UNIVERSITY MANAGEMENT

Курсов Проект

По

Дисциплината Архитектури на софтуерни системи

Барие Банцова, Елисавета Тодорова
СОФИЙСКИ УНИВЕРСИТЕТ "СВ. КЛИМЕНТ ОХРИДСКИ"
Факултет по математика и информатика
Магистърска програма „Софтуерни технологии"

Януари, 2020

СЪДЪРЖАНИЕ

1. **Въведение**

1.1 Организация на текущия документ

1.1.1 Предназначение на документа

Целта на документа е да представи софтуерната архитектура на системата University Management.

1.1.2 Списък на структурите

За всяка структура се пояснява какъв точно аспект на системата показва, кои елементи включва и какви са връзките между тях.

- Декомпозиция на модулите – показва обособяването на отделни модули (логически обособени единици) в рамките на системата. Най-общо системата е разделена на модулите: UMSSite, UMSServer, ReportAPI, UMSDatabase, External Systems. UMSSite е потребителският интерфейс на системата. Чрез него студентите, преподавателите и работниците в различните отдели на университета могат да извършват различни дейности. Той се свързва с UMServer, който държи бизнес логиката на приложението. UMServer-а се свързва към базата данни. ReportAPI е програмен интерфейс, който се използва за генериране на официални справки и публични събития. UMSServer-а може да си комуникира ReportAPI- то. UMSServer-а си комуникира и със всяка една от външните системи. Тази структура е описана в секция 2 на настоящия документ.

- Допълнителни структури – в секция 3 от настоящия документ се разглеждат следните структури:
  - Употреба на модули – Разглеждат се връзките между модулите, обособени в Декомпозицията. Това са връзки от “тип модул А използва модул Б”. Тази структура е описана в секция 3.1 на настоящия документ.
  - Структура на внедряването – показва връзките между софтуерните елементи със средата, в която ще бъде използвана системата. Елементите са приложения, сървъри и външни системи. Тази структура е описана в секция 3.2 на настоящия документ.
  - Структура на процесите – Разглеждат се някои от по-важните процеси в системата. Тази структура е описана в секция 3.3 на настоящия документ.
- Архитектурна обосновка – защитават се архитектурните решения, които са взети за системата, с оглед отделните изисквания и използваните тактики за постигането им. Съдържа се в секция 4 на настоящия документ.

1.2 Общи сведения за системата

University Management е софтуерна система за управление на процесите и студентската информация в един университет.

Системата обслужва няколко учебни звена – учебен, счетоводен и административен отдел, както и студентски съвет, преподаватели и студенти. Всяко отделно звено в системата и неговите потребители притежават определени права за създаване и модериране на дадената информация, с която те разполагат, като поддържа защита на всички лични и финансови данни от неоторизиран достъп.

Потребителите от учебен отдел могат да приемат предложения за учебни планове и програми от преподавателите. Предложенията за учебни планове (специалности) и програми могат да бъдат одобрявани от административния отдел.

Системата поддържа профили на студентите и преподавателите, в които се записват техните данни, както и информация за техните компетентности. Системата предоставя и възможност за обмяна на лични съобщения между потребителите. Потребителите от счетоводния отдел, могат да контролират финансовите операции, които засягат другите потребители, както и разплащания с външни изпълнители на услуги.

Всеки студент има своя електронна студентска книжка, където преподавателите да нанасят техните оценки. Всеки студент може да се запише за одобрен курс само в рамките на своята специалност и при условие, че профилът му отговаря на входните изисквания за компетентности за съответния курс. Системата предоставя на студентите възможност за генериране на различни видове официални справки за студентския им статус или да предостави такива на хартия.

1.3 Списък на софтуерните елементи.

