README-UserService-Prototype-v2

User Service – Prototype (Contract-first, C# / .NET 9)

> **Doel:** een *klein en werkend* prototype van de User Service dat alle drie de kerntaken dekt:
> **Authentication**, **Profile Management** en **Data Storage** (decks & cards) (zoals beschreven in Project 42 opdracht omschrijving) — lokaal draaiend, volledig contract-first, zonder externe database of security-implementaties.

---

1. Inleiding

Dit prototype demonstreert hoe gebruikers zich kunnen **registreren**, **inloggen**, een **profiel beheren**, en hun **decks en kaarten** kunnen opslaan binnen het microservices-systeem van Project 42.
Het prototype is ontworpen voor **contractvalidatie**: andere services (zoals de Controller) kunnen alvast integreren via de OpenAPI-specificatie (zie H4).

Belangrijke keuzes

- Alle data wordt **in-memory** opgeslagen.
- Wachtwoorden zijn **niet gehasht**.
- Tokens zijn **niet cryptografisch veilig**.
- Er wordt géén externe database of file-opslag gebruikt.

---

2. Scope

In scope

- Minimal API's (.NET 9) met OpenAPI-export (`/openapi/v1.json`)
- **Authentication**: registreren, inloggen, tokenverificatie
- **Profile Management**: profiel + statistieken bijhouden
- **Data Storage**: decks + kaarten per gebruiker
- In-memory opslag
- Koppelpunt naar de Controller via `HttpClient`

Niet in scope

- Echte beveiliging (hashing, JWT)
- Persistente opslag (SQL/NoSQL)
- Inputvalidatie, rate limiting of logging

---

3. Snel starten (Windows 11)

Vereisten:

- .NET SDK 9.x
- PowerShell of Terminal

```
dotnet restore
dotnet run
```

Standaard draait de service op een willekeurige poort (bijv. `http://localhost:5069`).

Vaste poort instellen

```
builder.WebHost.UseUrls("http://localhost:5000");
```

---

4. Endpoints overzicht

Endpoint	Methode	Doel	Status
`/_health`	GET	Healthcheck	✅
`/users/register`	POST	Gebruiker registreren	✅
`/users/login`	POST	Token verkrijgen	✅
`/users/{id}`	GET	Profiel opvragen	✅
`/users/{id}`	PUT	Profiel bijwerken	✅
`/auth/resolve`	POST	Token → UserId (Controller)	✅
`/cards`	GET	Opgevraagde kaarten van gebruiker	✅
`/cards`	POST	Nieuwe kaart toevoegen (mock)	✅
`/decks`	GET	Decks van gebruiker opvragen	✅
`/decks`	POST	Nieuw deck aanmaken	✅
`/decks/{deckId}/cards`	POST	Kaarten toevoegen/verwijderen	✅
`/_controller/ping`	GET	Verbinding naar Controller	✅

OpenAPI-document:
`http://localhost:<poort>/openapi/v1.json`

---

5. Architectuuroverzicht

Extern (context)

```
Client ──> User Service Prototype ──> Controller (stub)
```

Intern (structuur)

```
PrototypeUserService/
 ├── Program.cs
 ├── Endpoints/
 │    ├── UsersEndpoints.cs
 │    └── DecksEndpoints.cs
 ├── Contracts/
 │    ├── Users.cs
 │    ├── Profile.cs
 │    ├── Decks.cs
 │    ├── Cards.cs
 │    └── ErrorResponse.cs
 └── Services/
      └── MockUserService.cs
```

---

6. Data- en contractmodellen

Authentication

RegisterRequest

```
{ "username": "cas", "password": "1234" }
```

Response

```
{ "userId": "guid", "username": "cas" }
```

LoginRequest

```
{ "username": "cas", "password": "1234" }
```

Response 200

```
{ "token": "Y0y6Pp5..." }
```

Profile Management

UserProfile (GET / PUT /users/{id})

```
{ "id": "guid", "username": "cas", "wins": 2, "losses": 1 }
```

UpdateProfileRequest

```
{ "username": "cas", "wins": 3, "losses": 1 }
```

Data Storage (Decks & Cards)

Deck

```
{ "id": "guid", "ownerId": "guid", "name": "My First Deck", "cardIds": [] }
```

CreateDeckRequest

```
{ "ownerId": "guid", "name": "Deck Naam" }
```

ModifyDeckCardsRequest

```
{ "addCardIds": ["guid"], "removeCardIds": ["guid"] }
```

Card

```
{ "id": "guid", "name": "Fireball" }
```

OwnedCardsResponse

```
{ "userId": "guid", "cards": [ { "id": "guid", "name": "Fireball" } ] }
```

---

7. Functioneel Testplan

Test	Actie	Verwacht
T1	POST `/users/register`	`200 OK` + userId
T2	POST `/users/login`	`200 OK` + token
T3	POST `/auth/resolve` met token	`200 OK` + userId
T4	GET `/users/{id}`	Huidig profiel
T5	PUT `/users/{id}` met wins of username	Profiel bijgewerkt
T6	POST `/cards`	Kaart toegevoegd
T7	GET `/cards`	Lijst kaarten
T8	POST `/decks`	Nieuw deck aangemaakt
T9	POST `/decks/{deckId}/cards`	Deck aangepast
T10	GET `/decks`	Lijst decks
T11	GET `/_health`	`200 OK`

---

8. Test voorbeelden (curl)

Profiel bijwerken

```
curl -X PUT http://localhost:<poort>/users/<id> ^
 -H "Content-Type: application/json" ^
 -d "{"wins":5,"losses":2}"
```

Deck aanmaken

```
curl -X POST http://localhost:<poort>/decks ^
 -H "Content-Type: application/json" ^
 -d "{"ownerId":"<id>","name":"My Deck"}"
```

Kaart toevoegen

```
curl -X POST "http://localhost:<poort>/cards?ownerId=<id>&name=Fireball"
```

---

9. Beveiliging (prototype-niveau)

- Geen hashing of salting.

- Tokens zijn willekeurige Base64-strings.
- Geen sessie-beheer of autorisatie-rollen.

> *Toekomstige uitbreidingen:* JWT-tokens, veilige password-hashing, database back-end.

---

10. Configuratie

Controller stub:

```
builder.Services.AddHttpClient("Controller", client =>
{
    client.BaseAddress = new Uri("http://localhost:5099");
});
```

Poort vastzetten:

```
builder.WebHost.UseUrls("http://localhost:5000");
```

---

11. Bekende beperkingen

- In-memory data verdwijnt na herstart.
- Geen inputvalidatie.
- Geen concurrency-beveiliging voor write-operaties.

---

12. Roadmap

1. **JWT authenticatie**
2. **Password hashing + database**
3. **Validatie en error-handling**
4. **Swagger UI**
5. **Deck/Card-validatie** tegen echte Card Collection Service

---

13. Bronnen

- Microsoft Docs – OpenAPI in ASP.NET Core (.NET 9): https://learn.microsoft.com/aspnet/core/fundamentals/openapi

- Microsoft Docs – Minimal APIs: https://learn.microsoft.com/aspnet/core/fundamentals/minimal-apis
- Swashbuckle (Swagger UI): https://github.com/domaindrivendev/Swashbuckle.AspNetCore

---

Auteur & versie

- Project: Project 42 – User Service Prototype
- Versie: 2.0.0
- Datum: 11-11-2025