passando come prop il testo contenuto nella barra di ricerca cosicché essa possa reinizializzare il jumbotron con lo stesso valore:

```
mounted() {
    if(this.initialText != null) {
        this.text = this.initialText;
            this.$refs.search.focus();
    }
}
```

5.1.2 Vuex

Vuex è il package utilizzato da tutte le pagine del sito che tentano di accedere a stati condivisi come i dati reperiti dal backend o le informazioni sull'utente autenticato. In totale sono stati utilizzati quatto moduli:

- dates: permette di ottenere tutte le informazioni sulle date di uscita dei giochi
- games: permette di ottenere tutte le informazioni sui giochi
- game: permette di ottenere le informazioni riguardanti un singolo gioco
- user: gestisce tutte le interazioni con il backend che un utente autenticato può effettuare

Un esempio di modulo è rappresentato in seguito.

```
import axios from 'axios'
import moment from 'moment'

const state = {
    recentReleases: [],
    upcomingReleases: [],
    loadingStatus: false
};

const getters = {
    recentReleases: state => state.recentReleases,
    upcomingReleases: state => state.upcomingReleases,
    loadingStatus: state => state.loadingStatus
};

const actions = {
    async getRecentDates({ commit }) {
        commit('loadingStatus', true);

        let to = moment().add(28, 'days').unix();
        let from = moment().subtract(7, 'days').unix();
        let response = await axios.get("http://localhost:3030/api/releases", {
            params: {
                "from": from,
                "to": to,
                "sort": "date",
                "category": 0
            }
        });

        commit('setUpcomingReleases',
                response.data.filter(el => el.date > moment().unix()));
        commit('setRecentReleases',
                response.data.filter(el =>
                el.date <= moment().unix()).sort((a, b) => b.date - a.date));
```

```
        commit('loadingStatus', false);
    }
};

const mutations = {
    setRecentReleases: (state, recentReleases) =>
                        state.recentReleases = recentReleases,
    setUpcomingReleases: (state, upcomingReleases) =>
                        state.upcomingReleases = upcomingReleases,
    loadingStatus: (state, newLoadingStatus) =>
                        state.loadingStatus = newLoadingStatus
};

export default {
    state,
    getters,
    actions,
    mutations
};
```

5.1.3 Filtering, ordinamento e paginazione

La pagina Games implementa vue-infinte-handler, un package che permette lo scorrimento della pagina finché vi sono risultati. Tali risultati possono venire filtrati e ordinati secondo vari criteri impostati dall'utente. La funzione principale richiamata in seguito a ogni cambiamento rilevato è la seguente:

```
this.loadGames({
  params: {
        page: this.page,
        limit: 20,
        released: released,
        genres: this.selectedGenres.length > 0 ? this.selectedGenres : null,
        platforms: this.selectedPlatforms.length > 0 ?
                  this.selectedPlatforms : null,
        name: this.searchTitle,
        sort: this.selectedSort == "" ?
              null : this.selectedSort
        }
})
.then(loadState => {
  //if there were other pages left inc the page counter
  if(this.loadedAll == false) {
        this.page++;
        $state.loaded();
  } else {
```

```
    //else the request can either be complete or there wasnt any response
    if(this.page == 1 && this.games.length > 0) {
      $state.loaded();
      $state.complete();
    } else {
      $state.complete();
    }
  }
})
```

5.1.4 Router

Il componente router implementa una guard beforeEnter nel caso si tenti di accedere ad una pagina che richiede autenticazione.

```
{
  path: '/options',
  name: 'Options',
  component: Options,
  beforeEnter: async (to, from, next) => {
    await store.dispatch("fetchUserProfile");
    if(store.getters["getUserProfile"] == null) {
      next({ name: 'Home' });
      Vue.notify({
            group: 'notify',
            type: 'error',
            title: 'You need to be signed-in in order to see this page!'
      });
    }
    next();
  }
}
```

5.1.5 SASS

Il precompilatore SASS è stato utilizzato per realizzare interamente il css del frontend. Le funzioni principali sono riportate in seguito.

- ratingColor: mixin per generare una serie di classi con con valore numerico da 1 a 100, tali classi avranno una proprietà background-color con tonalità tendente al rosso per valori bassi, giallo intermedi e verde per valori prossimi al 100

```
@mixin ratingColor {
    @for $i from 1 through 100 {
        .r#{$i} {
            $hue: 120 * ($i / 100);
```

```
                background-color: hsl($hue,100%,50%);
            }
        }
    }
```

- hoverTransition e onHover: coppia di mixin utilizzata per generare una transizione ease nel :hover dell'elemento

```
@mixin hoverTransition {
    transition: opacity .5s ease;
    background-color: transparent;
}

@mixin onHover {
    opacity: 0.65;
}
```

