APLICACIÓ WEB PER A UN CLUB ESPORTIU

TREBALL DE FI DE GRAU

Autor: Víctor Cano Rodríguez
Dirigit per: Sr. Esteban Herreros

Grau de Enginyeria Informàtica

UNIVERSITAT ROVIRA I VIRGILI

Tarragona
2022

Índex

Índex de figures

Índex de codi

1 Introducció

1.1 Visió general

El Treball de Fi de Grau ha consistit en un projecte de programació web enfocat en el disseny i creació d'una base de dades relacional, programació d'una API[1] REST[2], nocions de seguretat a la xarxa i publicació de l'aplicació mitjançant eines de gestió de sistemes.

L'aplicació web permet a tots els usuaris d'un club (jugadors, entrenador i directius) accedir a eines com poden ser veure un calendari de partits, els horaris d'entrenaments, rebre notificacions, realitzar pagaments de les quotes, comprar equipació o veure les classificacions. La informació com partits i horaris d'entrenament és configurable pels directius del club i és visible pels jugadors i entrenadors per a que la puguin consultar.

La idea del projecte va sorgir perquè el meu pare és el president del Club Voleibol Reus i em va comentar la necessitat que tenien de tindre una pàgina web. Vaig pensar que si feia una aplicació web amb una intranet, un sistema de rols i amb eines d'utilitat per als membres del club podria ser un projecte interessant, i així es va escollir aquesta idea.

1.2 Objectius del projecte

Els objectius a l'hora de realitzar aquest projecte eren: consolidar els coneixements de desenvolupament d'APIs REST; practicar també la part de Front-end crear diferents pàgines HTML[3] amb CSS i JavaScript; crear una base de dades relacional amb SQL[4] i tindre-la en un servidor SQLExpress per aprendre del procés, realitzar connexions amb altres APIs per aprendre com es fa; aplicar conceptes de seguretat en xarxes per a investigar com es realitzen els diferents processos que proporcionen seguretat als usuaris en altres aplicacions web; aprendre com es posa en producció un projecte així; i iniciar-me en el framework **.NET Core**, que el vaig escollir perquè és de codi obert, és multiplataforma i és més modern que el framework en el que es basa, **.NET**.

[1] API: Application Programming Interface (Interfície de programació d'aplicacions).
[2] Rest: REpresentational State Transfer (Transferència d'estats de representació).
[3] HTML: Hyper Text Markup Language (Llenguatge de Marcat de Híper Text).
[4] SQL: Structured Query Language (Llenguatge de Consultes Estructurades).

2 Descripció general del projecte

2.1 Entorn de desenvolupament

Per a desenvolupar el projecte s'han utilitzat diferents entorns, primer que tot per a la major part del desenvolupament que ha implicat crear la base de dades o programar s'ha utilitzat un ordinador d'escriptori amb Windows 10 perquè és el sistema operatiu que utilitza el meu ordinador personal i al principi no es tenia cap necessitar d'utilitzar cap programa o framework que fos específic d'algun sistema operatiu. Per a crear la base de dades es va utilitzar MySql perquè és un dels sistemes de gestió de bases de dades més coneguts i té suport en múltiples plataformes i entorns i també perquè la meva idea era crear una base de dades relacional i per aquest motiu es va considerar el llenguatge SQL com a millor opció.

De totes maneres posteriorment es necessitaria un servidor on posar la base de dades per poder accedir desde el framework Entity FrameworkCore i llavors es va decidir utilitzar SQL Express, que és un sistema de gestió de bases de dades relacionals i és tracta d'una versió lleugera de Microsoft SQL Server i que és gratuïta.

Un cop es va tindre la base de dades muntada va començar la part de programar el Back-end de l'aplicació web. Pel projecte es volia utilitzar o bé el framework ASP.NET o ASP.NET Core perquè es tenia experiència prèvia amb ASP.NET i té avantatges molt útils com:

- La reducció de temps de programació
- Compta amb multitud d'eines i opcions per a configurar els projectes amb el IDE Visual Studio.
- El procés de posada en producció és senzill.
- Com és un framework molt popular també m'he pogut beneficiar de multitud de tutorials en internet, molts fòrums amb dubtes comuns resolts i una documentació extensiva del diferents components.

Al final es va decidir per ASP.NET Core perquè tot i que té alguns avantatges i alguns desavantatges, les seves virtuts cridaven l'atenció, algunes d'aquestes son:

- Core es basa en .NET però reescrit des de zero i aprofitant totes les característiques potents i traient les parts menys atractives de .NET.
- ASP.NET Core és un projecte de codi obert i per tant qualsevol persona pot fer aportacions al codi i, personalment, és una característica que valoro en els projectes. (Tot i que soc conscient que també comporta inconvenients).
- ASP.NET Core és multiplataforma, a diferencia del seu predecessor, tot i que utilitzo Windows durant la major part del desenvolupament, a la part final utilitzo Linux i llavors aquest és un punt molt fort per a ASP.NET Core.
- Core té un rendiment molt major, ja que va ser una de les prioritats en el seu desenvolupament.

Per a programar amb aquest framework s'ha decidit utilitzar el patró de disseny MVC[5] perquè és un patró molt potent per a servidors web i el qual ja havia utilitzat abans.
Conjuntament amb ASP.NET Core també es va escollir fer servir la eina Razor, que és una eina per a per a executar codi C# en una vista HTML. No és un punt que fos d'interès inicialment perquè la idea era fer vistes simples que obtinguessin continguts de la API REST però es va considerar que potser es podria fer servir puntualment per estalviar-se de crear funcions molt redundants que podrien no fer falta amb un parell de línies de Razor.
També es va fer servir un altre framework molt útil per a fer servir amb ASP.NET Core: Entity Framework Core. És molt útil per accedir a objectes de una base de dades simplificant molt el procés de fer consultes, insercions, actualitzacions...

Per a tindre un seguiment del codi i poder accedir a versions anteriors del projecte s'ha utilitzat el software de control de versions Git. S'ha fet servir amb una eina integrada en Visual Studio per a fer totes les comandes que es podrien fer a Git directament des de la interfície de Visual Studio. El projecte té un repositori que es va crear a GitHub per la senzillesa que té connectar-lo a la eina de Git de Visual Studio.

Per a la part final del projecte es va haver d'utilitzar Linux per a poder posar en producció el projecte i es va decidir utilitzar una màquina virtual en el meu propi ordinador perquè no disposava de cap ordinador amb una distribució de Linux. Per a la meva màquina virtual es va escollir la distribució Ubuntu perquè es una de les més populars i fàcils de fer servir a més de que ja comptava amb experiència amb aquesta. El programa que vaig fer servir per a tindre la màquina virtual va ser Virtual Box per ser un dels programes més potents d'aquest tipus.

El IDE utilitzat en Ubuntu va ser Visual Studio Code perquè és molt més ràpid i dinàmic que Visual Studio i anava millor per a fer pulls del repositori.

I per a tindre un proxy invers s'ha fet servir nginx, que ha permès redirigir el tràfic de el nom del domini al servidor web.

2.2 Necessitats del projecte

Durant el desenvolupament del projecte han sorgit diverses necessitats que s'han anat resolent com poden ser:

- Servei DDNS[6]: Quan es va començar la fase de producció del projecte el tutor va comentar a una reunió que un apartat del projecte podria consistir en utilitzar un servidor DDNS per a accedir a la aplicació web a partir de un domini i així em va sorgir aquesta necessitat.

[5] Model View Controller: Model Vista Controlador, és un patró de disseny de programació de software molt popular que serveix per a separar les dades de negoci de la interfície gràfica i connecta ambdues amb un controlador.
[6] Dynamic Domain Name Server: Servidor de Noms de Domini Dinàmc

- Obrir un port al meu router: Per a poder rebre peticions HTTP al meu servidor web des de un entorn remot es va tindre que modificar les opcions del router de casa meva per a acceptar tràfic HTTP i redirigir-lo a la ip privada que tenia la meva màquina virtual.
- Proxy Invers: Relacionat amb la necessitat anterior, també calia una forma de redirigir les peticions que arribaven a la meva màquina virtual al servidor web del meu projecte.

3 Requisits

3.1 Requisits funcionals

1. Poder registrar a usuaris.
2. Que usuaris puguin iniciar sessió.
3. Que els usuaris puguin tancar la sessió quan ho desitgin.
4. Tindre diferents pàgines accessibles per a tothom.
5. Tindre pàgines privades accessibles només per a usuaris.
6. Disposar d'un sistema de rols(jugadors,entrenadors i directius).
7. Que els jugadors puguin veure el seu horari.
8. Que els jugadors puguin veure el seu calendari de partits.
9. Que els jugadors rebin notificacions les setmanes que tenen partits.
10. Els punts 6, 7 i 8 han d'estar replicats corresponentment als entrenadors, tenint en compte que un entrenador pot tindre més d'un equip.
11. Els directius han de poder donar d'alta usuaris.
12. Els directius han de poder edit les dades dels usuaris.
13. Els directius han de poder llegir les dades del usuaris.
14. Els directius han de poder eliminar usuaris.
15. Els directius han de poder establir els horaris d'entrenament pels jugadors i entrenadors.
16. Els directius han de poder posar les dates dels partits per a que apareguin als calendaris dels jugadors.
17. Hi ha d'haver un apartat públic que mostri les classificacions de tots els equips.
18. L'aplicació web ha de poder enviar missatges als usuaris per Telegram.
19. La web ha de ser accessible des de qualsevol dispositiu.
20. Les pàgines han de ser responsives.
21. Els directius han de poder veure les quotes.
22. Els directius han de poder editar les quotes.
23. Els directius han de poder crear quotes.
24. Els directius han de poder eliminar quotes.
25. Les contrasenyes dels usuaris han d'estar emmagatzemades de forma segura.
26. Una de les pàgines públiques ha de consistir en un blog.
27. Els directius han de poder crear entrades pel blog.
28. Els directius han de poder editar entrades del blog.
29. Els directius han de poder eliminar entrades del blog.

3.2 Requisits no funcionals

1. S'han de poder fer els pagaments de les quotes via Telegram.
2. S'han d'emmagatzemar les transaccions.
3. Les transaccions han de ser visibles pels directius.
4. Els directius han de poder eliminar transaccions.
5. La pàgina ha de contar amb una secció de highlights.
6. Cada usuari ha de ser capaç de penjar els seus highlights i la pàgina ha de mostrar els més recents i més impactants.
7. La pàgina ha de contar amb una petita secció d'estadístiques sobre la pròpia aplicació web.
8. Els directius han de poder realitzar sortejos dins de la pàgina.
9. L'aplicació ha d'utilitzar el protocol HTTPS.
10. El servidor web ha d'estar allotjat en alguna màquina que estigui disponible gairebé el 100% del temps.

