Stockholms
Tekniska Institut

DotaProject, ett verktyg för att hämta information

Av Eric Norrwing

Examensarbete

Stockholms Tekniska Institut

Handledare: Kristoffer

Datum: 2024-12-27

Sammanfattning

Denna rapport innehåller beskrivning och tillvägagångsätt för uppbyggnaden av Dotaproject, en app för att scrapa information från Liquipedia till en databas.

Innehållsförteckning

1. Om projektet

1.1 Mål

Målet med Dotaprojekt var och förblir enkelt, ett verktyg att samla in data från en plats som inte tillandahåller ett öppet API för individer.
Det huvudsakliga målet var att så kallat “Scrapa” hemsidan (Liquipedia.net) för information gällande professionella esportspelare och lag och få in informationen till en lätttillgänglig databas.

1.2 Tekniker och verktyg

En snabb lista av Tekniker som projektet använt:
.NET SDK 8.0
.ASPNET CORE 4.8.1
.NET ENTITY-CORE-FRAMEWORK
Microsoft SQL och SSMS
Docker

Rider IDE
Postman

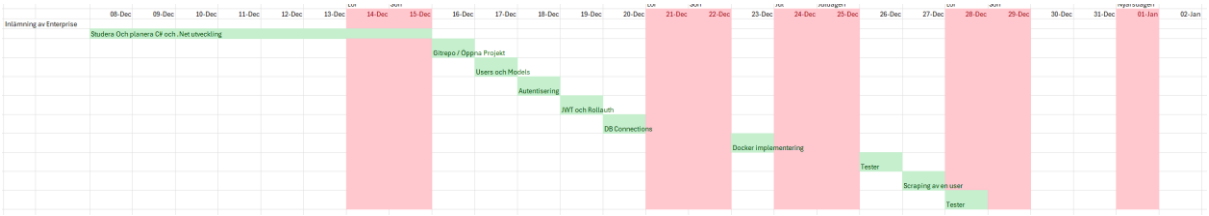
Med hjälp av:
Stack Overflow
ChatGPT,
Youtube(Nick Chapsas, Les Jackson m.m)
Microsoft Learning
Github Blog
m.m

1.3 Avgräsningar

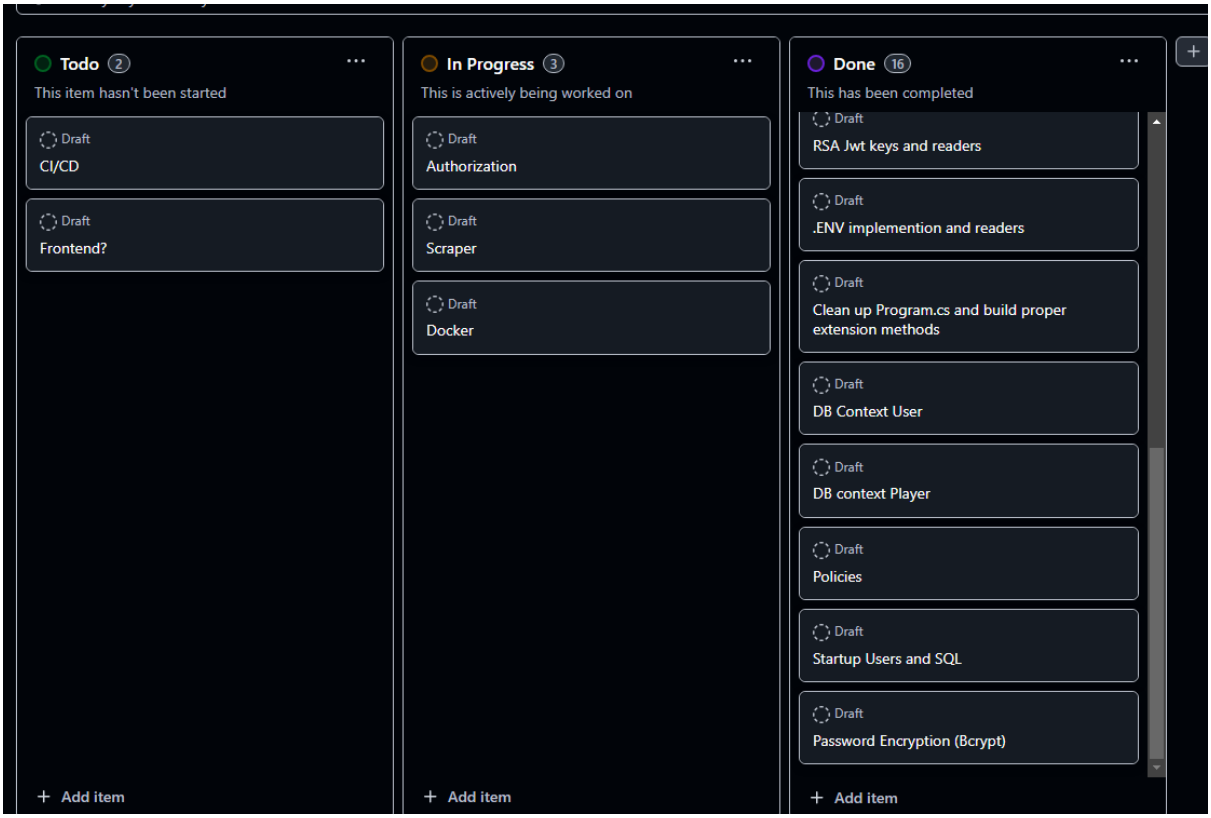
Detta projekt har en oändlig potential för att hämta mer eller mindre information från hemsidan. Projektets huvudsakliga mål är att bygga en version som sparar NÅGON form av information på ett strukturerat sätt.
Generella scrapers blir snabbt udaterade och kommer behöva uppdateras kontinuerligt för att detta projekt ska överleva, och därför kommer detta projekt att fokusera på att hämta informationen genom an autoriserad app (för att förminska kraven på Liquipedias servrar) men bara information om individuella spelare eller lag.

