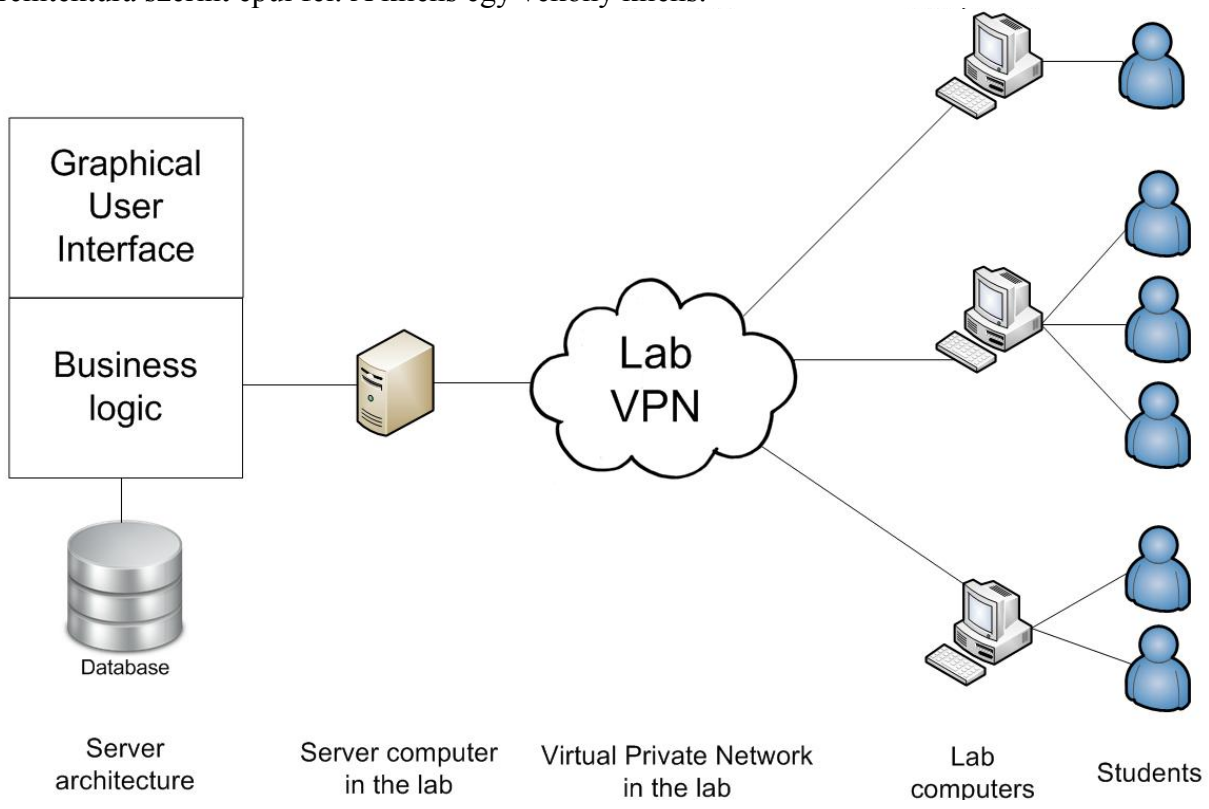
Rendszerterv

1. Architekturális felépítés

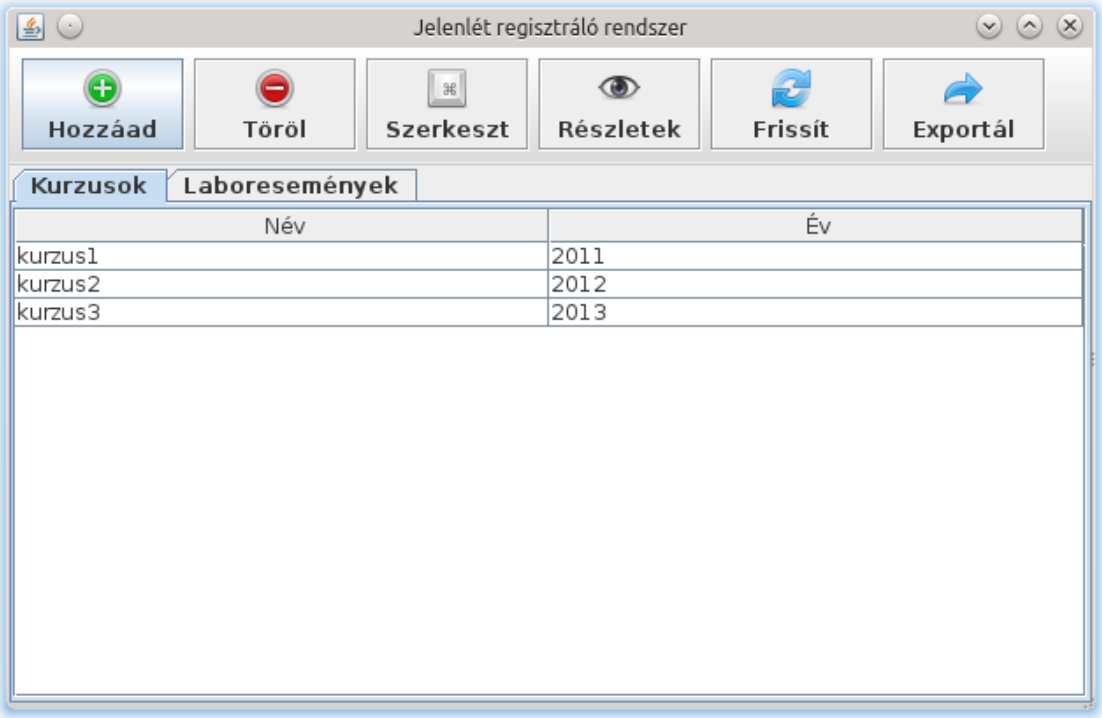
A rendszer egy szerverből és a hozzá csatlakozó kliensekből áll. A szerver háromrétegű architektúra szerint épül fel. A kliens egy vékony kliens.