4 Disseny

4.1 Arquitectura de l'aplicació

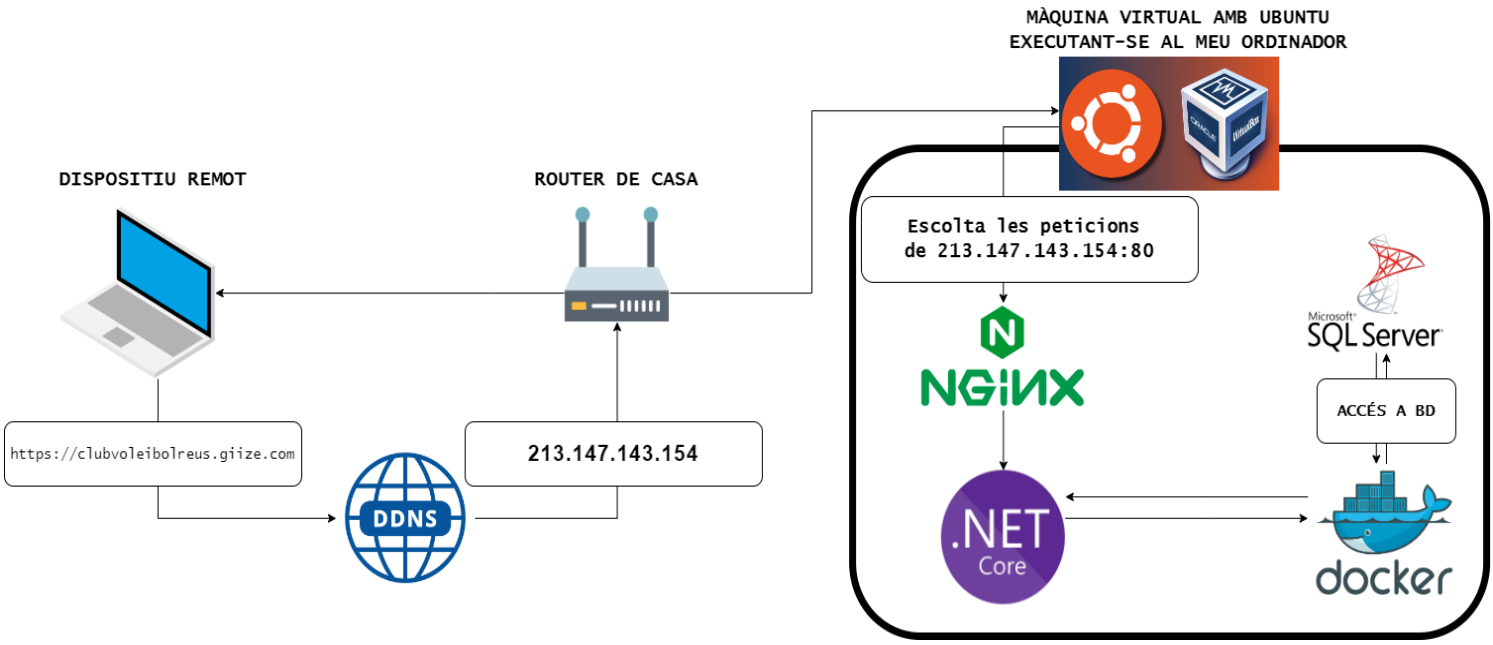


Figura 1. Esquema de l'arquitectura de l'aplicació.

L'arquitectura consisteix en:

- Un servidor DDNS per a utilitzar el domini **clubvoleibolreus.giize.com** en la IP pública del meu router (213.147.143.154). Per a tractar el problema de que les IPs públiques poden canviar amb el temps, s'ha implementat una tasca programada amb el crontab de Linux per a actualitzar la meva IP a la pàgina per si canvia. S'ha indicat que aquesta tasca s'executi cada 15 indicant-ho així amb una entrada nova al fitxer que s'edita amb la comanda **crontab -e**. Dins d'aquest fitxer s'ha afegit la comanda

```
*/15 * * * * wget -O dynulog -4
"https://api.dynu.com/nic/update?hostname=clubvoleibolreus.giize.com&ip=10.0.0.0&myipv6=no&username=bitoh&password=0163e0c50082f412cb9bdefb17730979"
```

Aquesta comanda fa una petició a la API de Dynu per a actualitzar la IP pública, ho fa amb els paràmetres de hostname sent el nom del domini, ip sent la IP que es vol canviar (que si es deixa a 10.0.0.0 s'agafa la pública detectada en aquell moment), ipv6 a no perquè no ens interessa i el usuari i la contrasenya de la conta. La contrasenya és un hash obtingut amb una eina de la pròpia pàgina de Dynu per a no passar la contrasenya en clar i exposar-la.

- El meu router, on s'ha configurat que accepti el tràfic de fora del protocol HTTP.

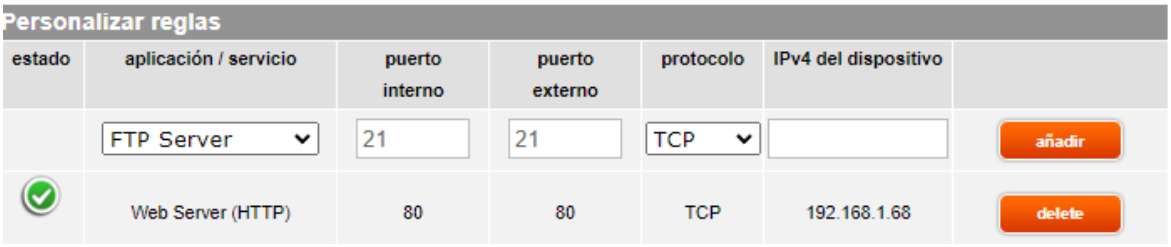


Figura 2. Regla per acceptar tràfic HTTP des de l'exterior de la xarxa.

- Una màquina virtual al meu ordinador amb **Ubuntu** i amb el administrador de màquines virtuals **VirtualBox.** També es va afegir una opció per a fer reenviament de ports per a enviar les peticions des del "host" a la màquina "guest".

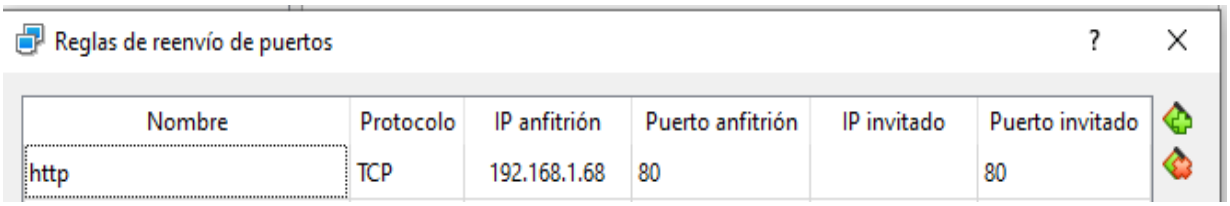


Figura 3. Regles de reenviament de ports a la màquina virtual.

- A la màquina virtual es va configurar un servidor nginx per a que escoltés el tràfic provinent de **http://clubvoleibolreus.giize.com** (però que arriba al router i es va reenviant fins la màquina virtual) i que el redirigeixi al localhost en el port 5001, que és on està escoltant les peticions la meva aplicació ASP.NET Core.
- El servidor web en ASP.NET Core és el s'ha programat per fer tota la funcionalitat principal del projecte, rep les peticions quan el servidor nginx li reenvia.
- Finalment, també tinc un docker a la màquina virtual on dins es té el servidor SQL Express engegat per a fer consultes i actualitzacions a la base de dades. El motiu per el que s'ha necessitat fer servir un docker i no tindre el servidor a la pròpia màquina virtual és que la versió de Ubuntu que tinc instal·lada (22.04) no és compatible amb la CLI de SQL Server encara, llavors es tenien dos alternatives: crear una altra màquina virtual amb una versió anterior de Ubuntu (20.04 o anterior) o bé utilitzar un docker. La primera opció donava més feina de traspassar el entorn i la segona és

una aportació interessant per al treball i a més es va plantejar la idea de en un futur traspassar el projecte sencer a un contenidor per minimitzar l'ús de disc que necessitaria el servidor i així optimitzar el servidor web. A més, Docker és una eina molt útil perquè facilitaria la feina de traspàs d'entorn simplificant molt significativament el procés de traslladar el projecte d'una màquina a una altra, per exemple.

4.2 Disseny de la interfície gràfica

Tot i que la interfície gràfica no és dels punts més forts o interessants d'aquest projecte, s'ha decidit fer-la mínimament estètica i responsiva.

4.2.1 Llibreries utilitzades

Per a la capa de Front-end s'ha fet servir dos llibreries:

1. Bootstrap:

Bootstrap és una biblioteca i set d'eines per al disseny de pàgines web i ja comptava amb experiència utilitzant-la i per això s'ha he fet servir. Els punts més forts que té per mi son:

- Eines útils per a fer responsius els dissenys: El grid de bootstrap et permet dividir el contingut de les pàgines en columnes on indiques quantes columnes hauria de tindre el disseny segons la mida del dispositiu on es visualitza la pàgina.
- Estils inclosos per a fer més visualment atractius alguns elements com: botons, checkboxes, taules...
- En general trobo que és una eina per a donar consistència a la meva interfície gràfica i fer-la més vistosa.

2. JQuery

JQuery és una lliberia de JavaScript que s'ha fet servir per al scripts que han fet falta en els meus fitxers HTML. Es molt útil perquè estalvia escriure molt de codi i fa més senzill l'ús de JavaScript per modificar el contingut de les pàgines. Trobo necessari inclou-re aquesta llibreria en el apartat de Disseny de la interfície gràfica perquè JavaScript forma part del Front-end d'una aplicació web.

4.2.2 Apartats de la interfície

Navbar:

Un navbar o barra de navegació és una secció d'una pàgina web que serveix per a proporcionar enllaços a tot el contingut de l'aplicació. S'ha creat un navbar tot i que no s'ha arribat a implementar totes les pàgines dels enllaços que apareixen al navbar perquè he considerat que aquesta seria la part menys interessant del projecte ja que no té dificultat.

Figura 4. Barra de navegació vista des de una resolució gran i ample.

Els enllaços als que es pot accedir des de el meu navbar són:

- Home page: A diferencia dels altres enllaços, aquest s'ha fet que sigui una foto ja que és mes vistós i opino que la major part de la gent entén que un logo en una racó superior esquerra de la pàgina et serveix per a tornar a la pàgina principal.
- Classificacions: En aquest apartat pots accedir a una pàgina on es poden veure els resultats de tots els equips federats del club.
- ¡Apuntat!, Els nostres equips: Aquestes dos seccions s'han posat per a omplir el navbar i que no quedi tan buit però no tenen cap contingut.
- Blog: Es una pàgina amb entrades de blog que poden crear els directius del club i que pot visualitzar tothom.
- Registrat: Aquesta seria la secció del navbar més interessant, ja que és la que et permet registrar-te o iniciar sessió i posteriorment accedir a seccions privades de la intranet del club. Un cop has iniciat sessió, el text d'aquesta secció canvia a "Hola, *Usuari*".