2. Planering och genomförande

2.1 Projektmetod:
Projektets metodik är agil utveckling, och har en grundläggande struktur som evalueras och utvecklas över tid vartefter produkten produceras. Den överläggande planen såg ut som nedan:

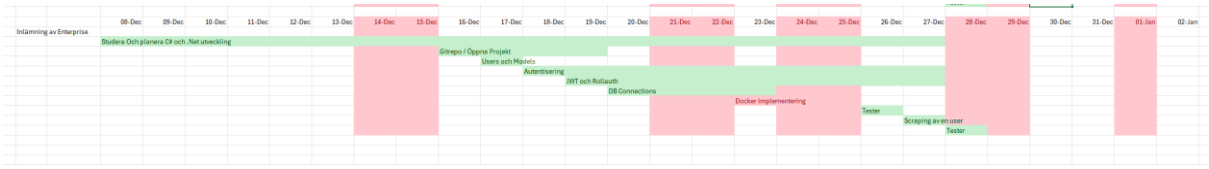


Userstories och projektstegen kan hittas på https://github.com/users/EricNorrwing/projects/2 men här kommer en kort summering:



Denna planering höll INTE och blev väldigt stressigt, och därför är det mycket oplanerat arbete och arbete som gjordes utan att skapa issues/Userstories om det. Detta är ett brist i långtidsplanerningen av inlärningen av .NET och .ASPNET Core web API.

2.2 Struktur



Den mer sanningsärliga bilden av projektet. Många steg av scaffolding tåg längre tid eller helt enkelt inte fans med i slutprodukten. Målet med projektet kan inte säga att det uppnåddes utöver det absoluta MVP.

Planerningen gjordes tidigt men följdes redan första veckan inte enligt plan då jag hade problem med sätta upp strukturen i projektets helhet. Därför hamnar vi efter och fick redan första veckans byggfas dra ner på målen.

Teknikvalen gjordes baserat på LIA plats som jag fick påskrivet den 11 dec för att matcha arbetsplatsens val av verktyg.

2.3 Problem och lösningar

En stor del av problemen med detta kommer från .Net's autentisering och auktorisering samt deras lösningar Databas implementeringar.
Ett lätt exempel på problem är att ladda in data till databasen rekommenderas att köra en "Migration" som är ett sätt att bygga ett databas schema automatiskt från dina models. Men det var ett steg som tekniskt sätt inte behövdes eftersom de genererades endå, men inte 100% rätt alltid. Det gjorde att kod som borde fungerat ej efterföljde planerade schema och behövdes rättas till efter mycket buggfixande.