2.2. UMSSite

2.2.3.1. Account Manager

2.2.3.1.1. Register

2.2.3.1.2. Login

2.2.3.1.3. Logout

2.2.3.2. Payment Manager

2.2.3.3. Profile Manager

2.2.3.3.1. User

2.2.3.3.1.1. Student

2.2.3.3.1.1.1 View Student Book

2.2.3.3.1.1.2 Register for course

2.2.3.3.1.1.3 Request Enquire

2.2.3.3.1.2. Teacher

2.2.3.3.1.2.1 Add Course

2.2.3.3.1.2.2 Create Request

2.2.3.3.1.2.3 Add student mark

2.2.3.3.1.3. Student Council

2.2.3.3.1.3.1 Create Request

2.2.3.3.1.4. Administrative Worker

2.2.3.3.1.4.1 Review request

2.2.3.3.1.4.2 Review/Send report data

2.2.3.3.1.5. Accounting Worker

2.2.3.3.1.5.1 Execute Payment

2.2.3.3.1.6. Education Worker

2.2.3.3.1.6.1 Review program course

2.2.3.3.1.6.1 Review/Send report data

2.2.3.3.1.7. Report Manager

2.2.3.3.1.7.1 View Report

2.2.3.3.1.7.2 Generate Report

2.3. UMSServer

2.3.3.1. Database Manager

2.3.3.2. Account Manager

2.3.3.2.1. Register

2.3.3.2.2. Login

2.3.3.2.3. Logout

2.3.3.3. Course Manager

2.3.3.3.1. Create

2.3.3.3.2. Update

2.3.3.3.3. Delete

2.3.3.3.4. Get information

2.3.3.3.5. Assign participants

2.3.3.4. Event Manager

2.3.3.4.1. Add

2.3.3.4.2. Delete

2.3.3.5. Enquire Manager

2.3.3.5.1. Generate report

2.3.3.5.2. Send student report

2.3.3.5.3. Content management

2.3.3.6. Encryption

2.3.3.6.1. Encrypt

2.3.3.6.2. Get decryption key

2.3.3.7. Student Manager

2.3.3.7.1. Register

2.3.3.7.2. Manage profile

2.3.3.7.3. Get information

2.3.3.7.4. Update student profile

2.3.3.7.5. Assign curricilum

2.3.3.8. Messaging Manager

1.4 Разширен терминологичен речник

Софтуер – съвкупността от цялата информация от инструкции и данни, необходими за работата на всяка електронноизчислителна машина.

Софтуерна система – краен продукт от комуникиращи помежду си компоненти (програми), които са част от една компютърна система

Компютърна система – съвкупност от хардуер, софтуер и информационни данни, които са необходими за функционирането на една система

Външна система – отдалечена софтуерна система, която е източник на данни или функционалности, които се използват от настоящата система

Операционна система – система, която управлява и координира ресурсите на хардуера и софтуера и обслужва изпълняваните компютърни програми.

Сървър – софтуерна услуга, стартираща на специално предназначен за нея компютър.

Application server – сървър, държащ главната бизнес логика на системата

Database server – сървър, на който се помещава база от данни

Web server – сървър, който приема и връща отговор на заявки, направени от клиент като редоставя информацията на клиента под формата на HTML документ.

База от данни – представлява колекция от логически свързани данни в конкретна предметна област, които са структурирани по определен начин.

Интерфейс – споделена граница между два разделени компютърни компонента, обменящи информация.

Модул – логически обособена софтуерна единица

Процес - съвкупност от стъпки, която изгражда логическо действие и стига определена цел

Потребител – човек, който използва компютърна или мрежова услуга

Клиент – част от компютърна или софтуерна система, която достъпва услуга, предоставена от сървър

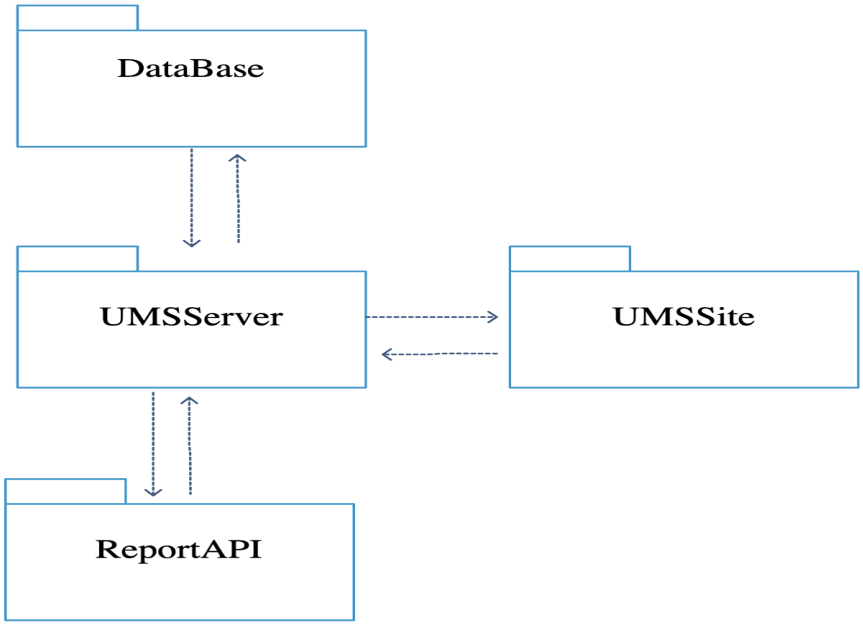
Криптиране – кодиране (скриване) на информация

