Софийски университет „Св. Кл. Охридски”

Факултет по математика и информатика

Магистърска програма
„Софтуерни технологии”

Предмет: Обектно-ориентиран анализ и проектиране на софтуерни системи

Зимен семестър, 2024/2025 год.

Тема № 38: Система за управление на студентската информация

Курсов проект

Автори:

Даниел Халачев, фак. № 4MI3400603

Стефан Велев, фак. № 0MI3400521

януари, 2025 г.

гр. София

Съдържание

1. Въведение

В България има 52 акредитирани висши училища. Всяко едно от тях обучава хиляди студенти, за всеки от които се генерира и съхранява огромно множество от информация. За да бъдат тези данни използваеми, те трябва да бъдат съхранявани в подходящи структури и на подходящо място по единен начин.

За съжаление, твърде голяма част от съществуващите системи не решават изцяло този проблем, а в допълнение пораждат и други проблеми.

Ако разгледаме Системата за управление на студентската информация (СУСИ) на Софийския университет, тя притежава множество функционалности, които други системи (например УИСС на Техническия университет) не поддържат - записване на избираеми дисциплини, преглед на оценки и кредити по дисциплини, заплащане на семестриална такса, попълване на оценки. Същевременно обаче, подобно на голяма част от студентските информационни системи в България, тя все още не поддържа елементарни, но наистина необходими функционалности - преглед на седмичния график, заплащане на всички видове такси, изцяло електронно попълване на оценки и протоколи, съхраняване и категоризиране на цялата необходима информация за документиране на образователния процес.

В допълнение, нито една от разгледаните от нас информационни системи не спазва принципите на качествения потребителски интерфейс - достигане до функционалност с до 3 кликвания на мишката, акцент върху разпознаване, а не запомняне, интуитивно и смислено разположение на бутоните, и други. Затова не е изненада негативната обратна връзка относно потребителското изживяване - студенти и преподаватели често не харесват съществуващите системи (включително СУСИ), а някои посочват и съмнения в сигурността на използваната от тях система - мнения, които са основателни с оглед на използваните технологии (в случая със СУСИ - ASP.NET, 2003 година).

Предложената от нас система - **SIMS (Student Information Management System)**, цели да разреши проблема с електронното управление на студентската информация с основен фокус върху Софийския университет. Вниманието е насочено към няколко основни насоки за развитие:

- подобряване на потребителското изживяване за студенти и преподаватели чрез съвременен многоплатформен достъп
- улесняване на **дейността на студентите** чрез предоставяне на необходимата за тях информация, включително успех от изпити, *седмичен график*, *изчисление на натрупаните и оставащите кредити*, *заплащане на всички видове такси онлайн, записване на избираеми дисциплини онлайн*
- улесняване на **дейността на преподавателите** чрез *изцяло електронно попълване на оценки и протоколи* без отражение върху сигурността,

съхраняване и автоматично предоставяне (при поискване) на администраторите на данни под формата на доклади полезна информация за учебната дейност
- улесняване на **дейността на администраторите на информация** чрез по-голямо дигитализиране на *административните услуги* и намаляване на човекопотока на място, както и улесняване на възможността за управление на дисциплини
- подобряване на **сигурността** чрез използване на съвременни технологии, чиято поддръжка не зависи от корпорации (или поне са с гарантиран срок на поддръжка /lifetime/).

Решението на проблема е представено по-долу с помощта на UML диаграми в средата за моделиране Visual Paradigm (и изключение на Timing и Object диаграмите, направени с PlantUML).

В секция *2. Анализ на решението* ще разгледаме въпросите, които са типични за обектно-ориентирания анализ на една софтуерна система. В подсекция *2.1 Потребителски изисквания и работен процес* разглеждаме софтуерните изисквания на системата – както функционалните, така и нефункционалните изисквания. Ще допълним изложението на подсекцията с диаграма на случаи на употреба /с потоци от събития/ и диаграми на дейностите. В подсекция *2.2 Примерен графичен интерфейс* прилагаме фигури на примерен графичен интерфейс на SIMS. В подсекция *2.3 Диаграми на анализа* представяме резултатите от анализа на проблема под формата на клас диаграми на анализа (с класове със стереотипи) и диаграми на последователността и на комуникацията. За класове и обекти с множество възможни състояния ще бъдат представени диаграми на състоянието. В подсекция *2.4 Модел на съдържанието/данните* ще опишем основния модел на данните.

В секция *3. Дизайн* ще представим проектирането на системата в резултат на дейностите по обектно-ориентирания дизайн на системата и избраната софтуерна платформа. Ще разгледаме схема на софтуерната архитектура на решението с диаграма на внедряването, диаграми на класовете на дизайна (с ограничения, описани на OCL), диаграми на времето (за задаване на времена за синхронизация и комуникация в решението) и компонентна диаграма, демонстрираща логическата организация и взаимовръзки между основните компоненти. Ще илюстрираме решението с извадки от генериран сорс код в секция *8. Апендикс*.

В секция *4. Тестване* ще маркираме какво трябва да включват тестовите случаи и какви видове тестване се предвиждат в реалното изпълнение на проекта.

В секция *5. Заключение и възможно бъдещо развитие* ще обобщим резултатите от работата по проекта, както и ще споменем предимствата и ограниченията на използваните технологии. Ще коментираме възможни алтернативи, както и каква е използваемостта на подобни решения в практиката. Ще предложим насоки за бъдещо развитие на конкретното решение.

В секция *6. Разпределение на работата* ще опишем как е била разпределена работата между авторите на проекта.

В секция *7. Използвани ресурси* ще цитираме всички ресурси, които са били необходими на авторите за създаването на проекта.

2. Анализ на решението

2.1 Потребителски изисквания и работен процес

2.1.1 Функционални изисквания

Потребителите на системата ще бъдат от следните роли (с посочени ключови характеристики):

- студент (лице, което учи в университета) - имена, факултет, специалност, факултетен номер
- преподавател (лице, което преподава в университета) - имена, дисциплини, които води
- администратор на данни (лице, което извършва административни дейности по отношение на информацията на другите потребители в системата)
- администратор на системата (лице, което отговаря за поддържането на хардуерните и софтуерните компоненти на системата)

Ще започнем по-подробното описване на функционалните системни изисквания от гледна точка на отделните роли в системата:

1. **Гост-потребител**
   - **1.1. Вписване:**
     - 1.1.1. Системата ще предоставя възможност за вписване в системата.
     - 1.1.2. За вход системата ще изисква валидни потребителско име и парола.
       - 1.1.2.1. Системата трябва да позволява имейл да бъде валидно потребителско име.
     - 1.1.3. Ако липсва запис за потребител в базата данни, системата трябва да визуализира съобщение за грешка.
     - 1.1.4. Системата ще предоставя възможност за възстановяване на забравена парола чрез изпращане на имейл за промяна на паролата към потребителя.
       - 1.1.4.1. Имейлът за възстановяване на паролата би могъл да съдържа линк, препращащ към страница за промяна на паролата.
       - 1.1.4.2. Имейлът за възстановяване на паролата би могъл да съдържа код, който системата ще очаква да бъде въведен, за да завърши промяната на паролата.
     - 1.1.5. След успешен вход потребителят ще бъде препратен към началната страница на своя профил.
   - **1.2. Преглед на новини:**

1.2.1. Системата би могла да показва публична информация за предстоящи и протичащи в момента на ползване кампании за плащане на такси, записване на избираеми и други.

2. Студент и преподавател - общи изисквания

2.1. Преглед на седмично разписание:

2.1.1. Системата ще предоставя на студентите и преподавателите персонализиран ежеседмичен график на занятията, в които участват, под формата на таблица.

2.1.2. Системата ще предоставя информация единствено за дисциплините, записани от студента или водени от преподавателя.

2.1.3. Всяко занятие ще съдържа данните: име на дисциплина, място и час на провеждане, и ще има списък от етикети за метаданни, зададени от администратора на данни - например лекция/упражнение и задължителна/избираема.

2.1.4. Студентите и преподавателите ще могат да редактират личното си седмично разписание.

2.1.4.1. Редактирането ще позволява промяна на времето на провеждане.

2.1.4.2. Редактирането ще позволява скриване на занятието в разписанието.

2.1.4.3. Системата ще позволява връщане към разписанието по подразбиране.

2.2. Преглед на съобщения:

2.2.1. Студентите и преподавателите ще могат да преглеждат получените съобщения.

2.3. Изпращане на съобщения:

2.3.1. Потребителите ще могат да изпращат съобщения, въвеждайки:

2.3.1.1. адресат;

2.3.1.2. тема;

2.3.1.3. съдържание на съобщението.

2.3.2. Системата би могла да позволява прикачване на файлове към съдържанието на съобщенията.

2.3.3. Ако потребителят е преподавател, системата ще позволява адресиране на повече от един потребител - група, курс, специалност.

2.4. Системата би могла да предоставя възможност за настройка на имейл типа известия, които потребителят да получава спрямо ролята си

2.4.1. ***Типове известия за студент***: нова семестриална оценка, нова такса, успешно платена такса, ново съобщение, нова кампания, промяна на позиция в опашка за записване на избираема дисциплина, промяна в седмичното разписание.

2.4.2. ***Типове известия за преподавател***: ново съобщение, приет протокол, отхвърлен протокол, промяна в седмичното разписание.

2.5. Системата би могла да предоставя персонализирана начална страница с най-използваните дейности на потребителя.

2.5.1. Системата би могла да предоставя възможност за настройка на видимостта на преките пътища от началната страница.

2.6. Системата ще позволява на потребителите да виждат списък на последните 20 вписвания в профила си в системата.

2.6.1. Прегледът на вписванията ще включва информация за датата и часа, браузъра и типа на устройството.

2.6.2. Прегледът би могъл да включва информация за IP адреса на устройството, от което е направено вписването.

2.7. Смяна на паролата:

2.7.1. Системата ще предоставя изглед за смяна на паролата.

2.7.2. Системата ще изисква двуфакторно удостоверяване преди въвеждането на новата парола.

2.8. Изпращане на молба:

2.8.1. Системата би могла да предоставя възможност за електронно попълване и изпращане на заявление до администрацията на университета.

2.8.1.1. Потребителят би могъл да избира измежду готов списък от бланки

2.8.1.2. Администраторите биха могли да бъдат лицата, отговорни да качват готовите бланки в системата.

2.8.1.2.1. Системата би могла да предоставя начин за създаване на готовите бланки от администратор на данни с цел да се избегне нуждата от HTML специалист за създаването им.

3. Студент

3.1. Преглед на избираемите дисциплини:

3.1.1. Системата ще предоставя на студент списък с всички налични за записване избираеми дисциплини.

3.1.1.1. Системата ще предоставя наличните за текущия семестър избираеми дисциплини, ако има активна кампания за избираеми дисциплини.

3.1.1.2. Системата ще предоставя наличните за предстоящия семестър избираеми дисциплини, ако няма активна кампания за избираеми дисциплини.

3.1.1.3. При липса на въведена информация за бъдещи избираеми дисциплини, системата ще визуализира избираемите дисциплини от съответния семестър на предишната година.

3.1.2. Списъкът ще включва името на дисциплината, описанието ѝ (ако е попълнено), изискванията за записване (ако има такива), броя кредити, които носи, и титуляра, от когото се води дисциплината.

3.1.3. Системата би могла да включва други метаданни за дисциплините в списъка с избираеми дисциплини.

3.2. Маркиране на избираеми дисциплини:

3.2.1. Системата ще позволява на студент маркиране (отбелязване) на избираеми дисциплини от списъка с налични избираеми дисциплини по време на преглед на избираемите дисциплини.

3.2.2. Системата ще позволява премахване на маркирането

3.2.3. Системата ще позволява неограничен брой повторни маркирания

3.2.4. Системата ще предоставя изглед за преглед на маркираните избираеми дисциплини

3.2.5. Списъкът с маркирани избираеми автоматично ще отстранява маркираните избираеми дисциплини, които са записани окончателно, след края на кампанията за избираеми дисциплини.

3.3. Записване на избираеми дисциплини:

3.3.1. Системата ще позволява на студентите да запишат избираема дисциплина:
3.3.1.1. директно, ако има свободни места (при зададен лимит)
3.3.1.2. в опашка, ако няма свободни места (при зададен лимит)
3.3.1.3. само ако отговарят на критериите за записване
3.3.1.4. в момента е активна кампания за избираеми дисциплини
3.3.2. Системата ще позволява на студентите да филтрират списъка с избираемите дисциплини по следните критерии: име на дисциплината,титуляр, брой кредити.
3.3.3. Системата би могла да позволява филтриране и по други критерии.
3.3.4. Системата ще позволява на студентите да сортират избираемите дисциплини по следните критерии: име на дисциплината, титуляр, брой кредити.
3.3.5. Системата би могла да позволява сортиране и по други критерии.
3.3.6. Системата трябва автоматично да записва студент, който е първи в опашката, при освобождаване на свободно място в списъка за дадената дисциплина.
3.3.7. Системата би могла да не съхранява данните за опашката след края на кампанията за избираеми дисциплини.
3.3.8. Системата ще предоставя неограничен брой възможности за отписване на записаните дисциплини до края на текущия етап в кампанията за избираеми дисциплини.
3.3.9. Системата ще предоставя неограничен брой възможности за повторно записване на избираемата дисциплина, при отписване по време на един етап в кампанията за избираеми дисциплини.
3.3.10. Системата няма да позволява повторно записване на избираема дисциплина, ако тя е била записана и изкарана от студента.

3.4. Плащане на такси:

3.4.1. Системата ще позволява на потребителите да извършват плащания на всички такси чрез системата ePay.
3.4.2. Системата ще показва точния размер на таксата, както и основание за таксата.
3.4.3. Системата ще позволява заплащане на семестриални такси
3.4.4. Системата ще позволява заплащане на такси за повторно явяване на изпити.
3.4.5. Системата би могла да позволява заплащане на други такси.
3.4.6. Системата би могла да уведомява студентите за сроковете за плащане на семестриални такси:
3.4.6.1. два дни преди началото
3.4.6.2. два дни преди края на кампанията
3.4.7. Системата би могла да скрива такси, които са заплатени или не могат да бъдат заплатени вече

3.5. Обобщен преглед на дисциплини:

3.5.1. Системата ще предостави на студентите възможност за преглед на получените оценки от дисциплините, които са карали
3.5.2. Прегледът ще включва име на дисциплина, титуляр, етикет задължителна/избираема, семестър на полагане, оценка, кредити (ECTS).

3.5.3. Изгледът за преглед би могъл да предоставя и други метаданни.
3.5.4. Дисциплините в прегледа могат да се филтрират по:
3.5.4.1. Семестър (текущ / минали)
3.5.4.2. Задължителни / избираеми
3.5.4.3. Взети / невзети / взети на поправка
3.5.5. Изгледът би могъл да позволява допълнителни критерии за филтриране.

3.6. Преглед на здравно осигуряване:

3.6.1. Системата би могла да предоставя изглед за преглед на здравното осигуряване по месеци.

4. Преподаватели

4.1. Преподавател ще може да редактира анотацията на дисциплина, която води.

4.2. Попълване на протоколи:

4.2.1. Системата ще позволява на преподавателите да генерират протоколи в края на семестъра за всяка водена от тях дисциплина
4.2.2. Системата ще позволява да се вписват оценки във вече генериран протокол:
4.2.2.1. или индивидуални (за конкретен студент) директно от интерфейс на системата;
4.2.2.2. или масово импортирани от .csv, .xsl, или .xslx файл във вида “факултетен номер; оценка”
4.2.3. Системата ще прави резервно копие на въведените до момента оценки на всеки 5 попълвания на данни или на всеки 3 минути (по-малкото от двете).
4.2.4. Изгледът за редакция на оценки ще позволява редакция на оценката само за един студент в даден момент.
4.2.5. Системата ще позволява на преподавателя да отбелязва етикети за статуса на протокол - *Незавършен*, *Изпратен, Приет* и *Отхвърлен*.
4.2.5.1. Етикетът *Изпратен* ще бъде автоматично задаван от системата при изпращане за валидация от администратор на данни.
4.2.5.2. Етикетът *Отхвърлен* ще бъде автоматично задаван от системата при отхвърляне на протокола от администратор на данни поради необходимост от корекции.
4.2.5.3. Етикетът *Приет* ще бъде автоматично задаван от системата при приемане на протокола от администратор на данни след преглед.
4.2.6. Системата ще позволява на преподавателя да изпраща протокола за верификация.
4.2.6.1. Системата няма да позволява редактиране на протокола след изпращането му.
4.2.6.2. Системата ще изисква двуфакторно удостоверяване преди изпращане на (нова версия на) протокола.