5.2 Backend

5.2.1 APIService

Richieste GET

Per il reperimento delle informazioni riguardanti le date o i giochi, data la numerosa quantità di possibili filtri applicabili, le funzioni realizzate interrogano il database in base a tutte le impostazioni reperite dalla query della richiesta. Un esempio è la funzione getGames rappresentata in seguito.

```
let query = {};

let options = {
    page: req.query.page != null ? Number(req.query.page) : 1,
    limit: req.query.limit != null ? Number(req.query.limit) : 500,
    populate: ["release_dates","platforms","genres"],
    sort: req.query.sort != null ? req.query.sort : "",
};

if(req.query.released == "true") {
    let dates = await Release_Dates.find({category: 0,
                                          date: { $lte: moment().unix() }});
    query["release_dates"] = {
        $in: dates.map(el => el.id)
    };
} else if (req.query.released == "false") {
```

```
    let dates = await Release_Dates.find({category: 0,
                                          date: { $gt: moment().unix() }});
    query["release_dates"] = {
        $in: dates.map(el => el.id)
    };
} else if (req.query.released == "TBA") {
    let dates = await Release_Dates.find({category: {$ne: 0}});
    query["release_dates"] = {
        $in: dates.map(el => el.id)
    };
}

if(req.query.minrating != null && req.query.maxrating != null) {
    query["aggregated_rating"] = {
        $gte: req.query.minrating,
        $lte: req.query.maxrating
    };
}
if(req.query.category != null) {
    query["category"] = {
        $in: req.query.category
    };
}

...

Games.paginate(query, options, function(err, games) {
    if(err || games == null) {
        res.send("Error");
    } else {
        res.json(games);
    }
});
```

Autenticazione

L'autenticazione è stata implementata mediante il package passport e la relativa Strategy passport-google. Una volta configurato passport (come rappresentato in seguito), per le route che necessitano o meno di autenticazione viene usata una funzione requireAuth o requireGuest.

```
passport.use(new GoogleStategy({
  clientID: process.env.GOOGLE_CLIENT_ID,
  clientSecret: process.env.GOOGLE_CLIENT_SECRET,
  callbackURL: "/api/auth/google/callback"
},
```

```
  async (accessToken, refreshToken, profile, done) => {
    const newUser = {
      googleId: profile.id,
      mail: profile.emails[0].value,
      nickname: profile.displayName,
      firstName: profile.name.givenName,
      lastName: profile.name.familyName,
      image: profile.photos[0].value
    };
    try {
      let user = await User.findOne({ googleId: profile.id });
      if (!user) {
        user = await User.create(newUser);
      }
      done(null, user);
    } catch (e) {
      console.error(e);
    }
  }
));
```

5.2.2 IGDBService

Il servizio IGDBService viene eseguito una volta al giorno alle ore 00:00 per controllare se sono stati aggiungete nuovi giochi o date.
Dato che il database IGDB impone un limite di richieste mensili, con un numero di risultati pari a 500 e un offset massimo pari a 5000 per richiesta, tutte le richieste sono create in modo da ricevere il maggior numero di risultati minimizzando il numero di richieste mensili utilizzate.

IGDBDataDownloader

IGDBDataDownloader è il sorgente contenente tutte le funzioni che interagiscono con il database pubblico IGDB.
La funzione principale è denominata "downloadAllData" e permette di scaricare tutti i dati in base ad un url e una query forniti come parametri; è anche possibile specificare una funzione di callback. Le richieste vengono effettuate utilizzando puppeteer, tenendo conto del limite e offset impostati, come segue:

```
let allResults = [];
for(let i = 0; i < Math.ceil(maxoffset/limit); i++) {
    let page = await browser.newPage();
    await page.setRequestInterception(true);
    await page.setDefaultNavigationTimeout(0);

    let query;
```

```
    if(basicQuery.includes("limit")) {
        query = basicQuery + "offset " + (limit * i) + ";";
    } else {
        query = basicQuery + "limit " + limit + "; offset " + (limit * i) + ";";
    }

    page.on('request', interceptedRequest => {
        let data = {
            'method': 'POST',
            'postData': query
        };

        interceptedRequest.continue(data);
    });

    await page.goto(proxyURL + url);
    let result = await page.evaluate(() =>  {
        return JSON.parse(document.querySelector("body").innerText);
    });

    //exit if there are no more results
    if(!Object.keys(result).length){
        break;
    }

    allResults.push(result);
}
```

IGDBDataHandler

Una volta ricevuta una lista di tutti i videogiochi, il sorgente IGDBDataHandler riconosce tutti i giochi o date non ancora presenti nel database. L'esempio rappresentato in seguito mostra come vengono reperite tutte le piattaforme ignote per i giochi appena reperiti (tale procedimento è ripetuto anche per i generi, cover, screenshot).

```
// Game platforms
let platformsSet = [...new Set((data.filter(el => el.platforms != null)
                                    .map(el => el.platforms))
                                    .flat(1))];
let platforms = [];
if(platformsSet.length > 0) {
  platforms = await Platforms.find(({code: {$in: platformsSet} }),
  function(err, response) {
    if(err) {
          console.log(err);
```

```
        } else {
          return response;
        }
  });

  // If the db doesn't contain all the platforms
  if(platformsSet.length != platforms.length) {
    let newPlatforms = platformsSet.filter(el => !platforms.map(x => x.code)
                                                           .includes(el.toString()));

    console.log("Downloading new platforms: " + newPlatforms);
    let newPlatformsData = await getSpecificData("https://api-v3.igdb.com/platforms",
                                                 "fields name;",
                                                  newPlatforms);

    for(i in newPlatformsData){
      let saved = await saveToDB(Platforms, {
        name: newPlatformsData[i].name,
        code: newPlatformsData[i].id
      });
      platforms.push({ id: saved.id, name: saved.name, code: saved.code});
    }
  }
}
```