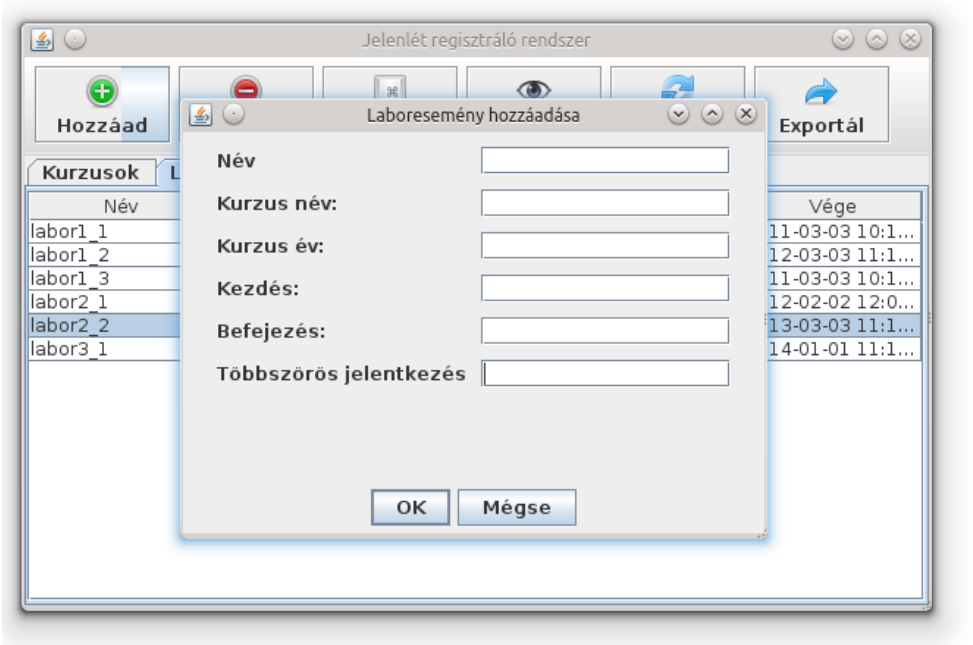
2. Kezelői felület terve

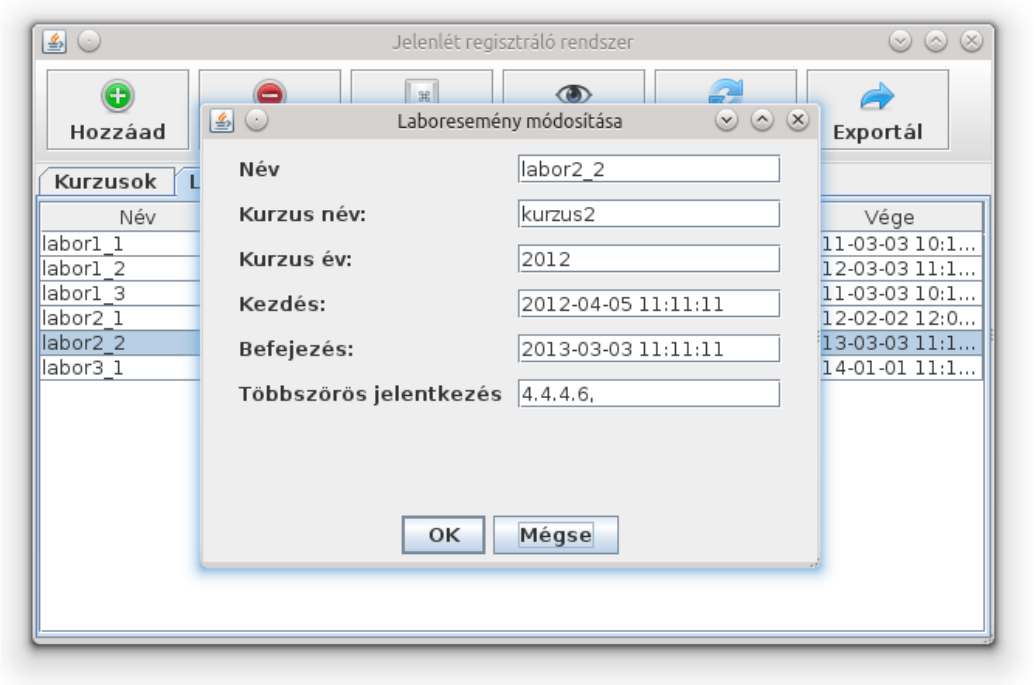
1. A Szerver alkalmazás felülete

A szerver alkalmazás grafikus felhasználói felülete a Java Swing API-jára épül. Az alkalmazás főképernyője két részből áll. Lent füleken a kurzusok és laboresemények, fent pedig az aktuális fülön lévő adatokat manipulálni képes gombok.

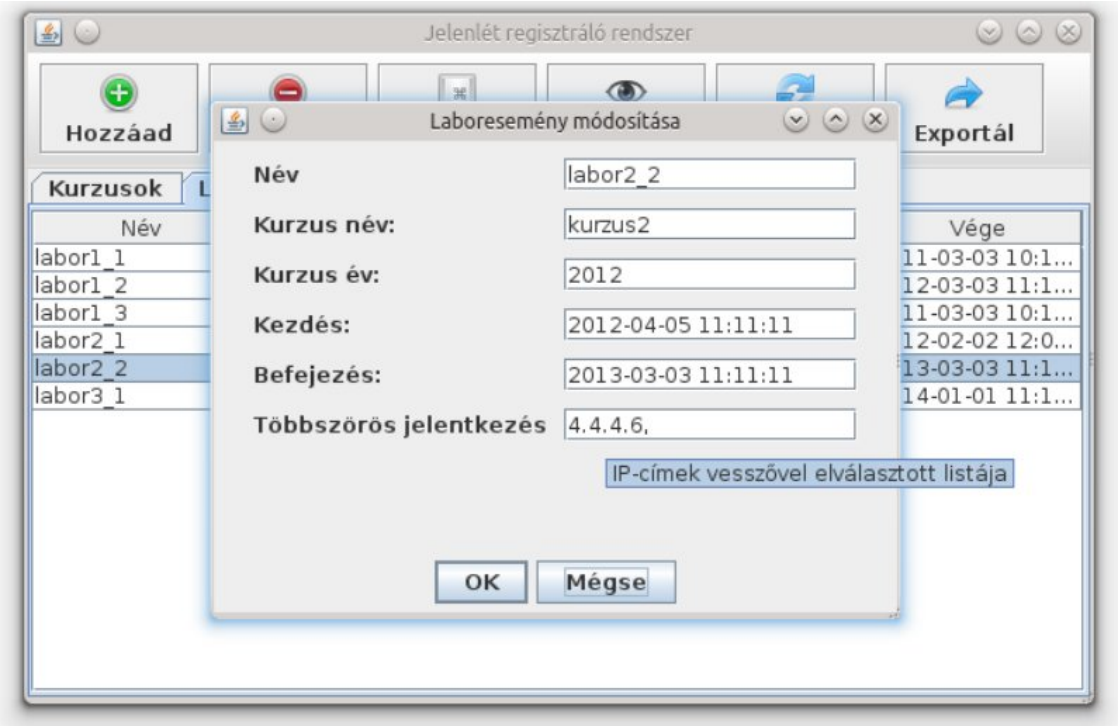


Mind a már létező kurzusok, mind a már létező laboresemények egy-egy fülön, táblázatos formában jelennek meg. A hozzáad és a szerkeszt gombra egy új ablak nyílik, mellyel szerkeszteni lehet az éppen kijelölt elemet, vagy új elemet lehet hozzáadni.