Декриптиране – разкодиране на информация

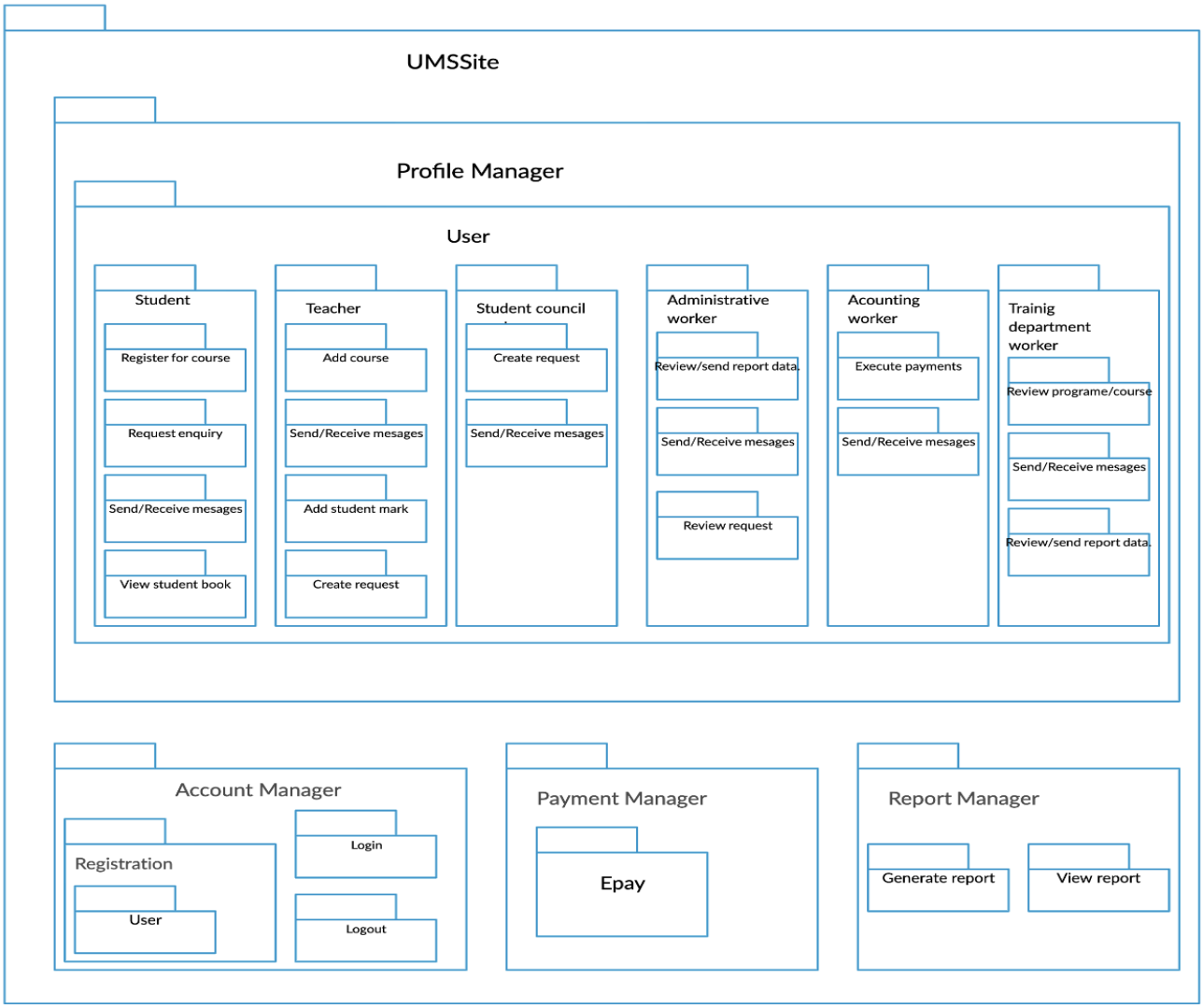
Декомпозиция - софтуерна структура, показваща как системата се разделя на отделни модули.

2 Декомпозиция на модулите

2.1 Общ вид на Декомпозицията на модули за системата



2.2 **UMSSITE**



2.2.1 Предназначение на модула

Модулът UMSSite представлява уеб приложение, чрез което различните видове потребители могат да управляват профилите си след регистрация.