Un aspecte interessant és que quan accedeixes a la pàgina des de un dispositiu com un mòbil, que és vertical en lloc de horitzontal, el navbar s'adapta per a que els continguts càpiguen i apareguin de forma distribuïda, gràcies a la llibreria de **Bootstrap**.

Figura 5. El navbar inicialment apareix plegat.

Figura 6. El navbar es desplega al donar-li al botó del llistat.

Pàgina de classificacions:
Per a aquesta pàgina he utilitzat un navbar en conjunt amb una taula per a mostrar les classificacions dels diferents equips. Amb el navbar es pot accedir als diferents grups d'una lliga (els que es juguen l'ascens o els que es juguen mantindre la categoria) i la taula que mostra els equips i els diferents punts, partits jugats i sets a favor o en contra. Des de el navbar pots fer clic a altres grups i es carreguen a la taula els resultats d'aquell grup. Trobo que és una forma molt intuïtiva de mostrar els resultats a més de dinàmica.

Figura 7. Pàgina de Classificacions que mostra els resultats del equip de l'usuari loggejat o els resultats de tots els equips si no estàs loggejat.

Seccions d'usuari:

Les seccions d'usuari es la pàgina a la que pots accedir únicament si has iniciat sessió o t'has registrat de forma correcta. La pàgina et mostra unes seccions o unes altres depenent de quin rol tinguis. És bastant simple ja que només consta d'uns botons estilitzats amb **Bootstrap.**

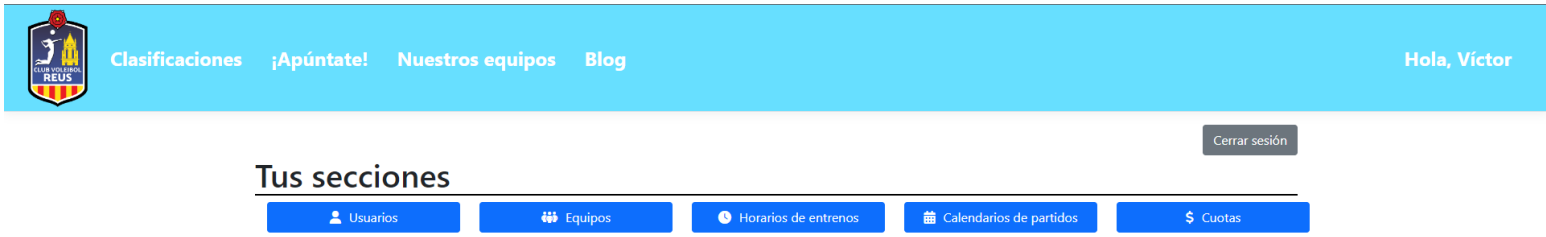


Figura 8. Pàgina de seccions d'usuari.

Secció d'usuaris i d'equips:

Aquestes dues seccions son molt semblants ja que consisteixen en un CRUD[7] on es poden llistar els usuaris o equips, modificar-los, eliminar-los o afegir de nous. Igual que a la pàgina de classificacions he fet ús de una taula amb estils de **Bootstrap.**

[7] Create Read Update Destroy: Sigles molt utilitzades en programación quan s'utilitza una base de dades, volen dir Crear Llegir Actualitzar i Destruir, que son les operacions bàsiques que es pot fer amb dades que es guarden.

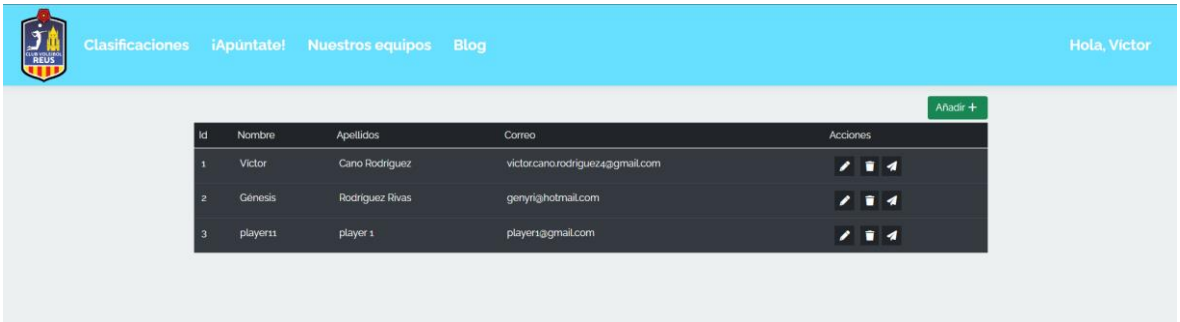


Figura 9. Pàgina d'usuaris.

4.3 Disseny de la base de dades

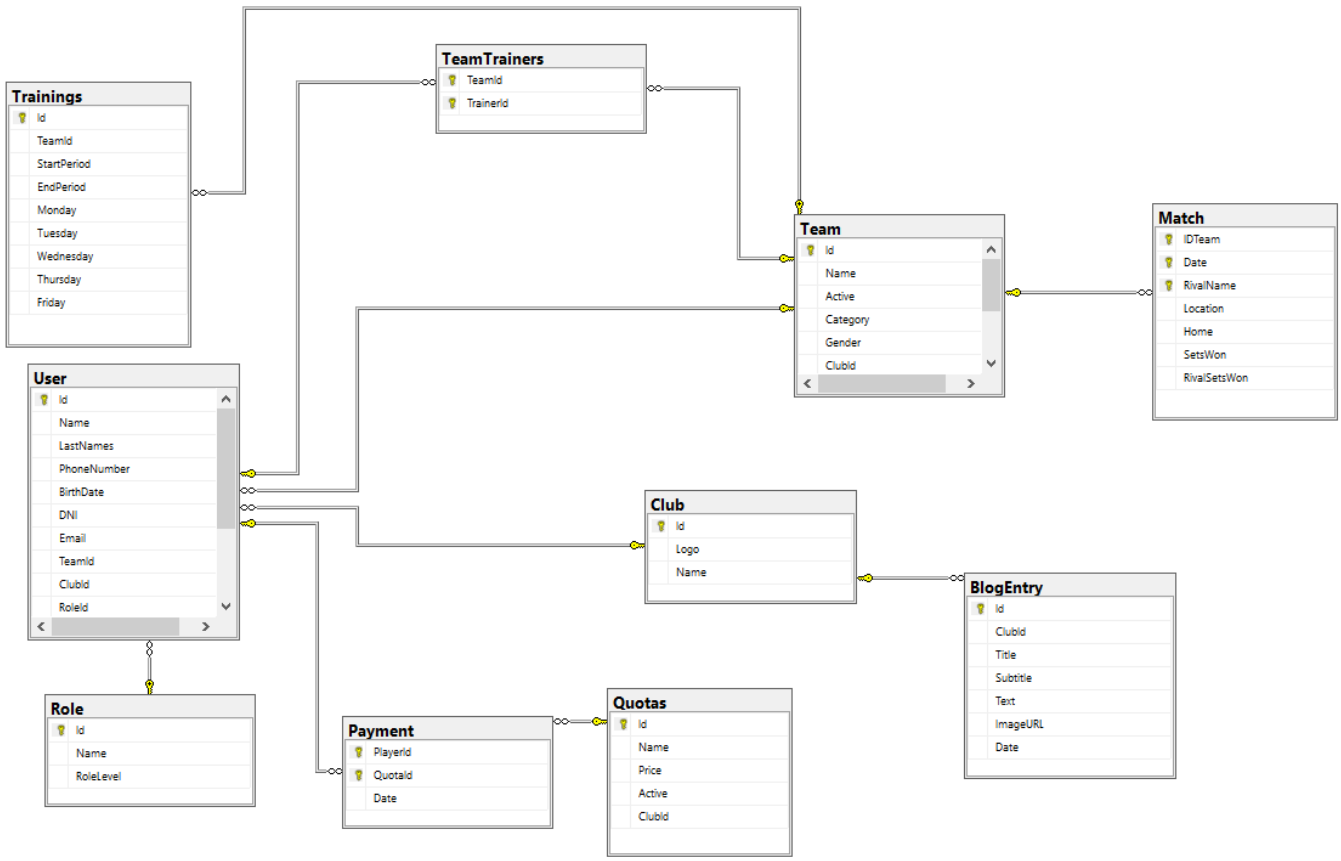


Figura 10. Diagrama de la base de dades

La base de dades es la primera part del projecte que es va fer. Es va començar per aquí perquè en el treball, és un dels aspectes més importants i havia d'estar bastant polit des del principi, si no cada cop que es fes un canvi a la base de dades també s'hauria de modificar el codi.

Vaig començar pensant quines taules necessitaria i a continuació explicaré algunes de les més rellevants:

1. **Club:** La taula club sorgeix perquè si es vol tindre informació rellevant del club guardada s'ha d'emmagatzemar a la base de dades. I tindre dades com: el correu del club, el telèfon, direcció, nom, son aspectes que han d'estar guardats per a facilitar la edició d'aquests des de un únic punt. Conta amb les següents dades:

- Id:És un enter que s'autogenera cada cop que s'afegeix un club. La idea inicial és només tindre un club però ho posat així per si en un futur desenvolupo més el projecte i ho ofereixo a més clubs.
- Logo: És una URL a la imatge del logo del club. S'emmagatzema com a string i al final no he utilitzat aquest camp.
- Nom: Guardo el nom del club també en format string per si s'arribés a canviar.
- PhoneNumber: Número de telèfon de un dels directius, ho guardo en format string encara que es podria guardar com un enter per si algú volgués posar "+34 123456789" o algun format de número de telèfon que no fos directament el número.
- Address: Tot i que no està en ús, inicialment ho vaig posar a la base de dades, és una direcció postal per si calgués guardada en format string.
- Email: Aquets si que és un camp molt important i que utilitzo a diferents llocs. És el correu electrònic del club emmagatzemat en format string.

2. **User:** La taula usuari és una que originalment es tenia pensat fer-la en tres taules, una per a jugadors, un altre per a entrenadors i una per a directius. Es va plantejar que utilitzant herència podria quedar interessant però al final s'ha decantat per utilitzar una única taula per simplicitat i perquè realment es poden estalviar dos taules si s'utilitza una única per a usuaris i s'afegeix una per al rol o directament s'afegeix el rol del usuari dins de la taula. Al final s'ha decidit creat una taula més per als rols perquè pot ser interessant emmagatzemar més informació dins d'aquesta.
   La taula conté els següents camps:

- Id: Enter autogenerat que fa de clau primària de la taula.
- Name: Nom de l'usuari emmagatzemat com a string.
- LastNames: Cognoms de l'usuari emmagatzemats com a string. És opcional.
- PhoneNumber: Número de telèfon de l'usuari emmagatzemat com a string. És opcional.
- BirthDate: Data de naixement de l'usuari emmagatzemada com a tipus "date" de SQL. És opcional.
- DNI[8]: Document Nacional d'Identitat de l'usuari emmagatzemat com a string de tamany màxim 9 caràcters. És opcional.
- Email: Adreça de correu electrònic emmagatzemada com a string. És opcional.
- TeamId: Enter, clau forana a la taula de Teams, concretament al seu Id. La relació és de **User N ---- 0-1 Team (un equip pot tindre múltiples usuaris però un usuari pot tindre només un equip o cap).** És opcional.