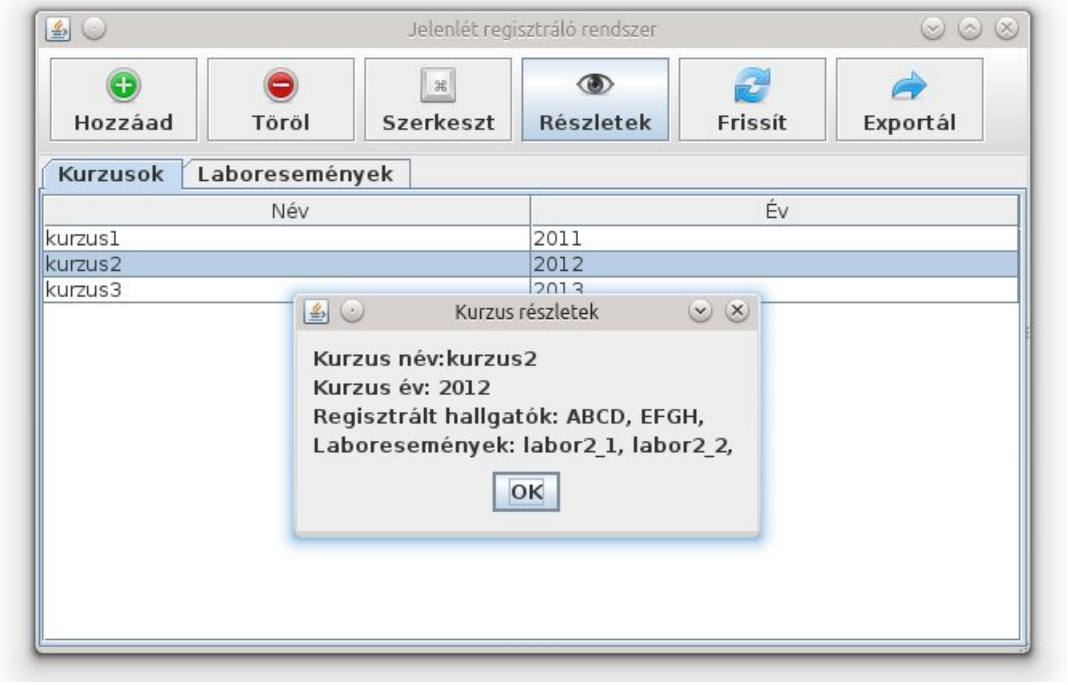


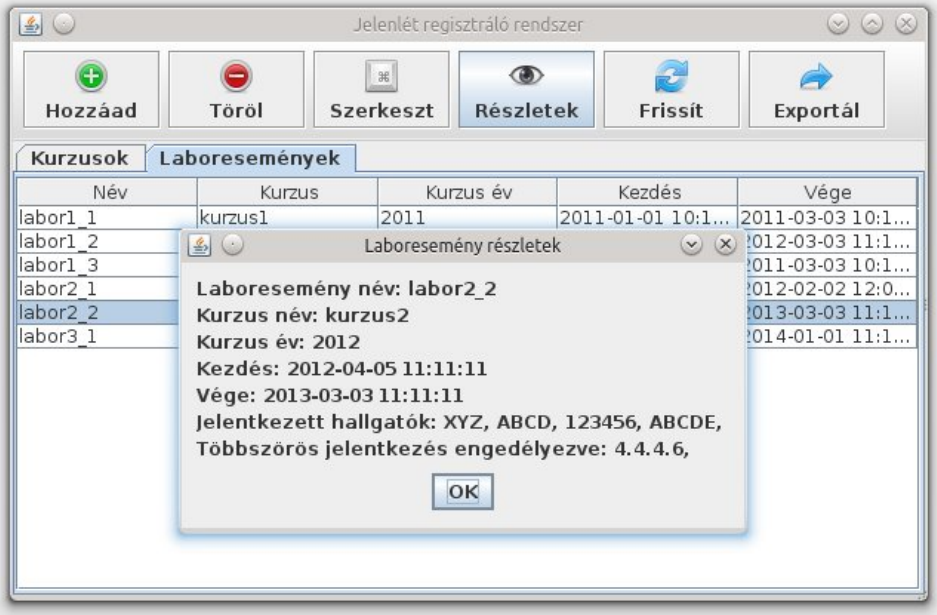
A kurzort az egyes beviteli mezők fölé víve, egy felugró üzenet jelenik meg, amely segít a beviteli formátum helyes megadásában.



A töröl gombra kattintva a táblázatban kijelölt elem törlődik, a frissít gombra kattintva pedig újra betöltődnek a táblázatba a kurzusok/laboresemények, attól függően hogy melyik fül volt aktív.

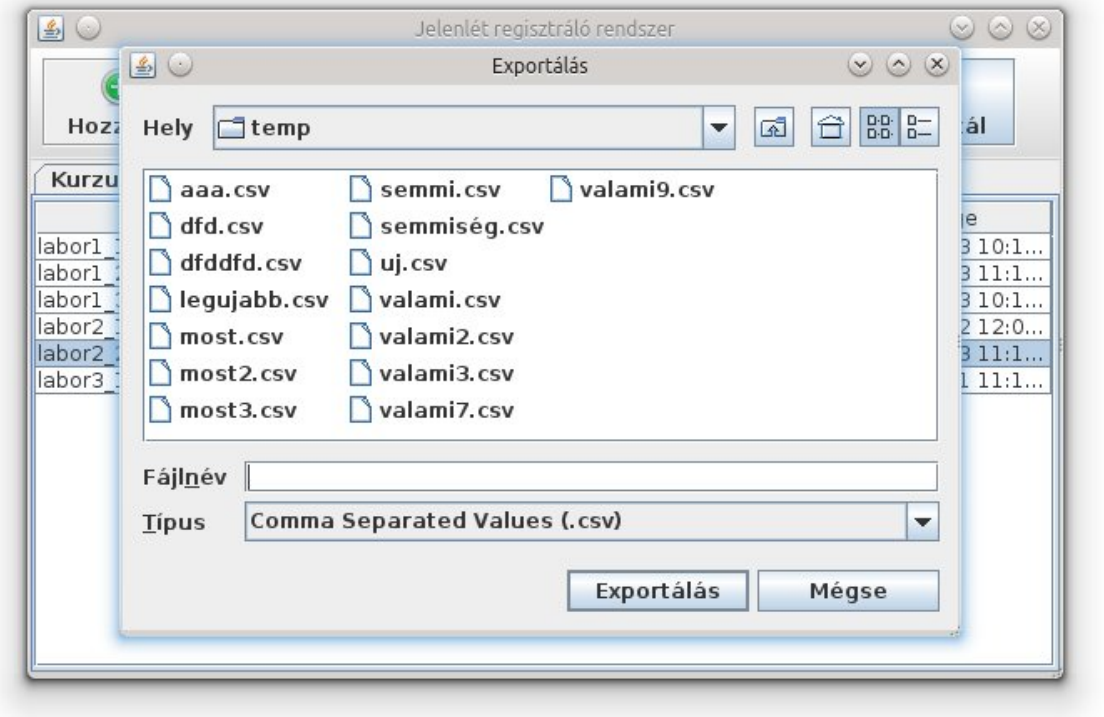
A részletek gomb egy új ablakot nyit, amely a táblázatban kijelölt elemről jelenít meg részletes információkat.





Az exportálás gomb egy fájlválasztó ablakot nyit, ahol a felhasználó megadhatja, hogy hova szeretné exportálni az adatbázisban található laboreseményeknek részvételi adatait. Az exportálás CSV formátumba történik (vesszőkkel elválasztva), amibe a laboreseményeik adatain kívül az

adott laboreseményen résztvevő hallgatók Neptun-kódjai is bekerülnek. A megjelenő ablakban a fájlnév mezőbe a kívánt fájlnevet kell beírni, amit a program kiegészít a ".csv" kiterjesztéssel. Ez a CSV fájl később könnyen beolvasható és kezelhető a Microsoft Excel programmal.