5. Администратори на данни

5.1. Смяна на паролата:

5.1.1. Системата ще предоставя изглед за смяна на паролата.
5.1.2. Системата ще изисква двуфакторно удостоверяване преди въвеждането на новата парола.

5.2. Създаване на нови акаунти:

5.2.1. Системата ще предоставя на администраторите функционалност за създаване на нови акаунти за студенти и преподаватели в системата.
5.2.2. Системата ще изисква от администраторите да въведат необходимата информация за новия потребител, която включва задължително: име, фамилия, ЕГН, адрес, имейл адрес, номер на мобилен телефон, роля.
5.2.3. При регистриране на нов студент освен задължителната информация, ще се изисква задължително и факултетен номер, ОКС, специалност, начин на финансиране.
5.2.4. При регистриране на нов преподавател освен задължителната информация, ще се изисква задължително и информация за неговата степен, както и катедра, от която ще бъде част.
5.2.5. Системата би могла да изисква и други данни, които са в съответствие с политиката на съответното висше училище.
5.2.6. Системата ще генерира парола по подразбиране.
5.2.7. Системата ще изпраща на новия потребител автоматично генерирано известие по email с информация за акаунта
5.2.8. Системата би могла да задължи потребителя да смени паролата си по подразбиране при първо влизане в системата.

5.3. Създаване на нови дисциплини:

5.3.1. Системата ще осигурява възможност за администратор на данни да създава нова дисциплина за дадена академична година.
5.3.2. Системата ще дава възможност за създаване на чисто нова дисциплина, за което се изисква посочване на: име, списък от целеви специалности, титуляр (преподавател), анотация, конспект, хорариум, задължителни/препоръчителни предварителни изисквания, списък от етикети.
5.3.3. Системата ще дава възможност за създаване на нова дисциплина като копие на друга с възможност за промяна на данните.

5.4. Управление на кампании:

5.4.1. Системата ще позволява на администратор на данни създаването на кампании, сред които кампании за избираеми дисциплини и кампании за плащане на такси.
5.4.2. Системата ще позволява създаване на кампании за плащане на семестриални такси.
5.4.2.1. При създаването на кампанията, системата автоматично ще генерира такса за всеки студент.
5.4.3. Системата ще позволява създаване на кампания за избираеми дисциплини от администратор.
5.4.3.1. Системата ще позволява разделянето на етапи, при създаването на избираема дисциплина.
5.4.3.2. Системата ще позволява избор между следните правила за етапите на избираеми дисциплини:
5.4.3.2.1. Записване на избираеми
5.4.3.2.2. Отписване на избираеми
5.4.3.2.3. Отписване на избираеми от предходен етап
5.4.3.3. Правилата за отделните етапи ще могат да бъдат определяни от съответното висше училище.

5.4.3.4. Системата ще изисква въвеждане на списък с избираеми дисциплини, които са налични за конкретната кампания

5.4.4. Системата ще позволява редактиране на списъка с избираеми дисциплини преди и по време на кампания за избираеми дисциплини.

5.4.4.1. Системата ще позволява редактиране на началната дата на кампания за избираеми дисциплини преди настъпване на предишната поставена дата.

5.4.4.2. Системата ще позволява редактиране на етапи, ако не са започнали.

5.4.4.3. Системата ще позволява редактиране единствено на крайната дата на етапи, които са започнали.

5.5. Редактиране на дисциплина:

5.5.1. Системата ще позволява промяна на името, описанието, преподавателите и анотацията на дадена дисциплина

5.5.2. Системата ще позволява промяна на критерия за подреждане на записани студенти за дисциплина

5.5.2.1. Системата ще позволява избор между следните критерии за записване:

5.5.2.1.1. Лимит и подредба по последователност на записване

5.5.2.1.2. Лимит и подредба по среден успех

5.5.2.1.3. Без лимит (например при провеждане на входен тест)

5.5.3. Промяна на лимита за избираема дисциплина.

5.5.4. Системата ще позволява преглед, добавяне и премахване на студенти от списък на записани за дисциплина

5.6. Верифициране на протоколи:

5.6.1. Системата ще позволява администратор на данни да верифицира протокол, изпратен от преподавател.

5.6.2. Системата ще позволява администратор на данни да отхвърля протокол, изпратен от преподавател за верификация.

5.6.3. Системата ще връща отхвърлен от администратора на данни протокол на преподавателя за редакция.

5.6.4. Срокът за обработка на протокол е 14 дни от получаването му.

5.7. Създаване на справка:

5.7.1. Системата ще позволява на администраторите на данни да създават справки за студенти и преподаватели.

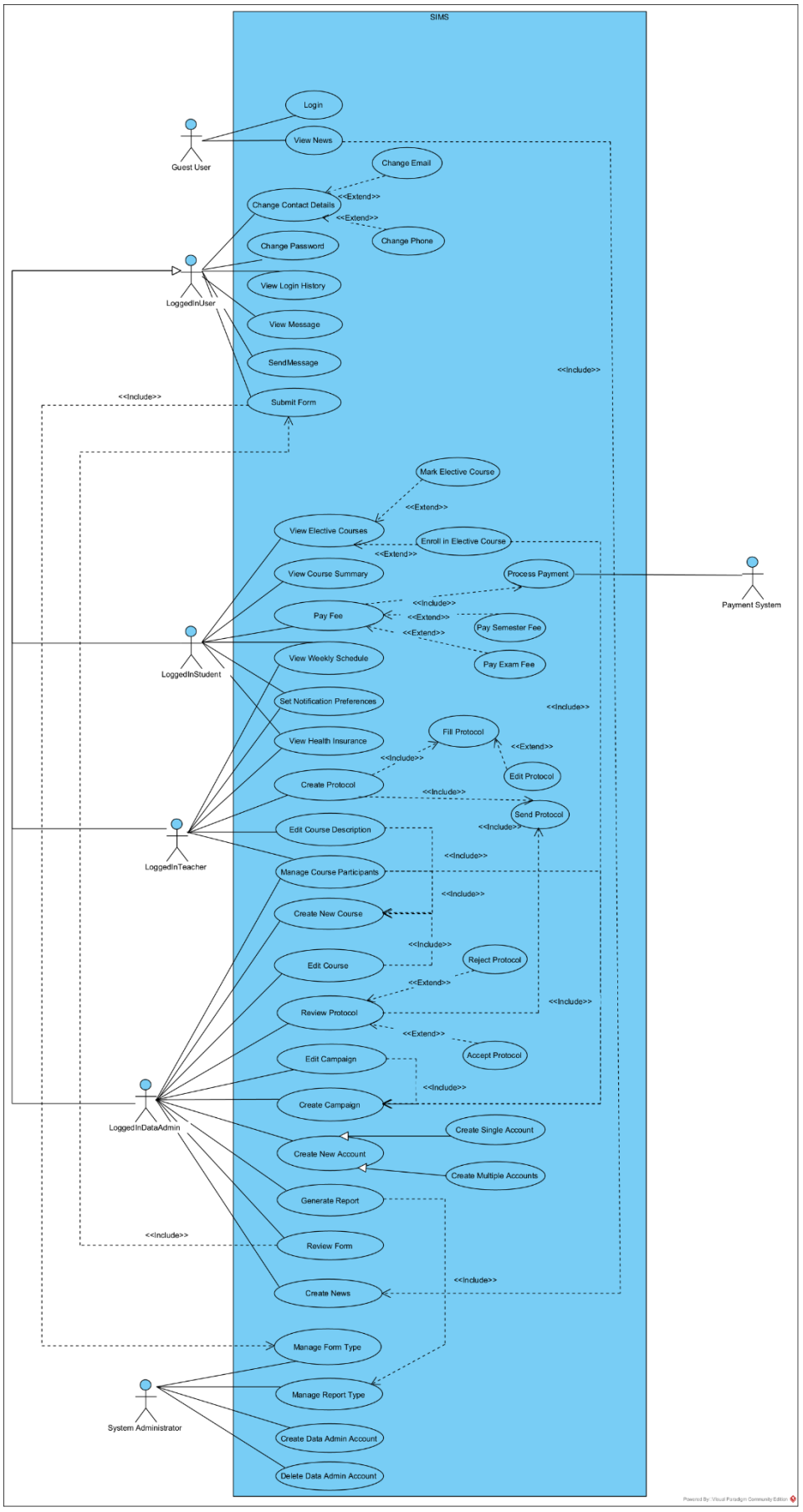
5.7.2. Системата ще позволява справките да се композират динамично спрямо ролята на лицето, за което се прави справката и данните за него в базата данни.

Пример: *избиране на роля → ако е преподавател се появяват полета за избор на дисциплини и хорариум за тях; ако е студент - отново избор на дисциплини, но с избор на оценки и кредити за тях.*

6. Администратори на системата

6.1. Системата ще позволява администратор на системата да **създава профили** от тип **“администратор на данни”** чрез директни права за достъп до базата данни.

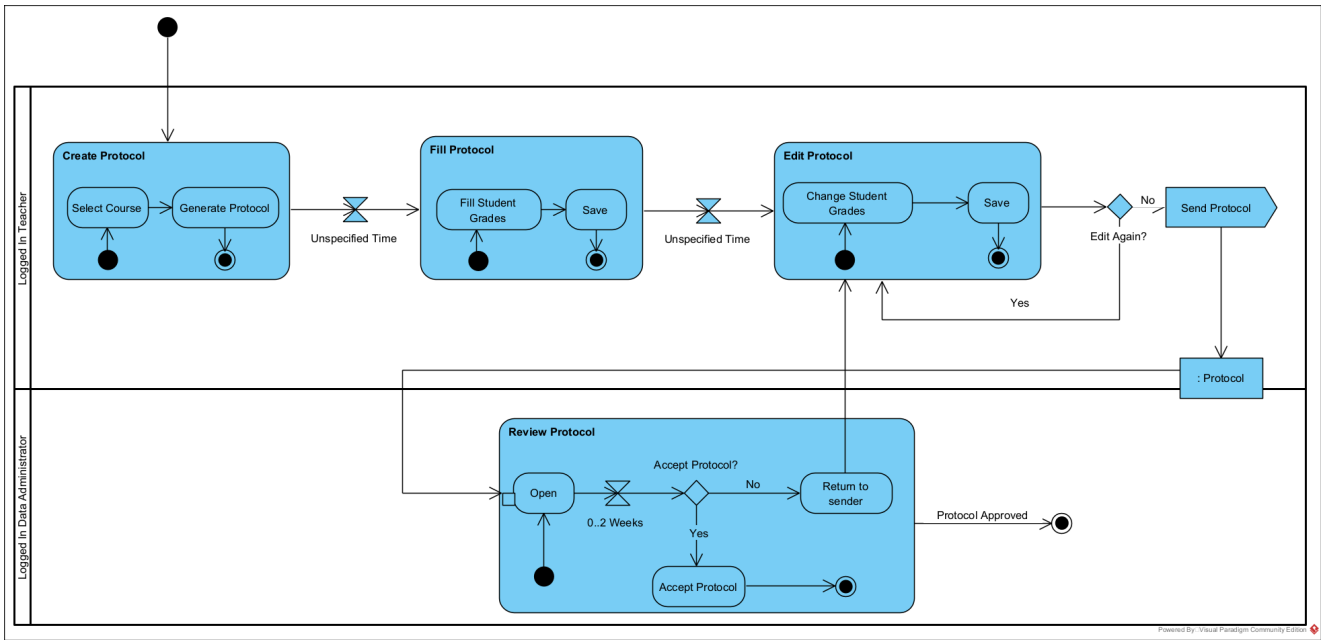
6.2. Системата ще позволява администратор на системата да **изтрива профили** от тип **“администратор на данни”** чрез директни права за достъп до базата данни.


SIMS
Login
View News
Guest User
Change Email
Change Contact Details
<<Extend>>
<<Extend>>
Change Phone
Change Password
LoggedInUser
View Login History
View Message
SendMessage
Submit Form
<<Include>>
<<Include>>
Mark Elective Course
<<Extend>>
View Elective Courses
Enroll in Elective Course
<<Extend>>
View Course Summary
Process Payment
Payment System
<<Include>>
Pay Fee
<<Extend>>
<<Extend>>
Pay Semester Fee
Pay Exam Fee
LoggedInStudent
View Weekly Schedule
Set Notification Preferences
<<Include>>
Fill Protocol
View Health Insurance
<<Include>>
<<Extend>>
Edit Protocol
Create Protocol
<<Include>>
Send Protocol
<<Include>>
Edit Course Description
LoggedInTeacher
<<Include>>
Manage Course Participants
<<Include>>
Create New Course
<<Include>>
Edit Course
Reject Protocol
<<Extend>>
Review Protocol
<<Include>>
<<Extend>>
Accept Protocol
Edit Campaign
<<Include>>
Create Campaign
LoggedInDataAdmin
Create Single Account
Create New Account
Create Multiple Accounts
Generate Report
<<Include>>
Review Form
<<Include>>
Create News
Manage Form Type
Manage Report Type
System Administrator
Create Data Admin Account
Delete Data Admin Account

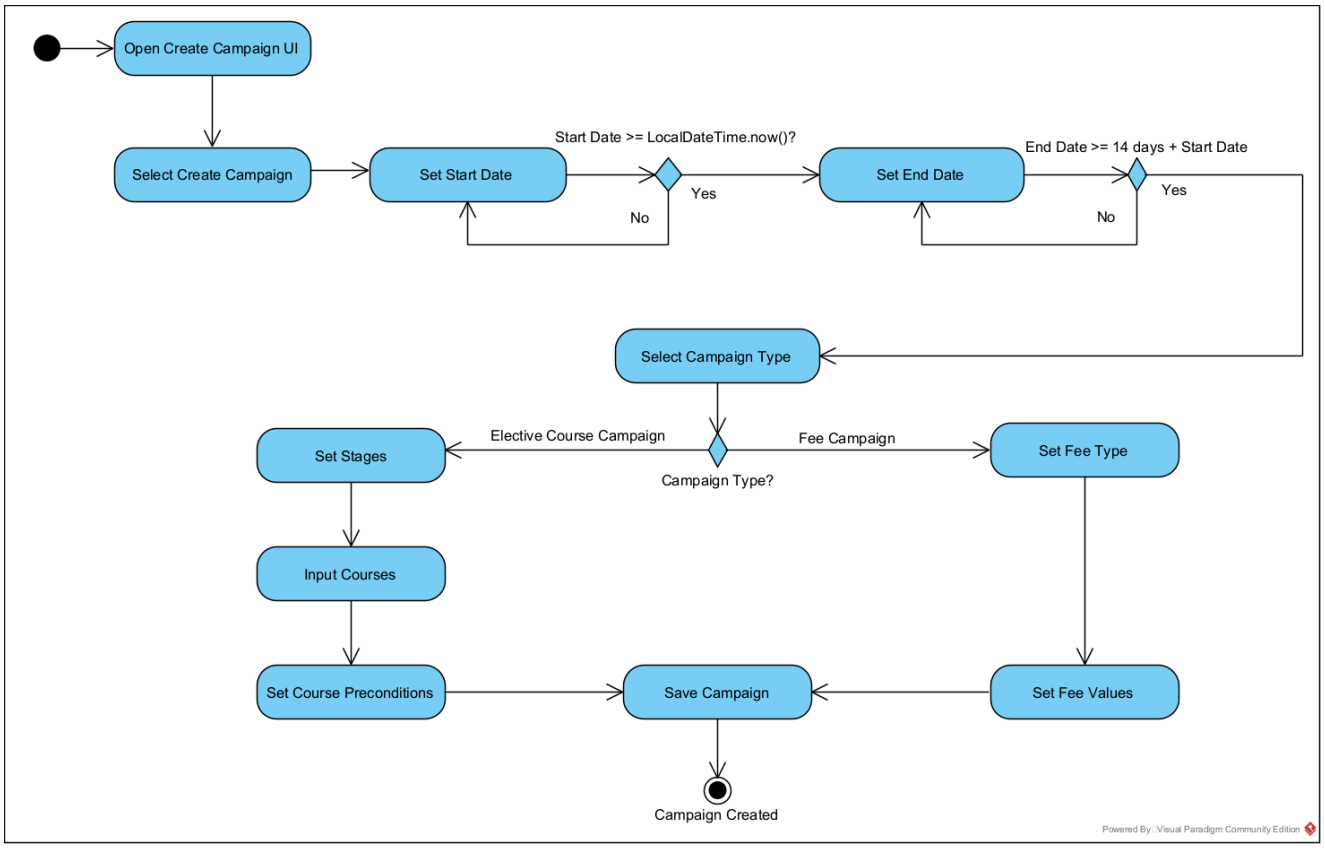
Описание на диаграмата на потребителските случаи: Диаграмата съдържа всички потребителски случаи в системата, защото те са взаимосвързани и разделянето на диаграмата не е възможно без частична загуба на информация. Случаите са организирани визуално според актьорите, които ги инициират. Чрез наследяване на актьори се преизползват потребителски случаи, общи за всички логнати потребители. Чрез наследяване на потребителски случаи се разглеждат частни случаи на потребителския случай за създаване на акаунт на потребител от страна на Администратор на данни. Описания на всеки потребителски случаи са от таблички от вида:

Потребителски случай	
Участващи актьори	
Поток от събития	
Начални условия	
Крайни условия	
Изисквания за качество	

и могат да бъдат намерени във файла *UseCaseTables*.



