**Rendszermodellezés**

Bevezető:

Az LIN034-es tárgykóddal rendelkező Rendszermodellezés elnevezésű tantárgy elvégzése érdekében elkészítettem egy [UML](https://www.geeksforgeeks.org/unified-modeling-language-uml-introduction/) diagramokon alapuló rendszertervet, amely egy elképzelt kórházi applikáció [szerveroldali](https://www.computerhope.com/jargon/s/server.htm) működését hivatott modellezni.

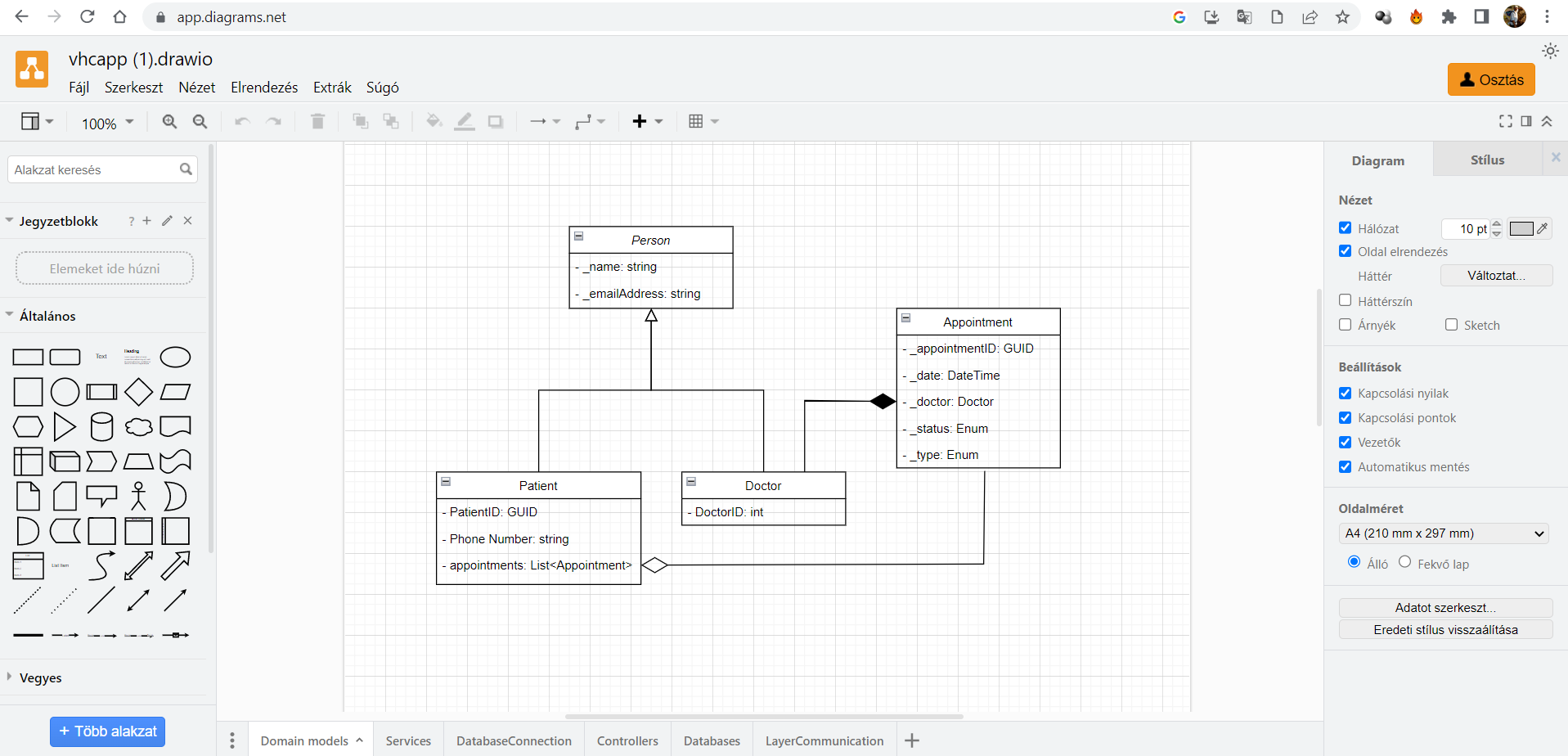
A jelenlegi elvárások (és az elkészült rendszermodell) szerint ennek a szervernek képesnek kell lennie arra, hogy az alapvető [CRUD](https://www.codecademy.com/article/what-is-crud) műveleteket (Create, Read, Update, Delete) a Person és az Appointment [entitás](https://www.techopedia.com/definition/14360/entity-computing#:~:text=An%20entity%20is%20any%20singular%2C%20identifiable%20and%20separate,languages%2Fconcepts%2C%20database%20management%2C%20systems%20design%20and%20other%20arenas.) esetén el tudja végezni a definiált service és repository osztályok segítségével.