Ett annat exempel på problem är att min Docker implementering aldrig riktigt fungerade som det ska, (Över 10h totaltid mellan 2 personer för att lista ut detta) och nu fungerar den, men den kör inte via individuella microservices som är rekommenderat inom .Net.

Listan på problem är oändligt lång och gjorde att manga delar av projektet mer eller mindre lagts på hold och inte kommer att återupptas.

2.4 Hinder

Det överlägset största hindret i projektet var tidsramen. Med tanke på att jag inte garanterat fått min LIA plats föränn 11 Dec kunde jag inte på ett vettigt sätt påbörja projektet tidigare, och då var det första steget att lära mig grundläggande delar av skillnader mellan språken. Men även om vi bortser från detta har vi en väldigt snar tidsram för hela detta projektet även om man har låga mål från början.

Andra hinder bestod i större skillnader mellan tidigare .NET och nyare versioner, ej kompatibla ramverk och funktionalitet och helt enkelt större problem att lära mig alla nyanser av språket och vad man kan göra.

3. Verktyg och Metoder

3.1 Verktyg

Kort sammanställning av verktyg:

Rider IDE
Microsoft SQL Server
SSMS

Docker
Github
Github Actions
Github Projects

Och för inlärning:
StackOverflow
Youtube (Framför allt: Les Jackson, Nick Chapsas, Patrick God)
ChatGPT (Framför allt syntax och bugfix)
2 Utvecklare vänner när de hade tid att hjälpa.

Den största skillnaden är att jag valde att köra Rider IDE över att köra VS Code (Som Arbetsplatsen jag går till använder), då jag är van med IntelliJ som är snarlikt. Annars har jag valt att använda allt som de använder på arbetsplatsen.

3.2 .NET

.NET är microsofts huvudsakliga ramverk för C# kod och har använts här med.

.NET Authorization och Authentication

Deras inbyggda lösningar för att verifera users (Motsvarigheten till SpringSecurity)
Det generella är att vi har egna users som vi autentiserar mot appen med hjälp av JWT Tokens. Summering nedan:
Användaren skapar en användare med en Http-Post mot en endpoint där du skickar en body med “username, password”. Detta godkänns om syntax och krav är uppfyllade och skickas till databasen. Värden valideras under denna process i service largret. Den hashar även password med Bcrypt.
Sedan skickar användaren en Http-Post till en login endpoint, som returnerar en JWT token. Denna token används som autentisering och auktorisering för resten av timmen innan du behöver registrera en ny token på Login. I Denna token ingår dina Policies (Roller) som avgör vilka endpoints du får besöka. En vanlig user kan t.ex. inte edita andras profiler, men admins kan det.
Appen är Stateless.
Databasen har 2 default users, en admin och en utan.

För säkerhetensskull har vi alltid 2 lager mellan user och databasen, även för scrapern. User > UserService > UserRepository > Databas. Och databas connectionen använder sig av interfacet “Disposable” som stänger ner databas connectionen när den inte längre behövs. Det gör att appen blir långsammare, men betydligt säkrare.
Denna app BÖR använda sig av CORS, men gör för TILLFÄLLET när jag skriver detta INTE det pga problem med docker.

Den är annars 100% Valdierat och escape vänlig mot databasen.

Flera metoder utvärderas och kastades bort när jag valde gällande autentisering. Original planen var at använda Okta, men krävde autentiserat HTTPS certifikat, eller Keycloak men i brist på tid ville inte sätta mig in i det utan byggde något snabbt. Det gör att det säkerligen finns brister i detta projekt som kan ha missats.

3.3 MicrosoftSQL
Microsoft SQL valdes, som nämnt tidigare, pga framtida arbetsplats och har en del egenheter som Postgre (som var andra alternativet för mig) inte har. En del problem uppstod här när jag läste connectionstrings från min .ENV fil pga att alla generella readers på nätet splittar på “=”, men strängarna innehåller = som en del av koden för att kontakta databasen.

Strukturen på databasen är att HELT separera Users från allt annat. Detta verktyg ska (i en ideel värld som ej finns) Säljas med ett autentiserings paket från mig eller någon annan. Jag har stor respekt för Liquipedia.net och vill ej att de ska få problem för att min appa spammer dem.