Описание на диаграма на дейностите за създаване на протокол от Преподавател: Диаграмата моделира дейностите и действията по създаване и валидиране на един протокол, което включва участието както на преподавател, така и на администратор на данни. За удобно разделение и четимост вместо стереотипи са използвани swimlanes.



Описание на диаграма на дейностите за създаване на кампания от Администратор на данни: Диаграмата моделира дейностите и действията по създаване на кампания. Отделено е внимание на валидирането на входа за датите на начало и край на кампанията. Останалите стъпки по избор предполагат избор от падащи списъци, което позволява валидиране на входните данни още с графичния интерфейс. Разгледани са различни случаи спрямо вида на кампанията (за такси или за избираеми дисциплини), видовете такси и др.

2.1.2 Нефункционални изисквания

Ще разгледаме някои от по-важните нефункционални изисквания за системата:

1. Сигурност:

1.1. Системата предоставя еднофакторно удостоверяване чрез JSON Web Token с валидност 1 седмица, който може да бъде получен чрез вписване с потребителско име и парола.

1.2. Системата трябва да използва двуфакторно удостоверяване, когато еднофакторното не е достатъчно сигурно:

1.2.1. нанасяне на оценки в протоколи

1.2.2. верифициране на протоколи

1.2.3. създаване на кампании

1.2.4. плащане на такси

1.3. Двуфакторното удостоверяване на системата трябва да се състои от парола и допълнителна стъпка - OTP код, изпратен по SMS или имейл, чрез които се издава elevated token с валидност 5 минути.

1.3.1. Изпращането на кода ще отнема не повече от 60s

1.4. Генерирането на token, ще отнема не повече от 1-2s.
1.5. Сървърът трябва да използва TLS протокол версия 1.2 или по-висока за всички комуникации с клиентите.
1.6. Системата трябва да криптира всички лични данни на потребителите по време на съхранение.

2. Мащабируемост:

2.1. Системата трябва да може да обслужва до 2500 потребители едновременно без нарушаване на производителността.

3. Наличност:

3.1. Системата трябва да е налична в 99,5% от времето за месец.
3.2. Позволява се интервал за поддръжка от до един час веднъж месечно (по възможност в неработни за университета дни) между 23:00 и 06:00 часà..

4. Използваемост:

4.1. Дизайнът на потребителския интерфейс трябва да отговаря на правилата за добро потребителско изживяване:
  4.1.1. Най-използваните 3 функционалности от всяка потребителска група трябва да бъдат достъпни с до 3 кликвания на мишката.
  4.1.2. Изгледът за студенти и преподаватели трябва има десктоп и мобилна версия.
  4.1.3. Потребителят не трябва да въвежда данни, които се съхраняват от системата в личния му профил и могат да бъдат изведени автоматично.
4.2. Системата трябва да предоставя уместни съобщения на потребителите:
  4.2.1. Системата трябва да връща обратна връзка при невъзможно извършване на операция поради наложено ограничение от организацията.
  4.2.2. Системата трябва да информира потребителя за настъпили грешки и изключения, като капсулира техническата информация и посочва единствено източник на грешката (локален компютър, мрежа или сървър) и възможност за разрешаване (презареждане на страницата, достъпване на системата в различно време, повторно вписване в системата).

5. Достъпност:

5.1. Системата трябва да бъде проектирана в съответствие със стандартите за достъпност на уеб съдържанието (WCAG) 2.1, ниво AA.

6. Съвместимост:

6.1. Системата трябва да позволява достъп до всички функционалности през следните браузъри: Firefox, Chromium, Chrome, Brave, Edge, Safari.

7. Производителност:

7.1. Системата трябва да има максимално време за отговор от 2 секунди за всяка операция или транзакция

8. Разработка на системата трябва да се осъществи чрез инструмент за контрол на версиите.

9. Възможност за поддръжка:

9.1. Системата ще създава лог файлове, описващи всички сривове в системата.
9.2. Поддръжка на системата чрез система за контрол на версиите.
  9.2.1. Препоръчителни системи за контрол на версиите са Git и GitHub.

10. Цялост на данните:

10.1. Системата трябва да поддържа периодични резервни копия на базата данни - веднъж на 3 дни.

11. Защита на данните - GDPR:

11.1. Системата ще съхранява минималното достатъчно количество данни, необходимо за безпроблемната работа на всички функционалности на системата - пълна идентификация на потребителя, извършване на учебната дейност и поне един метод за контакт.

11.2. Университетът, внедряващ системата, ще изисква писмена декларация за съгласие за ползване на системата преди предоставянето на данните за вход по подразбиране на всеки потребител.

11.2.1. Декларацията ще съдържа информация за данните, които системата събира за потребителите.

11.3. Университетът, внедряващ системата, ще въведе план за уведомление и действия, при изтичане на лични данни извън рамките на системата (с изключение на задължителните данни за плащане чрез външната система за плащане).

11.4. Системата ще съхранява личните данни в съответствие с GDPR и законите на Република България.

11.4.1. Системата ще съхранява личните данни на всички потребители през цялото време на тяхната активна длъжност в университета.

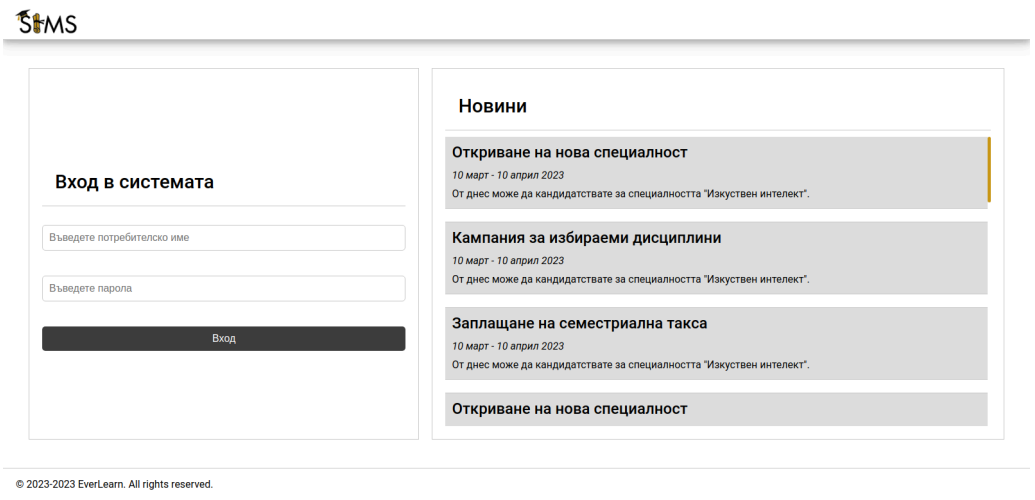
11.4.2. Системата ще съхранява личните данни на всички студенти в продължение на времето, посочено в Закона за висшето образование.

11.4.3. Системата ще съхранява личните данни на преподавателя 10 години след прекратяване на трудовата дейност в университета.

11.4.4. Системата ще съхранява личните данни на администратор на данни на системата 5 години след прекратяване на трудовата му дейност.

11.4.5. Университетът, който внедрява системата, се задължава да обнови сроковете на съхранение на личните данни на потребителите съгласно всякакви промени в GDPR или в законите на Република България.

2.2 Примерен графичен интерфейс



SIMS

График | Оценки | Избираеми дисциплини | Съобщения | 62547

Разписание на занятията

Дисциплина	Място	Начало
Английски език	ФМИ, 313	09:00
Бази данни	ФМИ, 325	11:00
Увод в програмирането	ФМИ, 200	13:00
ОО програмиране	ФМИ, 313	15:00
Операционни системи	ФМИ, 325	17:00

Такси

Такса	Сума	Статус
Семестриална такса	450лв.	Платена
Exam retake tax	50.00	Плати

Маркирани избираеми дисциплини

Име	Записана
Фрактали	☑
Увод в статистиката	☑
Функционално програмиране	☐

Записани избираеми дисциплини

Дисциплина	Преподавател	Кредити
Фрактали	Милко Такев	8
Увод в статистиката	Мая Желязкова	5

Съобщения

Добре дошли в курса по Джава
Евгений Кръстев, Преди 2 часа

Провеждане на курса по Увод в програмирането
Александър Димов, Вчера

Гост-лектор
Боян Бончев, 19.03 9:00

Оценки

Дисциплина	Преподавател	Оценка
Дискретни структури	Андрей Сариев	4.50
Алгебра	Боряна Ангелова	5.00
Комуникативни умения	Симеон Петров	4.00
Структури от данни	Боян Димитров	5.50

SIMS

График | Оценки | Избираеми дисциплини | Съобщения | 62547

Избираеми дисциплини

Потърсете избираема дисциплина

☑ Име ☑ Преподавател ☑ Кредити ☑ Оценка ☑ Препоръчителен семестър ☑ Рейтинг ☑ Маркиране | Маркирани и немаркирани | Всички | CSV | PDF

Име	Преподавател	Кредити	Оценка	Препоръчителен семестър	Рейтинг	Маркиране	Записване
Функционално програмиране	Доц. д-р Иван Иванов	5	Изпит/Текуща оценка	1	4.6	☑	Записване
Английски език 1	Преп. Мария Петрова	3	Текуща оценка	1	4.3	☐	Записване
Основи на базите данни	Проф. д-р Николай Николов	4	Изпит/Текуща оценка	2	4.5	☐	Записана в опашка
Криптография	Проф. д-р Стефан Стефанов	6	Изпит	4	4.8	☐	Записване в опашка
Интернет програмиране	Доц. д-р Мария Георгиева	5	Изпит/Текуща оценка	3	4.4	☐	Записване
Дискретни структури	Проф. д-р Петър Петров	5	Изпит/Текуща оценка	2	4.5	☐	Записване в опашка
Фрактали	Доц. Милко Такев	5	Изпит/Текуща оценка	1	4.6	☑	Записана
Увод в статистиката	доц. Мая Желязкова	3	Текуща оценка	1	4.3	☑	Записана

SIMS

График | Протоколи | Управление на курсове | Кампания за избираеми дисциплини | Справки | Съобщения | проф. д-р Иван Георгиев

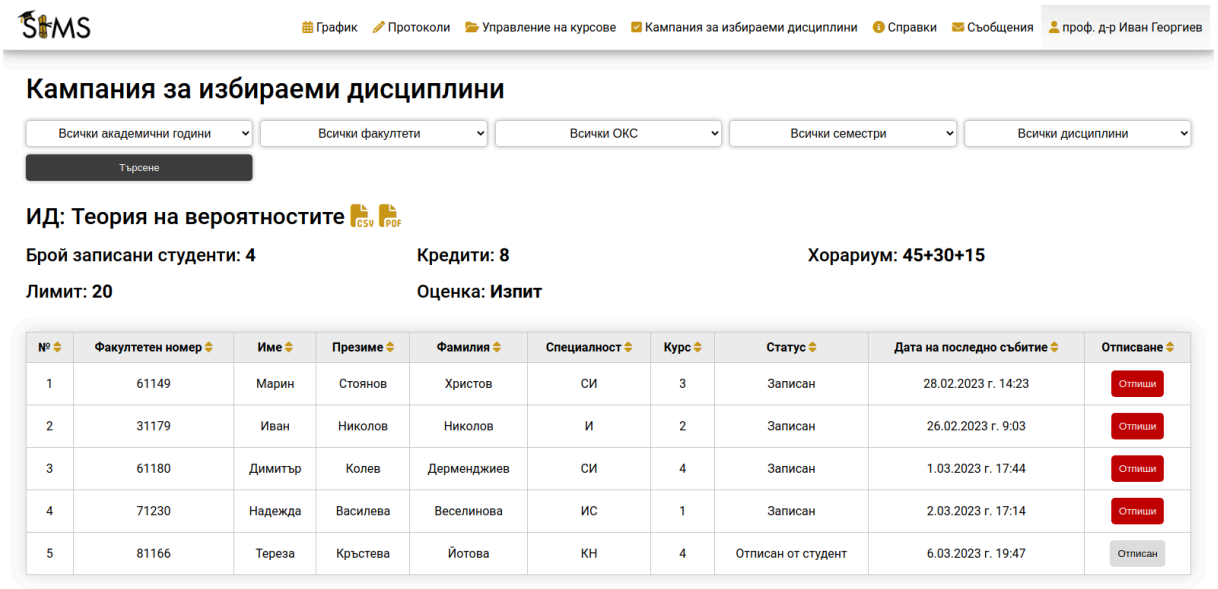
Редактиране
Премахване
Смяна на парола

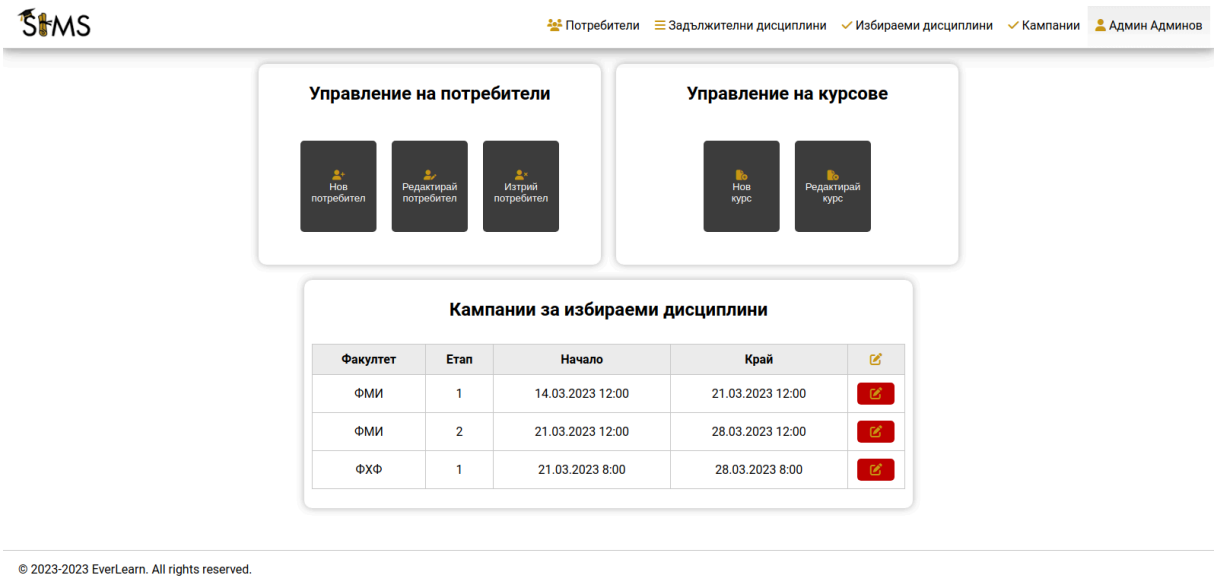
Име	Иван Христов Георгиев
Титла	проф. д-р
Факултет	ФМИ
Катедра	Вероятности и статистика
Имейл	ivan.georgiev@fmi.uni-sofia.bg
Телефон	0887896666
Адрес	гр. София, ул. "Хан Крум" 34

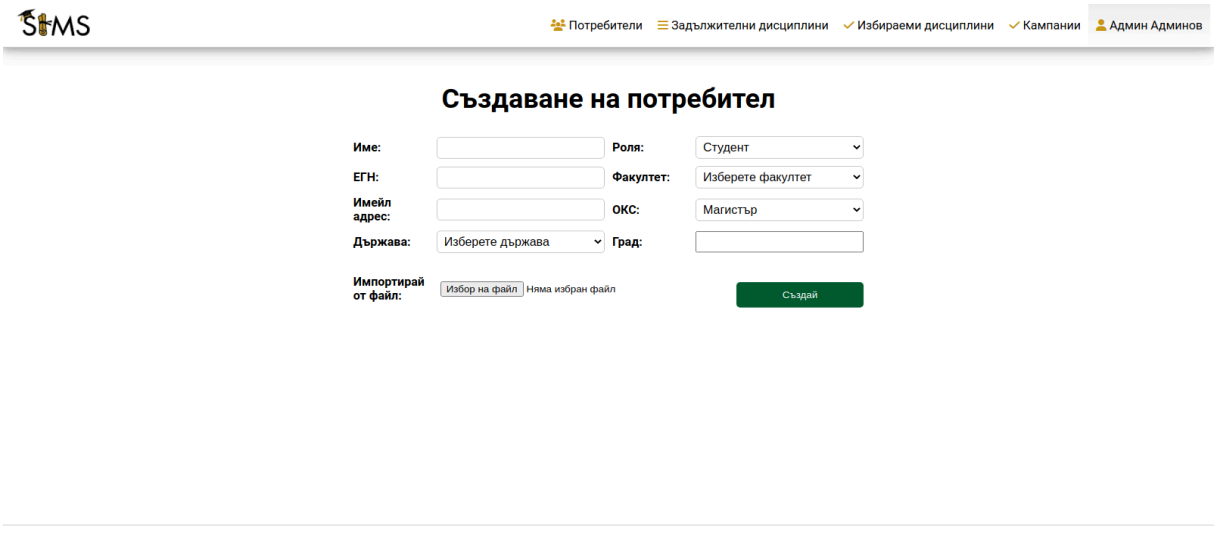
Здравно осигуряване | Логвания | Настройки на началната страница | Настройки на известията | Тема