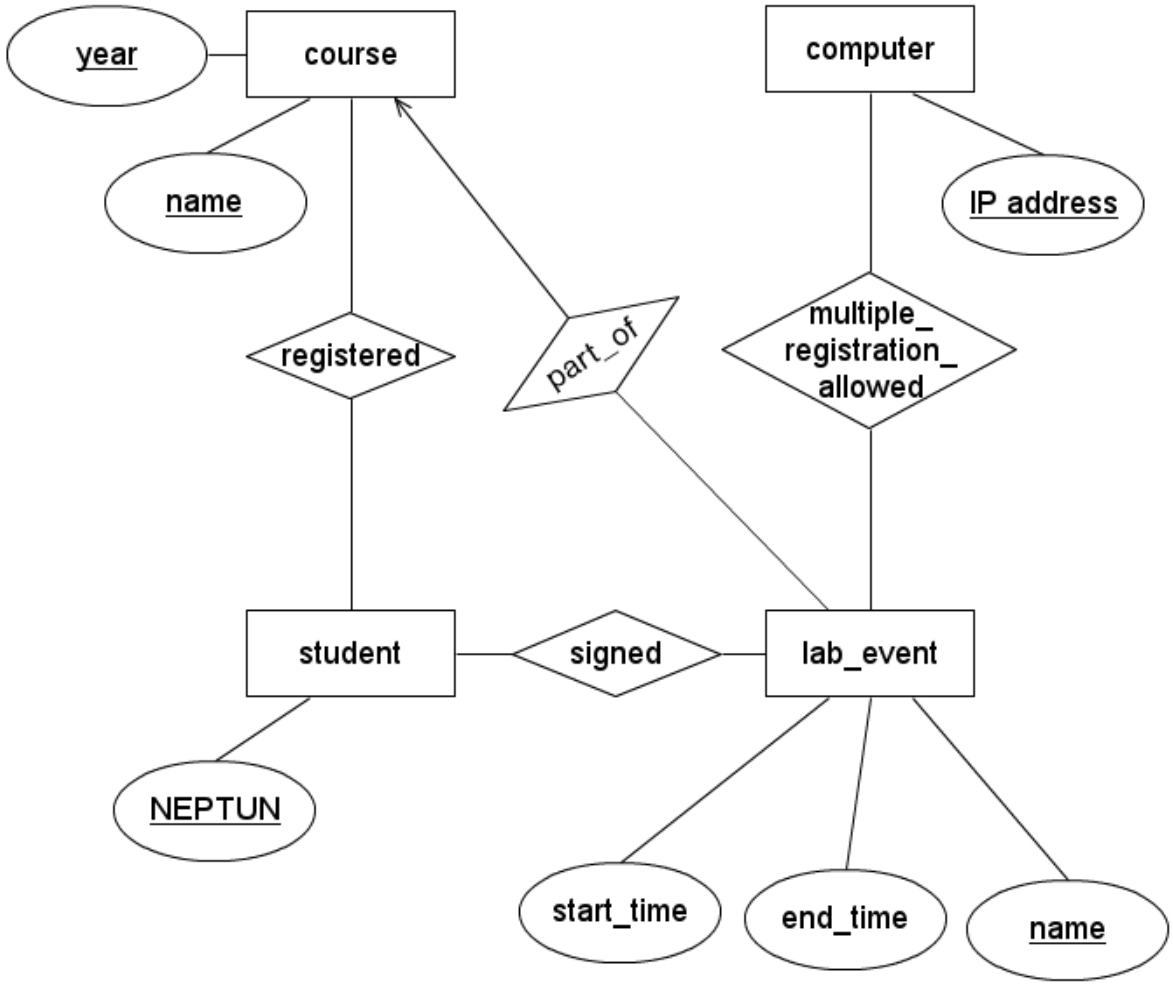
2. A kliens felülete

A kliens egy statikus HTML oldal. Két input mező található rajta, az egyikbe a felhasználó beírhatja a Neptun-kódját, míg a másikba annak a laboreseménynek a nevét, amire jelentkezni szeretne. Ezután a jelentkezés gombra kattintva a kérés elküldésre kerül a szerver felé.

Ha a hallható nem regisztrált Neptun-kóddal próbál meg jelentkezni, akkor egy figyelmeztető üzenetet kap. Ekkor a jelentkezés újbóli elküldésével megerősíti a jelentkezési szándékot.

3. Adatbázis terv

A szerver alkalmazás alatt egy SQLite adatbázis fut, melyet a szerver a JDBC Java API-n keresztül ér el. Az adatbázis entitás reláció diagrammal reprezentált modellje:



A szükséges adattábák és hozzá tartozó attribútumok

A következőkben a táblák vannak felsorolva, zárójelben az attribútumokkal. Az aláhúzott attribútumok kulcsot jelentenek az adott táblában. A *dőlt* attribútumok másik tábla kulcsát jelentik.

- course(year, name)
- student(neptun)
- computer(ip_address)
- lab_event(name, *part_of_course_name*, *part_of_course_year*, start_time, end_time)
- registered(*course_name*, *course_year*, *student_neptun*)
- signed(*lab_event_name*, *student_neptun*)
- multiple_registration_allowed(*lab_event_name*, *computer_ip_address*)

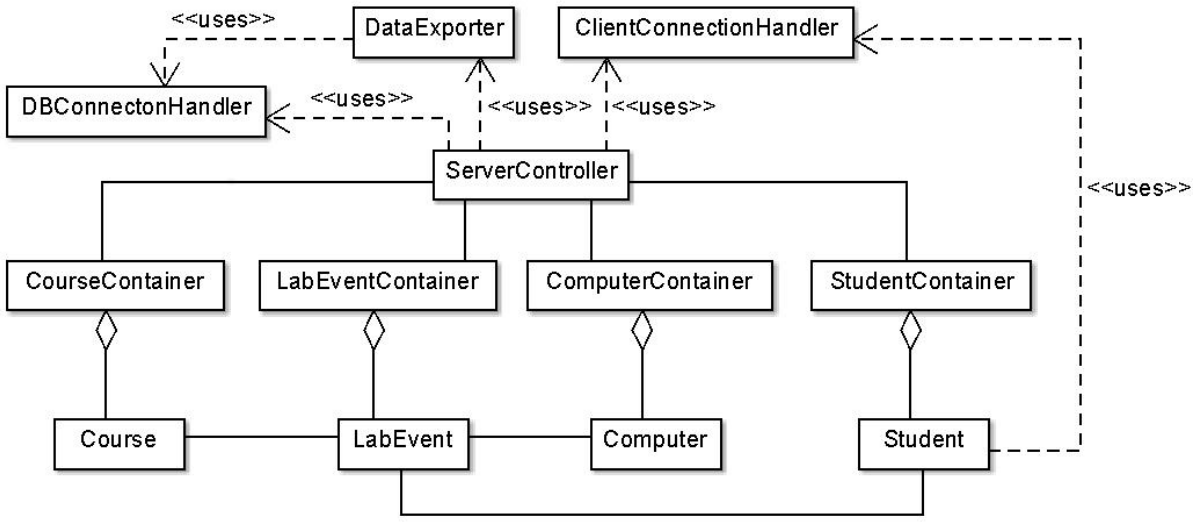
4. Kliens és szerver közötti kommunikáció

A kliens és szerver közötti kommunikáció HTML GET paramétereken keresztül történik. A szerver két paramétert fogad el a gyökér vagy az index.html lekérésénél. Ez a két paraméter a hallgató Neptun-kódja, valamint a laboresemény neve, amelyikre jelentkezni kíván. A hallgató a jelentkezés eredményét az erre kapott válaszból tudja meg. A jelentkezés eredménye lehet:

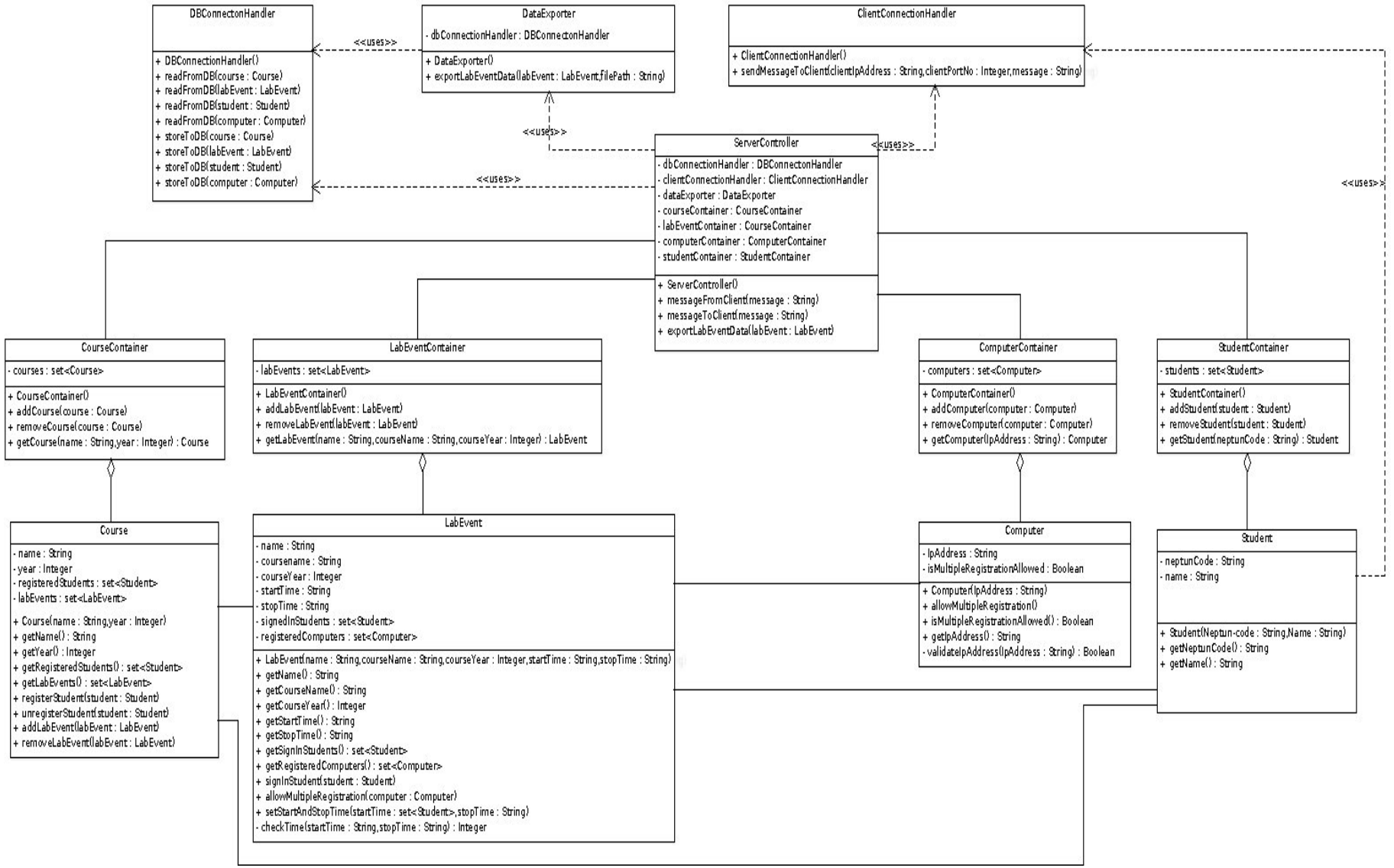
- Sikeres: a hallgató feliratkozik a laboreseményre.
- Sikertelen, ha a laboresemény jelenleg nem aktív.
- Sikertelen, ha a hallgató érvénytelen vagy tartományon kívüli IP-címmel próbál jelentkezni.
- Sikertelen, ha az adott laborgépről már történt jelentkezés az adott laboreseményre, és nincs engedélyezve a többszörös jelentkezés.
- Sikertelen, ha a hallgató már jelentkezett az adott laboreseményre.