Därför är users och UserClaims (som används för autentisering) separata tabeller med nycklar till varandra, mer om det finns i koden om exakt hur schemat är uppbyggt.

Alla delar av databasen är byggd med så kallat “Migrations” som är ett EntityCore Framwork (EF) struktur för att skapa automatiska schemas.

Microsoft SQL i mitt fall valideras med Windows login, vilket gör att databasen måste köras på en windows burk eller i en container /Azure.

3.4 Git
Git används under alla delar av projektet, men inte till den helhet som planerades från början. Målet var att allt skulle köras i en DockerContainer via Github Actions och även innefattas CI/CD principer och Devops. Denna del blev aldrig realiserad och det finns lite leftovers i koden (CI/CD docker.yaml osv) som jag lämnade för att visa problemen som uppstod. En hel del tid slösades här.

Jag har även använt Github Projects som en enkel skiss för userstories för att följa vad som byggts och nästa steg.

Github har naturligvis används som versionshantering, även om det var tankar kring att använda Azure Devops.

4. Verktyg och Metoder

Slutresultat är tyvärr till mestadelas misslyckande. All integrering som var planerad finns en tillgängligt och en del funktionalitet saknas (men Kanske fixas under helgen efter detta är skrivet)

https://github.com/EricNorrwing/DotaProject

Projektet är i slut funktionalitet:
Ett 100% Autentiserat och Auktoriserat program, stateless med JWT token login baserat på 2048 RSA private key och en public key för att lösa (industristandard). Users skapas och interagerar med min AuktoriseringsDatabas (AuthDb) och laddas kontinuerligt med Repo och Service lager, strukturen är uppsatt för skalning.

Det finns endpoints för att lägga in EN spelare baserat på URL och samla in data från denna (Men koden är skriven i typescript och hemsidan har förändrats så den körs inte 100% korrekt).

Docker implementationen körs och kan användas via Docker-compose för att öppna rätt portar till appen, annars körs den via Http för enkelhetens skull.

Tester kommer så snart jag fixat scrapern.

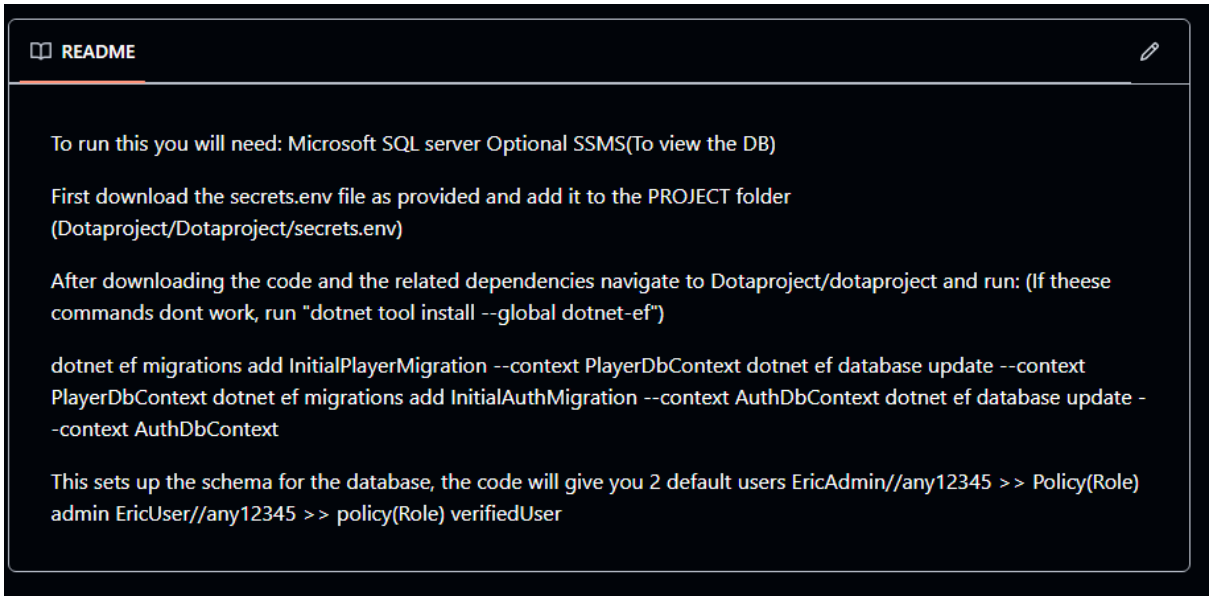
5. Egna Reflektioner

Jag är, så klart, missnöjd med min prestation den senaste månaden i helhet både i detta projekt och Enterprise. Varje gang jag fick avsätta ett mål pga tidsbrist gjorde det att jag blev mer och mer ointresserad och detta arbetet kändes bara stressigt. En hel del av detta är dock otroligt lärorikt och jag är glad över den mängd arbete som gjorts och hur arbetet sattes upp.