Az entitásokat [“hot” és “cold” backupokban](https://blog.toadworld.com/hot-and-cold-backups-overview#:~:text=A%20Cold%20backup%2C%20that%20is%2C%20one%20done%20with,read%20consistent%20copy%2C%20but%20doesn%27t%20handle%20active%20transactions.) tároljuk a gyorsabb kiszolgálás és az adatdisztribúció elérése érdekében. Egy [relációs adatbázis](https://www.oracle.com/database/what-is-a-relational-database/) fog cold backupként szolgálni (pl.: [PostgreSQL](https://www.postgresql.org/), [MsSQL](https://learn.microsoft.com/en-us/sql/?view=sql-server-ver16)), míg a hot backup egy key-value elven működő [NoSQL](https://www.geeksforgeeks.org/introduction-to-nosql/) adattár lesz (pl.: [Redis](https://redis.io/)). Hogy pontosan miért döntöttem ezen megoldás mellett, azt a későbbi fejezetekben részletezem.

Az elkészült rendszermodell alapján a forráskód megírását/előállítását kódgeneráló szoftvernek, nyelvi modellnek vagy egy junior backend fejlesztőnek oda is adhatjuk, hiszen a tervezést mi magunk már elvégeztük.

Rendszermodell megnyitása, lépések:

1. A mellékelt vhcapp nevű fájl-t (.drawio kiterjesztés) le kell tölteni.
2. Rákeresni a draw.io ([diagrams.net](https://app.diagrams.net/)) oldalra, majd itt kiválasztani az “Open Existing Diagram” opciót.
3. Kiválasztani a letöltött vhcapp nevű fájlt (vö. <https://miau.my-x.hu/miau/298/vhcapp%20(1).drawio>)