2.2.2 Основни отговорности в системата

Задачата на модула е да визуализира по подходящ начин данните, които получава от сървъра, да предоставя необходими форми за въвеждане и модериране на данни от потребителите, да изпраща тези данни на сървъра и да визуализира получения отговор на потребителя.

2.2.3 Описание на интерфейсите на модулите

2.2.3.1. **Account Manager** - модула събира наличните функционалности за автентикация и ауторизация на потребителите на системата.

public User GetCurrentUser()

public bool IsAuthenticated()

2.2.3.1.1. **Registration** - отговаря за попълването на необходимите данни от потребителя и изпращането им до модул Сървър, където се случва самата регистрация. На негово ниво трябва да се валидират данните.

public string GetUsername()

public string GetPassword()

public string GetEmail()

public bool ValidateUsername (string username)

public bool ValidatePassword (string password)

public bool ValidateEmail (string email)

public void SendRegisterRequest (string username, string password, string email)

Вход: Потребителско име, Имейл, Парола

Изход: Съобщение дали операцията е успешна или не

2.2.3.1.2. **Login** - предоставя форма за попълване на данните от потребителя, след което ги изпраща до модул Сървър. Този модул е отговорен и за показване по подходящ начин на отговора, който сървъра е върнал, а именно дали влизането в системата е успешно или не.

public string GetUsername()

public string GetPassword()

public void SendLoginRequest (string username, string password) public bool ValidateUsername (string username)

public bool ValidatePassword (string password)

Вход: Потребителско име, Парола

Изход: Съобщение дали операцията е успешна или не

2.2.3.1.3. **Logout** – реагира на заявка за изход от системата от потребителя като изпраща заявка до Сървъра. При успех изтрива всички данни за текущия потребител, които пази в браузъра, сесията и връща потребителя в екрана за влизане в системата.

public void SendLogoutRequest ()

Вход: void
Изход: Съобщение дали операцията е успешна или не

2.2.3.2. **Payment Manager** - Този модул трябва се свързва с външна система(ePay) за осъществяване на транзакции. При успешно свързване и осъществяване на транзакцията информацията се изпраща на сървъра.

public void TransferFunds (decimal money, User sender, User receiver)

Вход: Сумата (големината) на превода; Потребителя, изпращащ сумата; Получателя на сумата
Изход: Съобщение дали операцията е успешна и/или списък от грешки

2.2.3.3. Profile Manager е достъпен след като потребителя бива ауторизиран и неговата самоличност - устанивена. Функционалностите му са разделени на групи, в зависимост от ролята, която играе потребителя в системата. Следващите описани методи са общи за подфункционалностите на профил мениджъра.

public void SendMessage (String message, User receiver)

Вход: Съобщението; Получателя на съобщението
Изход: Съобщение дали операцията е успешна и/или списък от грешки

public void MakeEventPost (Post postEvent)

Вход: Обект съдържащ всички входни параметри на поста.
Изход: Съобщение дали операцията е успешна и/или списък от грешки

2.2.3.3.1. **User**

2.2.3.3.1.1 **Student**

2.2.3.3.1.1.1 R**egister for course** - потребителя изпраща заявка за записване в дадена дисциплина. При задействането му се праща информация на сървъра, който определя дали записа е успешен и връща отговор, който модула трябва да визуализира по подходящ начин.
public void SendRegisterRequest (String courseName)

Вход: Името на курса, който потребителя иска да запише

Изход: Съобщение дали операцията е успешна и/или списък от грешки

2.2.3.3.1.1.2 **View student book -** модула изпраща заявка до сървъра и той връща необходимата информация. Може да се разглеждат цялата налична информация за предметите и оценките на един студент.
public void ViewStudentBook (String studentName)

Вход: Име на търсения студент
Изход: Визуализация на студентската книжка

2.2.3.3.1.1.3 **Request Enquire -** модула изпраща заявка до сървъра за извеждане на заявки, като уверения.

public void RequestNewEnquire (String typeOfEnquire, EnquireData enquireData)

Вход: Тип на заявката; Информация касаеща заявката
Изход: Съобщение дали заявката е приета или не

2.2.3.3.1.2 **Teacher**

2.2.3.3.1.2.1 **Add Course** - потребителя изпраща заявка за създаване на нов курс/дисциплина. При задействането му се праща информация на сървъра, който изпраща заявката до Курс Мениджъра. Той определя дали записа е успешен и връща отговор дали курса е регистриран.
На преподавателя се визуализира форма за попълване на данни.