5.2.3 MailService

Il servizio MailService viene avviato ogni giorno alle ore 00:00 per avvisare gli utenti dell'imminente uscita di un gioco da essi seguito. Il codice per effettuare l'invio delle mail è rappresentato in seguito.

```
let from = moment().startOf("day").unix();
let to = moment().endOf("day").unix();

let { data } = await axios.get("http://localhost:3030/api/releases", {
    params: {
        "from": from,
        "to": to,
        "category": 0
    }
});

let users = await User.find({"datesFollowed": {
    $in: data.map(el => el._id)
}, "mailNotifications": true});
```

```
users.forEach(user => {
    let mailDates = data.filter(el => user.datesFollowed.includes(el._id.toString()));

    let mailOptions = {
        from: process.env.MAIL,
        to: user.mail,
        subject: "WhenCanIPLAYIt release reminder",
        template: "dailyReminder",
        context: {
            nick: user.nickname,
            dates: mailDates
        },
        attachments: [{
            filename: 'logo.png',
            path: __dirname +'/public/logo.png',
            cid: 'logo'
        }]
    };

    transporter.sendMail(mailOptions, function(error, info){
        if (error) {
            console.log(error);
        } else {
            console.log('Email sent: ' + info.response);
        }
    });
});
```

La stilizzazione delle mail è stata effettuate mediante il package nodemailer-handlebars e un esempio del risultato finale è mostrato in figura Figura 5.1.

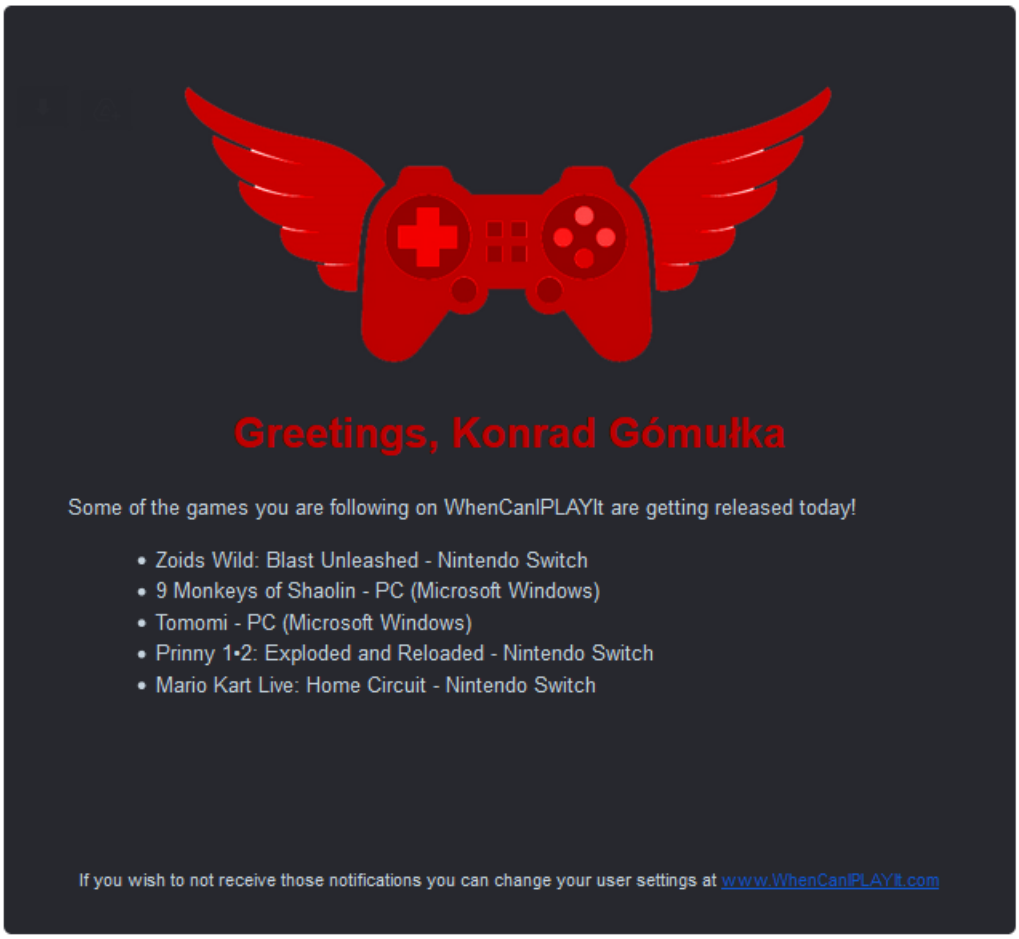

Figura 5.1: Mail di notifica inviata all'utente.

Capitolo 6

Test

6.1 Test sul codice

6.1.1 APIService

Per il testing del servizio APIService è stato utilizzato il tool Postman, effettuando richieste per tutte le possibili route e controllando la validità delle risposte.

6.1.2 IGDBService

Per il servizio IGDBService, in una fase iniziale, sono stati utilizzati vari log per controllare la validità dei dati ottenuti dal database IGDB, mettendoli a confronto con delle richieste effettuate manualmente tramite Postman. Una volta assicurato che i dati ottenuti fossero integri, campionando il numero di giochi richiesti è stato effettuata un'accurata analisi dei dati inseriti nel database MongoDB, confrontandoli con il risultato desiderato.

6.1.3 MailService

Per testare il servizio MailService, inizialmente sono state inviate delle mail di prova per controllare la formattazione delle mail. Una volta conclusa la prima fase è stata verificata anche la correttezza dei dati ricevuti controllando se i titoli dei giochi ricevuti corrispondessero realmente a quelli seguiti e in uscita il giorno stesso.

6.2 Test con l'utente

Durante la fase di sviluppo sono stati presentati i vari mockup a vari conoscenti, considerati come target prinicipale dell'applicazione, e raccogliendo le varie opinioni in merito a aspetto e funzionalità desiderate.