[8] Document Nacional d'Identitat: Document que acredita a les persones físiques.

- ClubId: Enter, clau forana a la taula Club necessària per a acedir a les dades del club al que pertany un usuari. La relació és de **User N ---- 1 Club (un club pot tindre múltiples usuaris però un usuari pot tindre només un club).** És opcional.
- RoleId: Enter, clau forana a la taula Role, necessària per comprovar el rol d'un usuari. La relació és de **User N ---- 1-0 Role (un rol pot tindre múltiples usuaris però un usuari pot tindre només un rol).** És opcional.
- Salt: Sal guardada com a string per a la contrasenya d'un usuari, en l'apartat de la implementació s'explica en més profunditat aquest camp.
- Hash: Hash de la contrasenya d'un usuari emmagatzemat com a string, en l'apartat de la implementació explico en més profunditat aquest camp.
- TelegramUser: El nom d'usuari de telegram de l'usuari, aquest camp és opcional i s'emmagatzema com a string. Aquets camp és utilitzat per l'enviament de missatges de telegram i pagaments.
- TelegramChatId: Un enter llarg que es l'Id del chat a la base de dades de Telegram. És necessari emmagatzemar-lo per poder enviar missatges. Igual que el camp anterior, és opcional.

Column Name	Data Type	Allow Nulls
Id	int	☐
Name	nvarchar(25)	☐
LastNames	nvarchar(60)	☑
PhoneNumber	nvarchar(15)	☑
BirthDate	date	☑
DNI	nvarchar(10)	☑
Email	nchar(200)	☑
TeamId	int	☑
ClubId	int	☑
RoleId	int	☑
Salt	nvarchar(30)	☑
Hash	nvarchar(MAX)	☑
TelegramUser	nvarchar(255)	☑
TelegramChatId	bigint	☑
		☐

Figura 11. Taula User visualitzada desde SQL Server Management Studio.

3. **Team:** La taula Team guarda informació de cada equip del club fent distinció de categoria i gènere, és a dir, poden existir el equip Sènior masculí i Sènior femení a l'hora.

   La taula conté els següents camps:

- Id: Enter autogenerat que fa de clau primària per la taula.
- Name: Nom de l'equip, emmagatzemat com a string.
- Category: Nom de la categoria emmagatzemat com a string, vaig plantejar fer una taula només per a les categories però no li vaig veure utilitat.

- Gender: Gènere del equip, emmagatzemat amb un caràcter, pot ser ‘M’, ‘F’ o ‘X’ si es Masculí, Femení o Mixte.
- ClubId: Enter, clau forana a la taula Club. La relació és de **Team N ---- 1 Club (un equip pot tindre un únic club però un club pot tindre múltiples equips).**
- Competition: Competició en la que juga l’equip (federació catalana, consell esportiu del baix camp...) emmagatzemat com a byte per estalviar espai.
- FederationSlug: Camp “auxiliar” per a poder realitzar la connexió amb la API de la federació per obtindre els resultats. És tracta d’un string que especifica la categoria de la competició.
- FederationName: Igual que el camp anterior però per al tipus de competició (volei platja, pista...).

5. Implementació

La implementació del projecte ha suposat utilitzar entorns variats i entrar en contacte amb diversos temes dels que no tenia cap experiència prèvia. Aquestes àrees son: registre d'usuaris, emmagatzemant segur de contrasenyes a la base de dades, publicació de l'aplicació web, creació d'un servidor web a una màquina virtual i connexió amb altres APIs web. M'agradaria aprofundir sobre cadascun d'aquests temes individualment i de les peculiaritats de la seva implementació.

5.1 Seguritat web

En aquest apartat es parla sobre el registre d'usuaris i el emmagatzemament de les seves contrasenyes com he mencionat abans.

Quan vaig començar amb el projecte no sabia com es gestionaven les sessions dels usuaris quan entres a qualsevol pàgina web. Un dels objectius del treball era aprendre com funciona i implementar-lo d'alguna manera al meu projecte tot i que no fos extremadament segur, em conformava amb una implementació que no exposés les contrasenyes dels usuaris i un sistema de sessions prou segur.

Tot i que s'ha fet servir alguna llibreria per a les sessions la funcionalitat principal la he programat jo.

Emmagatzemament segur de contrasenyes d'usuaris:

Tot i que no sembla un tema destacable trobo que està bé explicar com s'ha he fet perquè trobo interessant la part com s'ha fet.

Quan un usuari es vol registrar a la web, se li demana tres camps: email, nom i contrasenya. I quan s'envia el formulari de registre primer que tot es revisa que no hi hagi un altre usuari amb el mateix email, en aquest cas s'indica a la pantalla que el email ja està en ús, en cas que sigui un email nou es genera un "Salt" que és una cadena de caràcters aleatòria que serveix per a fer més difícil esbrinar les contrasenyes en un atac de força bruta. Un cop generada aquesta cadena aleatòria, s'assigna al nou usuari per a guardar-la en la base de dades. Després s'utilitza la llibreria System.Security.Cryptography de ASP.NET Core per a generar la contrasenya, concretament utilitzo la funció **Rfc2898DeriveBytes** que pren per paràmetre un "Salt" una contrasenya i un número d'iteracions per a obtindré la contrasenya. És interessant consultar el RFC[9] 2898 perquè documenta el procés en el que es basa aquesta funció.

Aquesta funció retorna un objecte de tipus **Rfc2898DeriveBytes**, que també forma part de la llibreria però aquesta classe implementa la funció GetBytes() amb la que puc agafar finalment el hash (que ara consta de la salt i la contrasenya) i guardar-lo a la base de dades de forma segura (convertint els bytes a un string codificat en dígits base-64).

També es interessant explicar com s'ha implementat la funció de verificar si un usuari ha introduït correctament la seva contrasenya:

Quan un usuari fa un intent de login es mira si existeix un usuari amb el email introduït i si existeix s'agafa la seva salt i el seu hash. Es converteixen els dos en arrays de bytes

[9] Request For Comments: Publicacions que busquen posar en comú aspectes de la enginyeria de internet.

ja que s'havien guardat com a strings codificats. Es torna a utilitzar la funció **Rfc2898DeriveBytes** com al fer el registre amb la sal guardada, la contrasenya introduïda i les mateixes iteracions que al registre. Si el resultat d'aquesta funció és el mateix hash que es tenia guardat és que la contrasenya que ha introduït l'usuari és correcta i per tant es pot crear una sessió per l'usuari.

Implementació de sessions

Per al projecte volia que al navegar per les diferents pàgines es pogués saber si el usuari navegant ha iniciat sessió o no. Per fer això el meu tutor em va suggerir que utilitzes un Json Web Token, a partir d'ara referits com a JWT. Els JWT son un manera estandarditzada basada en objectes amb la notació JavaScript que permeten la propagació de permisos i privilegis. Es poden fer servir com una cookie, guardant-los en el disc de la persona que accedeix a un servidor web o guardant-los en la sessió HTTP. Totes dos implementacions tenen unes avantatges i uns inconvenients:

Guardar el JWT en una cookie

+ El navegador envia automàticament el JWT al servidor.
+ El mateix token pot ser utilitzat desde diferents pestanyes del navegador.
- Aquesta implementació és vulnerable a atacs XSS o Cross-Site Scripting[10] podent robar la cookie si no s'implementa amb la prou seguretat.

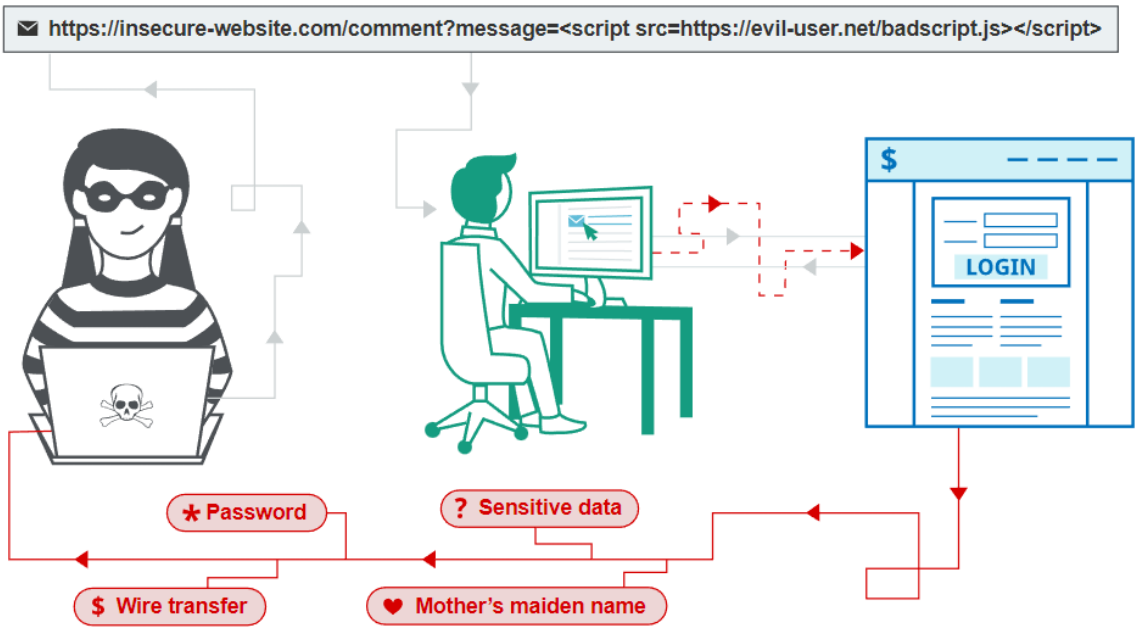


Figura 12. Esquema d'un atac XSS. Font: https://portswigger.net/web-security/cross-site-scripting

[10] AtacXSS o Cross-Site Scripting: Es tracta d'un atc en el que un atacant injecta scripts maliciosos en una URL del lloc web i ho envia a usuaris que si accedeixen desde aquesta URL a la página poden posar en vulnerabilitat les cookies o altres dades.

Guardar el JWT al disc del client

+ Invulnerable a atacs CRSF[11].
+ El mateix token pot ser utilitzat desde diferents pestanyes del navegador.
- Vulnerable a atacs XSS.
- S'ha d'implementar un mecanisme per a enviar el token.