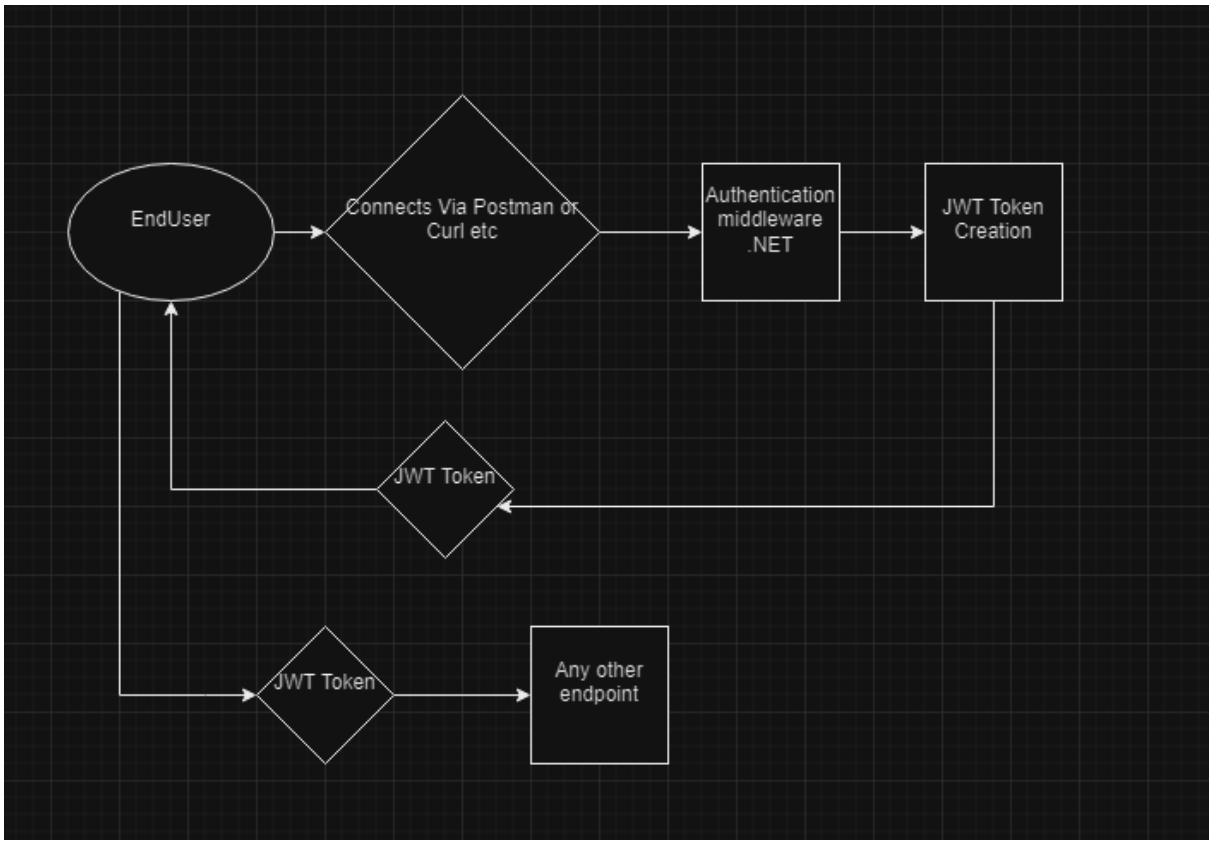
Den huvudsakliga sammanfattningen är att pga LIA stress, job och Jul jag har helt enkelt inte haft den tiden jag ville för att bygga den appen jag velat och jag hoppas i framtiden att mer tid kan allokeras mellan Enterprise projektet och Examensarbetet, även om jag är 100% ombord med att LIA borde vara en lång period i slutet.