public void SendRegisterNewCourseRequest (CourseData courseData, String sender, String senderResoning)

Вход: Данни за създаване на новия курс. Име на заявителя; Основание за заявката

Изход: Съобщение дали операцията е успешна и/или списък от грешки

2.2.3.3.1.2.2 **Create Request -** модула изпраща заявка до сървъра за създаване на general requests и той връща необходимата информация.

public void CreateGeneralRequest (GeneralRequestData requestData)

Вход: Данни за създаване на нова заявка
Изход: Съобщение дали операцията е успешна и/или списък от грешки

2.2.3.3.1.2.3 **Add Student Mark -** модула изпраща заявка до сървъра за въвеждане на оценка по даден предмет на студент.

public void CreateStudentMark (String studentName, int mark, String courseName)

Вход: Име на студента; Оценка; Име на курса
Изход: Съобщение дали заявката е приета или не и дали оценката е запазена

2.2.3.3.1.3 **Student Council**

2.2.3.3.1.3.1 **Create Request -** модула изпраща заявка до сървъра за създаване на general requests и той връща необходимата информация.

public void CreateGeneralRequest (GeneralRequestData requestData)

Вход: Данни за създаване на нова заявка
Изход: Съобщение дали операцията е успешна и/или списък от грешки

2.2.3.3.1.4 **Administrative Worker**

2.2.3.3.1.4.1 **Review Request -** модула изпраща заявка до сървъра за преглеждане и одобряване на дадена заявка, като например след одобрение за запис на нов курс. Той връща необходимата информация дали заявката е одобрена или не .

public void ReviewRequest (RequestData requestData)

Вход: Данни на заявката
Изход: Съобщение дали заявката е удобрена или не

2.2.3.3.1.4.2 **Review/Send Report Data -** модула изпраща заявка до сървъра за преглеждане и одобряване на даден report. Той връща необходимата информация дали заявката е удобрена или не .

public void ReviewReport (ReportData requestData)

Вход: Данни на репорта
Изход: Съобщение дали заявката е удобрена или не

2.2.3.3.1.5 **Accounting Worker**

2.2.3.3.1.5.1 **Execute Payments -** Задейства механизъм за извършване на плащания към служители или други потребители. При задействането му се праща информация на сървъра, който определя дали плащанията са успешни и връща отговор, който модула трябва да визуализира по подходящ начин.

public void ExecutePaymentRequest (User user, Payment payment)

Вход: Изпращач на парите; Данни на заявката
Изход: Съобщение дали заявката е изпълнена или не

2.2.3.3.1.6 **Education Department**

2.2.3.3.1.6.1 **Review Program Course -** модула изпраща заявка до сървъра за преглеждане и одобряване на дадена заявка, като например

регистрация на нов курс от Преподавател и връща необходимата информация дали заявката е одобрена или не .

public void ReviewProgrameCourse (RequestData requestData, User user)

Вход: Данни на заявката; Изпращач на заявката
Изход: Съобщение дали заявката е одобрена или не

2.2.3.3.1.6.2 **Review/Send Report Data -** модула изпраща заявка до сървъра за преглеждане и одобряване на даден report. Той връща необходимата информация дали заявката е удобрена или не .

public void ReviewReport (ReportData requestData)

Вход: Данни на репорта
Изход: Съобщение дали заявката е удобрена или не

2.2.3.4. Report Manager

2.2.3.4.1 **View Report -** Модула изпраща заявка до сървъра и той връща необходимата информация.

public void ViewReport (String reportType)

Вход: Типа на репорта, който бива търсен
Изход: Съобщение дали операцията е успешна и/или списък от грешки

2.2.3.4.2 **Generate Report -** модула изпраща заявка до сървъра за създаване на нов report. Той връща необходимата информация дали заявката е удобрена или не .

public void CreateNewReport (ReportData requestData)

Вход: Данни на репорта
Изход: Съобщение дали заявката е удобрена или не

2.2.4 Други

2.2.4.1 Ограничения при употребата

За достъп до системата е необходима връзка с Интернет и съвременен браузър.

2.2.4.2 Грешки и изключения

При възникване на грешка по време на работата на приложението, тя да се визуализира по подходящ начин на потребителя.