5. Osztálydiagram

1. Egyszerűsitett osztálydiagram



2. Teljes osztálydiagram


DBConnectonHandler
+ DBConnectionHandler()
+ readFromDB(course : Course)
+ readFromDB(labEvent : LabEvent)
+ readFromDB(student : Student)
+ readFromDB(computer : Computer)
+ storeToDB(course : Course)
+ storeToDB(labEvent : LabEvent)
+ storeToDB(student : Student)
+ storeToDB(computer : Computer)
<<uses>>
DataExporter
- dbConnectionHandler : DBConnectonHandler
+ DataExporter()
+ exportLabEventData(labEvent : LabEvent,filePath : String)
<<uses>>
ClientConnectionHandler
+ ClientConnectionHandler()
+ sendMessageToClient(clientIpAddress : String,clientPortNo : Integer,message : String)
<<uses>>
<<uses>>
ServerController
- dbConnectionHandler : DBConnectonHandler
- clientConnectionHandler : ClientConnectionHandler
- dataExporter : DataExporter
- courseContainer : CourseContainer
- labEventContainer : CourseContainer
- computerContainer : ComputerContainer
- studentContainer : StudentContainer
+ ServerController()
+ messageFromClient(message : String)
+ messageToClient(message : String)
+ exportLabEventData(labEvent : LabEvent)
<<uses>>
CourseContainer
- courses : set<Course>
+ CourseContainer()
+ addCourse(course : Course)
+ removeCourse(course : Course)
+ getCourse(name : String,year : Integer) : Course
LabEventContainer
- labEvents : set<LabEvent>
+ LabEventContainer()
+ addLabEvent(labEvent : LabEvent)
+ removeLabEvent(labEvent : LabEvent)
+ getLabEvent(name : String,courseName : String,courseYear : Integer) : LabEvent
ComputerContainer
- computers : set<Computer>
+ ComputerContainer()
+ addComputer(computer : Computer)
+ removeComputer(computer : Computer)
+ getComputer(IpAddress : String) : Computer
StudentContainer
- students : set<Student>
+ StudentContainer()
+ addStudent(student : Student)
+ removeStudent(student : Student)
+ getStudent(neptunCode : String) : Student
Course
- name : String
- year : Integer
- registeredStudents : set<Student>
- labEvents : set<LabEvent>
+ Course(name : String,year : Integer)
+ getName() : String
+ getYear() : Integer
+ getRegisteredStudents() : set<Student>
+ getLabEvents() : set<LabEvent>
+ registerStudent(student : Student)
+ unregisterStudent(student : Student)
+ addLabEvent(labEvent : LabEvent)
+ removeLabEvent(labEvent : LabEvent)
LabEvent
- name : String
- coursename : String
- courseYear : Integer
- startTime : String
- stopTime : String
- signedInStudents : set<Student>
- registeredComputers : set<Computer>
+ LabEvent(name : String,courseName : String,courseYear : Integer,startTime : String,stopTime : String)
+ getName() : String
+ getCourseName() : String
+ getCourseYear() : Integer
+ getStartTime() : String
+ getStopTime() : String
+ getSignInStudents() : set<Student>
+ getRegisteredComputers() : set<Computer>
+ signInStudent(student : Student)
+ allowMultipleRegistration(computer : Computer)
+ setStartAndStopTime(startTime : set<Student>,stopTime : String)
- checkTime(startTime : String,stopTime : String) : Integer
Computer
- IpAddress : String
- isMultipleRegistrationAllowed : Boolean
+ Computer(IpAddress : String)
+ allowMultipleRegistration()
+ isMultipleRegistrationAllowed() : Boolean
+ getIpAddress() : String
- validateIpAddress(IpAddress : String) : Boolean
Student
- neptunCode : String
- name : String
+ Student(Neptun-code : String,Name : String)
+ getNeptunCode() : String
+ getName() : String

6. Osztályok leírása

1. ServerController
A szerver irányításáért felelős osztály. A szerver osztályai rajta keresztül érik el az adatbázist, valamint tartja a kapcsolatot a különböző gyűjtőosztályok (containers) között. További feladata a klienstől érkező üzenetek kezelése a ClientConnectionHandler osztály segítségével.

2. ClientConnectionHandler
A klienstől érkező, valamint a kliensnek küldendő üzenetek kezelésére szolgáló osztály. Továbbítja az üzeneteket a megfelelő kliensnek (IP-cím, portszám szükséges a címzéshez).

3. DataExporter
Lehetővé teszi a laboreseményeknek adatainak háttértárra való mentését. Az adatokat az adatbázsiból olvassa ki, majd ezután a megadott útvonalra egy CSV fájlba menti.

4. DBConnectionHandler
Az SQLite adatbázis és az üzleti logikát biztosító réteg közötti interfészt megvalósító osztály. Az üzleti logikát biztosító rétegtől kapott parancsokat SQL utasításokká fordítja át, melyeket továbbít az adatbázis felé.

5. CourseContainer
A kurzusokat tartalmazó gyűjtőosztály.

6. Course
A kurzusokat reprezentáló osztály. A laborvezetőnek lehetősége van kurzusokat létrehozni és törölni, laboreseményeket rendelni a kurzusokhoz és ezeket törölni, valamint hallgatókat rendelni a kurzushoz, és őket törölni. Ha a laborvezető nem adja hozzá a hallgatókat a kurzus hallgatóinak listájához, akkor a hallgatók jelentkezéskor figyelmeztető üzenetet kapnak.

7. LabEventContainer
A laboreseményeket tartalmazó gyűjtőosztály.

8. LabEvent
A laboreseményeket reprezentáló osztály. A laborvezetőnek lehetősége van kurzushoz rendelt laboreseményt létrehozni, melynek létehozásakor kötelező jelleggel meg kell adnia a laboresemény kezdési és befejezési idejét. Ezen időintervallumon belül jelentkezhetnek a hallgatók az adott laboreseményre. Ezt az időintervallumot a kezdeti időpont előtt a laborvezető módosíthatja.

9. ComputerContainer
A laborgépeket tartalmazó gyűjtőosztály.

10. Computer
A laborgépeket reprezentáló osztály. Laborgépeket abból a célból szükséges azonosítani, hogy a laborvezető megadhat olyan laborgépeket, amelyekről elfogadunk többszörös jelentkezést is (alapértelmezésben egy laborgépről egy laboreseményre egy jelentkezés fogadható el,

viszont előfordulhat, hogy egy laborgépen több hallgató is dolgozik, valamint vannak olyan hallgatók, akik saját laptoppal dolgoznak az órákon, így nekik is lehetőséget lehet biztosítani a laboreseményre történő jelentkezéshez.) A többszöri jelentkezés engedélyezése mindig csak az adott laboreseményhez kerül beállításra, amit a laborvezető később törölhet is. Egy újabb laboreseménynél újra be kell állítani azokat a laborgépeket, amelyekről engedélyezve van a többszörös jelentkezés.