Andra mer positive reflektioner är att jag är jättenöjd med funktionalitet i .NET och ser verkligen stora skillnader i hur ramverket fungerar och det har varit väldigt lärorikt i logging (Stort Tack till SeriLog) för all hjälp man kan få som utvecklare med problemlösning om koden ger dig möjlighet.

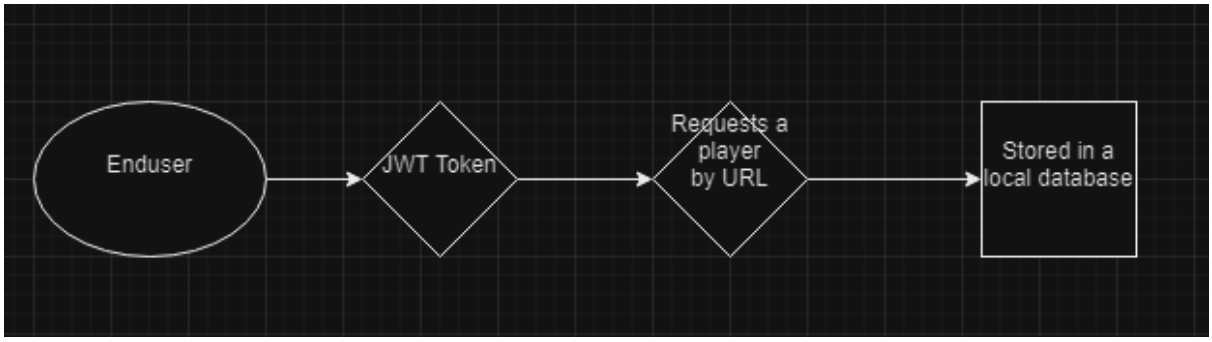
6. Diagram och beskrivningar



Github Readme för instruktioner om att köra appen, Rekommenderas för testing att köra Http.