A képen szöveg, képernyőkép, beltéri, számítógép látható

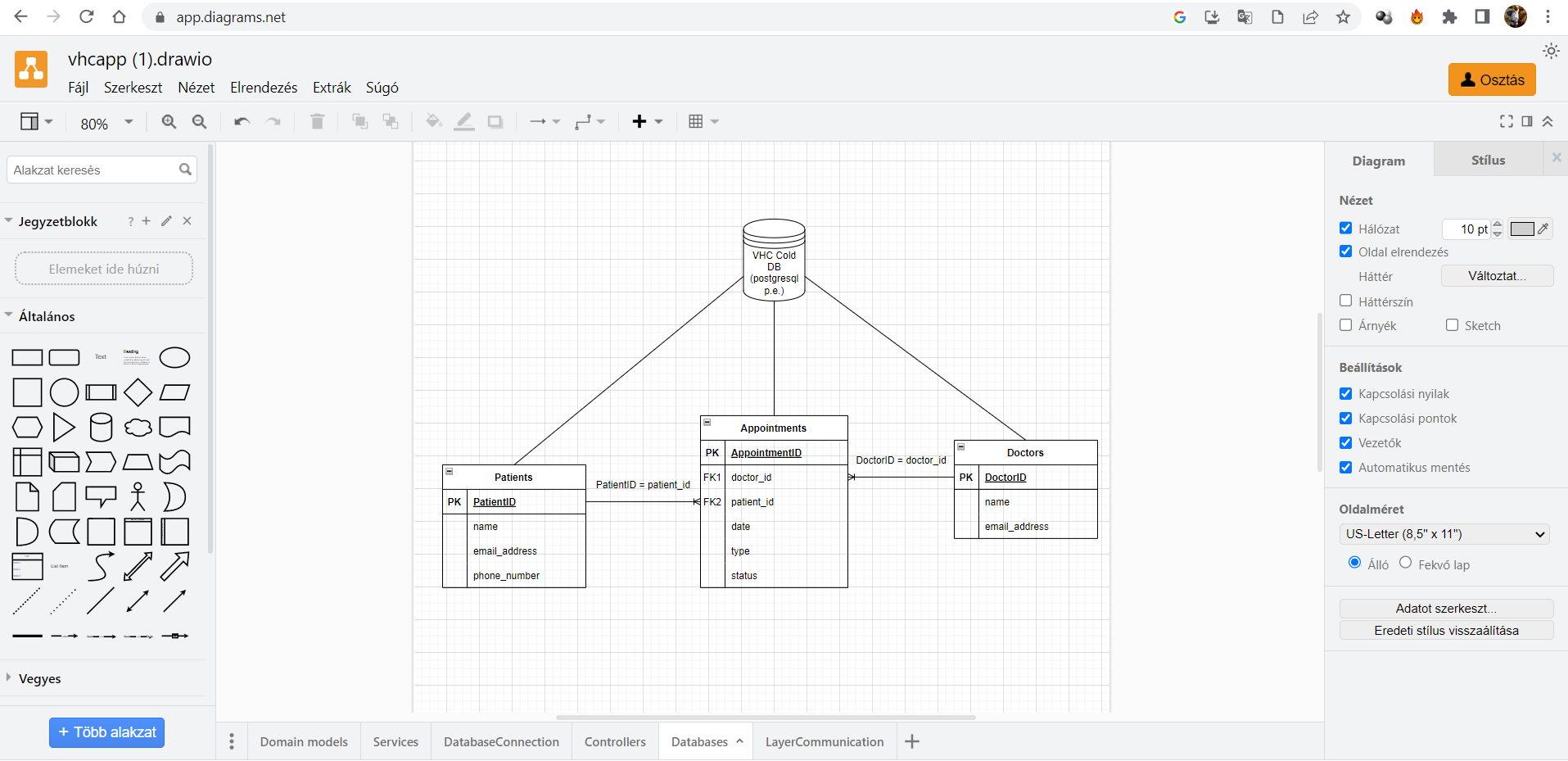
Automatikusan generált leírás

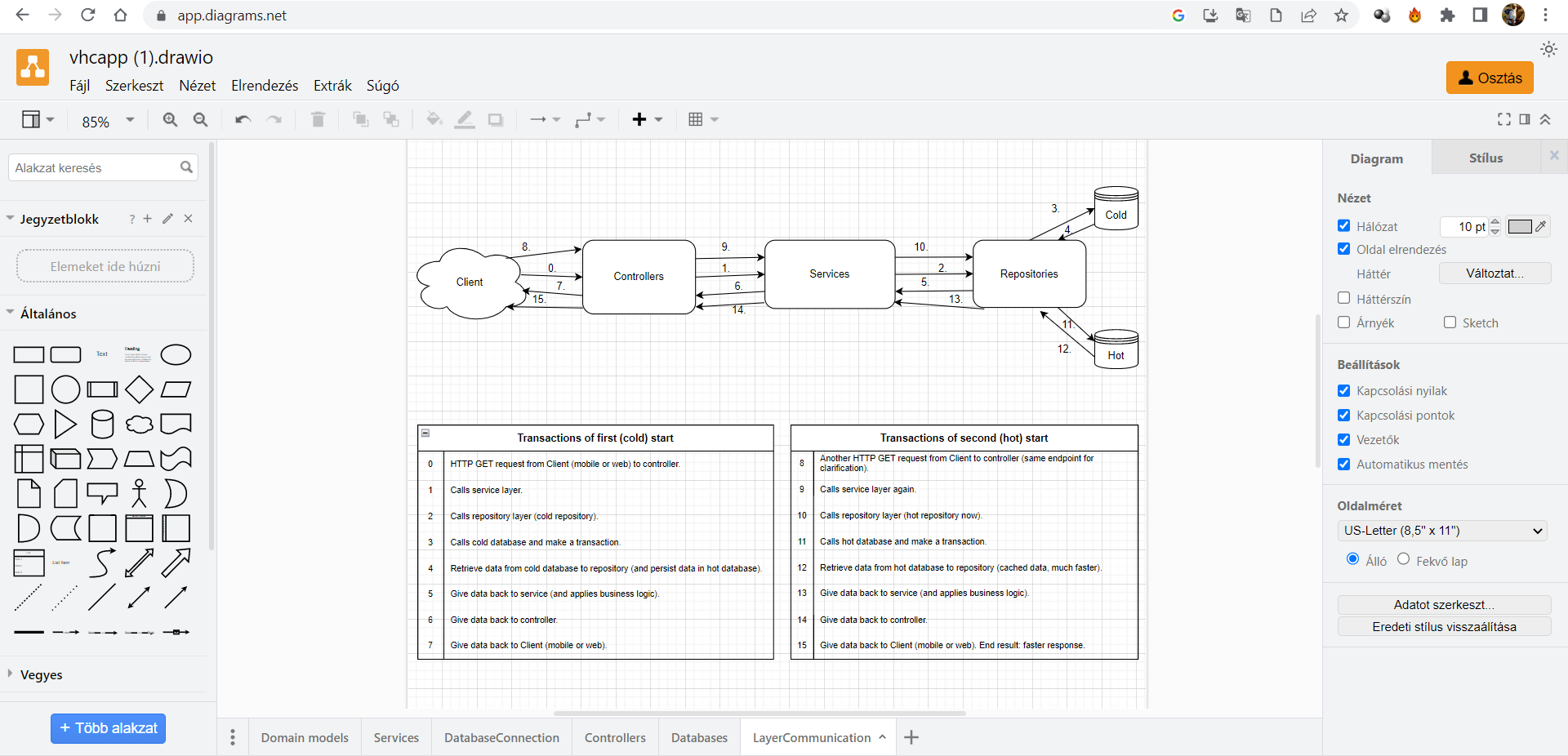
A képen szöveg, képernyőkép, beltéri látható

Automatikusan generált leírás

A képen asztal látható

Automatikusan generált leírás





Dokumentáció:

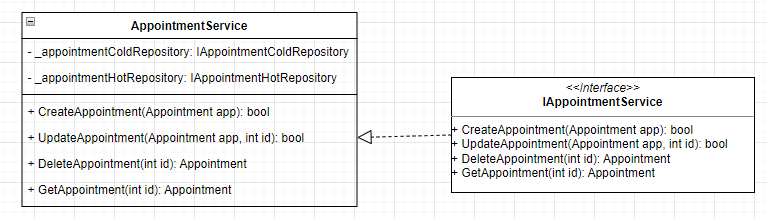
* Jelmagyarázat:

Az elkészült rendszerdiagramban található modellek igényelnek jelmagyarázatot is, ugyanis például az eltérő megjelenésű nyilak más és más típusú relációkat, egy-egy domain osztály tulajdonsága (legyen az property vagy method) előtti mínusz vagy plusz jel eltérő hozzáférhetőséget hivatottak jelképezni. Ennek fényében fontosnak tartottam kitérni ezek jelentésére.

* + Térbeli elhelyezkedés:

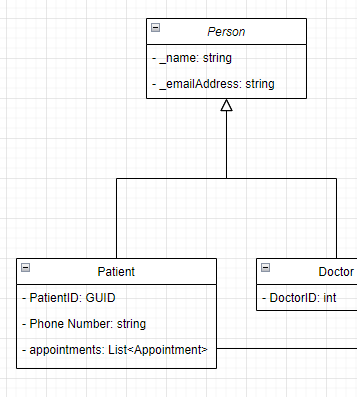
A legegyszerűbb elsődlegesen végig menni ezen aspektuson, ugyanis a térbeli elrendezésnek nincsen mögöttes jelentéstartalma. Itt csupán az esztétika volt szempont, ennek mentén kívántam valamilyen módon elhelyezni a különböző modelleket.