11. StudentContainer

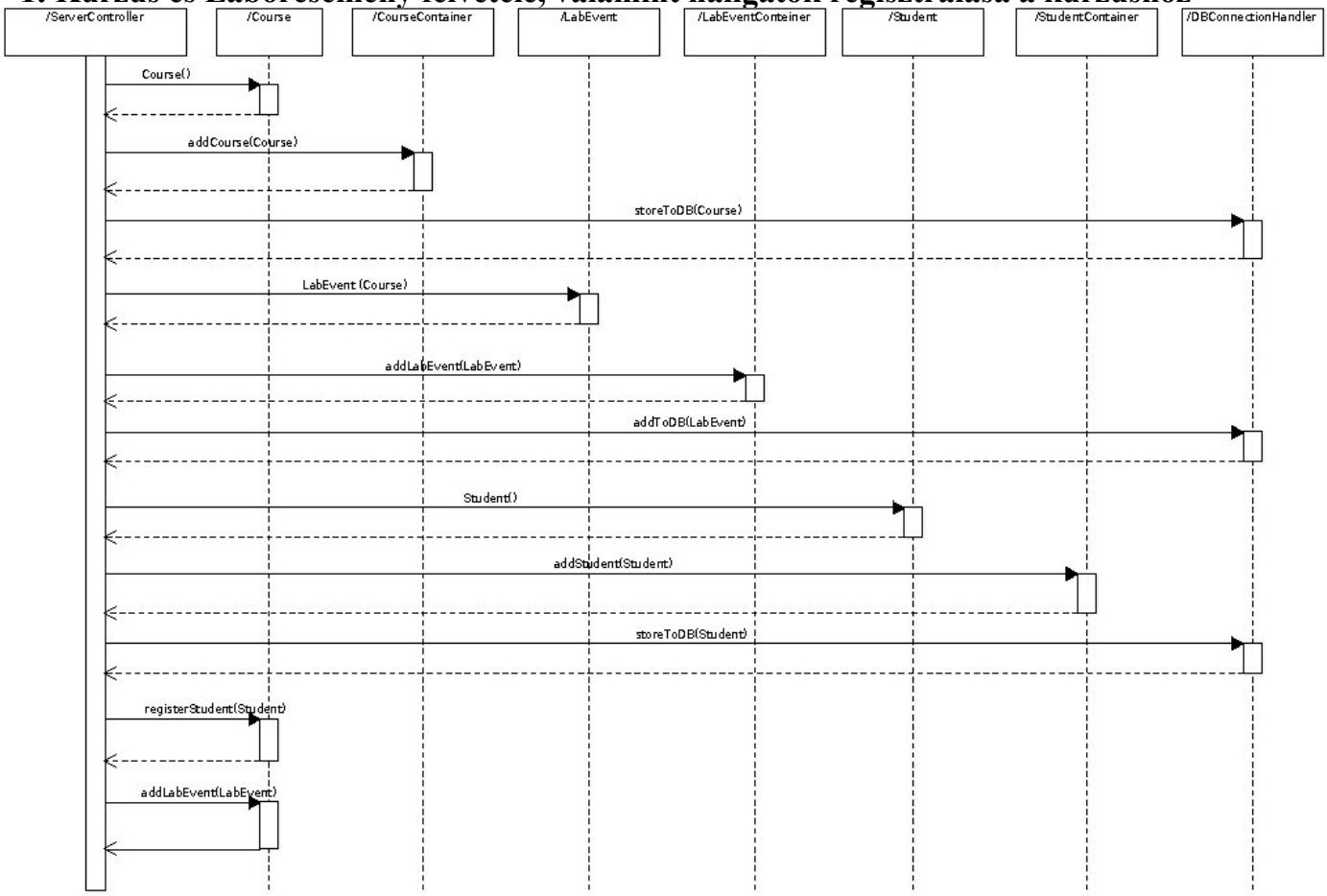
A hallgatókat tartalmazó gyűjtőosztály.

12. Student

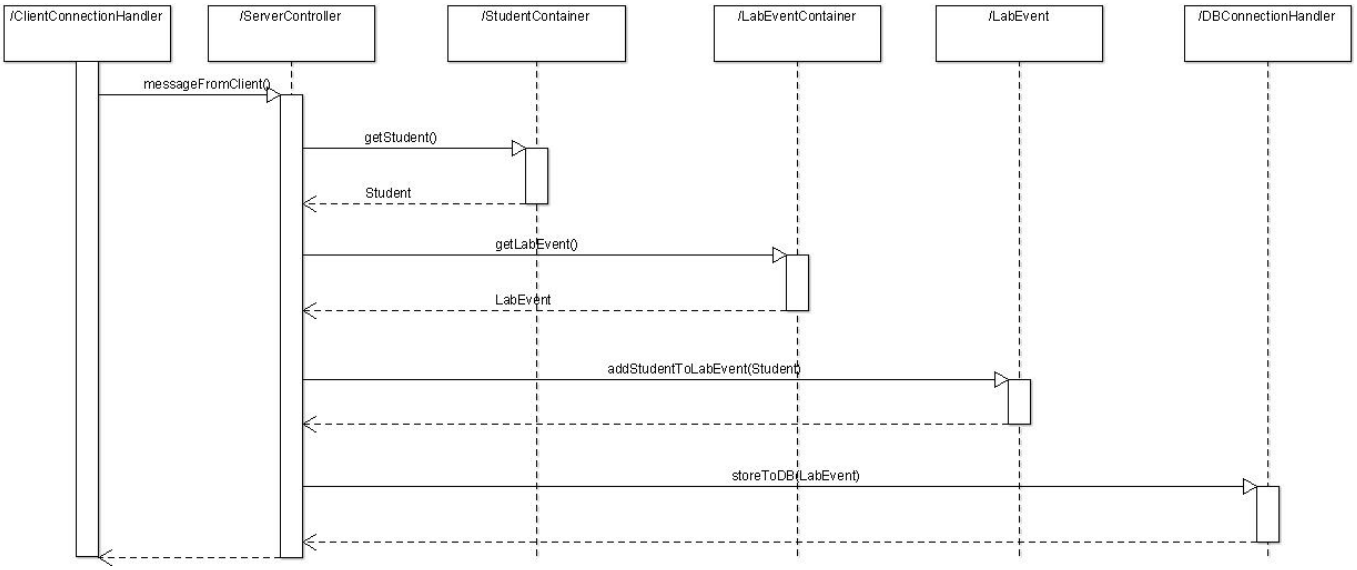
A hallgatókat reprezentáló osztály. Egy hallgatóról a Neptun-kódját tároljuk, később ez alapján azonosítja őt a rendszer. Jelentkezéskor a hallgatók Neptun-kódja ellenőrzésre kerül, ha a laboreseményhez tartozó kurzusra a laborvezető korábban nem regisztrálta a hallgatót, akkor a hallgató jelentkezéskor figyelmeztetést kap.