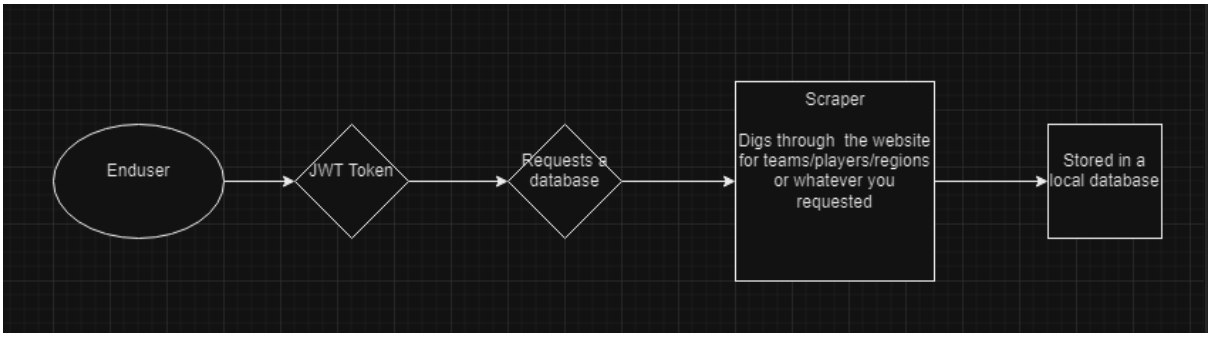


Login Flow

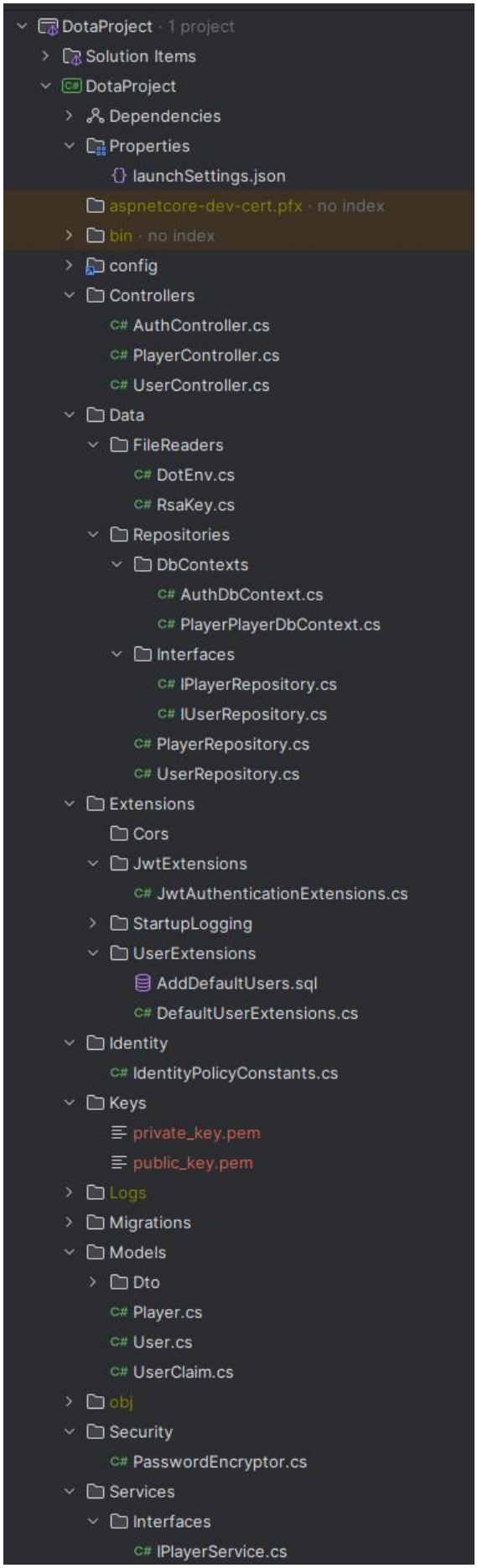


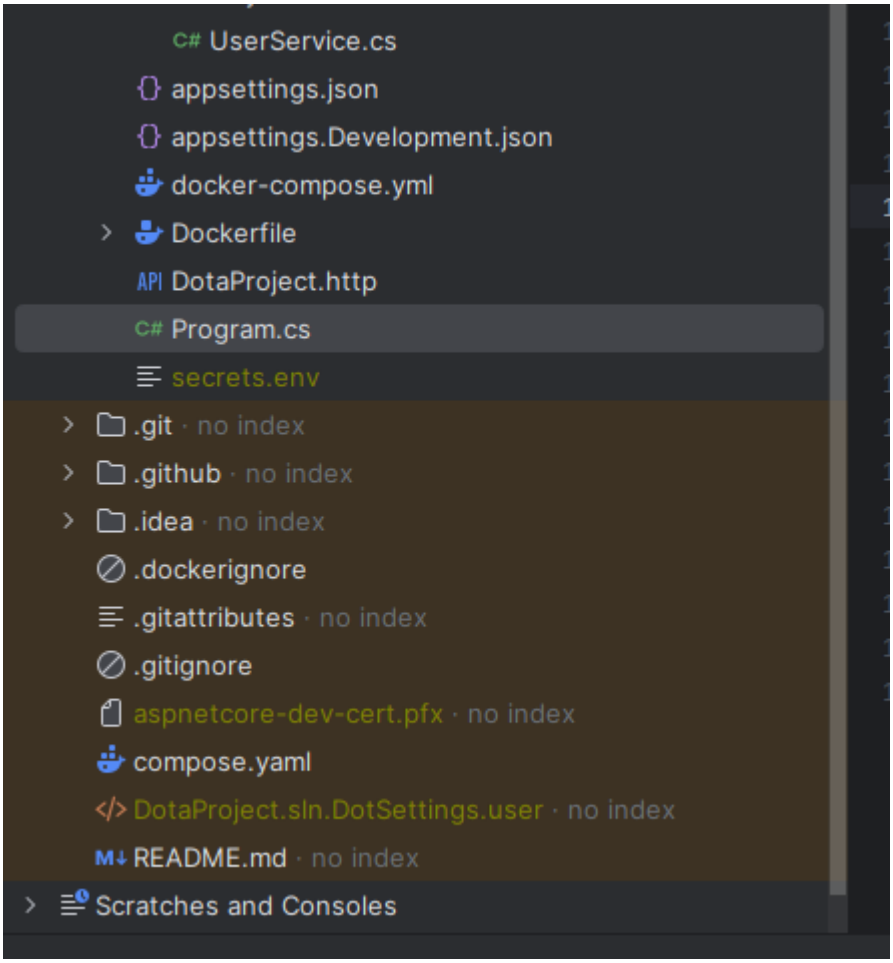
MVP
End usage (Once you have acquired an JWT Token for authentication)

Ideal End product



Till exempel borde scrapern kunna hämta data från denna plats: https://liquipedia.net/dota2/Portal:Players/Americas för att hitta alla spelare i Nord+Syd Amerika.