* + [Arrows](https://learn.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-2015/modeling/uml-class-diagrams-reference?view=vs-2015):
    - Interfész implementáció:

Amennyiben egy osztály implementál egy interfészt, abban az esetben a Classifiert kell használnunk. (Példa: AppointmentService <-- IAppointmentService)

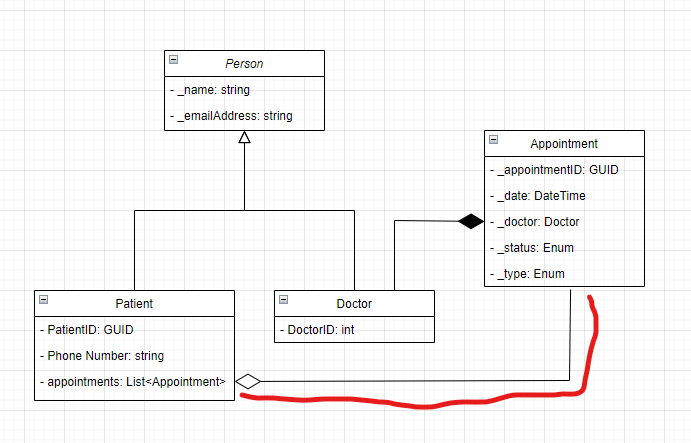
* + - Öröklődés (Is-a relationship):

Amennyiben egy osztály rendelkezik ősosztállyal, abban az esetben a Generalizationt kell használjuk. (Példa: Person 🡨 Patient)



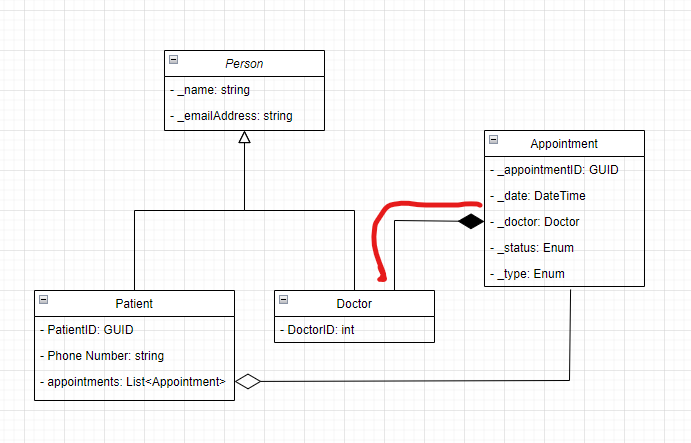
* + - [Aggregation](https://www.uml-diagrams.org/aggregation.html) ([Has-a relationship](https://www.bestprog.net/en/2020/02/27/c-types-of-relationships-between-classes-is-a-has-a-uses-examples-aggregation-composition/)):

Amennyiben egy osztály rendelkezik egy olyan property-vel, amely egy másik osztályunk példányát tartalmazza és ezen példány megléte **nem elengedhetetlen** az osztályunk működésében. Itt az Aggregation arrow-t kell használnunk. (Példa: Patient 🡨 Appointment)



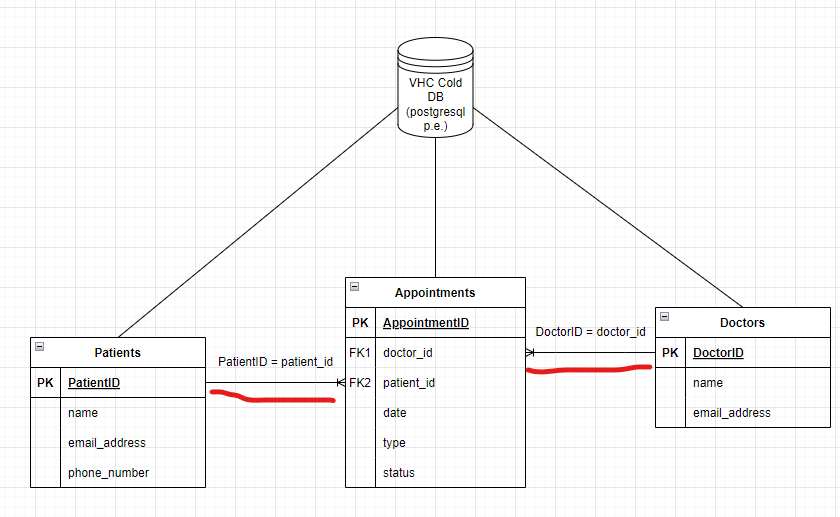
* + - [Composition](https://www.uml-diagrams.org/composition.html) (Has-a relationship):