7. Szekvenciadiagramok

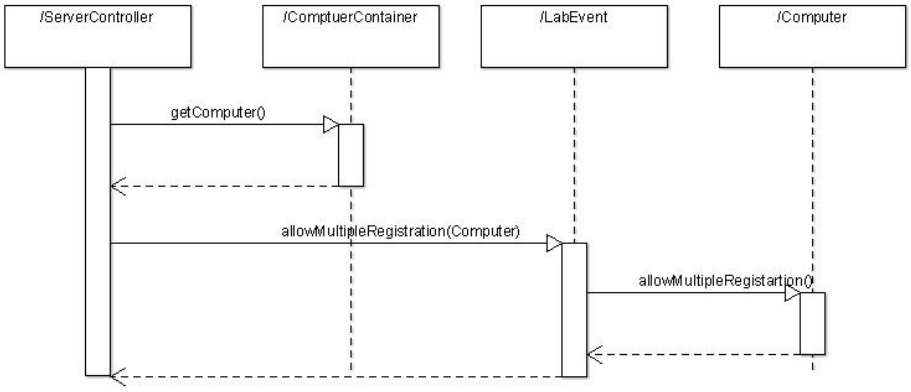
1. Kurzus és Laboresemény felvétele, valamint hallgatók regisztrálása a kurzushoz



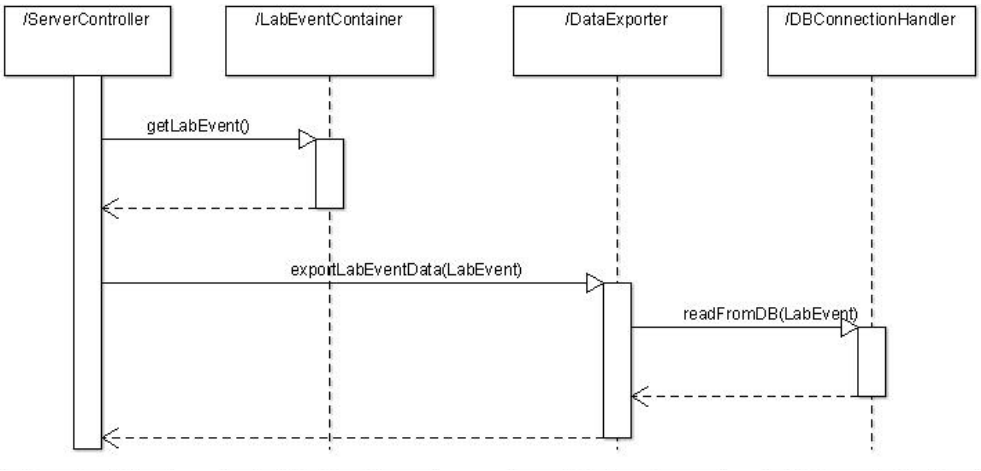
2. Hallgató jelentkezése a laboreseményre



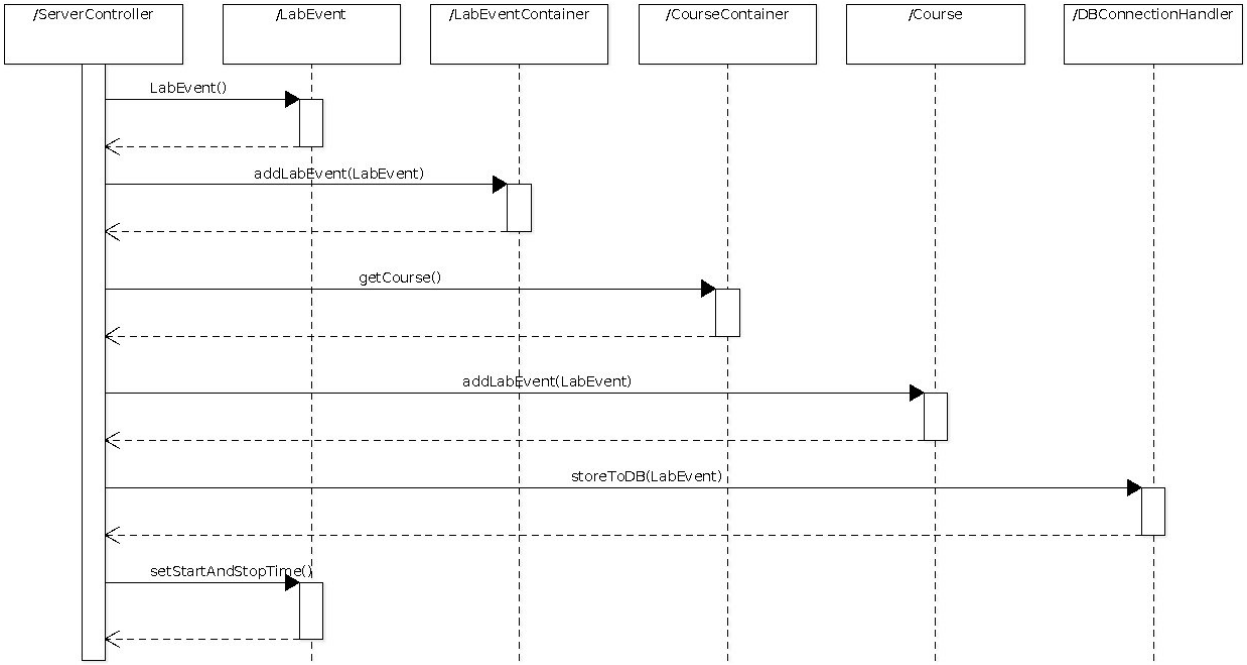
3. Többszörös jelentkezés engedélyezése



4. Laboresemény adatainak CSV fájlba mentése

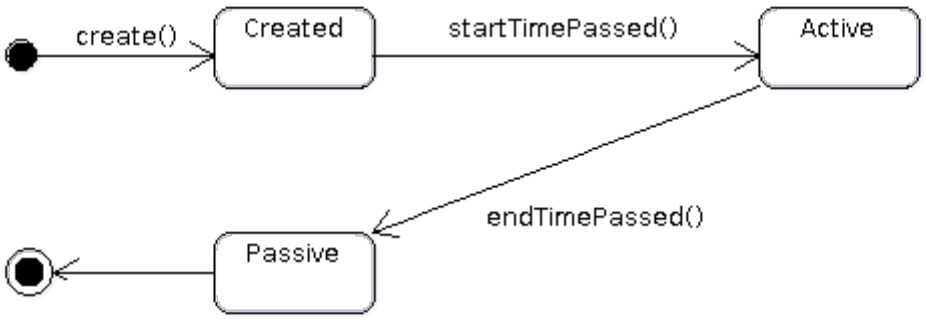


5. Laboresemény létrehozása és időintervallum módosítása



8. Állapotdiagramok

1. A létrehozott laboreseményeknek állapotai (LabEvent oszály)



9. Tesztelési terv

1. Manuális integrációs tesztesetek

■ A felhasználó regisztrált Neptun-kóddal sikeresen jelentkezik a laboreseményre

Előkövetelmény

A szerver egy előre meghatározott tartalmú adatbázissal fut. Az adatbázis tartalma: 2 kurzus, mindkettőhöz 2-2 laboresemény. Az egyik laboreseményhez regisztrálva van két Neptun-kód, melyből az egyik a tesztben megadott, a többi laboreseményhez is 2-2 a tesztben megadottól külöböző Neptun-kód. Olyan gép, melyről többen is regisztrálhatnak, nincs megadva.

Teszt lépései

- A felhasználó megnyitja a weblapot egy böngészőben
- Megadja a Neptun-kódot a felületen

A teszt sikeressége

A megfelelő Neptun-kód, mint a megfelelő laboreseményhez jelentkezett jelenik meg az adatbázisban, és a szerver felhasználói felületén (frissítés után).

■ A felhasználó előre nem regisztrált Neptun-kódot ad meg, és figyelmeztetés után sikeresen regisztrál a laboreseményre

Előkövetelmény

A szerver egy előre meghatározott tartalmú adatbázissal fut. Az adatbázis tartalma: 2 kurzus, mindkettőhöz 2-2 laboresemény. Minden laboreseményhez regisztrálva can 2-2 a tesztben megadottól külöböző Neptun-kód. Olyan gép, melyről többen is regisztrálhatnak, nincs megadva.

Teszt lépései

- A felhasználó megnyitja a weblapot egy böngészőben
- Megadja a Neptun-kódot a felületen
- A figyelmeztetés ellenére jelentkezik

A teszt sikeressége

A megfelelő Neptun-kód mint a megfelelő laboreseményhez jelentkezett jelenik meg az adatbázisban, és a szerver felhasználói felületén (frissítés után).

■ Egy számítógépről több felhasználó is megpróbál jelentkezni, azonban ez nem megengedett

Előkövetelmény

A szerver egy előre meghatározott tartalmú adatbázissal fut. Az adatbázis tartalma: 1 kurzus, ahhoz egy laboresemény. A kurzushoz két Neptun-kód van regisztrálva. Nincs megadva olyan gép, melyről többen is jelentkezhetnek.