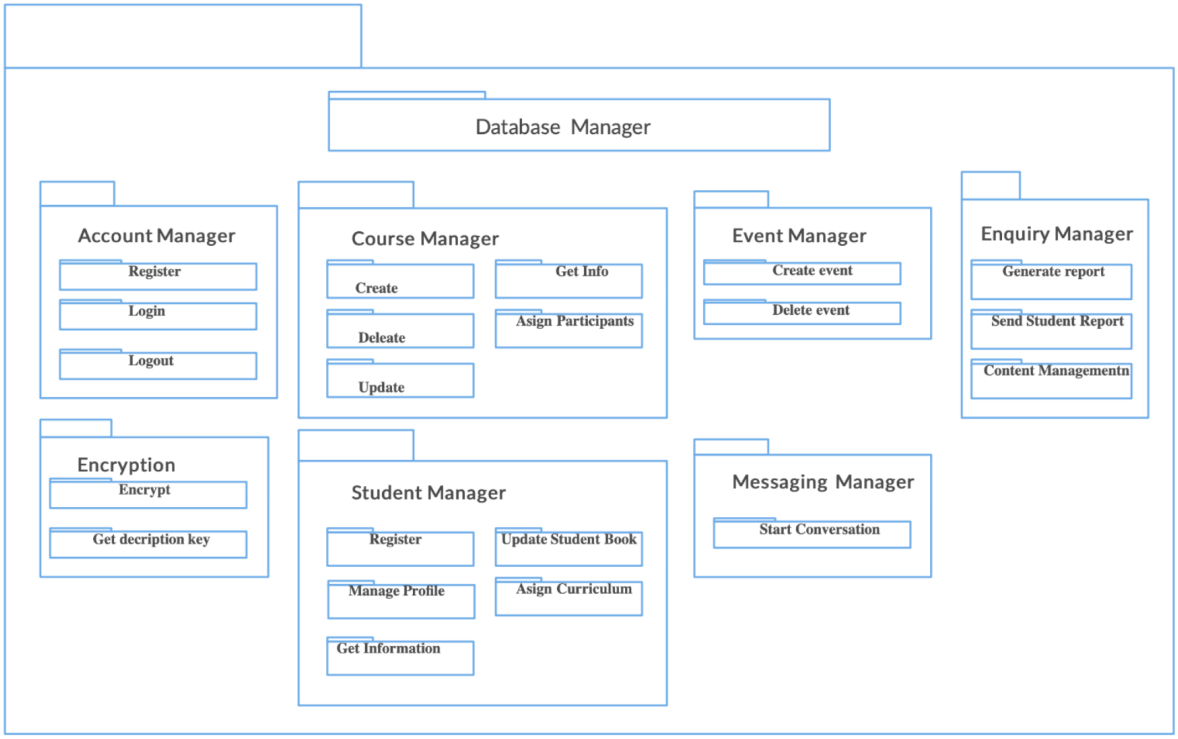
2.2.4.3 Зависимости от други елементи

Модулът зависи от UMSServer и Базата от данни. При грешка със свързването до тях, приложението спира работа, но визуализира съответното съобщение за грешка.

2.2.4.4 Описание на възможните вариации

При необходимост към Payment Manager могат да се добавят връзки и с други външни системи за плащане освен ePay..

2.3 UMSServer



2.3.1 Предназначение на модула

Тук се съхранява основната бизнес логика на системата. Този модул е и връзката с базата от данни за останалите модули, които го използват.

2.3.2 Основни отговорности в системата

UMSServer приема заявки от web интерфейсът(UMSSite), обработва ги и връща подходящ отговор. При необходимост се обръща към Базата от данни и взима или записва информация в нея.

2.3.3 Описание на интерфейсите на модулите

2.3.3.1. **Database Manager** – отговаря за връзката на останалите модули от MODServer с базата от данни.

2.3.3.2. **Account Manager**

2.3.3.4.1. **Register** - проверява дали дадения потребител не се е регистрирал вече (зает email или потребителско име, име на изпълнител...) и дали въведените данни са валидни

(достатъчно дълго име, парола...). Ако да - връща отрицателен отговор заедно с причината за грешката. Ако не - запазва данните в базата и връща положителен отговор.
public ActionResult Register (string username, string password, string email)

Вход: информация за потребителя (потребителско име, парола, емайл адрес)

Изход: *ActionResult,* съдържащ информация дали регистрацията е успешна или не.

private bool ValidateUsername (string username)

Вход: потребителско име на потребителя

Изход: Истина или лъжа в зависимост от това дали потребителското име е валидно или не.

private bool ValidatePassword (string password)

Вход: парола на потребителя

Изход: Истина или лъжа в зависимост от това дали паролата е валидна или грешна.