Amennyiben egy osztály rendelkezik egy olyan property-vel, amely egy másik osztályunk példányát tartalmazza és ezen példány megléte **elengedhetetlen** az osztályunk működésében. Itt a Composition arrow-t kell használnunk. (Példa: Appointment 🡨 Doctor)



* + - [One-to-many](https://vertabelo.com/blog/one-to-many-relationship/) kapcsolat:

A "one-to-many" kapcsolat egy adatbázis-tervezési koncepció, amely az adatok közötti kapcsolatot írja le. Ebben a kapcsolattípusban egy adatokból álló csoport (több adat) kapcsolódik egyetlen másik adathoz (egy adat). Ezt a kapcsolatot a "has-a" kapcsolat jellemzi, mivel az egyik adatobjektum "tartalmaz" másik adatobjektumokat. A kapcsolatot egy [foreign key](https://www.w3schools.com/SQL/sql_ref_foreign_key.asp) létrehozásával hoztuk létre. (Példa az adattáblánkban: Patient 🡨 Appointment)



* + Osztály tulajdonságainak [hozzáférhetősége](https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/access-modifiers):

Egy osztály rendelkezhet property-vel és method-dal. Mindkét esetben meghatározhatjuk, hogy az osztály ezen elemei milyen hozzáférhetőségekkel/elérhetőségekkel rendelkezzenek. Konvenció szerint a property-k private, míg a method-ok public értéket kaptak.

A jelölésekről bővebben:  
A képen asztal látható

Automatikusan generált leírás

* Rétegek ismertetése:

A rétegek lazán kapcsolódnak egymásról, illetve csak a saját belső működésükről tudnak. Pédául könnyen tudjuk cserélni az adatbázisunkat más típusú adatbázisra, erről egyedül a repository osztályunknak kell tudnia. Amennyiben egy ilyen módosítást elvégzünk, az nem fogja a többi rétegben implementált osztályainkat érinteni. A rétegek kommunikációjáról a LayerCommunication fülön tudunk áttekinteni 1-1 példahívás leírásával egyetemben.

* + Controller:

A controller class-ban definiált endpointon keresztül a küldött request beérkezik a szerverünkhöz. Amennyiben az HTTP requesttel elindított folyamat sikeresen zárul, 200 – OK HTTP response lesz a visszatérési érték.

* + Service:

Mivel a service réteg a controller (felelősség: bejövő hívások kezelése) és a repository (adatbáziskapcsolat) között helyezkedik el, így ebben a layerben kell implementáljuk a business logikát.

* + Repository:

Az adatbáziskapcsolatért felelős, ez a forráskódunk legbelsőbb rétege.

* + - Cold repository:

Amennyiben a kliens olyan adatlekérést (Read művelet) hajt végre, amely adatlekérési eredményt korábban nem mentettünk el a hot adattárunkban, ebben az esetben a folyamat a cold adatbázisunkból fogja előhívni a kért record-ot.

* + - Hot repository:

Amennyiben a kliens olyan adatlekérést (Read művelet) hajt végre, amely adatlekérési eredményt korábban már lementettük a hot adattárunkban, ebben az esetben ebből a hot repository-ból fogja előhívni a kért record-ot. Ez gyorsabb válaszidőt eredményez.