Логвания

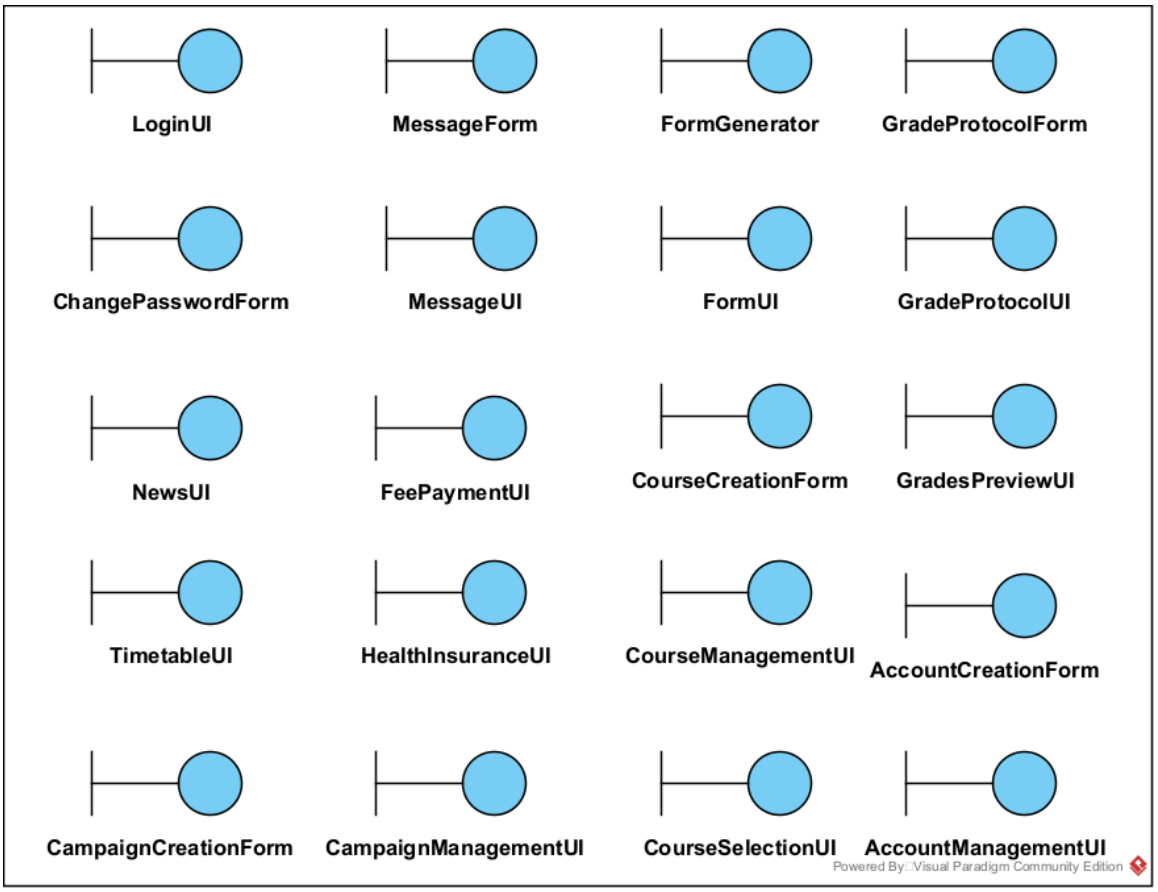
Дата и час	Устройство	IP адрес
18/05/2023 17:07:16	Windows 11 / Chrome 97.0.4692.99	192.168.2.1
16/05/2023 09:16:16	Mac OS X / Safari 15.1	172.16.0.2
09/05/2023 17:31:11	Windows 10 / Firefox 95.0	10.0.0.1
06/05/2023 22:09:45	iOS 15 / Safari 15.1	192.168.0.10
01/05/2023 11:15:33	Android 12 / Chrome 96.0.4664.93	192.168.1.100
26/04/2023 16:22:01	Windows 10 / Edge 98.0.1108.56	192.168.1.50
16/04/2023 09:43:10	Linux / Firefox 95.0.1	172.16.0.100
10/04/2023 14:58:21	Mac OS X / Chrome 97.0.4692.99	192.168.1.2
06/04/2023 11:30:05	iOS 14 / Safari 14.1	10.0.0.2
26/03/2023 16:06:16	Windows 11 / Edge 98.0.1108.56	192.168.1.20


SIMS
График
Протоколи
Управление на курсове
Кампания за избираеми дисциплини
Справки
Съобщения
проф. д-р Иван Георгиев
Кампания за избираеми дисциплини
Всички академични години
Всички факултети
Всички ОКС
Всички семестри
Всички дисциплини
Търсене
ИД: Теория на вероятностите
CSV
PDF
Брой записани студенти: 4
Кредити: 8
Хорариум: 45+30+15
Лимит: 20
Оценка: Изпит
№
Факултетен номер
Име
Презиме
Фамилия
Специалност
Курс
Статус
Дата на последно събитие
Отписване
1 61149 Марин Стоянов Христов СИ 3 Записан 28.02.2023 г. 14:23 Отпиши
2 31179 Иван Николов Николов И 2 Записан 26.02.2023 г. 9:03 Отпиши
3 61180 Димитър Колев Дерменджиев СИ 4 Записан 1.03.2023 г. 17:44 Отпиши
4 71230 Надежда Василева Веселинова ИС 1 Записан 2.03.2023 г. 17:14 Отпиши
5 81166 Тереза Кръстева Йотова КН 4 Отписан от студент 6.03.2023 г. 19:47 Отписан
© 2023-2023 EverLearn. All rights reserved.


SIMS
Потребители
Задължителни дисциплини
Избираеми дисциплини
Кампании
Админ Админов
Управление на потребители
Нов потребител
Редактирай потребител
Изтрий потребител
Управление на курсове
Нов курс
Редактирай курс
Кампании за избираеми дисциплини
Факултет
Етап
Начало
Край
ФМИ 1 14.03.2023 12:00 21.03.2023 12:00
ФМИ 2 21.03.2023 12:00 28.03.2023 12:00
ФХФ 1 21.03.2023 8:00 28.03.2023 8:00
© 2023-2023 EverLearn. All rights reserved.


SIMS
Потребители
Задължителни дисциплини
Избираеми дисциплини
Кампании
Админ Админов
Създаване на потребител
Име:
Роля:
Студент
ЕГН:
Факултет:
Изберете факултет
Имейл адрес:
ОКС:
Магистър
Държава:
Изберете държава
Град:
Импортирай от файл:
Избор на файл
Няма избран файл
Създай
© 2023-2023 EverLearn. All rights reserved.

2.3 Диаграми на анализа



Описание на клас диаграма на гранични (boundary) класове на анализа: На представената UML диаграма са показани по-голямата част от примерните гранични класове на анализа. Те моделират взаимодействието между системата и нейните актьори като играят ролята на посредник. Основните отговорности на граничните класове са както следва:

LoginUI и **ChangePasswordForm** – отговарят за основната логика по страницата за вписване и формата за смяна на паролата

MessageUI и **MessageForm** – отговарят за основната логика по страницата за изпращане на съобщения и формата за създаване на съобщения

FormUI и **FormGenerator** – отговарят за основната логика по страницата за управление на формуляри и модула за тяхното генериране

GradeProtocolUI и **GradeProtocolForm** – отговарят за основната логика по страницата за изпращане на протоколи и формата за тяхното попълване

NewsUI – отговаря за основната логика по страницата за преглед на съобщения

HealthInsuranceUI – отговаря за основната логика по страницата за платено здравно осигуряване

FeePaymentUI – отговаря за основната логика по страницата за плащане на такси

TimetableUI – отговаря за основната логика по страницата за визуализиране на разписанието

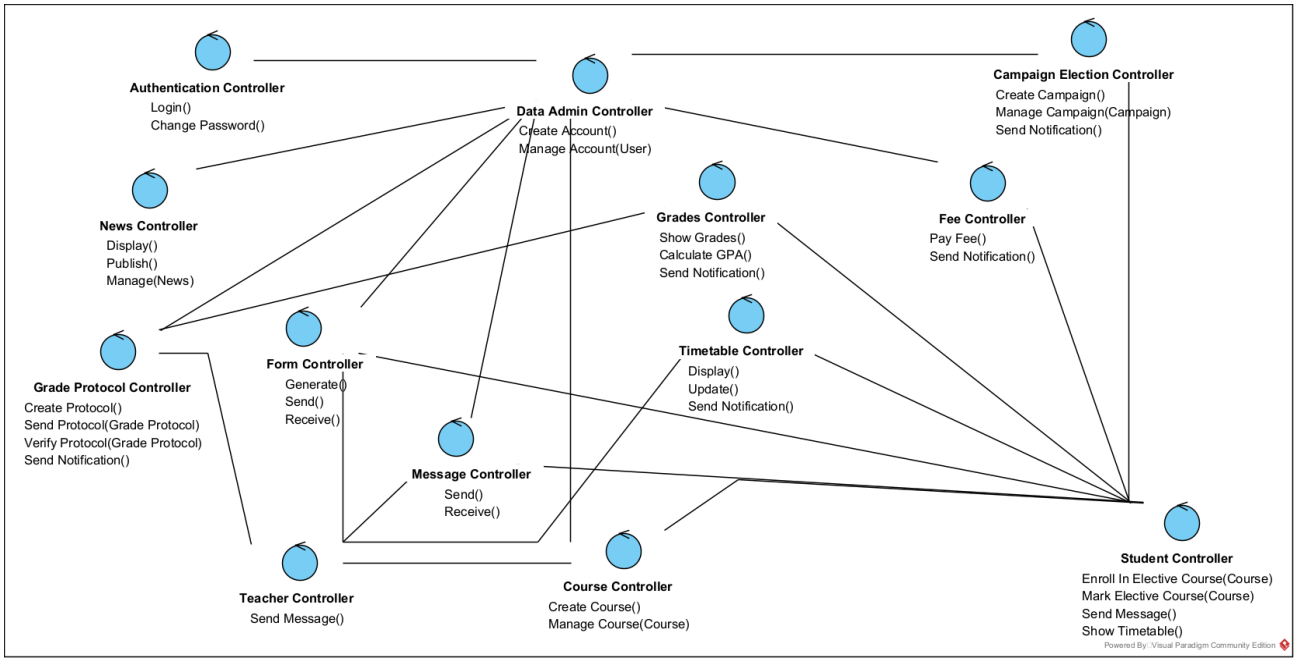
GradesPreviewUI – отговаря за основната логика по страницата за преглед на завършени курсове и получените от тях оценки

CourseSelectionUI – отговаря за основната логика по страницата за избор на избираеми курсове при активна кампания за избираеми дисциплини

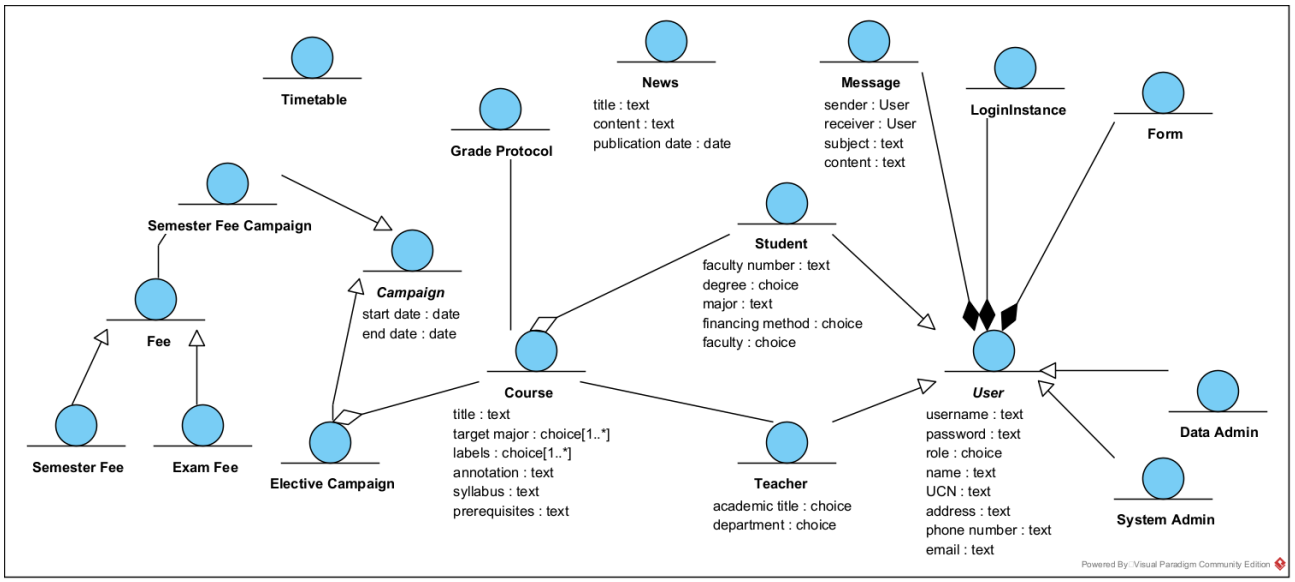
CourseManagementUI и **CourseCreationForm** – отговарят за основната логика по страницата за управление на дисциплини и формата за тяхното създаване

AccountManagementUI и **AccountCreationForm** – отговарят за основната логика по страницата за управление на акаунти и формата за тяхното създаване

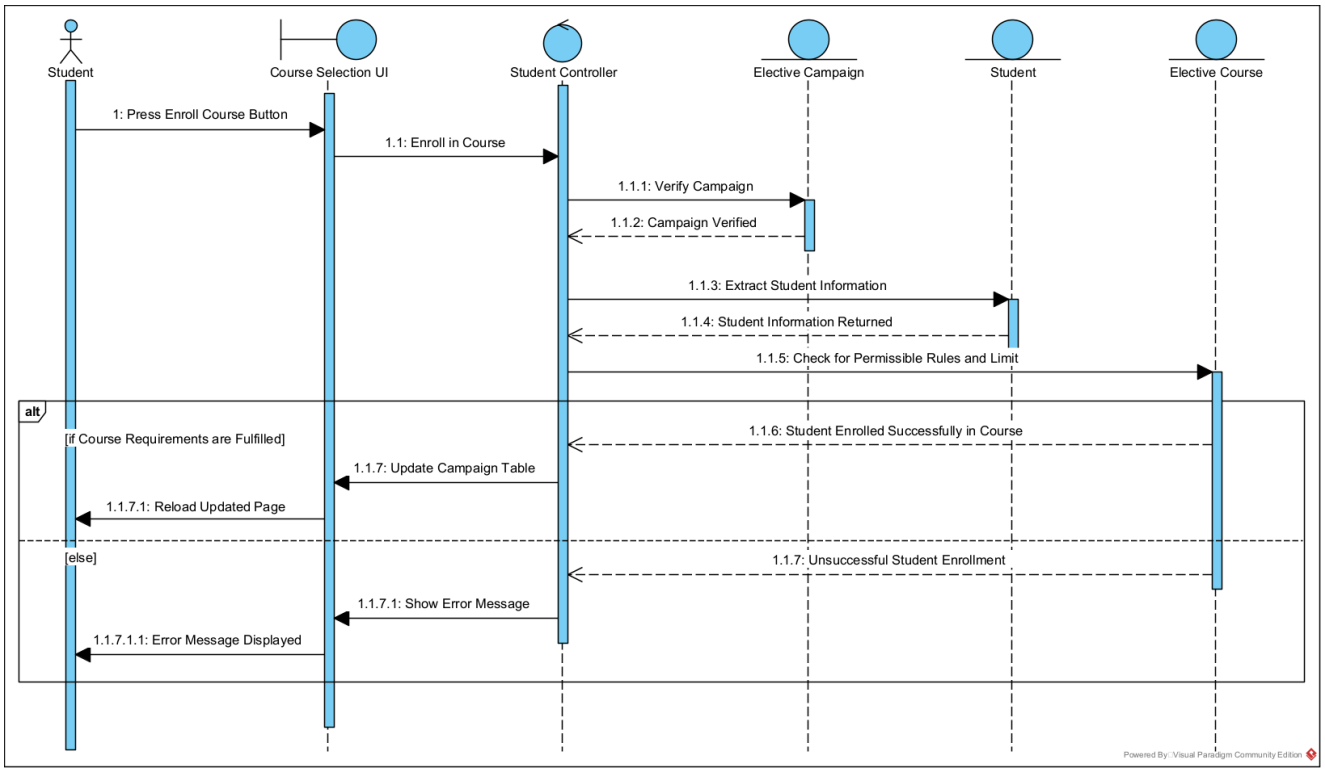
CampaignManagementUI и **CampaignCreationForm** – отговарят за основната логика по страницата за управление на кампании и формата за тяхното създаване



Описание на клас диаграма на контролни (control) класове на анализа: На представената UML диаграма са показани по-голямата част от контролните класове на анализа. Тяхната цел е да се моделира контролното поведение, вземайки предвид основните потребителски случаи. Диаграмата дава само общ поглед върху контролните класове на системата, като подробности могат да бъде видени в съответната диаграма на класовете на дизайна. Основните контролни класове са контролери за управление на студенти, преподаватели, администратори на данни, оценки, курсове, съобщения, такси, разписание, кампания за избираеми дисциплини, формуляри, протоколи, новини, автентикация. Методите на отделните класове съответстват на основните функционалности на системата.