private bool ValidateEmail (string email)

Вход: емайл адрес на потребителя

Изход: Истина или лъжа в зависимост от това дали емайлът е валиден или не

private bool CheckIfUsernameAvailable (string username)

Вход: потребителско име на потребителя

Изход: Истина или лъжа в зависимост от това дали потребителското име е свободно или е вече заето

private boolean CheckIfEmailAvailable (string email)

Вход: емайл адрес на потребителя

Изход: Истина или лъжа в зависимост от това дали емайл адресът е свободен или е вече зает

2.3.3.2.2. **Login** - проверява дали данните са валидни (съществуващ потребител, съответстваща парола). Ако не - връща отрицателен отговори и причината за грешката. Ако да - логва потребителя (връща token, cookie...).

public ActionResult Login (string username, string password)

Вход: потребителско име и парола на потребителя

Изход: резултат със стойност успех или провал в зависимост дали входът е успешен или не

private boolean CheckIfUsernamePasswordMatch(string username, string password)

Вход: потребителско име и парола на потребителя

Изход: истина или лъжа в зависимост от дали съществува потребител с такова потребителско име и парола

private boolean ValidateUsername (string username)

Вход: потребителско име

Изход: истина или лъжа в зависимост от това дали съществува потребител с подаденото потребителско име

private boolean ValidatePassword (string password)

Вход: парола на потребителя

Изход: истина или лъжа в зависимост от това дали съществува потребител с подаденото парола

2.3.3.2.3. **Logout** - отнема правата за достъп на потребителя (премахва token-а, cookie-то…)

public ActionResult Logout ()

Вход: нямаме входни данни

Изход: резултат със стойност успех или провал в зависимост дали изходът е успешен или не

2.3.3.3 **Course Manager**

2.3.3.3.1 **Create** – създаване на нов курс, като преди да бъде създаден се валидират входните данни за курса, дали вече съществува курс със подаденото име. След като бъде успешно валидиран курс той бива записан в базата от данни.

public ActionResult Create(Course course)

Вход: обект от тип Course съдържащ в себе си данните за курса

Изход: резултат със стойност успех или провал в зависимост дали добавянето на курс е успешно или не

private boolean validateCourse(Course course)

Вход: обект от тип Course съдържащ в себе си данните за курса

Изход: истина или лъжа в зависимост от това дали данните за курса са валидни или не

private boolean isCourseNameAlreadyExists(String courseName)

Вход: име на курса

Изход: истина или лъжа в зависимост от това дали курс с токова име вече съществува

2.3.3.3.2. **Update** – променяне на данните на вече съществуващ курс. Променят се само данните, които са подадени, останалите данни на курса остават непроменени

public ActionResult Update(Course course)

Вход: обект от тип Course съдържащ в себе си данните за курса

Изход: резултат със стойност успех или провал в зависимост дали обновяването на данните на курс е успешно или не

private boolean validateCourse(Course course)

Вход: обект от тип Course съдържащ в себе си данните за курса

Изход: истина или лъжа в зависимост от това дали данните за курса са валидни или не

private Course getCorseByName(String courseName)

Вход: името на курса който искаме да обновим

Изход: обект от тип Course, ако бъде намерен курс с подаденото име в базата от данни, ако не се хвърля изключение, че такъв курс не съществува.

2.3.3.3.3. **Delete** – по подадено име на курс то бива изтрит от базата от данни

public ActionResult Delete(String courseName)

Вход: името на курсът който ще бъде изтрит

Изход: резултат от тип ActionResult със стойност успех или провал в зависимост от това дали курсът е изтрит или не

private boolean checkIfCourseExists(String courseName)

Вход: името на курсът

Изход: истина или лъжа в зависимост от това дали съществува курс с такова име

2.3.3.3.3. **Get Info** - получаване на информация от базата данни за даден курс/ за всички курсове

public List<Course> getInfo()

Вход: няма входни данни

Изход: списък от Course обекти съдържащ данните за всички записани в базата данни курсове

public Course getCourseInfo(String courseName)

Вход: име на курса

Изход: обект от тип курс съдържащ данните за курса

private boolean checkIfCourseExists(String courseName)

Вход: името на курсът

Изход: истина или лъжа в зависимост от това дали съществува курс с такова име

2.3.3.3.4 **Assign participants** – добавяне на участници към курса

public ActionResult AssignParticipants(String courseName, List<String> participanstUsernames)

Вход: името на курсът, списък от потребителските имена на студентите които ще бъдат записани за този курс