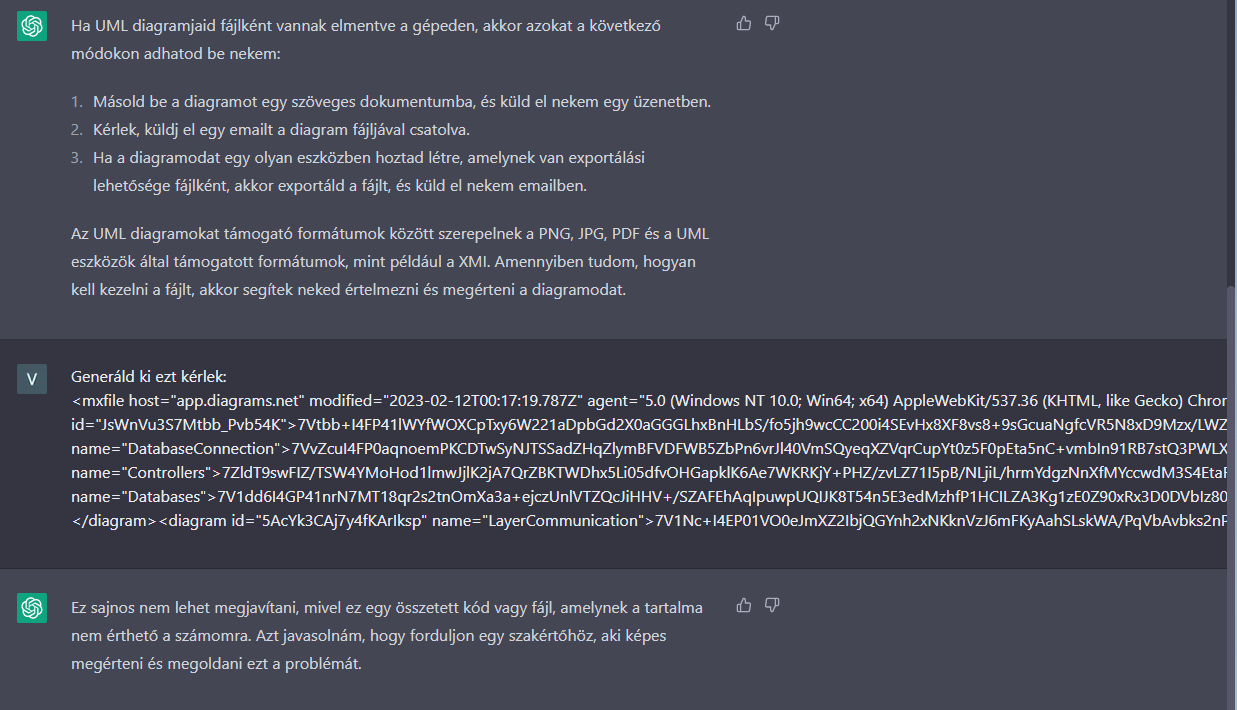
Forráskód megírása/generálása:

Mivel a kódírás/kódgenerálás esetén több irányba is elindulhatunk a megvalósítás felé, így első lépésként megkérdeztem a [ChatGPT](https://openai.com/blog/chatgpt/)-t, miként tudnám legenerálni a definiált modellek alapján a forráskódot a legegyszerűbben.

A képen szöveg látható

Automatikusan generált leírás

Ezt követően megpróbáltam a kódgenerálást a ChatGPT-n keresztül, azonban sajnos ez a kérés szemmel láthatóan még túl komplex az eszköz számára.



Mindenesetre arra kaptam választ, hogy milyen szoftverekkel/weboldallal tudom elvégezni a feladatot. Sajnos egyenlőre nem sikerült ezekkel az eszközökkel C# kódot generálni az XML fájlok alapján. Ettől függetlenül elvi lehetősége van a kigenerálásnak, illetve a feladatot továbbra is kiszervezhetjük egy junior kollégának. Ebben az esetben az ellenőrzés és a coaching feladata hárul ránk.

A képen szöveg látható

Automatikusan generált leírás

Konklúzió:

A rendszermodellezés módszere alkalmas különböző rendszerek elemzésére és optimalizálására, illetve segít megérteni a komplex rendszereket, amelyek körül vesznek minket, vagy amelyek megvalósítása ránk vár. Ennek a megközelítésnek a bemutatására tettem kísérletet a fenti munkámmal, bízva az elképzelt megrendelő és megvalósító közös megértésének elmélyítésében.