Teszt lépései

- A kliens megnyitása egy böngészőben
- Első megadott Neptun-kóddal jelentkezés
- Második megadott Neptun-kóddal jelentkezés

A teszt sikeressége

Az első jelentkezés sikeres a megfelelő laboreseményre, azonban a második jelentkezés sikertelen. Erről a második felhasználót hibaüzenet tájékoztatja. A jelentkezés látható mind az adatbázisban, mind a szerver felhasználói felületén (frissítés után).

1. Egy számítógépről több felhasználó is sikeresen jelentkezik

Előkövetelmény

A szerver egy előre meghatározott tartalmú adatbázissal fut. Az adatbázis tartalma: 1 kurzus, ahhoz egy laboresemény. A kurzushoz két Neptun-kód van regisztrálva. Egy olyan gép van megadva, melyről többen is jelentkezhetnek.

Teszt lépései

- A kliens megnyitása egy böngészőben, a megfelelő IP címmel rendelkező számítógépről, melyről többen is jelentkezhetnek
- Első megadott Neptun-kóddal jelentkezés
- Második megadott Neptun-kóddal jelentkezés

A teszt sikeressége

Mindkét jelentkezés sikeres. A jelentkezés látható mind az adatbázisban, mind a szerver felhasználói felületén (frissítés után).

2. Automatikus tesztesetek

Automatikus tesztesetek a szerverhez léteznek. A tesztesetek a Git repository laborreg_server/test mappájában találhatóak. A tesztek rövid leírásának a tesztesetek nevei felelnek meg, melyek tartalmazzák hogy az adott teszteset mit tesztel. Külön teszt leírás nem készül az egyes tesztesetekhez, azoknak elég egyszerűeknek és tisztáknak kell lenniük ahhoz, hogy egyből át lehessen látni az adott teszteset mit tesztel. A teszteknél különösen, de bármilyen forráskódnál a kód készítésére a Robert C. Martin által készített Clean Code: A Handbook of Agile Software Craftsmanship című könyve az irányadó.

10. Tesztelést támogató eszközök

1. Unit test framework

A program unit tesztelése a JUnit 4 framework használatával történik. Ez a standard Java része.

2. Mocking framework

A unit tesztben a mockok a JUnittal együttműködni képes Mockito framework 1.9.5-ös verzója segítségével készülnek. A Mockitot a következő honlapról lehet elérni: http://code.google.com/p/mockito/

11. Fejlesztést támogató eszközök

1. Forráskód

A forráskód verziókezelésére a Git verziókezelő rendszer használatos, a tárolás a GitHubon történik. A GitHub repositoryt a következő weboldalról lehet elérni: https://github.com/esgott/szoft_arch_hazi

2. Adatbázis elérés

A program az adatbázis eléréséhez egy JDBC-hez készített SQLite külső libraryt használja. Ez megtalálható a http://code.google.com/p/sqlite-jdbc/ weboldalon.

3. Eclipse WindowBuilder plugin

A GUI-k készítésére és szerkesztésére a Google által készített WindowBuilder Eclipse plugin került felhasználásra. Ez az Eclipse "Install new software" menüpontjából telepíthető. Az Eclipse fejlesztőkörnyezet a http://www.eclipse.org weboldalról telepíthető.

12. Biztonsági kérdések

1. Hálózati biztonság

A szerver és a kliens csak egyetemi laborhálózatból használható. Erről a hálózatról feltételezhetjük, hogy biztonságos, és tűzfallal védett, tehát csak a belső hálózaton tartózkodó számítógépek tudnak kommunikálni a szerverrel. Ebből a feltételezésből kiindulva, nem szükséges újabb hálózati védelmi szintet implementálni a rendszerbe.

2. LDAP/AD authentikáció

A hallgatók nem-adminisztrátor jogú felhasználói is domain userek, tehát nem kell jelszóval védeni a szerverre történő jelentkezést. Ezt egy támadó csak a domain-be történő belépéssel tudja megkerülni, viszont ennek a védelme az egyetemi hálózat feladata.

3. SQL injection

Népszerű, de egyre veszélytelenebb támadási forma a webes alkalmazások ellen. Ennek kivédésére a java.sql.PreparedStatement osztályt használja a szerver, amely minden SQL utasítás megadása előtt ellenőrzi az utasításban szereplő egyes paramétereket. Ha nem megfelelő paramétert talál (pl. egy másik SQL utasítás), akkor elutasítja a kérést, ezáltal a szerver teljesen védetté válik bármilyen SQL injection típusú támadás ellen.

4. HTTPS

A szerver-kliens kommunikáció http-t igényel. HTTPS protokollt általában bizalmas adatok továbbítására használunk, viszont a kliens által küldött Neptun-kódok nyilvános adatok (természetesen a hozzá tartozó név nélkül). A hallgatók kurzuson való részvétele is nyilvános, mivel ezt bárki megtekinthetni a Neptun hallgatói információs rendszerben. Mivel a szerver és a kliens között csak nyilvános adat kerül átküldésre, ezért felesleges lenne a bonyolultabb és összetettebb https protokoll használata.

5. IP tartomány védelme

A szerver indításakor megadható egy IP-cím tartomány, amelyről a szerver elfogadja a hallgatók jelentkezését (ennek hiányában egy alapértelmezett tartomány kerül megadásra). Az ezen kívülről történő jelentkezéseket a szerver kiszűri, viszont a válaszában nem említi meg az érvényes tartományt (ugyanis ez egy újabb támadási pont lehetne).

13. Továbbfejlesztési lehetőségek

1. Szerver oldali hibaüzenetek minőségének javítása

A szerver oldali hibaüzenetek (amelyek a java beépített Logger osztályának segítségével kerülnek kiírásra) néha elnagyolt, nem pontos információt adnak a történt eseményekről. Továbbá a szerver által dobott exception-ök sem mindig írják le elég pontosan a hibaesetet. Ezeken később javítani lehetne, akár egységesíteni is lehetne őket.

2. Dátumkezelés, érvénytelen dátum elutasítása

A java.util.Date osztály sajátosságait kihasználva a szerver egy laboresemény időablakának beállításakor elfogad érvénytelen dátumot is. Ilyen típusú dátum megadásánál a Date osztály automatikusan átalakítja a dátumot egy érvényes, de későbbi dátumra. A későbbiekben megfontolható lenne a dátumok más beépített osztály segítségével történő kezelése, vagy jobb hibakezelési eljárások írása a dátumok ellenőrzésére.

3. Nyelvek kezelése, többnyelvűsítés

A szoftver fejlesztése angol nyelven történt, míg a hallgatók felé történő kommunikáció magyar nyelven zajlik, ezért néhány helyen előfordulhat, hogy keveredik a magyar és az angol hibaüzenet. (pl. az exception szövege angolul van, a hiba pontos leírása magyarul.) Ezt a jövőben egységesíteni lehet.
Egy másik továbbfejlesztési lehetőség ezen a területen a program átalakítása többnyelvűvé. Elékpzelhető, hogy a laboreseményeket magyarul nem beszélő hallgatók is látogatják, az ő segítségükre be lehetne állítani a szerveren, hogy milyen nyelven kommunikáljon a hallgatókkal.

4. CSV fájlba történő exportálás bővítése

A szerver jelenleg az adatbázisban tárolt összes laboresemény adatait, valamint az ezeken részt vett hallgatók Neptun-kódjait tárolja. Ezt a jövőben finomítani lehetne például a kívánt laboresemények kiválasztásával, vagy csak a megadott kurzusokhoz tartozó laboresemények adatainak exportálásával. További ötletként felmerülhet egy laboresemény lezárultával annak automatikus exportálása.

5. Kliens felület fejlesztése

A kliens jelenleg minimális design-nal rendelkezik, ugyanis az általa ellátott funkcióhoz nem szükségeltetik különös grafika. Manapság egy ilyen weboldal kereskedelmi forgalomban nem feltétlenül állná meg a helyét, ezért szükség lehetne akár statikus (pl. CSS) akár dinamikus (pl. AJAX) eszközök bevezetésére. Ez csak kis mértékben javítaná a kliens használhatóságát, viszont a kinézetét nagy mértékben szebbé tenné.