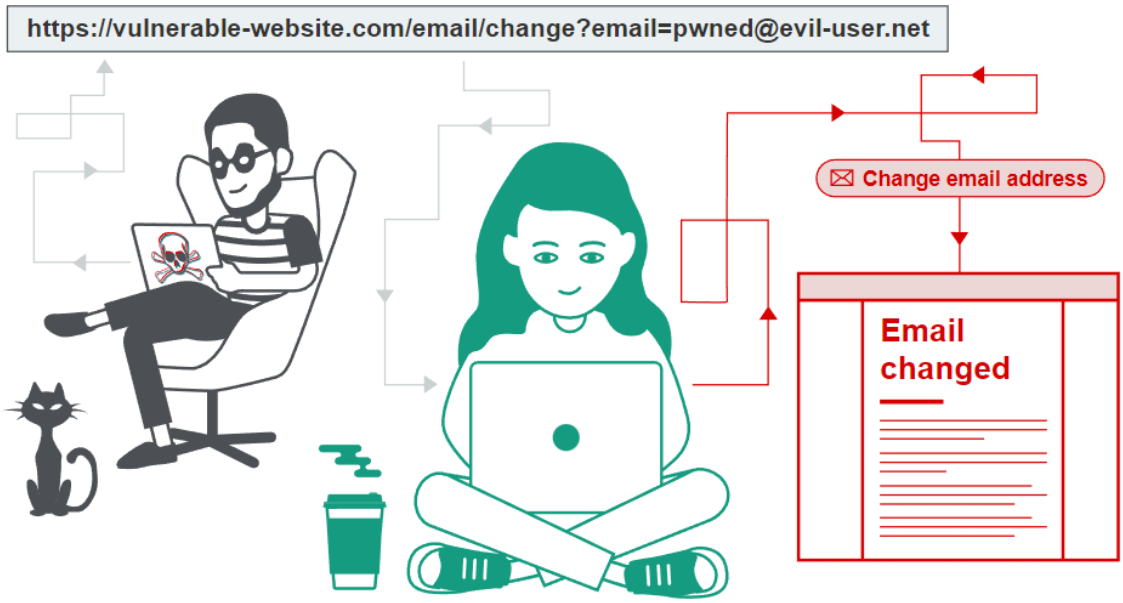


Figura 13. Esquema d'un atac CRSF. Font: https://portswigger.net/web-security/csrf

Guardar el JWT en la sessió HTTP

+ Invulnerable al atac CRSF.
+ Fa molt fàcil realitzar múltiples logins des de un mateix navegador.
- Vulnerable a atacas XSS.
- Al tancar la pestanya es perd tota la informació guardada a la sessió HTTP.

D'aquestes tres implementacions s'ha escollit la de guardar el JWT en una cookie perquè em va semblar més fàcil d'implementar i perquè ho vaig trobar interessant.
La implementació d'aquesta part del treball, a diferencia de la part anterior, depèn molt de la llibreria **System.IdentityModel.Tokens.Jwt,** ja que utilitzo la funció **CreateJwtSecurityToken** d'aquesta per a generar el JWT final. La part que si que m'agradaria comentar és la funció **ValidateUser** que he creat per a facilitar-me el desenvolupament del projecte i que ha resultat molt útil.
Era interessant pel projecte crear una funció que es podés cridar des de qualsevol controlador i amb la que es pogués verificar si hi ha un usuari amb una sessió iniciada

[11] Atac CRSF: CSRF vol dir Cross Site Request Forgery i es un tipus d'atac en el que el atacant es fa amb el navegador de la víctima i executa peticions HTTP (com pot ser una petició a una aplicación web per canviar la contrasenya) i aprofitant les sessions actives es fa amb la conta d'un

idealment sense necessitar de passar paràmetres per a mantindre el codi net i no haver de crear variables contínuament.

Es va arribar a una solució que complia aquest requisits:

```
static public User ValidateUser(HttpRequest request)
{
  using (VoleiReusContext ctx = new VoleiReusContext())
  {
    try
    {
      if (request.Cookies.Count < 1) return null;
      if (request.Cookies.Where(x => x.Key == "access_token").Count() ==
0)
        return null;
      string token =
            request.Cookies.FirstOrDefault(x => x.Key ==
      "access_token").Value;
      var tokenHandler = new JwtSecurityTokenHandler();
      TokenValidationParameters tvp = new TokenValidationParameters();
      tokenHandler.ValidateToken(token, validationParameters, out
          SecurityToken st);
      JwtSecurityToken jwt = (JwtSecurityToken)st;
      if (jwt.Payload.Count > 0)
      {
        string userEmail = jwt.Payload.First().Value.ToString();
        User user = ctx.Users.Include(x => x.Role).Include(x =>
            x.Team).FirstOrDefault(x => x.Email == userEmail);
        if (user != null)
        {
          return user;
        }

      }
      catch (Exception e)
    {

    }
    return null;
  }
}
```

Codi 1. Funció ValidateUser, la qual s'utilitza cada cop que es vol veure si el usuari està autenticat a un controlador.

La funció requereix un paràmetre "request" que és del tipus **HttpRequest** i es un objecte al que es pot accedir des de qualsevol controlador fent **HttpContext.Request** ja que **HttpContext** forma part de la interfaç ControllerBase, que implementen els controladors que he creat. Llavors simplement cridant la funció **ValidateUser(HttpContext.Request)** es pot obtenir l'objecte d'un usuari o **null** si no hi ha cap usuari loggejat, lo qual s'ha acabat utilitzant molt sovint en diversos controladors.Ç

5.2 Connexió amb altres APIs web

També s'ha dedicat una part del projecte a connectar la meva aplicació web amb dos APIs completament diferents. La primera es tracta de la API de la federació per a obtindre els resultats dels partits i les classificacions de la lliga, he utilitzat aquesta per a

obtindré i mostrar els resultats dels equips del Club Voleibol Reus a la meva web sense tindre que anar a una altra pàgina. L'altre API amb la que s'ha connectat la meva és la API dels bots de Telegram. Trobo que aquesta és molt més interessant perquè mentre que a la de la federació només faig un GET amb diferents paràmetres i després transformo les dades que em retorna, amb la API de Telegram he pogut crear un bot del club que envia missatges als usuaris i fins i tot permet realitzar pagaments (tot i que m'ha faltat desenvolupar més aquesta part).

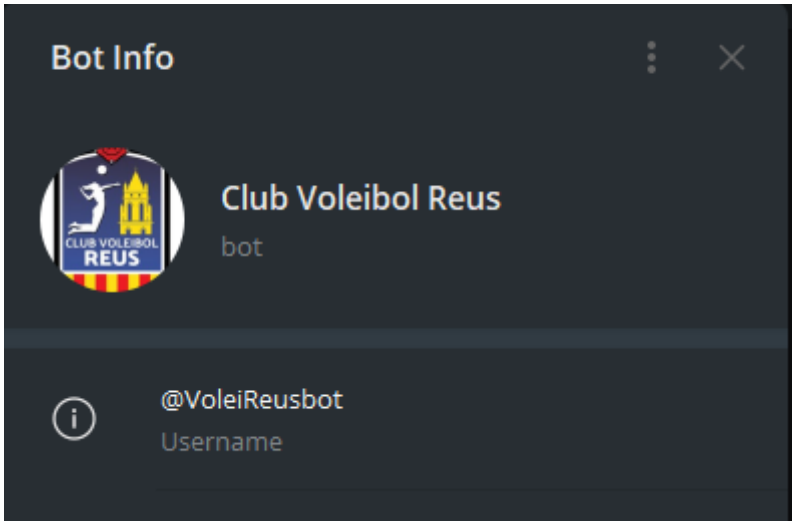


Figura 14. Informació relativa al bot de Telegram del club que he creat.

Connexió amb la API de la federació

Abans de començar el projecte es tenia clar que volia mostrar els resultats i classificacions dels diferents equips del club a la pàgina, però no tenia molt clar com fer-ho. Vaig investigar formes d'aconseguir-ho i la primera opció que vaig considerar era fer Web Scraping. El Web Scraping és una tècnica per a escanejar pàgines web i extreure la informació desitjada. Vaig trobar que seria un afegit interessant per al projecte fer això però també sabia que probablement m'acabaria portant bastant de temps realitzar-ho correctament.

Llavors és quan em va sorgir una idea, si la pàgina web de la federació on es mostren les classificacions té implementada una API Rest per a obtindré els resultats o té una implementació similar vol dir que la informació es carrega fent una petició a una API i llavors col·locant adientment aquets resultats a la pàgina. També podria estar implementada de forma que el renderitzat de la pàgina es fa al servidor en lloc de al client (utilitzant per exemple php o Razor amb C#). Si d'aquestes dues formes d'implementar la pàgina dels resultats ho fan de la primera manera que he mencionat podria provar a fer peticions jo mateix a la seva API i mostrar els resultats a la meva pàgina.

Afortunadament, vaig fer una prova i la pàgina estava implementada fent crides a una API seva. Ara calia provar si aquesta API seva permetia crides a les seves funcions des de altres APIs o només des de els seus servidors web.

Per a realitzar aquesta prova vaig utilitzar Google Chrome i vaig accedir a una pàgina de resultats qualsevol (concretament aquesta https://fcvolei.cat/voleibol/2A-DIVISIO-SENIOR-MASCULINA-2122/) i vaig obrir les eines per a desenvolupadors de Chrome

i vaig entrar a la secció **Network** aquí es mostren les peticions que es fan i les respostes que donen. Vaig verificar que hi havia una petició de les classificacions i resultats i que la resposta era un JSON que jo després podria manipular a voluntat. Fixant-me en les capçaleres de la resposta HTTP em vaig adonar de que la capçalera **access-control-allow-origin** tenia el valor * volent dir que qualsevol pot fer crides a aquesta API.