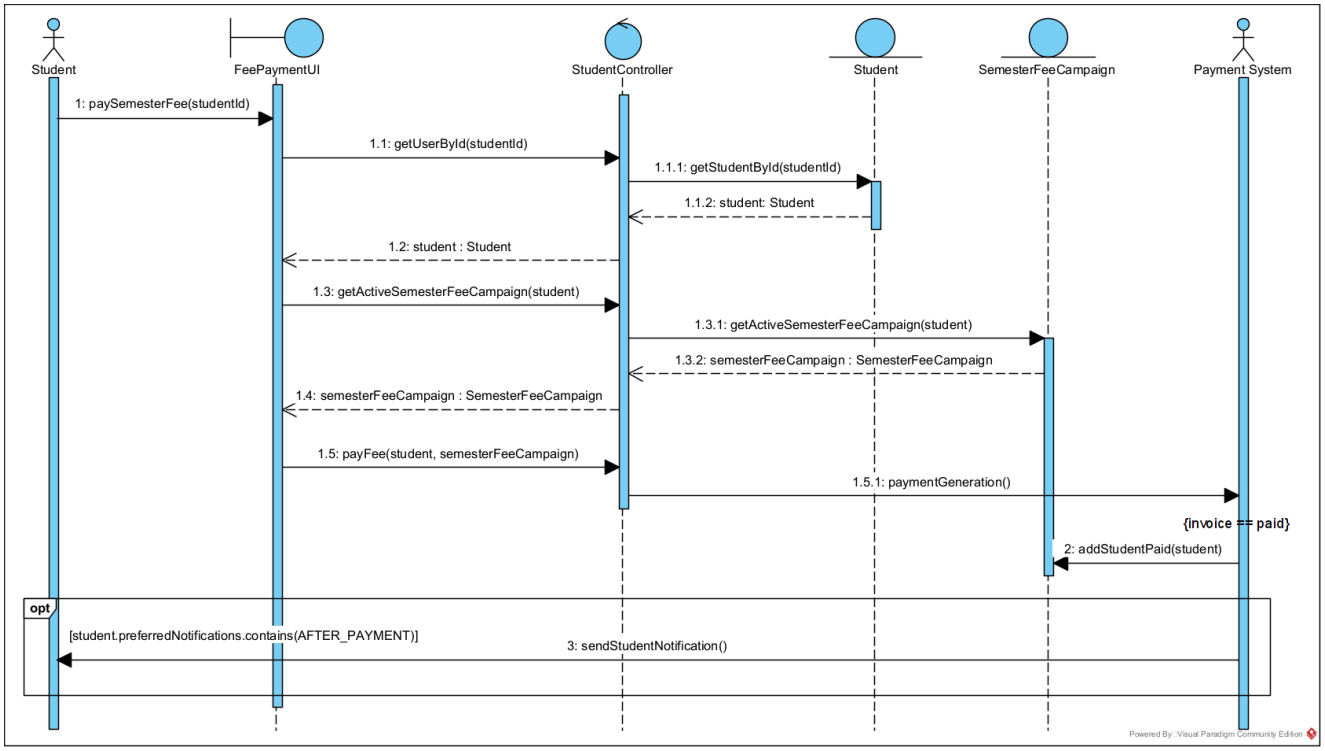


Описание на клас диаграма на обектни (entity) класове на анализа: На представената UML диаграма са показани основните обектни класове на анализа. Тяхната цел е да се моделира информацията и асоциираното поведение, което трябва да бъде съхранено в самите обекти. Основните обектни класове на анализа в SIMS са студенти, преподаватели, администратор на данни, администратор на системата, курсове, кампании за избираеми дисциплини, кампании за плащане на такси, протоколи, такси, разписания, новини, съобщения, формуляри, логове. Съответните техни атрибути са представени възможно най-общо, без да се задават конкретни типове данни на определен език за програмиране.

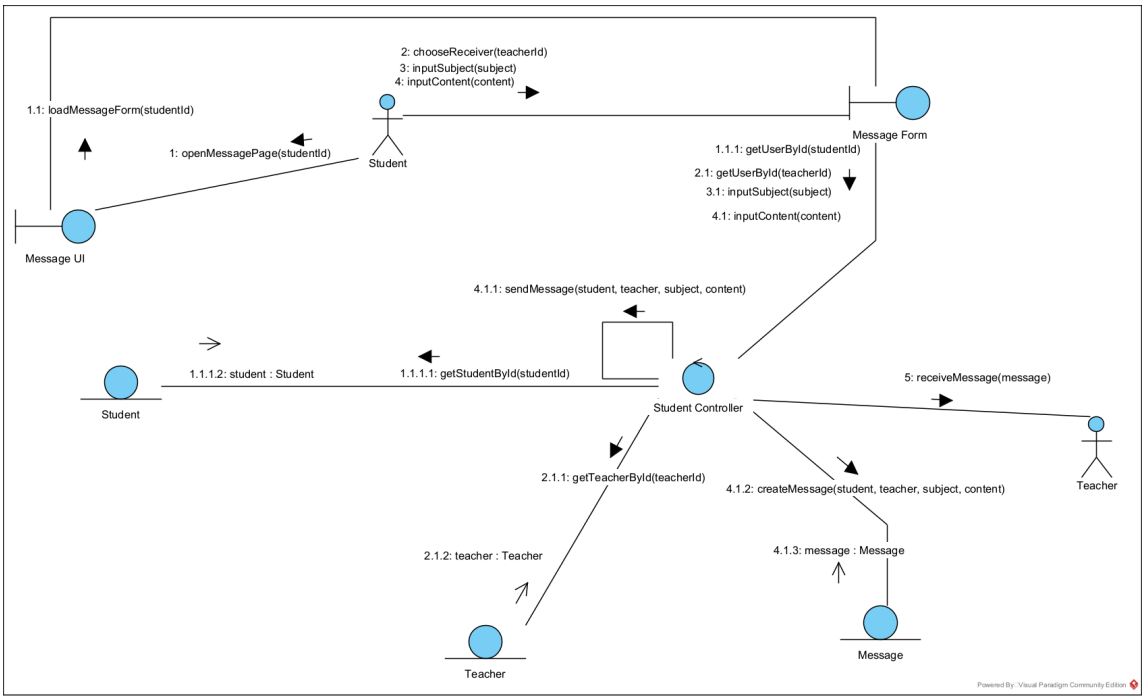


Описание на диаграма на последователността за записване на избираем курс от студент: На представената UML диаграма е показан процесът по записване на избираема дисциплина

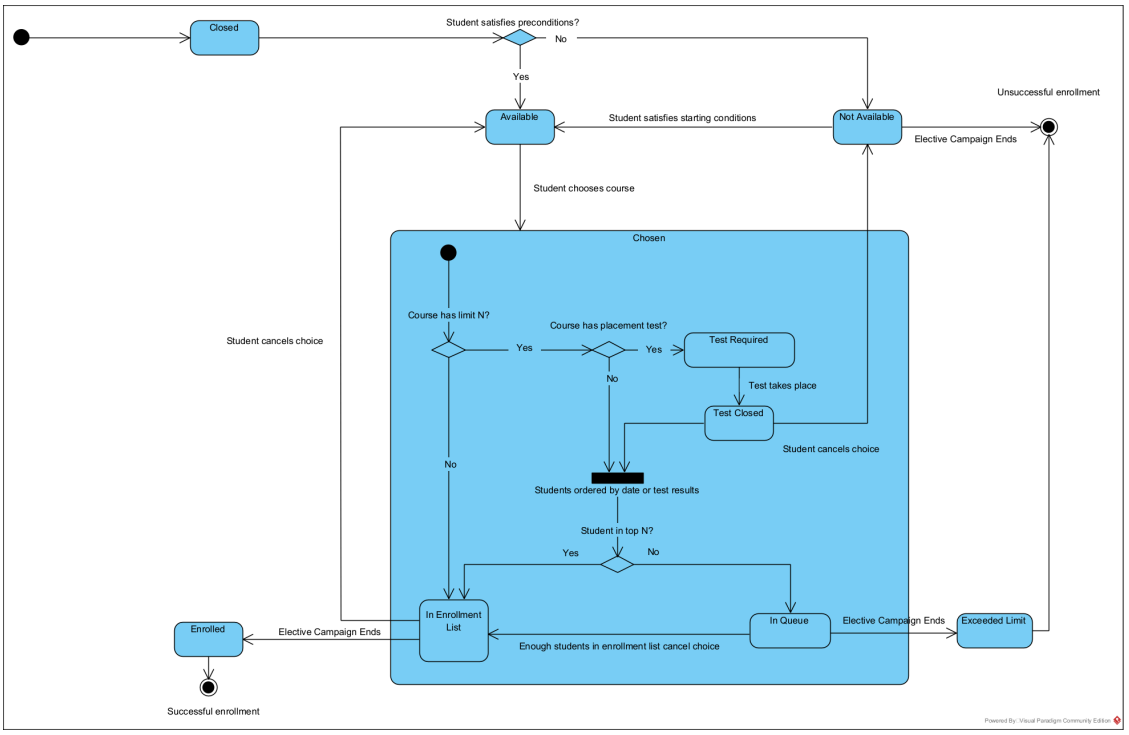
от студент по време на кампания за избираеми дисциплини. Основни участници в диаграмата (от ляво надясно) са студентът като актьор, граничният клас на анализа Course Selection UI, контролният клас Student Controller, обектните (entity) класове Elective Campaign, Student и Elective Course. Всички съобщения са синхронни и са описани под формата на текст. В диаграмата присъства и алтернативен блок (alt), който моделира два възможни изхода в зависимост от условието [if Course Requirements are Fulfilled].



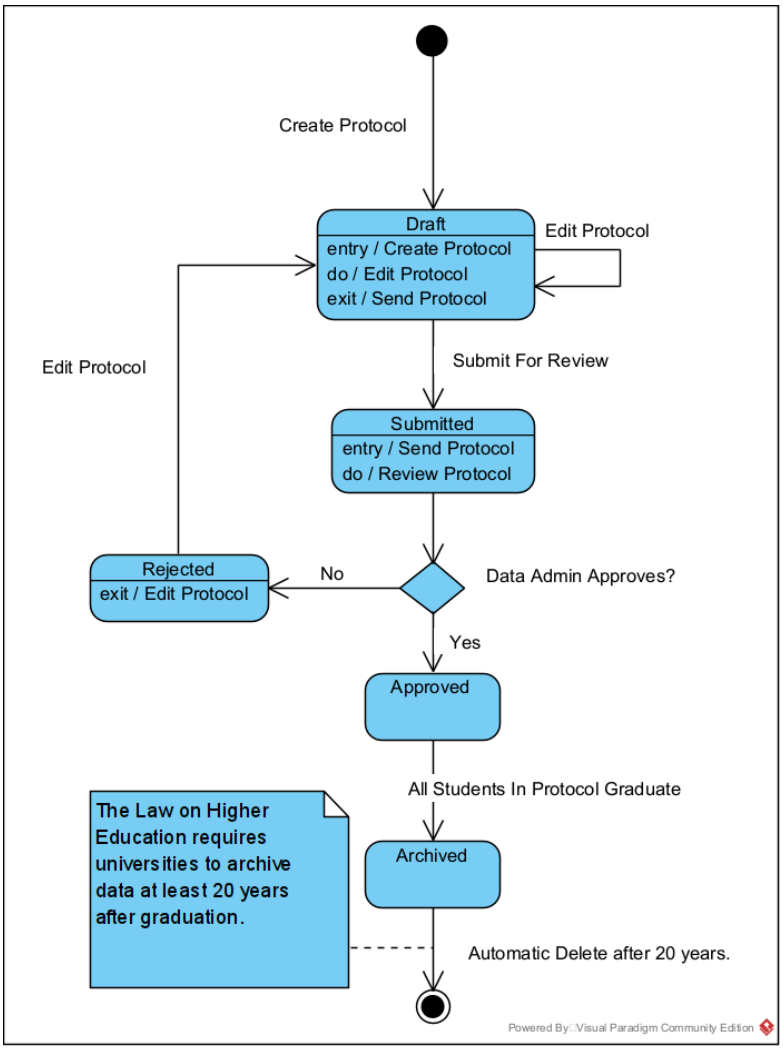
Описание на диаграма на последователността за плащане на семестриална такса от студент: На представената UML диаграма е показан процесът по плащане на семестриална такса от студент по време на кампания за плащане на такси. Основни участници в диаграмата (от ляво надясно) са студентът като актьор, граничният клас FeePaymentUI, контролният клас StudentController и обектните (entity) класове Student и SemesterFeeCampaign и актьорът Payment System. След начална заявка за плащане на такса от страна на студента се изисква цялата информация за самия студент и за активната кампания за плащане на такси. Тук съобщенията са под формата на методи, които да бъдат извикани. Чрез инвариант на състоянието (state invariant) е моделирано условието за успешно платена такса, без което не може да се продължи с по-нататъшното изпълнение на съобщенията от диаграмата. Чрез опционален фрагмент (opt) е заградено изпращането на нотификация за успешно платена такса. Това може да стане само ако е изпълнено условието student.preferredNotifications.contains(AFTER_PAYMENT).



Описание на диаграма на комуникацията за изпращане на съобщение от студент към преподавател: На представената UML диаграма е представен процесът по изпращане на съобщение от студент към преподавател. Основен акцент на диаграмата са представянето на връзките на асоциация между актьорите и отделните класове. Последователността на съобщенията (под формата на методи) може да бъде проследена с помощта на използваната номерация. След като бъдат извлечени всички необходими атрибути за създаване на съобщение, то бива изпратено и получено от съответния преподавател. Потокът на информацията бива моделиран чрез насочените стрелки.



Описание на диаграмата на състоянията на една избираема дисциплина от гледна точка на студента: Диаграмата на състоянията разглежда възможните състояния на записване на една избираема дисциплина по време на кампания за избираеми дисциплини, от гледна точка на студента, който я записва. В зависимост дали кампанията е започнала или не, дали има входен тест или не, дали има лимит за записване и дали той е запълнен или не, статусът на записване на избираема дисциплина минава през различни състояния за всеки един студент.