DotaProject · 1 project
Solution Items
DotaProject
Dependencies
Properties
launchSettings.json
aspnetcore-dev-cert.pfx · no index
bin · no index
config
Controllers
AuthController.cs
PlayerController.cs
UserController.cs
Data
FileReaders
DotEnv.cs
RsaKey.cs
Repositories
DbContexts
AuthDbContext.cs
PlayerPlayerDbContext.cs
Interfaces
IPlayerRepository.cs
IUserRepository.cs
PlayerRepository.cs
UserRepository.cs
Extensions
Cors
JwtExtensions
JwtAuthenticationExtensions.cs
StartupLogging
UserExtensions
AddDefaultUsers.sql
DefaultUserExtensions.cs
Identity
IdentityPolicyConstants.cs
Keys
private_key.pem
public_key.pem
Logs
Migrations
Models
Dto
Player.cs
User.cs
UserClaim.cs
obj
Security
PasswordEncryptor.cs
Services
Interfaces
IPlayerService.cs



Nuvarande projektstruktur.

7. Källförteckningar

Denna lista är inte 100% utförlig och mycket information har delats mellan mig vänner och en lång rad rubberducks.

Docker:
https://medium.com/@jakubrzepka/building-an-asp-net-8-web-api-docker-image-container-8a64f8635275
https://www.docker.com/
https://hub.docker.com/
https://docs.docker.com/build/ci/github-actions/
https://medium.com/rewrite-tech/how-to-create-custom-ci-cd-with-github-and-docker-495e8ff87c7e
https://medium.com/@avash700/ci-cd-made-easy-github-actions-docker-compose-and-watchtower-60a698d24f27
https://docs.docker.com/compose/
https://github.com/docker/compose
https://docs.docker.com/compose/how-tos/environment-variables/set-environment-variables/

Github
https://github.com/features/actions
https://docs.github.com/en/actions
https://marketplace.visualstudio.com/items?itemName=GitHub.vscode-github-actions (Innan jag byte till Rider)
https://github.blog/enterprise-software/ci-cd/build-ci-cd-pipeline-github-actions-four-steps/
https://docs.github.com/en/actions/about-github-actions/understanding-github-actions
https://docs.github.com/en/issues/planning-and-tracking-with-projects/learning-about-projects/about-projects
https://docs.github.com/en/actions/security-for-github-actions/security-guides/using-secrets-in-github-actions
https://docs.github.com/en/codespaces/managing-codespaces-for-your-organization/managing-development-environment-secrets-for-your-repository-or-organization

Youtube guider:
https://www.youtube.com/watch?v=mgeuh8k3I4g
https://www.youtube.com/watch?v=AKjG2tjI07U
https://www.youtube.com/watch?v=sW1OGB5ztZI
https://www.youtube.com/watch?v=Cxftp90K_ek
https://www.youtube.com/watch?v=jT8eA9A7qXE
https://www.youtube.com/watch?v=ygJ11fzq_ik
https://www.youtube.com/watch?v=fmvcAzHpsk8
https://www.youtube.com/watch?v=DgVjEo3OGBI
https://www.youtube.com/watch?v=cQ7WmUMbqe4
https://www.youtube.com/watch?v=dh1NFAIoZCI
https://www.youtube.com/watch?v=1LTwpTi04j4
https://www.youtube.com/watch?v=gCfBxN4m5gw
https://www.youtube.com/watch?v=lcaDDxJv260
https://www.youtube.com/watch?v=f0lMGPB10bM
https://www.youtube.com/watch?v=TPbCKUwJ_hE

Och allt detta med mera. Jag tar inte fram indivduella stackoverflow trådar eftersom de flesta besöktes bara under några minuter. Nästan all information är baserat på Officiell documentation men youtube har hjälpt mig mycket och framför allt Les Jackson’s långa videos bör kollas igenom. Varning: Denna lista är säkert 35h+ om jag fått med allt.