Una volta terminato, l'applicativo è stato presentato ad un gruppo di persone presenti fisicamente in sede di sviluppo (seppur limitato considerando le circostanze), richiedendo di testare la versione desktop nonché quella mobile e fornendo un feedback. Per ottenere un maggior numero di riscontri l'applicativo è stato anche presentato ad altri conoscenti mediante la condivisione schermo del software Discord, mostrando le varie funzionalità offerte dall'applicativo e chiedendo eventuali riscontri.

Capitolo 7

Deployment

Il deployment dell'applicazione è stato realizzato mediante docker. All'interno del dockerfile viene inizializzato npm, installate tutte le dipendenze di chromium necessarie a far funzionare il package puppeteer e infine richiamato il comando deploy contenuto in package.json.

All'interno di package.json vi sono due script:

- postinstall: richiamato in automatico al termine dell'installazione di npm. Installa tutte le dipendenze necessarie per far funzionare i vari servizi e sposta il contenuto di Frontend all'interno di ApiService, permettendo di esporre tutto alla stessa porta tramite il package 'connect-history-api-fallback'
- deploy: richiamato una volta terminato il setup del container, esegue tutti i servizi

7.1 Avvio

Dato che tutti i servizi vengono avviati sullo stesso container, per l'avvio è necessario clonare la repo e eseguire il seguente comando dalla cartella src
docker build . -t app
per poi avviare il servizio con
docker run -d -p 3030:3030 app

Note importanti:

- Nel caso si voglia avviare il servizio da docker è necessario sostituire la porta del frontend nelle variabili ambientali contenute in ApiService
- Data la notevole dimensione delle cartelle contenti le immagini, esse non sono contenute nella repository ma vanno scaricate a parte mediante il link

https://drive.google.com/file/d/1M4xbcCWL8Vlh$_xOkFDqjaGXH5fxdpQrp$
e inserite nella cartella public di IGDBService

Capitolo 8

Conclusioni

In conclusione mi ritengo estremamente soddisfatto dalle possibilità offerte dallo stack MEVN nonché dalla sua semplicità nell'utilizzo. In particolare sono rimasto sorpreso dalla quantità di packages di Vue reperibili dal web, semplificando notevolmente lo sviluppo sia del frontend che backend.
Sono sicuro che dopo tale esperienza di sviluppo mediante lo stack MEVN continuerò ad utilizzarlo per qualsiasi progetto futuro.

Unico rimpianto è che essendo in una situazione particolare e sviluppando il progetto da solo ho dovuto limitarne le possibilità, dovendo rinunciare a numerose funzionalità che sicuramente sarebbero state interessanti da implementare.

8.1 Prodotto finale

In seguito sono allegati gli screenshot dell'applicazione.

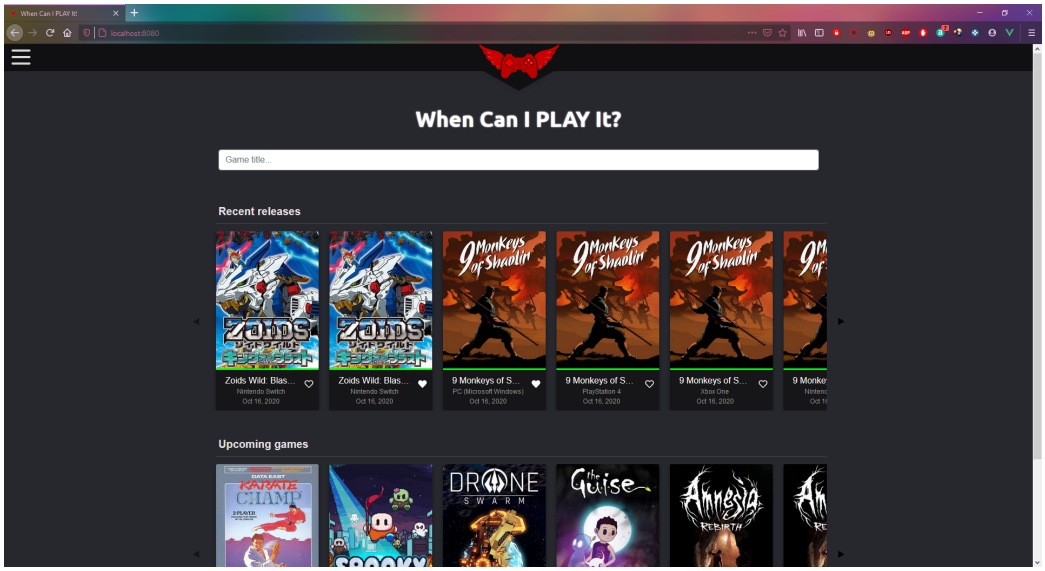

Figura 8.1: Home page in versione desktop.