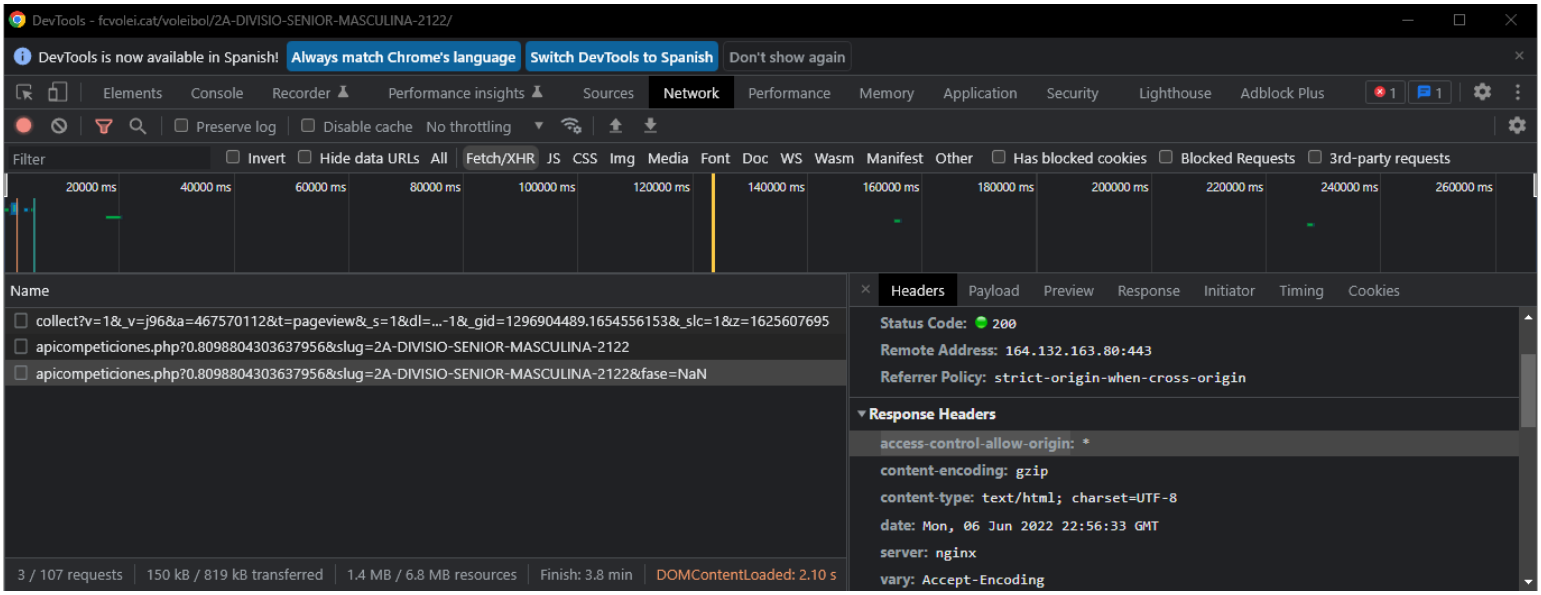


Figura 15. Les eines de desenvolupador de Google Chrome mostrant les capçaleres de la resposta a una crida a la API de la federació.

Tot i haver fet aquesta prova, abans de començar a programar vaig fer un petit test per veure que tot això que havia pensat funcionava realment i vaig utilitzar Postman amb la URL que apareix en les eines de Chrome, vaig fer la crida i vaig obtindre els resultats, concluïnt que podia fer crides a aquesta API des de la meva aplicació web.
La resta de la programació va ser simplement fer una crida AJAX[12] canviant el paràmetre de la competició i la categoria depenent de l'equip del que es vol mostrar resultats.

Connexió amb la API de Telegram Bots

El meu tutor del treball em va suggerir que podria utilitzar un bot de Telegram per a avisar cada setmana als jugadors dels partits que tenen aquella setmana. Vaig començar a investigar i vaig trobar que la documentació de la API és bastant complerta i no semblava massa difícil realitzar la connexió. Lo primer que tot que vaig tindre que fer és crear un bot a Telegram, és una tasca extremadament senzilla ja que es pot fer des de la mateixa aplicació de Telegram.

[12] Asynchronous JavaScript And XML: És un conjunt de tecnologies que serveix per fer crides HTTP sense tindre que tornar a carregar la página un altre cop.

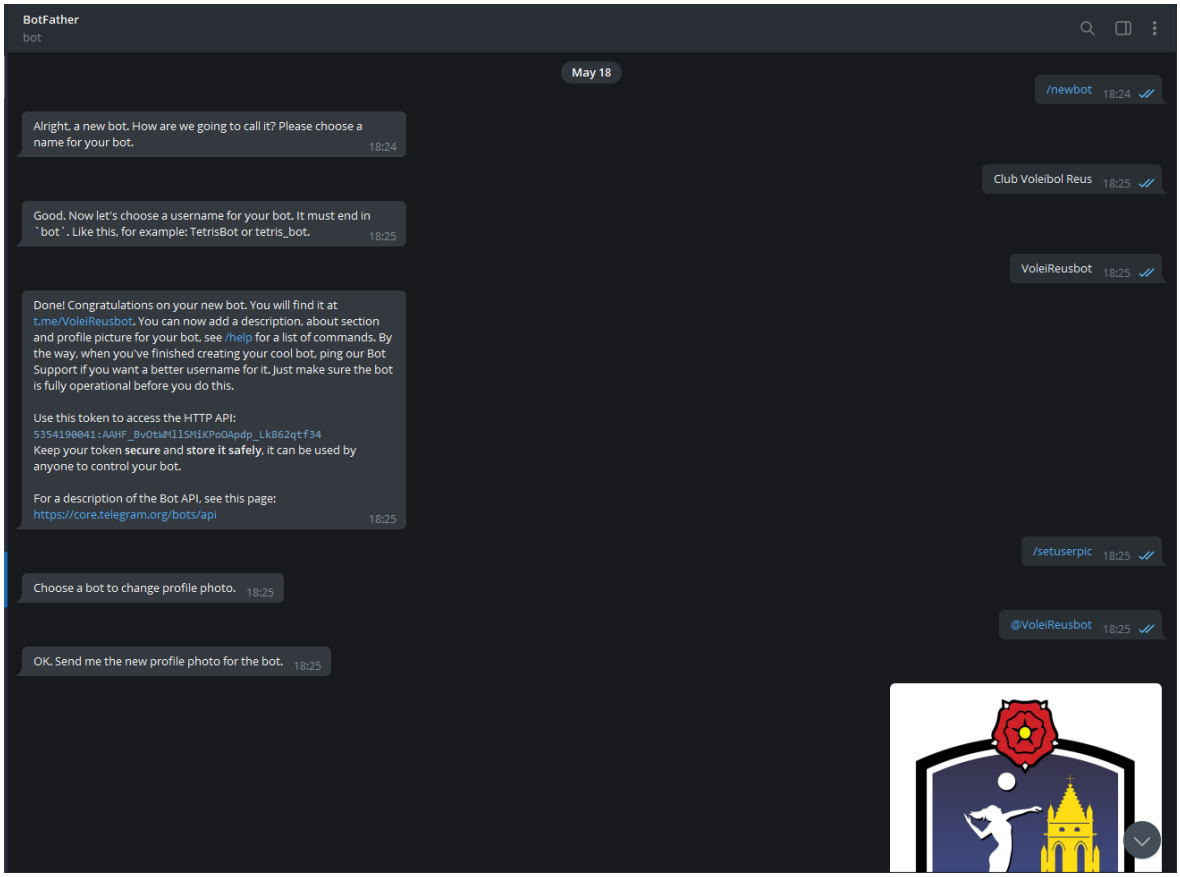


Figura 16. Procés per a crear un bot de Telegram.

A la Figura 16 es pot veure com el BotFather de Telegram (que és un bot per a crear bots) m'envia un token d'acces a la API per al bot que he creat. Aquest token el faig servir en totes les crides per autenticar el meu bot.

Per a fer la programació en ASP.NET Core he fet servir una lliberia anomenada **Telegram.Bot** la qual només utilitzo per a no tindre que crear els models dels objectes que retorna Telegram.

La primera funció que volia implementar era la de que el bot enviés missatges als usuaris, però em vaig trobar que per a fer servir aquesta funció, necessitava tenir implementada una abans: la de rebre actualitzacions.

GetUpdates

Aquest és un mètode que et retorna actualitzacions dels missatges que ha rebut el teu bot en les últimes 24 hores. Aquesta funció em resulta útil perquè no només et retorna els missatges, si no que a més et diu el id del xat amb els usuaris que han intercanviat missatges. Això es útil ja que posteriorment necessitaré saber aquest id de xat per a poder enviar missatges a usuaris.

Utilitzo aquesta funció en una tasca programada que executo cada 24 hores per veure quins usuaris han interactuat amb el bot i si ni han de nous, guardar els seus ids de xat a la base de dades per poder enviar-lis missatges en el futur. Per fer això, però, l'usuari ha d'haver configurar a l'aplicació web del club el seu usuari de Telegram, perquè si no no podré associar l'usuari de Telegram amb l'usuari de l'aplicació web.

La tasca programada funciona ja que s'han implementat les funcions en una classe que implementa la interfície IHostedService que està especialment pensada per a ser utilitzada amb tasques en segon pla. Aquesta interfície proporciona les següents funcions: **StartAsync** i **StopAsync**. Amb la funció **StartAsync** executo la tasca i inicio un timer cada 24 hores que es repeteix de forma periòdica. El temporitzador l'he fet servir amb la classe Timer de l'assemblat System.Timers.

SendMessage

Aquest mètode és el que permet enviar missatges a usuaris, accepta dos paràmetres obligatoris i uns altres d'opcionals però jo només utilitzo els obligatoris. Aquests dos paràmetres son el id del xat i el text. Com he mencionat abans, el id del xat només el puc obtenir fent prèviament un GetUpdates i el text puc enviar el que vulgui.

Per tant, aquest mètode requereix que s'hagi executat prèviament el GetUpdates i que hi hagi usuaris a la base de dades amb el id del xat de Telegram.

He utilitzat el procediment en una tasca programada per a que s'executi setmanalment i revisi per a cada equip, quins partits té per aquella setmana i per a que notifiqui a cada jugador de l'equip dels partits que té si tenen configurat el usuari de Telegram i el id del xat configurats a l'aplicació web.

Crec que aquesta funcionalitat del projecte destaca bastant ja que és una cosa realment útils per a un club i que pot resultar molt útils per als jugadors i entrenadors.

SendInvoice

Aquest mètode permet enviar una petició de pagament a un usuari, he aprofitat això per a implementar la funcionalitat de pagar les quotes del club.

Per poder utilitzar aquest mètode he necessitat fer configuracions addicionals, per començar havia de connectar un proveïdor de pagament al meu bot. Alguns dels diferents proveïdors disponibles són Stripe, Smart Glocal, Payme, Paymega.. D'aquests només coneixia Stripe, i només em sonava, tampoc l'havia fet servir mai, vaig crear una conta de Stripe i vaig activar el mode de test per a poder fer proves. Abans de poder fer les proves però, vaig tindre que connectar el meu bot amb la conta que acabava de crear en Stripe, però també va ser un procés senzill, semblant al de crear el bot.

Després d'haver fet aquesta configuració, puc utilitzar la funció per a fer una sol·licitud de pagament, però la he de proveir de 7 paràmetres obligatoris:

- Chat_id: Id del chat on es vol enviar el Invoice, és el mateix id que s'utilitza a sendMessage.
- Title: Títol per al Invoice.
- Description: Decripció del Invocie.
- Payload: Camp per a utilitzar en processos interns.
- Provider_token: Token de pagaments proporcionat pel BotFather de Telegram.
- Currency: String amb la divisa del pagament en ISO 4217.
- Prices: JSON serialitzat que conté el llistat de preus.