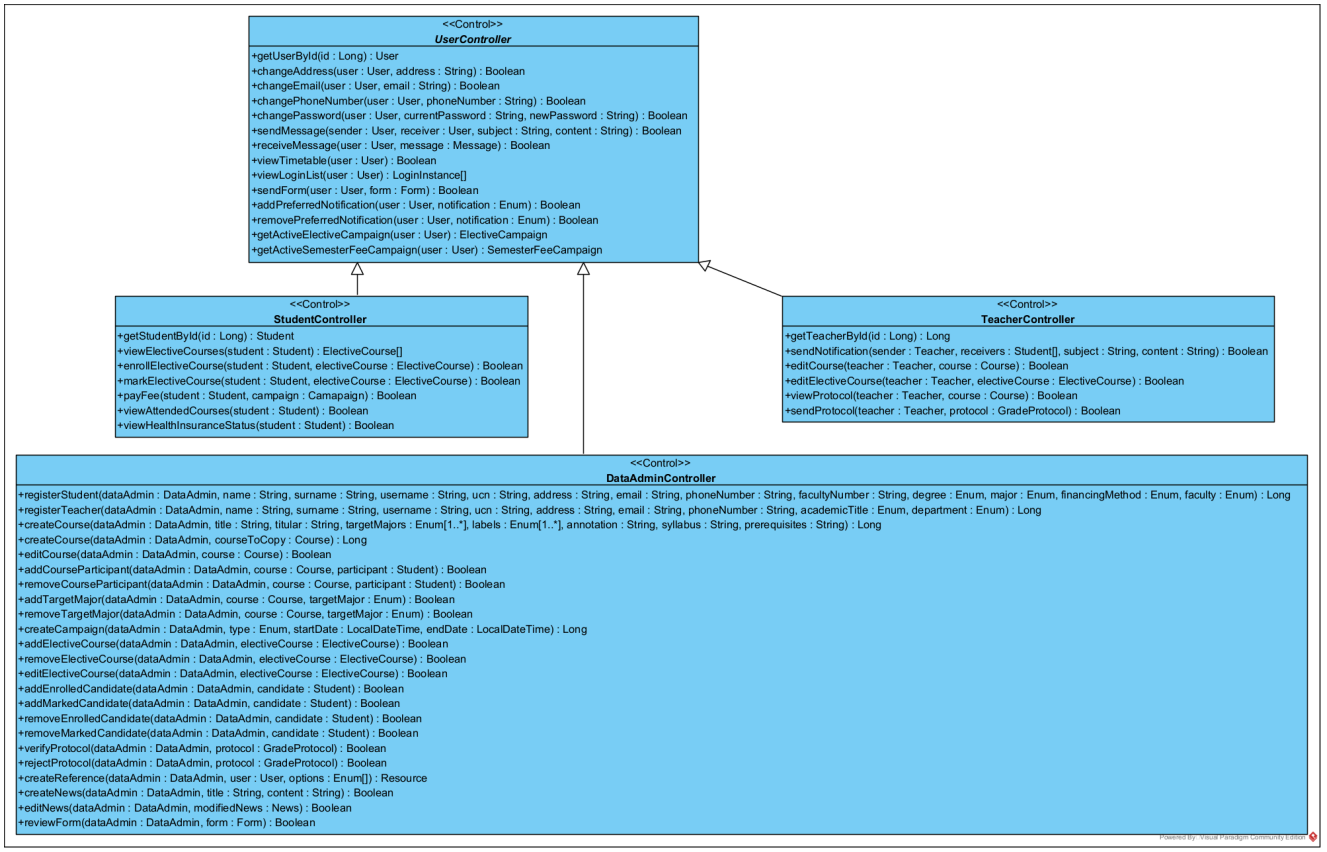
Описание на диаграмата на състоянията на един протокол: Тази диаграма на състоянията моделира различните състояния, в които може да се намира даден протокол в различните етапи от създаването, валидирането и съхраняването му, както и какви дейности настъпват във всяко състояние.

2.4 Модел на съдържанието/данните

Проектът използва множеството от икони *FontAwesome*, което е налично чрез референция към онлайн JavaScript скрипт. Не е необходимо локално копие. Ресурсите под формата на икони се зареждат с HTTP заявки при отваряне на HTML документ, който ги използва.

3. Дизайн

Избраната софтуерна архитектура е клиент-сървър. Софтуерната платформа е *Java* с помощта на *Spring Framework*. За целта всички типове данни и методи на класовете на дизайна са съобразени със синтаксиса на езика. Схемата на софтуерната архитектура ще бъде представена с помощта на пакетна диаграма, компонентна диаграма и диаграма на внедряването.

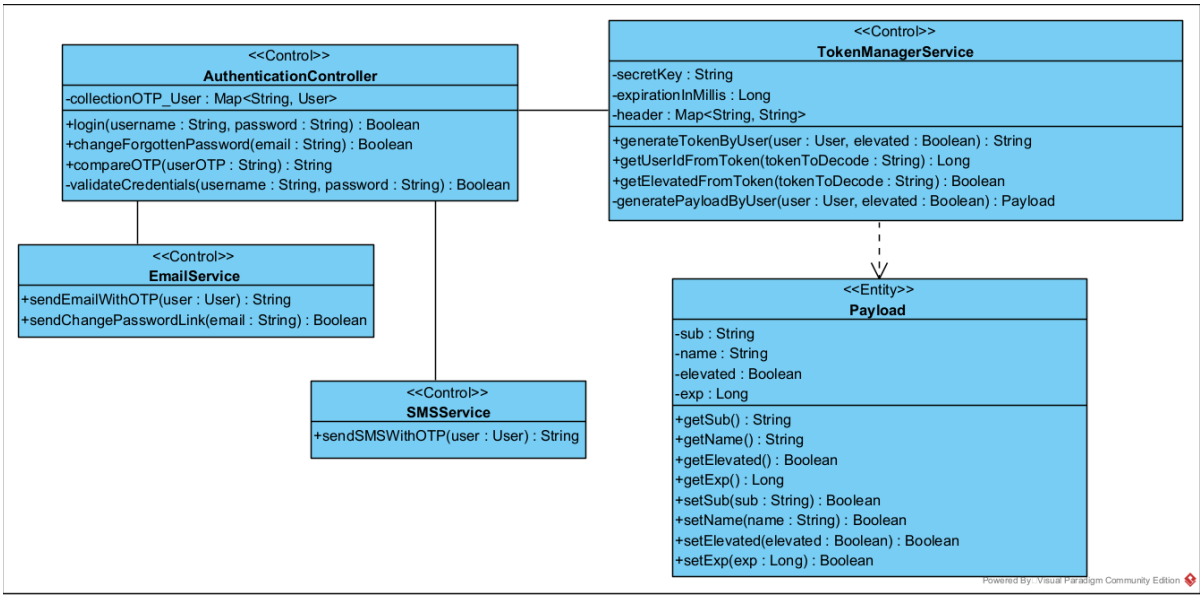


Описание на клас диаграма на контролни (control) класове на дизайна: На представената UML диаграма са показани основните контролни класове на дизайна в студентската информационна система. UserController е абстрактен клас, който бива наследен от контролерите на студенти, преподаватели и администратори на данни. Методите в UserController представят общата бизнес логика, която е обща за всички роли потребители в системата. За да се удостоверят процесите по автентикация и оторизация, всички методи, за които е необходим вход в системата, очакват като първи аргумент съответния потребител.

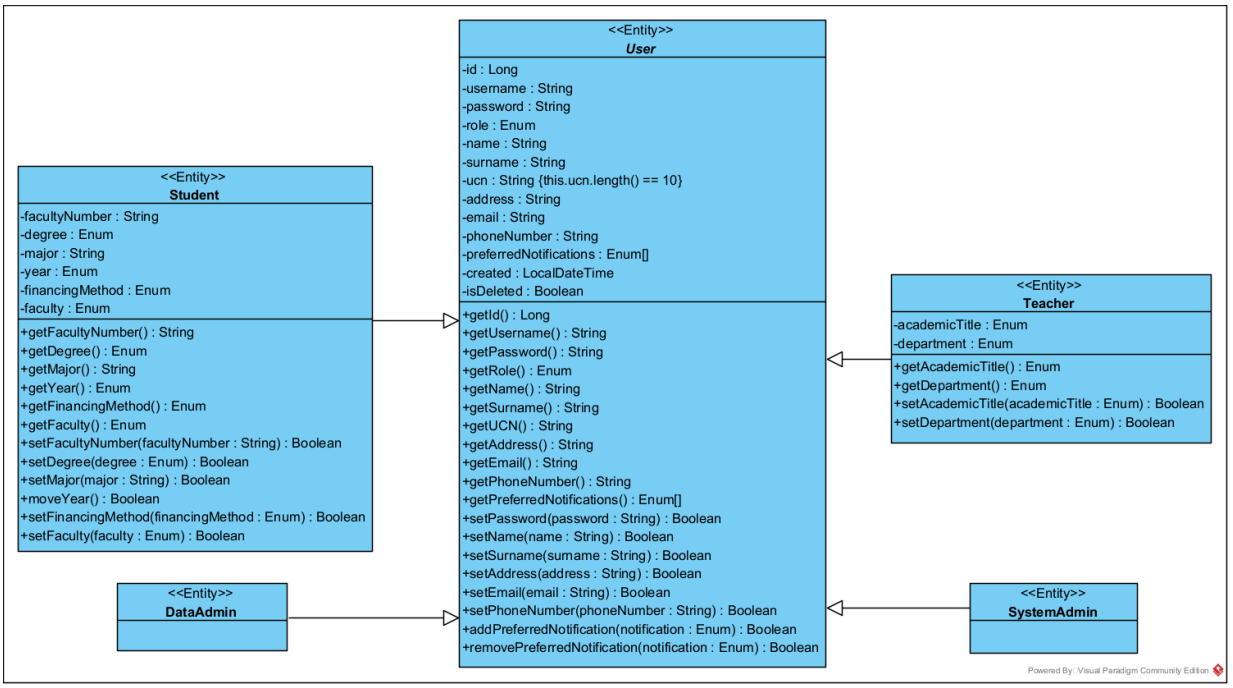
Примерно OCL ограничение:

context DataAdminController::createCampaign(dataAdmin : DataAdmin, type : Enum, startDate : LocalDateTime, endDate : LocalDateTime): Long

pre: startDate.isBefore(endDate)



Описание на клас диаграма на класове на дизайна за процесите по автентикация: На представената UML диаграма са показани основните контролни класове на дизайна, свързани с процесите по автентикация и оторизация в студентската информационна система. В AuthenticationController се намират методите за вход и възстановяване на забравена парола. В контролния клас се осъществява и генерирането на OTP за двуфакторна автентикация. Класовете EmailService и SMSService отговарят за изпращането на имейл и съобщение с кода при двуфакторната автентикация. Класът TokenManagerService отговаря за управлението на токените (tokens). За тяхното генериране се използва методът generateTokenByUser(...), който приема като параметри съответния потребител, за който трябва да се генерира токен (token), както и булев флаг elevated, който означава, че ще се използва за привилегирована операция (изпращане на протокол и др.). Обратната операция се извършва чрез методите getUserIdFromToken(...) и getElevatedFromToken(...). Класът Payload характеризира втората част от съдържанието на всеки токен (token).



Описание на клас диаграма на обектни (entity) класове на дизайна: На представената UML диаграма са показани основните обектни класове на дизайна за отделните роли потребители в системата. Общите атрибути са изнесени в абстрактния клас User. Сред тях са идентификатор (id), потребителско име (username), парола (password), роля (role), име (name), фамилия (surname), ЕГН (ucn), адрес (address), имейл (email), телефонен номер (phoneNumber), предпочитани нотификации (preferredNotifications), дата на създаване на акаунта (created), информация дали акаунтът е изтрит (isDeleted). Класът Student допълнително притежава като член-данни факултетен номер (facultyNumber), образователно-квалификационна степен (degree), специалност (major), година (year), метод на финансиране (financingMethod), факултет (faculty). Класът Teacher допълнително притежава като член-данни академична титла (academicTitle) и катедрата, към която принадлежи (department). Класовете DataAdmin и SystemAdmin се характеризират единствено чрез общите атрибути от абстрактния клас User. За повечето атрибути са предоставени съответните селектори (getters) и мутатори (setters), които са необходими за работа с обектите на тези класове.

Примерни OCL ограничения:

context User

invariant validUCNLength:

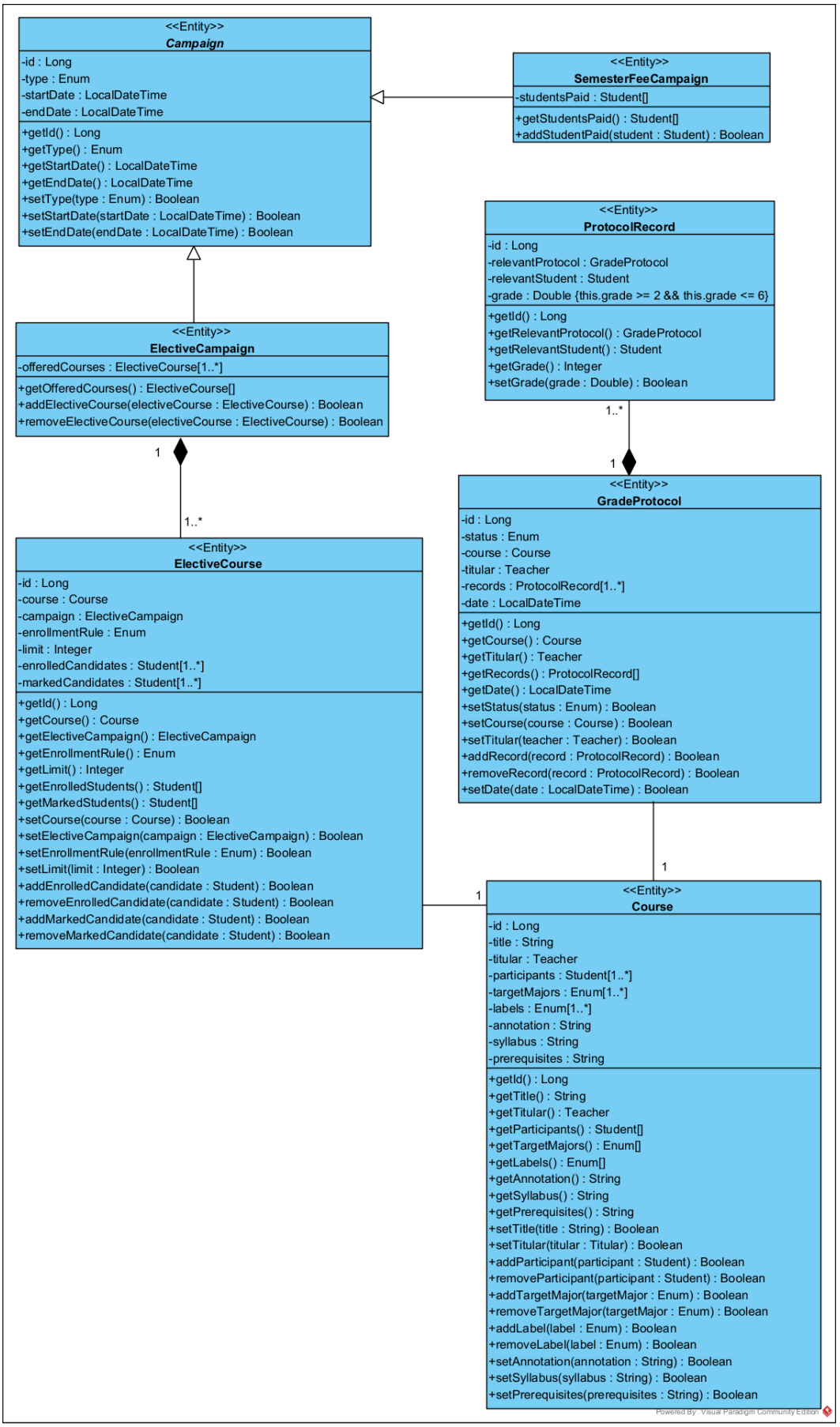
this.ucn.length() == 10

—----------

context User

invariant unqieUCNs:

User::allInstances() -> isUnique(ucn)