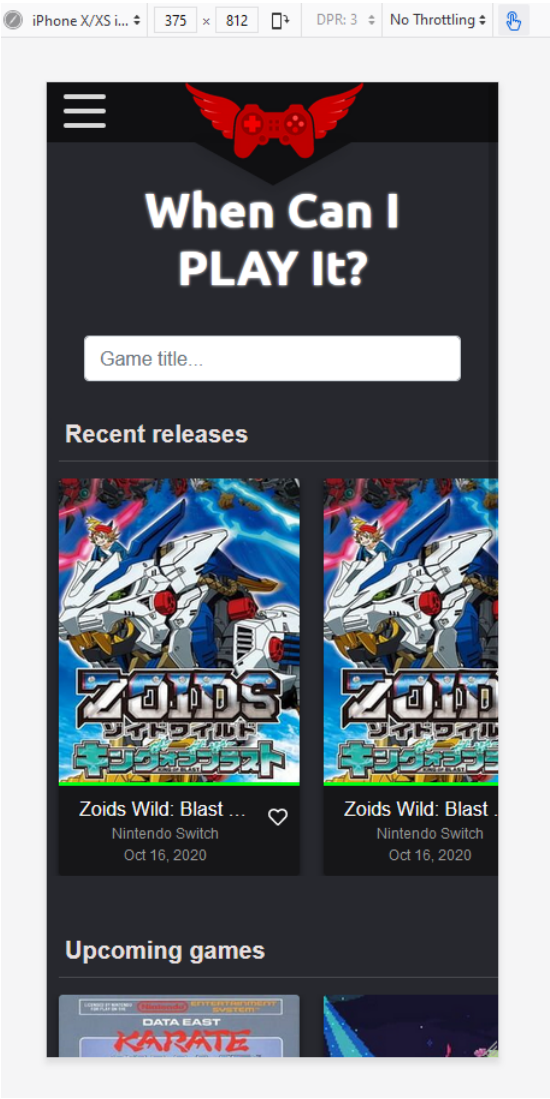

Figura 8.2: Home page in versione mobile.

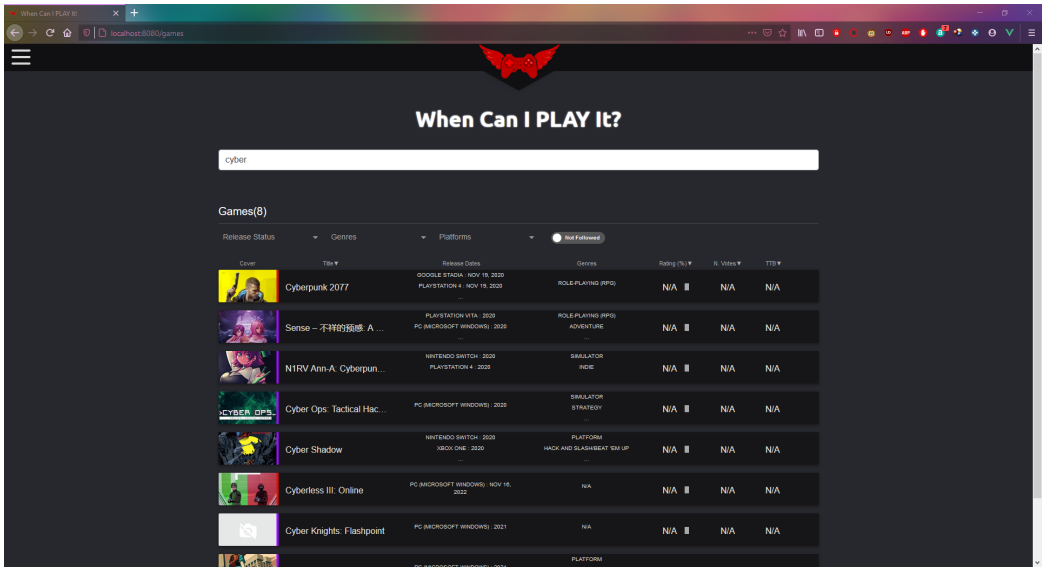

Figura 8.3: Pagina Games in versione desktop.

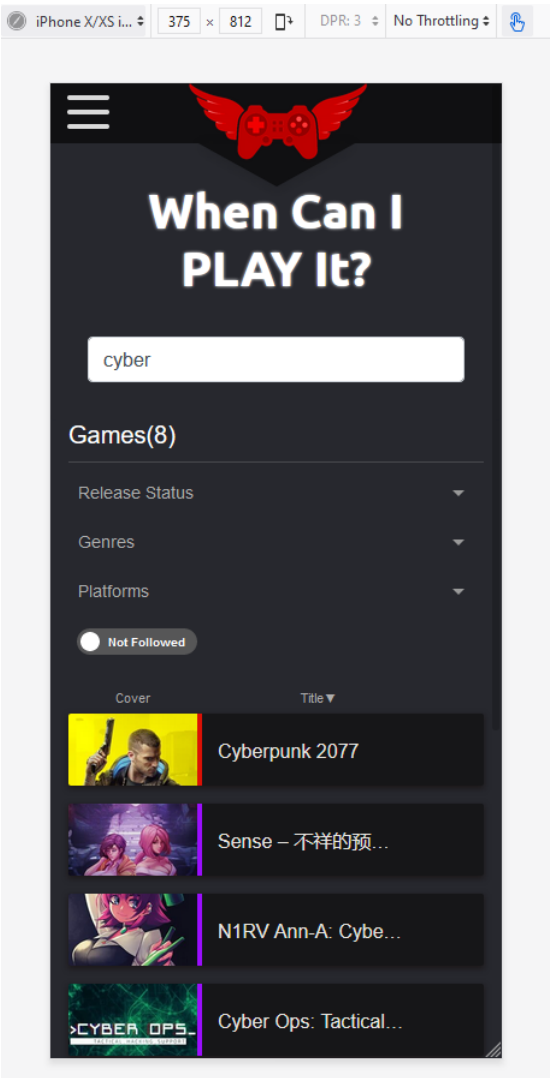

Figura 8.4: Pagina games in versione mobile.