Tot i que he posat en funcionament aquest mètode, no ho he implementat d'una forma que serveixi d'utilitat perquè si em centrés més en aquest punt em trauria massa temps del projecte, per tant el que he fet és afegir un botó a la pàgina de gestió dels usuaris per a poder enviar un Invoice de prova amb un preu de 5€.

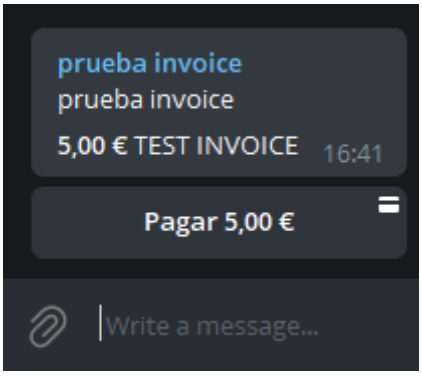


Figura 17. Invoice de prova que s'envia desde la pàgina de gestió d'usuaris.

He pogut fer proves del pagament gràcies a que Stripe permet fer proves sense gastar diners, simplement posant una targeta de crèdit amb unes dades en concret.

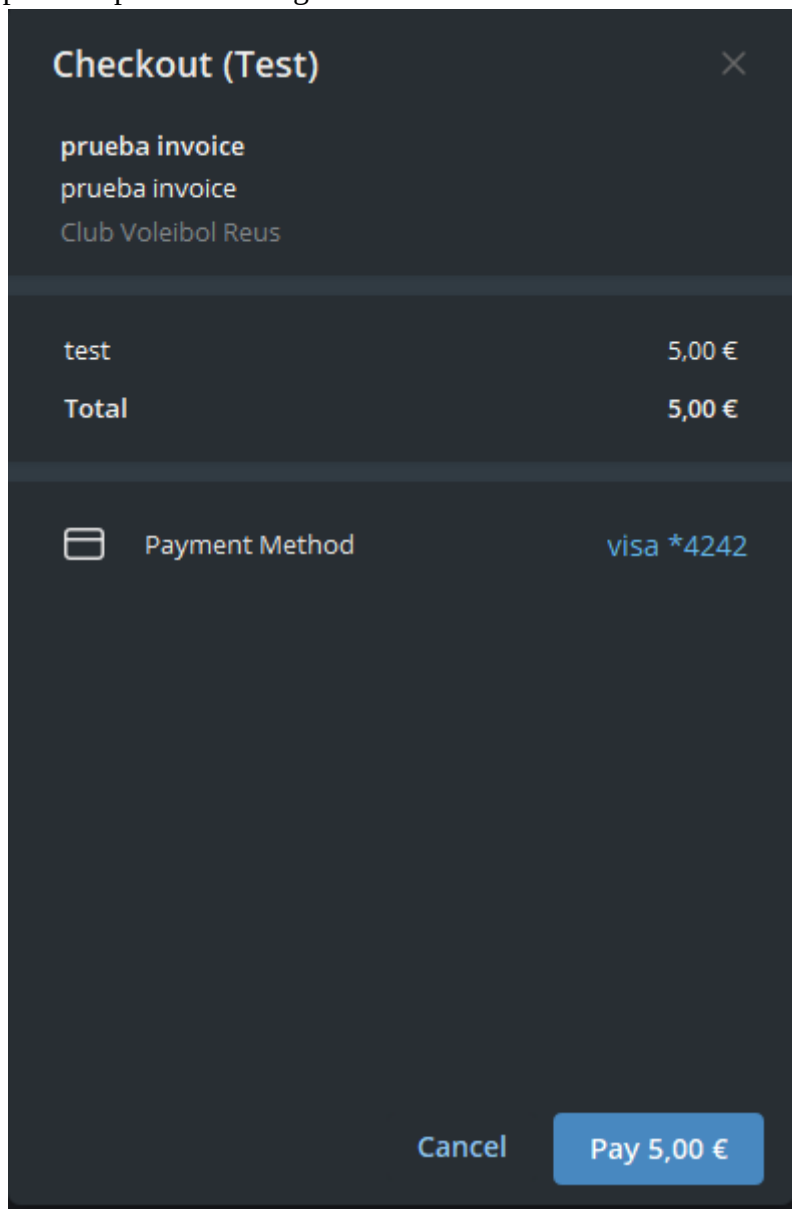


Figura 18. Detall del invoice quan vas a pagar.

Un cop li dones al botó de pagar de la figura 18 és necessita enviar un altra petició a la API de Telegram per a confirmar la rebuda del pagament.

answerPreCheckoutQuery

Es tracta de l'últim mètode que utilitzo per a integrar pagaments amb la meva aplicació web, és tracta d'una simple petició POST per indicar si s'accepta un pagament o no. Els paràmetres que accepta son:

- pre_checkout_query_id: Id de la query de pre checkout, s'obté fent la crida **getUpdates**.
- Ok: True o false segons si s'accepta o no el pagament.

Prèviament vaig mencionar que vaig crear una tasca programada per a que es fes una crida a **getUpdates** cada 24 hores per a registrar als usuaris, doncs bé, després de fer una crida d'enviar pagament, he fet que la mateixa funció que envia el pagament, esperi durant 15 minuts a que es realitzi el pagament, fent una crida a **getUpdates** cada 3 segons per veure si s'ha realitzat ja el pagament. Si durant aquest temps es rep resposta per al invoice enviat llavors s'envia un **answerPreCheckoutQuery** ara que ja coneixo el **pre_checkout_query_id** de la operació.

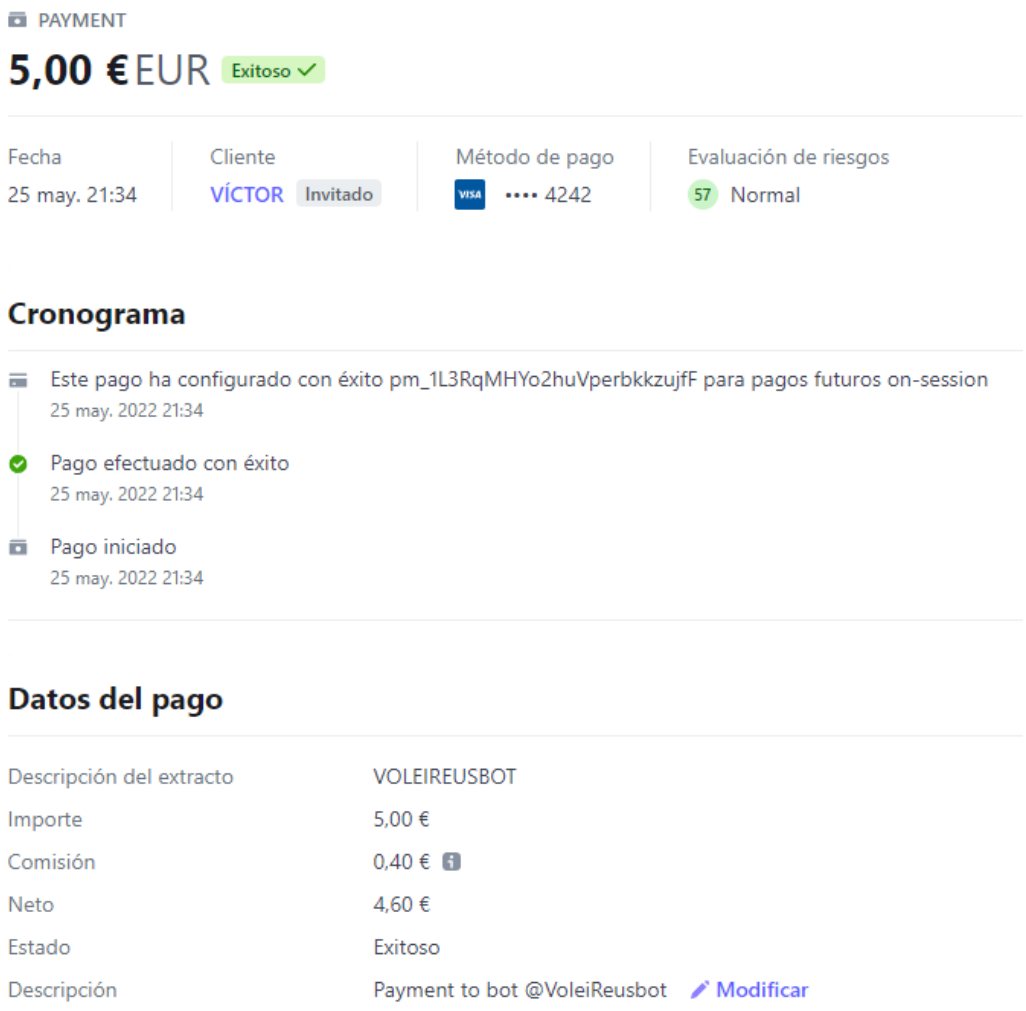


Figura 19. Dades del pagament rebut a Stripe.

5.3 Publicació de l'aplicació web

Posar en producció el meu projecte ha sigut per mi un dels punts més forts i interessants del treball, ja que és la part de la que crec que més he après i es un apartat imprescindible en el desenvolupament d'aplicacions web professionals. Ja he mencionat a l'apartat de Arquitectura de l'aplicació per damunt quines eines he utilitzat per a publicar el meu treball però m'agradaria entrar en detall d'algunes seccions:

Servidor DDNS amb Dynu

Per a poder tindre un domini vaig utilitzar la pàgina https://www.dynu.com/ per a poder tindre un domini gratuït. La pàgina et permet associar una ip pública a un domini del qual pot escollir el nom però has d'utilitzar alguns dels Top Level que et diuen i en el meu cas he escollit el "giize.com" per la seva brevetat. És per aquest motiu que a totes les pàgines de la meva aplicació web s'accedeix amb "clubvoleibolreus.giize.com".

Servidor nginx a la meva màquina virtual

També vaig mencionar que per a escoltar les peticions que es fan sobre el domini del club he utilitzat un servidor nginx, que és una eina molt potent que fan servir molts servidors web per a escoltar les peticions des de dispositius remots. En concret, el que he vaig fer es instal·lar-lo, configurar-ho editant el fitxer **/etc/nginx/sites-enabled/default** i engegar el servei.

Les configuracions que vaig haver de fer son: establir el domini on vull que escolti el nginx, indicar el port on vull que escolti (el port HTTP, el 80), indicar un **proxy_pass** que es on es redirigiran les peticions (vaig posar http://localhost:5001 perquè es on s'executa el meu servidor web en Release), i indicar informació addicional sobre les capçaleres HTTP.

```
victor@victor-VirtualBox:/var/www/clubvoleibolreus$ cat /etc/nginx/sites-enabled
/default
server {
        listen 80;

        server_name     clubvoleibolreus.giize.com;

        location / {
            proxy_pass  http://127.0.0.1:5001;
            proxy_http_version  1.1;
            proxy_set_header   Upgrade $http_upgrade;
            proxy_set_header   Connection "";
           # proxy_set_header    Host $host;
            proxy_cache_bypass $http_upgrade;
            proxy_set_header   X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header   X-Forwarded-Proto http;
            proxy_buffering    off;
        }
}
```

Figura 20. Fitxer de configuració de l'nginx per a escoltar peticions al domini de l'aplicació web del club.