<<Entity>>
Campaign
-id : Long
-type : Enum
-startDate : LocalDateTime
-endDate : LocalDateTime
+getId() : Long
+getType() : Enum
+getStartDate() : LocalDateTime
+getEndDate() : LocalDateTime
+setType(type : Enum) : Boolean
+setStartDate(startDate : LocalDateTime) : Boolean
+setEndDate(endDate : LocalDateTime) : Boolean
<<Entity>>
SemesterFeeCampaign
-studentsPaid : Student[]
+getStudentsPaid() : Student[]
+addStudentPaid(student : Student) : Boolean
<<Entity>>
ProtocolRecord
-id : Long
-relevantProtocol : GradeProtocol
-relevantStudent : Student
-grade : Double {this.grade >= 2 && this.grade <= 6}
+getId() : Long
+getRelevantProtocol() : GradeProtocol
+getRelevantStudent() : Student
+getGrade() : Integer
+setGrade(grade : Double) : Boolean
<<Entity>>
ElectiveCampaign
-offeredCourses : ElectiveCourse[1..*]
+getOfferedCourses() : ElectiveCourse[]
+addElectiveCourse(electiveCourse : ElectiveCourse) : Boolean
+removeElectiveCourse(electiveCourse : ElectiveCourse) : Boolean
1
1..*
1..*
1
<<Entity>>
GradeProtocol
-id : Long
-status : Enum
-course : Course
-titular : Teacher
-records : ProtocolRecord[1..*]
-date : LocalDateTime
+getId() : Long
+getCourse() : Course
+getTitular() : Teacher
+getRecords() : ProtocolRecord[]
+getDate() : LocalDateTime
+setStatus(status : Enum) : Boolean
+setCourse(course : Course) : Boolean
+setTitular(teacher : Teacher) : Boolean
+addRecord(record : ProtocolRecord) : Boolean
+removeRecord(record : ProtocolRecord) : Boolean
+setDate(date : LocalDateTime) : Boolean
<<Entity>>
ElectiveCourse
-id : Long
-course : Course
-campaign : ElectiveCampaign
-enrollmentRule : Enum
-limit : Integer
-enrolledCandidates : Student[1..*]
-markedCandidates : Student[1..*]
+getId() : Long
+getCourse() : Course
+getElectiveCampaign() : ElectiveCampaign
+getEnrollmentRule() : Enum
+getLimit() : Integer
+getEnrolledStudents() : Student[]
+getMarkedStudents() : Student[]
+setCourse(course : Course) : Boolean
+setElectiveCampaign(campaign : ElectiveCampaign) : Boolean
+setEnrollmentRule(enrollmentRule : Enum) : Boolean
+setLimit(limit : Integer) : Boolean
+addEnrolledCandidate(candidate : Student) : Boolean
+removeEnrolledCandidate(candidate : Student) : Boolean
+addMarkedCandidate(candidate : Student) : Boolean
+removeMarkedCandidate(candidate : Student) : Boolean
1
1
<<Entity>>
Course
-id : Long
-title : String
-titular : Teacher
-participants : Student[1..*]
-targetMajors : Enum[1..*]
-labels : Enum[1..*]
-annotation : String
-syllabus : String
-prerequisites : String
+getId() : Long
+getTitle() : String
+getTitular() : Teacher
+getParticipants() : Student[]
+getTargetMajors() : Enum[]
+getLabels() : Enum[]
+getAnnotation() : String
+getSyllabus() : String
+getPrerequisites() : String
+setTitle(title : String) : Boolean
+setTitular(titular : Titular) : Boolean
+addParticipant(participant : Student) : Boolean
+removeParticipant(participant : Student) : Boolean
+addTargetMajor(targetMajor : Enum) : Boolean
+removeTargetMajor(targetMajor : Enum) : Boolean
+addLabel(label : Enum) : Boolean
+removeLabel(label : Enum) : Boolean
+setAnnotation(annotation : String) : Boolean
+setSyllabus(syllabus : String) : Boolean
+setPrerequisites(prerequisites : String) : Boolean
Powered By: Visual Paradigm Community Edition

Описание на клас диаграма на обектни (entity) класове на дизайна: На представената UML диаграма са показани основните обектни класове на дизайна, отговарящи за моделирането на курсове, кампании за избираеми дисциплини, заедно с дисциплините към тях, кампания за плащане на такси, протоколи и отделните студентски записи към тях. Класът Campaign е абстрактен клас, който притежава общите атрибути – идентификатор (id), тип (type), начална дата (startDate), крайна дата (endDate). Неговите наследници ElectiveCampaign и SemesterFeeCampaign отговарят за кампаниите за избираеми дисциплини и кампаниите за плащане на такси. Агрегатният клас ElectiveCampaign съдържа множество от избираеми курсове (клас ElectiveCourse). Класовете ElectiveCourse притежават идентификатор (id), асоцииран курс (course), връзка към съответната кампания (campaign), правило за записване (enrollmentRule), ограничение на брой участници (limit), списък от записали към момента курса студенти (enrolledCandidates), списък от студенти, които са маркирали избираемата дисциплина (markedCandidates). Всеки курс (Course) се характеризира чрез идентификатор (id), заглавие (title), титуляр (titular), участници (participants), специалности, които имат възможност да посещават курса (targetMajors), списък от етикети към курса (labels), анотация (annotation), конспект (syllabus), предварителни условия (prerequisites). Към всеки курс се генерира протокол с оценки (GradeProtocol), който освен това притежава и идентификатор (id), статус на протокола (status), титуляр (titular), дата (date) и множество от записи (records), свързани чрез агрегация. Тези записи (ProtocolRecord) притежават идентификатор (id), връзка към конкретен протокол (relevantProtocol), връзка към съответен студент (relevantStudent), оценка (grade).

Примерни OCL ограничения:

context Campaign

invariant validStartDateBeforeEndDate:

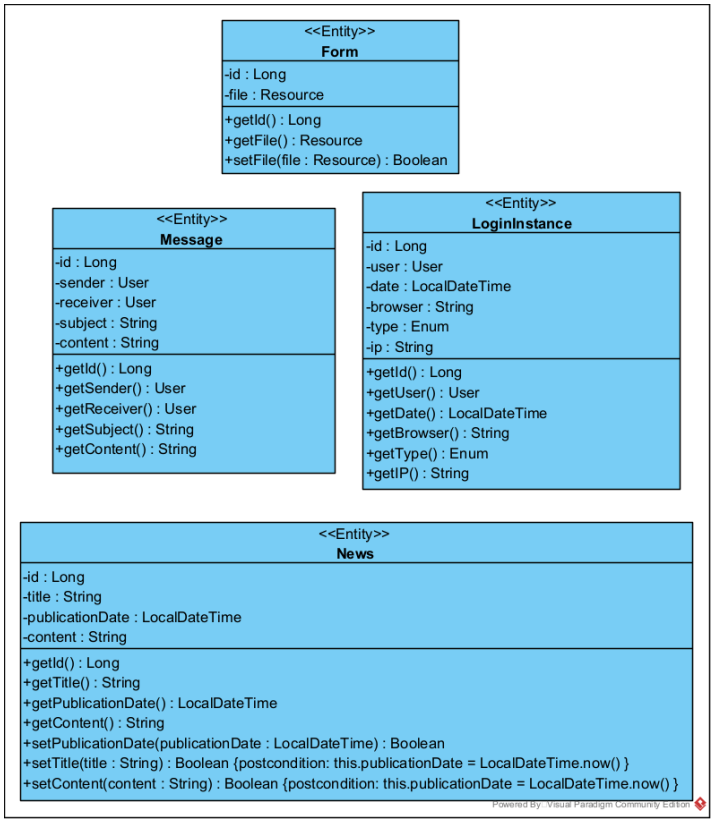
this.startDate.isBefore(this.endDate)

—----------

context ProtocolRecord

invariant validGrade

this.grade >= 2 && this.grade <= 6



Описание на клас диаграма на обектни (entity) класове на дизайна: На представената UML диаграма са показани основните обектни класове на дизайна, отговарящи за моделирането на новини, съобщения, логове, формуляри. Класът News притежава идентификатор (id), заглавие на новината (title), дата на публикуване (publicationDate) и съдържание (content). Класът Message притежава идентификатор (id), подател на съобщението (sender), получател на съобщението (receiver), тема (subject) и съдържание (content). Класът LoginInstance притежава идентификатор (id), потребител, за който се отнася записът (user), дата на записа (date), браузър, от който е осъществен входът (browser), вид на устройството (type), IP адрес на устройството (ip). Класът Form съдържа идентификатор на формуляра (id) и самият документ под формата на ресурс (file).

Примерни OCL ограничения:

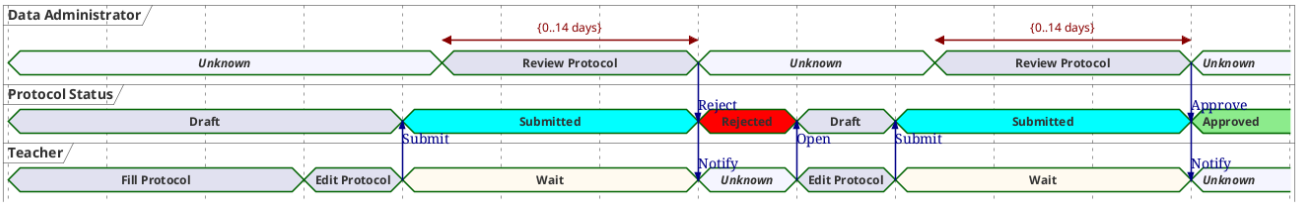
context News::setTitle(title : String): Boolean

post: this.publicationDate = LocalDateTime.now()

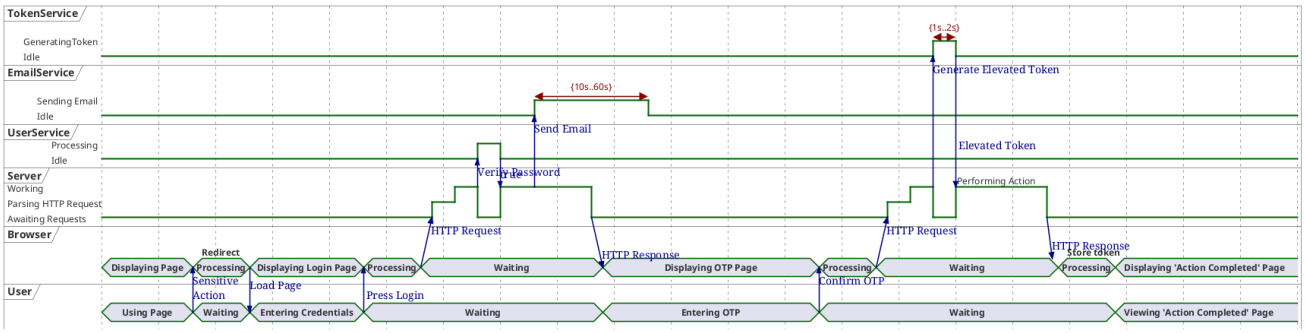
—----------

context News::setContent(content : String): Boolean

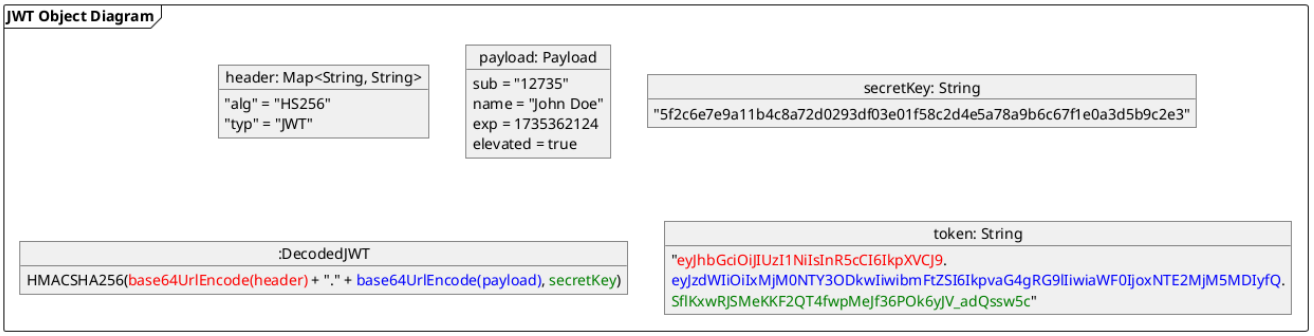
post: this.publicationDate = LocalDateTime.now()



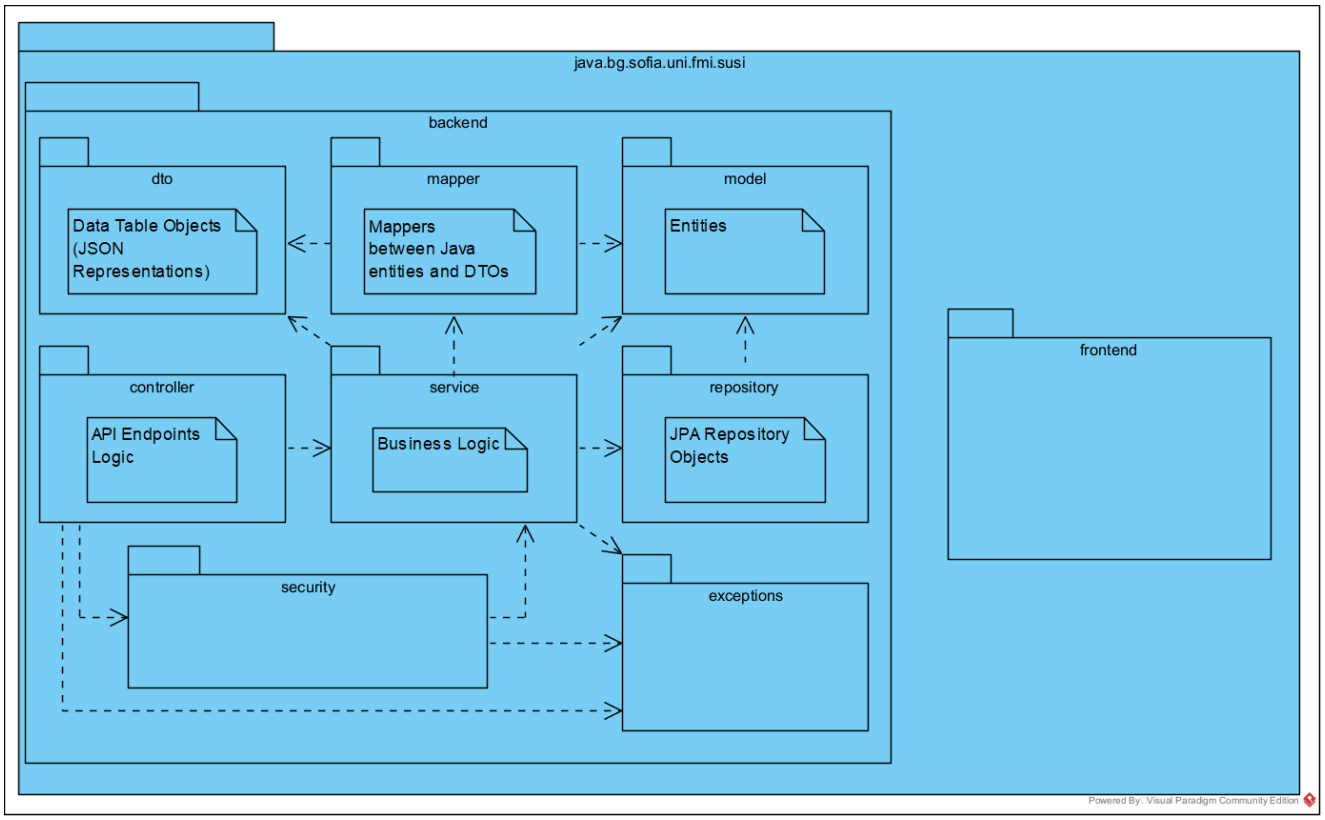
Описание на времевата диаграма на процеса по създаване и валидиране на протокол: Времевата диаграма разглежда отново различните състояния на един протокол в процеса на създаването и валидирането му (но не и архивирането му), но с акцент върху времевата последователност и взаимодействието между актьорите *Преподавател* и *Администратор на данни* и как техните действия променят *статуса на протокола*. Моделирани за изискванията за време за разглеждане на протокол от *Администратор на данни*.



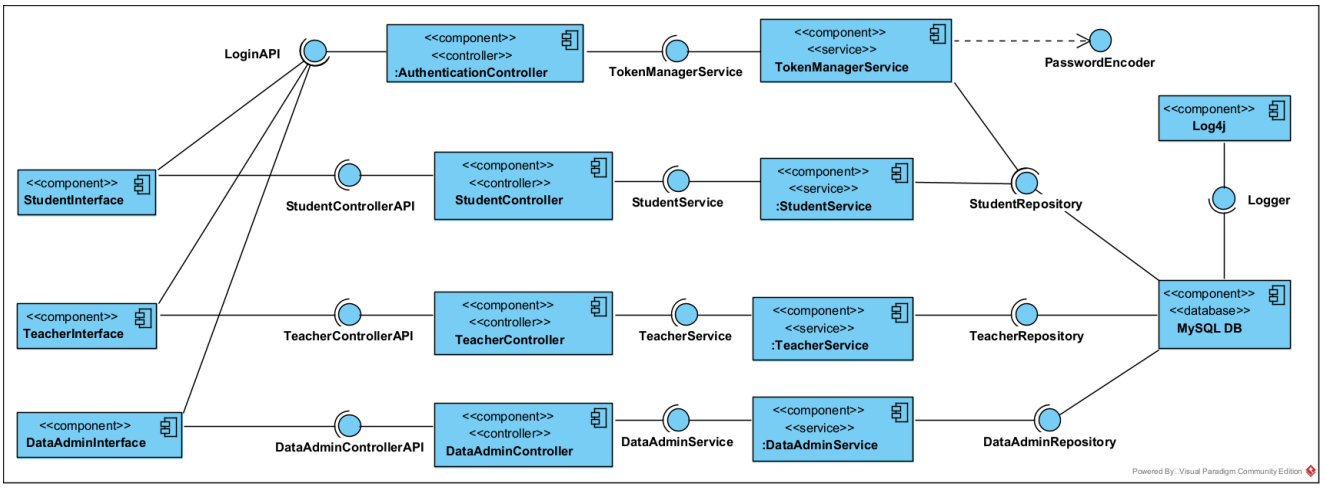
Описание на времевата диаграма на процеса на двуфакторна автентикация чрез издаване на elevated token: Времевата диаграма разглежда времевата последователност от състояния и събития, които настъпват, когато потребител желае да извърши действие, изискващо завишени привилегии - плащане на такса, попълване на протокол, валидиране на протокол и др. Моделирани са времевите изисквания за изпращане на кода и генерирането на JSON Web Token.