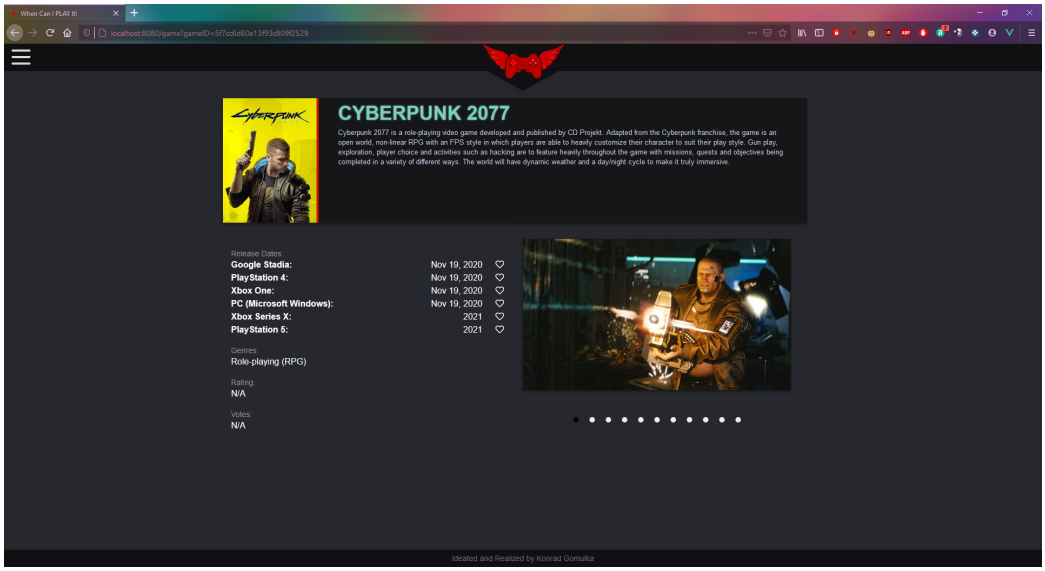

Figura 8.5: Pagina Game in versione desktop.

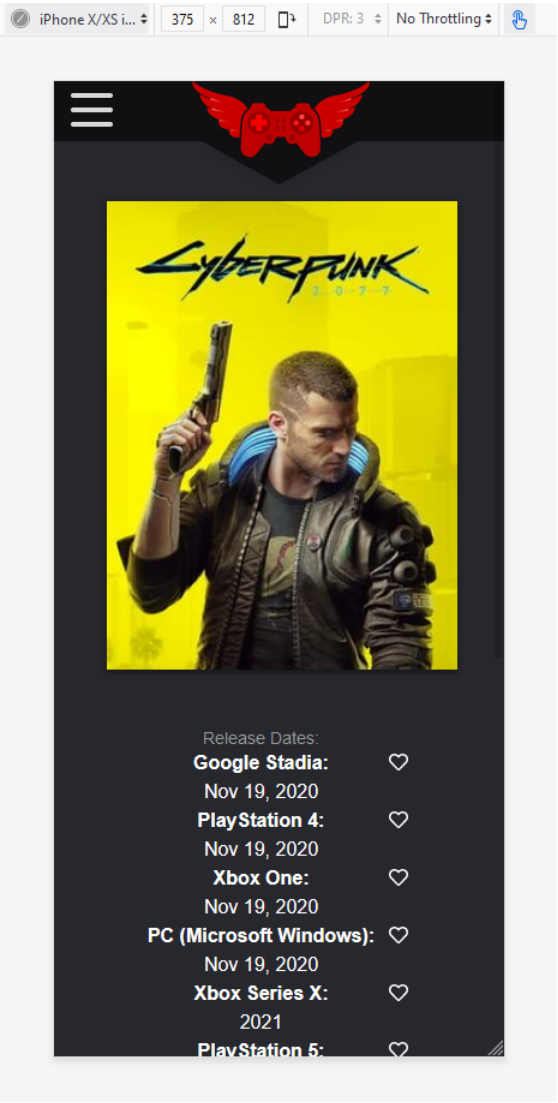

Figura 8.6: Pagina game in versione mobile.

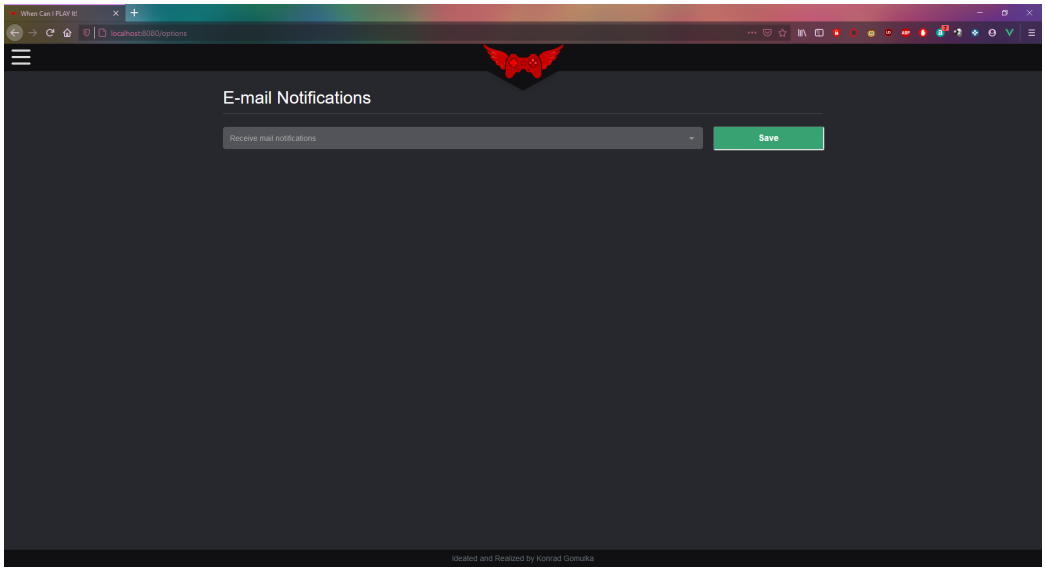

Figura 8.7: Pagina options in versione desktop.