6. Avaluació

Per a provar que tots els elements de l'aplicació web funcionen, encara que vaig anar fent proves a mesura que desenvolupava noves parts del projecte, vull fer una prova final per comprovar que tot funciona i dona els resultats esperats. Per a fer això he decidit fer una petita "simulació" d'un usuari final utilitzant la web:

Accés a la web des de dispositius remots

Quan un usuari entra a la pàgina web no hauria de tindre cap problema per a accedir encara que no hi sigui a la xarxa privada on tinc muntat el servidor. Per a realitzar la prova he utilitzat el meu telèfon mòbil desconnectat del wifi i ha funcionat perfectament, retornant la pàgina amb rapidesa. He aprofitat que he accedit amb un mòbil per a provar també que la interfície gràfica sigui responsiva i s'adapti a la resolució vertical del meu mòbil.

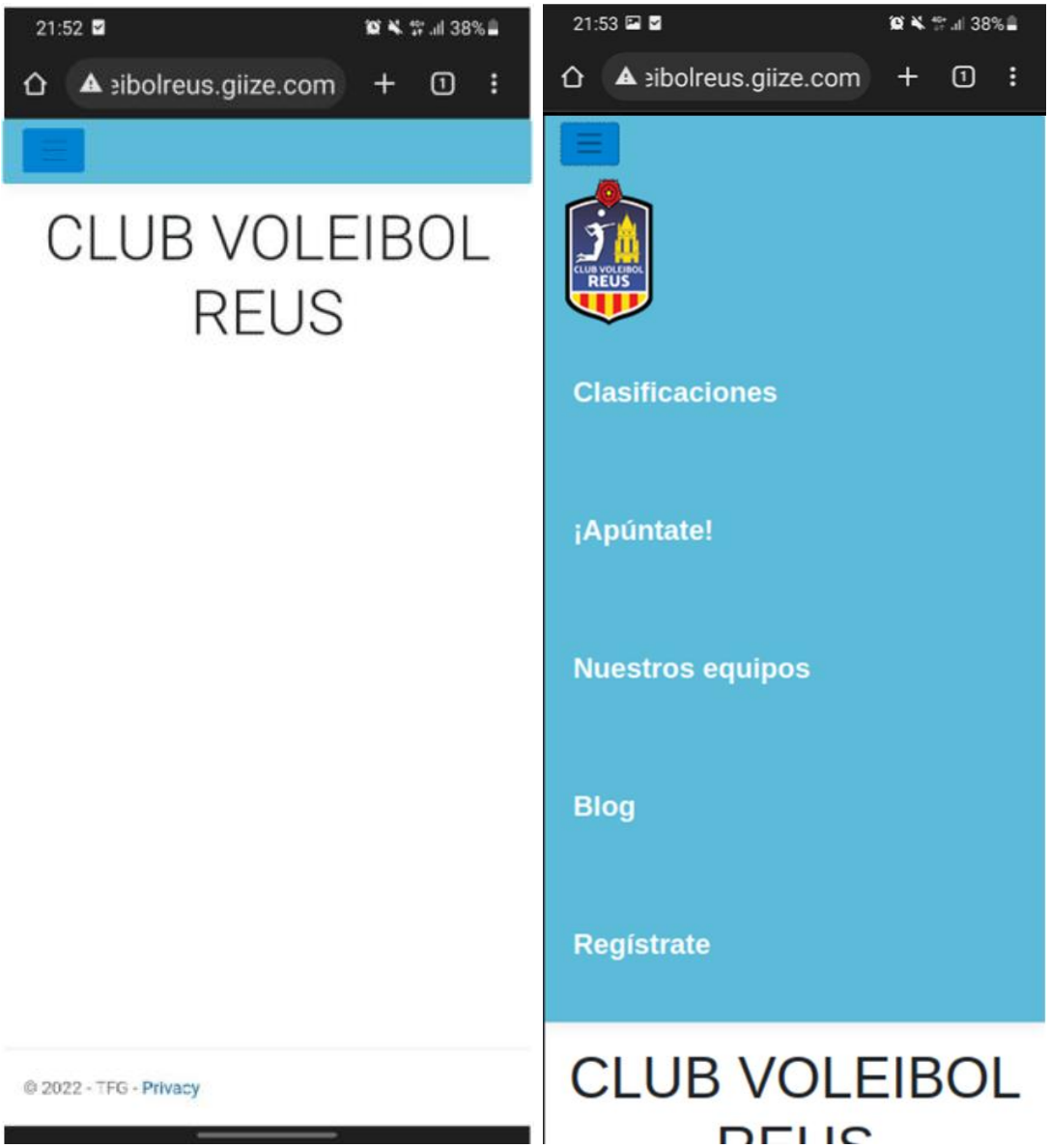
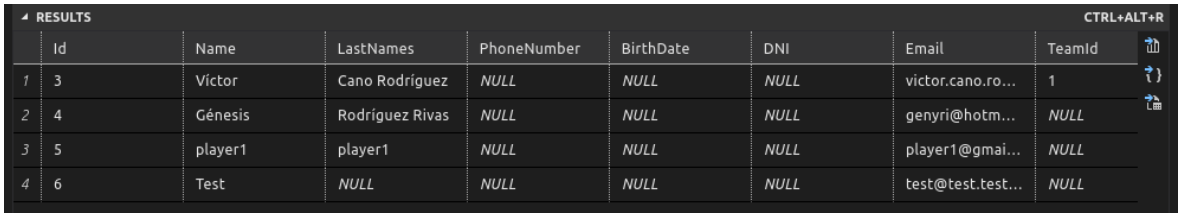


Figura 21 i 22. Accedint a l'aplicació web des de un dispositiu fora de la xarxa privada.

Registre de un nou usuari des de aquest dispositiu

Ara provaré el sistema de registrar usuaris, guardar-los a la base de dades i crear una sessió. Continuant des de el punt anterior, tinc oberta la pàgina al meu mòbil i he omplert el formulari per registrar-me. Ara puc comprovar si l'usuari s'ha creat mirant el contingut de la base de dades. Fent un simple select he vist que l'usuari s'ha afegit.



RESULTS CTRL+ALT+R

	Id	Name	LastNames	PhoneNumber	BirthDate	DNI	Email	TeamId
1	3	Víctor	Cano Rodríguez	*NULL*	*NULL*	*NULL*	victor.cano.ro…	1
2	4	Génesis	Rodríguez Rivas	*NULL*	*NULL*	*NULL*	genyri@hotm…	*NULL*
3	5	player1	player1	*NULL*	*NULL*	*NULL*	player1@gmai…	*NULL*
4	6	Test	*NULL*	*NULL*	*NULL*	*NULL*	test@test.test…	*NULL*

Figura 23. Usuaris a la base de dades després de la nova inserció.

Accés a seccions de la intranet

Per comprovar que el sistema de rols funciona correctament he iniciat sessió des de la meva conta amb rol de directiu per canviar el rol del nou usuari a directiu per a que pugui accedir a la major part de seccions de l'aplicació. Posteriorment he iniciat sessió un altre cop amb el nou usuari creat i he vist que el rol s'ha actualitzat correctament i em mostra les seccions visibles només per a directius.

Proves de la API

Seguint des de el punt anterior, he provat a fer les accions bàsiques sobre el llistat d'usuaris (llegir, crear, editar i eliminar) i he comprovat que les quatre funcionen correctament.

7. Conclusió

El resultat final del treball opino que demostra que he sabut implementar totes les fases del desenvolupament d'un projecte de programació web: Front-end, Back-end, Dev Ops i administració de bases de dades. Tot i que al projecte encara li falten moltes coses per implementar i polir per a que es pugui desplegar per al Club Voleibol Reus, crec que com a Treball de fi de grau ha quedat força bé.

He après molt sobretot sobre la part de publicar l'aplicació, ja que mai havia fet res similar i a més, no ha sigut tan senzill com a seguir un tutorial, si no que m'he basat en diferents vídeos, guies i documents per a fer-ho i m'han anat sorgint imprevistos que he tingut que resoldre i justificar com ho he fet. Concretament recordo que no contava amb cap documentació sobre com redirigir peticions remotes a una màquina virtual que s'estava executant en el meu ordinador, i d'aquesta manera vaig adquirir coneixements indirectament sobre com funciona la NAT o un dispositiu pont a les VirtualBox. Similarment, també vaig tindre molts problemes a l'hora d'instal·lar SQL Server a la màquina virtual i crec que la solució a la que vaig arribar, que es tracta en utilitzar un docker dins de la màquina virtual és una molt bona i amb la que també he adquirit coneixements indirectament.
I no només he après coses de la part de sistemes del projecte, com tampoc tenia experiència amb el framework ASP.NET Core he conegut de primera mà les diferencies que té amb ASP.NET i m'he trobat amb totes les peculiaritats que té i m'he adaptat per a fer-lo servir.

He aconseguit complir tots els requisits funcionals i fins i tot alguns de no funcionals, a més molts dels no funcionals encara que no els hagi implementat he creat una estructura amb el projecte per a poder afegir-los sense massa complicació.

Finalment, opino que aquest projecte ha sigut molt productiu per a mi per totes les coses que he après i perquè és un projecte que puc seguir desenvolupant de forma externa a aquesta assignatura per a tindre un projecte personal

Bibliografia

[1] RFC 2898: https://datatracker.ietf.org/doc/html/rfc2898

[2] Pàgina web https://es.wikipedia.org/wiki/JSON_Web_Token

[3] Pàgina web https://javascript.plainenglish.io/where-to-store-the-json-web-token-jwt-4f76abcd4577

[4] Pàgina web i imatge https://portswigger.net/web-security/csrf

[5] Pàgina web i imatge https://portswigger.net/web-security/cross-site-scripting

[6] Pàgina web https://www.ibm.com/docs/es/rational-soft-arch/9.6.1?topic=page-asynchronous-javascript-xml-ajax-overview

[7] Pàgina web https://core.telegram.org/bots/api

[8] Pàgina web https://core.telegram.org/bots/payments

[9] Pàgina web https://www.dynu.com/

[10] Pàgina web https://docs.microsoft.com/es-es/sql/linux/quickstart-install-connect-docker?view=sql-server-ver16&pivots=cs1-bash

[11] Vídeo https://www.youtube.com/watch?v=lIcVl16Cin8

[12] Pàgina web https://docs.docker.com/engine/install/ubuntu/