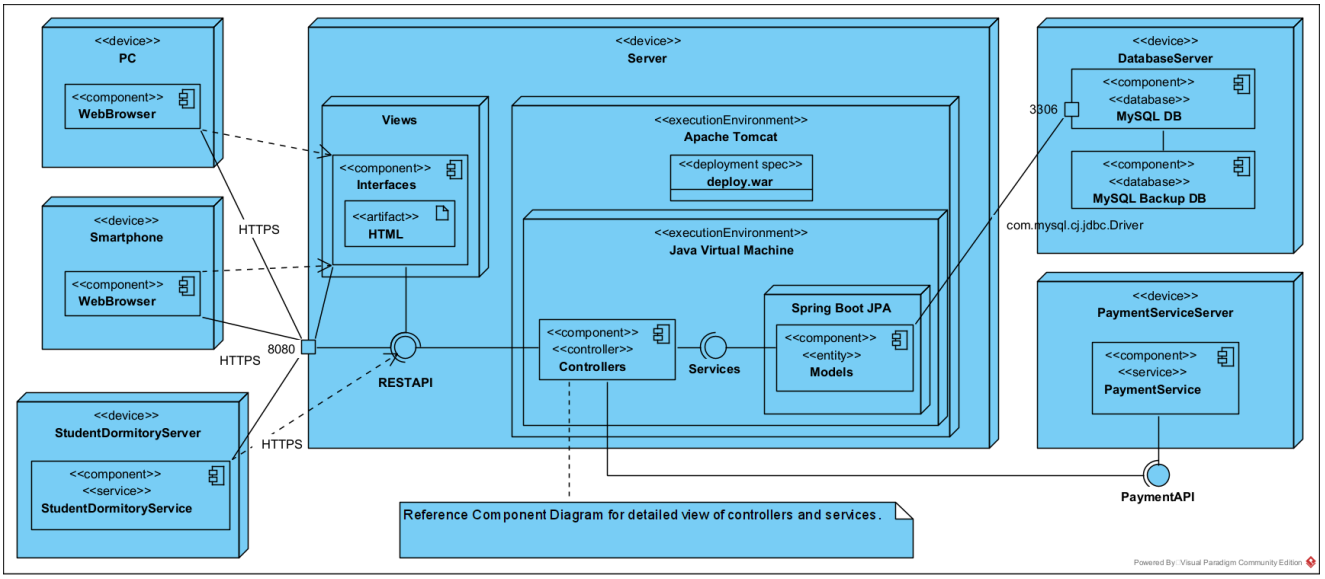
Описание на обектната диаграма на JSON Web Token: Обектната диаграма представя примерни инстанции на *Json Web Token* - специални кодирани (в случая с HS256) низове, които се съхраняват като бисквитки или сесии в браузъра и се използват за уникална автентикация на потребител за даден период от време (например 7 дни). Чрез полу-формална нотация е описана формулата за изграждане на низа. С различен цвят са изобразени съответните части преди и след прилагане на алгоритъма за кодиране.



Описание на пакетната диаграма на системата: Пакетната диаграма разглежда организирането на класовете на системата в пакети съгласно добрите практики за работа с Java и Spring Boot. Работната рамка е базирана на MVC, което обяснява наличието на пакети за Models, Views, Controllers. Работната рамка разширява концепцията за модели (класове в Java) чрез специални класове - *Data Table Objects* (DTOs), *Entities* и *Mappers* (преходи между първите два), чрез които данните се моделират не само като Java класове, но и като JSON и SQL обекти с автоматично преобразуване между различните представяния. В Spring Boot бизнес логиката всъщност се имплементира чрез класове услуги (Services), докато контролерите (Controllers) използват услугите и надграждат с логика за автентикация, за да предоставят endpoints за REST API.



Описание на компонентна диаграма за системата за управление на студентската информация SIMS: На представената UML диаграма е моделирана логическата организация и взаимовръзки между компонентите. Те представляват едно по-високо ниво на абстракция в сравнение с класовете. Отделните интерфейси за студент, преподавател и администратор на данни са моделирани чрез отделни компоненти, които чрез API си комуникират с контролни компоненти, които са също функционално обособени. Те от своя страна предават съобщенията на услуги, които си комуникират посредством Repository интерфейси с базата данни на SIMS. Log4j е компонент, който отговаря за отразяването на събития и изключителни случаи в системата. PasswordEncoder е интерфейс, който предоставя методи за криптиране на паролите. Той е необходим (required) интерфейс (моделиран с помощта на зависимост) за компонента TokenManagerService, който заедно с компонента AuthenticationController, отговаря за дейностите по автентикация и оторизация.



Описание на диаграмата на внедряването на системата SIMS: Диаграмата на внедряването разглежда разпределението на софтуера върху конкретен хардуер. Предвидено е поместването на бизнес логиката на един единствен хардуерен сървър, на който работи Spring Boot чрез Apache Tomcat контейнер. Посочената конфигурация може да удовлетвори изискванията за едновременно ползване от до 2500 потребители по едно и също време. Изгледите са статични HTML артефакти, които динамично се попълват с необходимите данни в браузъра на клиента чрез Javascript Ajax API заявки към REST API, предоставено от Tomcat сървъра. Хардуерният сървър очаква заявки към REST API или конкретни HTML страници (изгледи) на порт 8080. Предвиден е отделен хардуерен сървър за базата данни и резервната база данни, в случай на неизправност с първата. Сървърът за база данни е свързан със Spring чрез драйвера com.mysql.cj.jdbc.Driver и очаква SQL заявки на порт 3306. Заявките се генерират от възела Spring Boot JPA (Java Persistence API), който чрез видовете класове, споменати в описанието на пакетната диаграма, автоматично съпоставя на Java getters и setters SQL заявки. Такава архитектура позволява лесна скалируемост чрез разпределена база данни в бъдеще.

Моделирани са също различни видове клиенти, външната система за плащане и съществуващата външна система за администриране на студентските общежития, които консумират REST API на SIMS.

Така проектирана и внедрена системата чрез универсален REST API, тя позволява неограничени възможности за видовете клиенти, които могат да ползват системата, и информацията, която могат да достъпват от нея, стига да имат необходимите права.

4. Тестване

- **Функционално тестване** - тестване на всички основни контролни класове
    - Проверка на вход в системата с валидни и невалидни данни за вход.
    - Тестване на възстановяване на парола чрез имейл.
    - Тестване дали записаните/преподаваните курсове са видими в седмичното разписание.
    - Тестване на създаване, редактиране и изпращане на протоколи от преподаватели.
    - Уверяване, че студентите могат правилно да преглеждат, маркират и записват избираеми дисциплини.
    - Тестване на процеса на плащане на такси при различни сценарии (успешно плащане, върната грешка от външната система за плащане).
    - Проверка на изпращането на съобщения.
    - Проверка на REST API чрез Swagger
- **Интеграционно тестване**
    - Тестване на интеграцията между системата и външна система за плащане на такси.
- **Тестване на потребителския интерфейс (UI) и използваемостта**
    - Проверка на отзивчивостта и удобството на потребителския интерфейс както на десктоп, така и на мобилни устройства.
    - Тестване на съответствието с изискванията за достъпност по стандарт WCAG 2.1 (напр. съвместимост с екранни четци,, контраст).
- **Тестване на производителността**
    - Тестване на производителността на системата при едновременен вход на 2500 потребители.
    - Измерване на времето за реакция на операции с времеви лимит като изпращането на OTP код по имейл, генерирането на token и др.
- **Сигурност**
    - Тестване на преноса по мрежа чрез HTTPS.
    - Проверка на надеждността на двуфакторното удостоверяване.
    - Провеждане на тестове за проникване с цел идентифициране и отстраняване на потенциални уязвимости.
- **Тестване на натоварването**
    - Оценка на производителността на системата при високо натоварване, например по време на кампании за записване на дисциплини.

- **Тестване на базата данни**
  - Валидация на точността и консистентността на съхраняването и извличането на данни.
  - Тестване на процесите за архивиране и възстановяване за гарантиране на целостта на базата данни чрез резервната БД.
- **Съвместимост**
  - Проверка, че системата работи безпроблемно на различни устройства, операционни системи и браузъри.
  - Проверка, че системата успешно се съвместява със системата за управление на студентските общежития.

5. Заключение и възможно бъдещо развитие

SIMS е проект за нова студентска информационна система, която цели да предостави подобрени функционалности, сигурност и потребителско изживяване. За целта бяха посочени подробен набор от атомарни, консистентни и измерими изисквания, които такава система би следвало да спазва.

Бяха избрани публични технологии, работни рамки и стандарти с отворен код като Java, Spring Boot, TLS, HTTPS и др., които са с по-добра прозрачност в сравнение с работни рамки като .NET, които са с отворен код едва отскоро и развитието им се задава от една единствена корпорация (Microsoft). При това не се налага да се прави компромис с нефункционални изисквания като надеждност и скалируемост, защото Spring Boot Framework е с доказана ефективност в практиката.

При проектирането на системата взехме някои решения като създаване на резервна база данни, двуфакторна автентикация и други, които усложняват дизайна ѝ, но гарантират спазване на всички посочени изисквания, а също възможности за скалируемост, надграждане и подобрение в бъдеще. Допускаме, че може би съществуват технологии и техники, които биха били по-добри решения на поставения проблем, но с които не сме запознати към момента на проектирането на системата.

Системата е сложна за моделиране поради многото различни класове и многото изисквания. Нашият малък екип от 2-ма човека се фокусира само върху тези аспекти от модела, които са най-трудни за разбиране или са от ключова важност за системата, чрез 22 UML диаграми от различни видове, някои от които не бяха изисквани в шаблона на документа, но които ние включихме поради тяхната уместност в случая. Липсата на модели за абсолютно всички класове и процеси е единствено поради времевата невъзможност на екипа ни да моделира всички тях. Една възможна посока на подобрение на курсовия проект е включване на останалите класове и процеси.

Причината да изберем по-сложна тема за курсов проект беше, че искахме да изберем тема, която позволява да бъдат демонстрирани по-голямо множество различни видове UML диаграми.

Може също да се отбележи, че за Timing и Object диаграмите беше използван инструмент, различен от Visual Paradigm - PlantUML, което доведе до различен външен вид за тези диаграми. Това беше необходимо поради изтичане на пробния период на платената версия на Visual Paradigm и затрудненатa работа с безплатната версия, при която голяма част от опциите в графичния интерфейс бяха скрити. За щастие, опитът ни с PlantUML беше положителен и го препоръчваме за някои видове диаграми.

В заключение, курсовият проект демонстрира възможностите на екипа успешно да моделира обектно-ориентирани системи чрез UML диаграми чрез различни видове инструменти.

6. Разпределение на работата

Въведението, функционалните и нефункционалните изисквания, графичният интерфейс, тестовият план и генерирането на кода бяха разработени съвместно. Диаграмите бяха създадени, както следва:

Диаграма	Изпълнител
UML Use Case Diagram (1) и UseCaseTables	Даниел Халачев
UML Analysis Class Diagrams (Boundary, Control, Entity) (3)	Стефан Велев
UML Design Class Diagrams (5)	Стефан Велев
UML JWT Object Diagram (1)	Даниел Халачев
UML Activity Diagrams (2)	Даниел Халачев
UML Sequence Diagrams (2)	Стефан Велев
UML Communication Diagram (1)	Стефан Велев
UML Package Diagram (1)	Даниел Халачев
UML Component Diagram (1)	Стефан Велев
UML Deployment Diagram (1)	Даниел Халачев
UML State Diagrams (2)	Даниел Халачев
UML Timing Diagrams (2)	Даниел Халачев

7. Използвани ресурси

1. Visual Paradigm User's Guide, Visual-Paradigm.com, https://circle.visual-paradigm.com/docs
2. Timing Diagram Syntax and Features, PlantUML.com, https://plantuml.com/timing-diagram
3. Object Diagram Syntax and Features, PlantUML.com, https://plantuml.com/object-diagram

8. Апендикс

Извадка от генериран примерен сорс код на Java:

```
package java.bg.sofia.uni.fmi.susi.backend.model;


import java.time.LocalDateTime;


public class News {
  private Long id;
  private String title;
  private LocalDateTime publicationDate;
  private String content;

 // Constructor
  public News(Long id, String title, LocalDateTime publicationDate, String content) {
     this.id = id;
     this.title = title;
     this.publicationDate = publicationDate;
     this.content = content;
  }

  // Getters
  public Long getId() {
     return id;
  }
```

```
    public String getTitle() {
        return title;
    }

    public LocalDateTime getPublicationDate() {
        return publicationDate;
    }

    public String getContent() {
        return content;
    }

    // Setters
    public void setPublicationDate(LocalDateTime publicationDate) {
        this.publicationDate = publicationDate;
    }

    public void setTitle(String title) {
        this.title = title;
        this.publicationDate = LocalDateTime.now();
    }

    public void setContent(String content) {
        this.content = content;
        this.publicationDate = LocalDateTime.now();
    }
}
```

```
package java.bg.sofia.uni.fmi.susi.backend.model;


import java.time.LocalDateTime;

import java.util.ArrayList;

import java.util.List;


public abstract class User {

  private Long id;

  private String username;

  private String password;

  private Enum role;

  private String name;

  private String surname;

  private String ucn; // Universal Citizen Number (must be 10 characters)

  private String address;

  private String email;

  private String phoneNumber;

  private List<Enum> preferredNotifications;

  private LocalDateTime created;

  private Boolean isDeleted;


  // Constructor

    public User(Long id, String username, String password, Enum role, String name, String
surname,

                String ucn, String address, String email, String phoneNumber, List<Enum>
preferredNotifications) {

    this.id = id;

    this.username = username;

    this.password = password;

    this.role = role;
```

```
        this.name = name;
        this.surname = surname;
        setUCN(ucn);
        this.address = address;
        this.email = email;
        this.phoneNumber = phoneNumber;
            this.preferredNotifications = preferredNotifications != null ? preferredNotifications : new
ArrayList<>();
        this.created = LocalDateTime.now();
        this.isDeleted = false;
    }

    // Getters
    public Long getId() {
        return id;
    }

    public String getUsername() {
        return username;
    }

    public String getPassword() {
        return password;
    }

    public Enum getRole() {
        return role;
    }
```

```
public String getName() {
    return name;
}

public String getSurname() {
    return surname;
}

public String getUCN() {
    return ucn;
}

public String getAddress() {
    return address;
}

public String getEmail() {
    return email;
}

public String getPhoneNumber() {
    return phoneNumber;
}

public List<Enum> getPreferredNotifications() {
    return preferredNotifications;
}
```

```
// Setters
public boolean setPassword(String password) {
    if (password != null && !password.isEmpty()) {
        this.password = password;
        return true;
    }
    return false;
}

public boolean setName(String name) {
    if (name != null && !name.isEmpty()) {
        this.name = name;
        return true;
    }
    return false;
}

public boolean setSurname(String surname) {
    if (surname != null && !surname.isEmpty()) {
        this.surname = surname;
        return true;
    }
    return false;
}

public boolean setUCN(String ucn) {
    if (ucn != null && ucn.length() == 10) {
        this.ucn = ucn;
```

```
            return true;
        }
        return false;
    }

    public boolean setAddress(String address) {
        if (address != null && !address.isEmpty()) {
            this.address = address;
            return true;
        }
        return false;
    }

    public boolean setEmail(String email) {
        if (email != null && email.contains("@")) {
            this.email = email;
            return true;
        }
        return false;
    }

    public boolean setPhoneNumber(String phoneNumber) {
        if (phoneNumber != null && !phoneNumber.isEmpty()) {
            this.phoneNumber = phoneNumber;
            return true;
        }
        return false;
    }
```

```
    public boolean addPreferredNotification(Enum notification) {
        if (notification != null && !preferredNotifications.contains (notification)) {
            preferredNotifications.add(notification);
            return true;
        }
        return false;
    }

    public boolean removePreferredNotification(Enum notification) {
        if (notification != null && preferredNotifications.contains(notification)) {
            preferredNotifications.remove(notification);
            return true;
        }
        return false;
    }
}
```