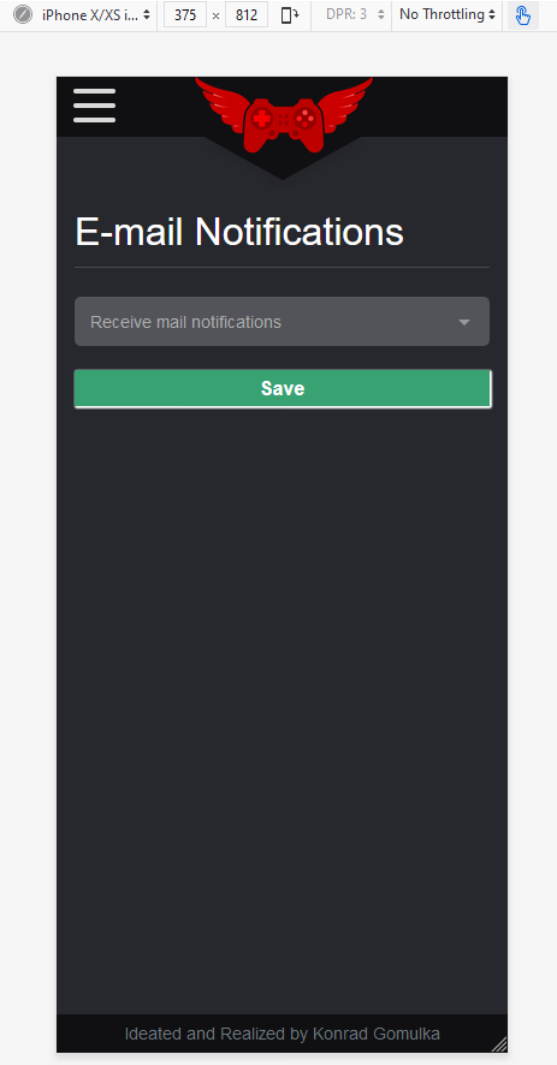

Figura 8.8: Pagina options in versione mobile.

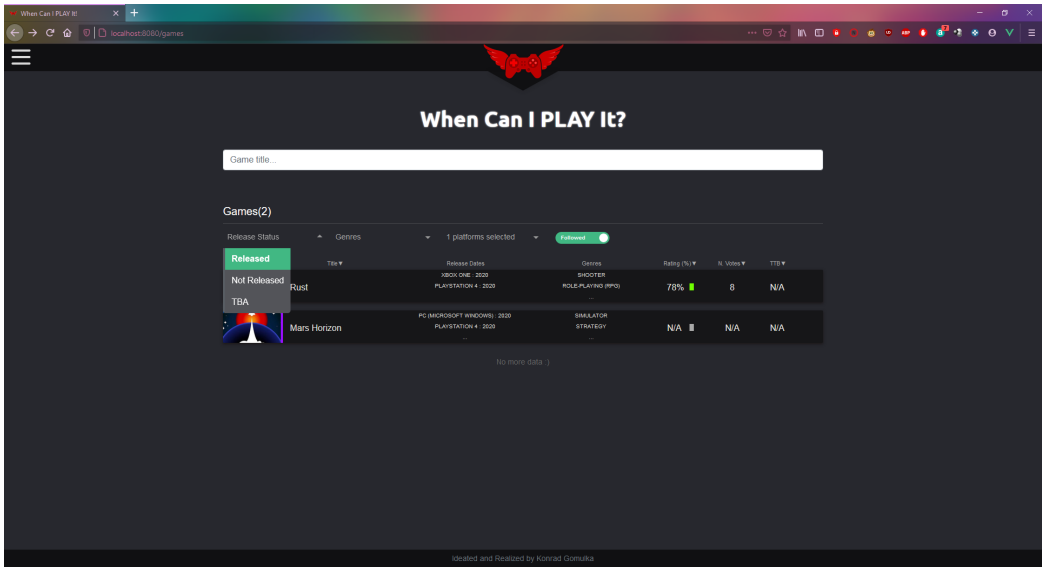

Figura 8.9: Filtering dei giochi in versione desktop.

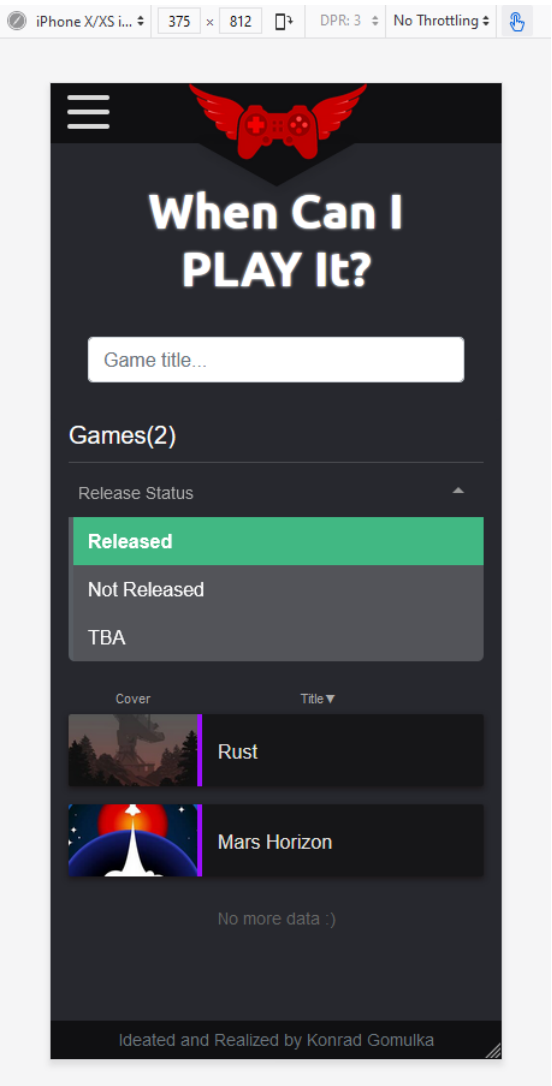

Figura 8.10: Filtering dei giochi in versione mobile.

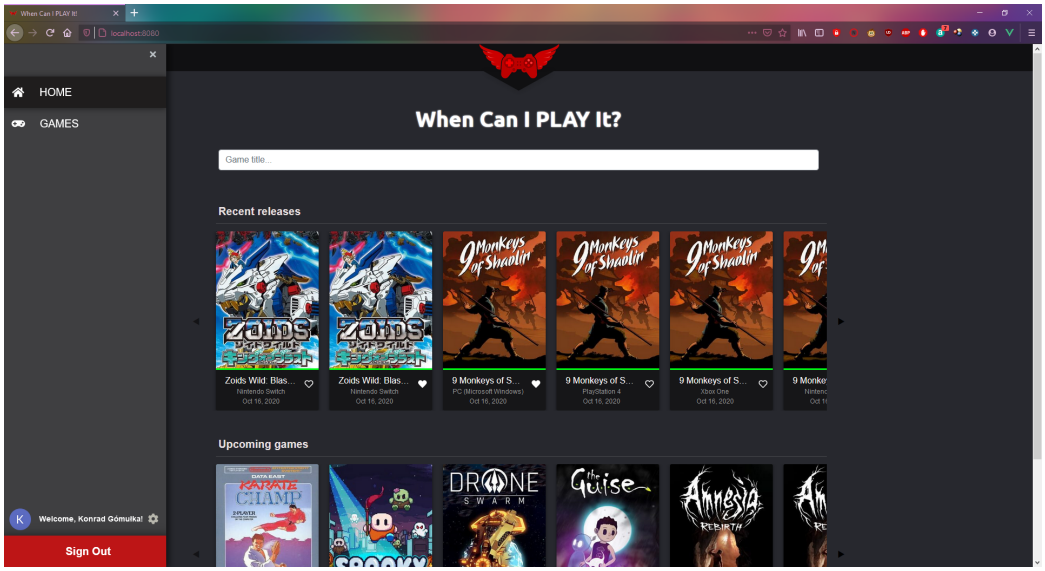

Figura 8.11: Sidebar in versione desktop.

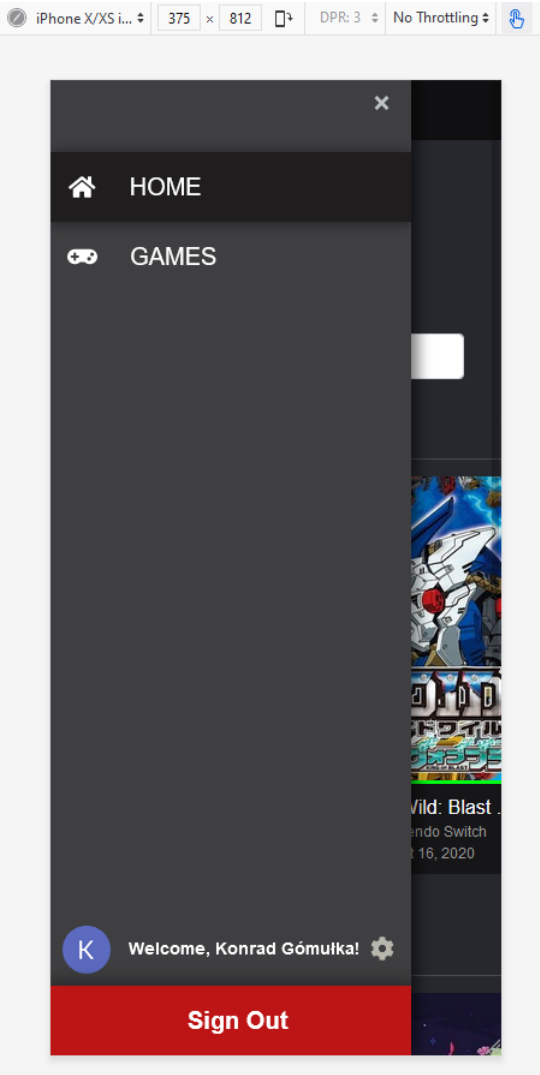

Figura 8.12: Sidebar in versione mobile.

Bibliografia

[1] IGDB. Internet game database.