Изход: резултат от тип ActionResult със стойност успех или провал в зависимост от това дали записването на потребителите е успешно или не

private Course getCorseByName(String courseName)

Вход: името на курса който искаме да обновим

Изход: обект от тип Course, ако бъде намерен курс с подаденото име в базата от данни, ако не се хвърля изключение, че такъв курс не съществува.

private boolean validateUsername (string username)

Вход: потребителско име

Изход: истина или лъжа в зависимост от това дали съществува потребител с подаденото потребителско име

2.3.3.4 **Event Manger** – отговаря за създаването и изтриването на събития

2.3.3.4.1 **Add Event** – валидира подадените данни за събитието и при успех го запазва в базата от данни.

public ActionResult addEvent(Event even)

Вход: обект от тип Event съдържащ данните за събитието

Изход: резултат от тип ActionResult със стойност успех или провал в зависимост от това дали добавянето на събитие е успешно или не

private boolean isEventDataValid(Event event)

Вход: обект от тип Event съдържащ данните за събитието

Изход: истина или лъжа в зависимост дали подадените данни са валидни или не

2.3.3.4.2 **Delete Event** – изтриване на вече съществуващ обект

public ActionResult deleteEvent(String eventName)

Вход: името на събитието

Изход: резултат от тип ActionResult със стойност успех или провал в зависимост от това дали изтриването на събитие е успешно или не

private boolean checkIfEventExists(String eventName)

Вход: името на събитието

Изход: истина или лъжа в зависимост от това дали съществува събитие с подаденото име в базата от данни

2.3.3.5 **Enquiry Manager** – отговаря за генерирането, изпращането на рипорт към отделните външни системи, запазване и изтриване на вече съществуващи рипорти

2.3.3.5.1 **Generate Report -** генериране на рипорт по подаден тип на рипорта

public Report generateReport(String reportType)

Вход: типът на рипортът който ще бъде генериран

Изход: обект от тип Report, съдържащ данните за изискания

2.3.3.5.2 **Send student report –** генериране и изпращане на рипорт за студентите на към държавните публични регистри

public Report sendStudentReport()

Вход: нямаме входни данни

Изход: обект от тип Report който съдържа данните за студентите

private Report generateReport()

Вход: нямаме входни данни

Изход: обект от тип Report който съдържа данните за студентите

2.3.3.5.3 **Content management –** създаване, изтриване на рипорт

public ActionResult createReport(Report report)

Вход: обект от тип Report съдържащ данните за рипорта

Изход: резултат от тип ActionResult със стойност успех или провал в зависимост от това дали добавянето на рипорта е успешно или не

public ActionResult deleteReport(String reportName)

Вход: име на рипорта който трябва да бъде изтрит

Изход: резултат от тип ActionResult със стойност успех или провал в зависимост от това дали изтриването на рипорта е успешно или не

2.3.3.6. **Encryption** - отговаря за криптиране и декриптиране на данните

2.3.3.6.1. **Encrypt** - криптиране на рипорт с използването на публичен ключ

public Report Encrypt(Report report)

Вход: обект от тип Report, който ще бъде криптиран

Изход: при успех се връща криптирания рипорт, в случай на провал се хвърля изключение

2.3.3.6.2. **GetDecryptionKey** - Издаване на частен ключ за декриптиране за даден потребител - уникален за потребителя

public String GetDecryptionKey(User user)

Вход: обект от тип User

Изход: при успех генерираният публичен ключ, при неуспех се хвърля изключение

2.3.3.7. **Student manager** - отговаря за операциите свързани със студентите, регистриране, обновяване, добавяне на отценка за даден курс, добавяне на студент към даден курс.

2.3.3.7.1 **Register** – регистрация на нов студент

public ActionResult register(Student student)

Вход: обект от тип Student съдържащ данните за студента

Изход: резултат от тип ActionResult със стойност успех или провал в зависимост от това дали регистрацията на студент е успешна или не

private boolean validateStudent(Student student)

Вход: обект от тип Student съдържащ данните за студента

Изход: истина или лъжа в зависимост от това дали подадените данни да валидни или не

private isStudentAlreadyExist(String studentName)

Вход: името на студент

Изход: истина или лъжа в зависимост от това дали студент с такова име съществува или не

2.3.3.7.2 **Manage Profile** - обновяване и изтриване на студентски профили

public ActionResult updateStudentProfile (Student student)

Вход: обект от тип Student съдържащ данните за студента

Изход: резултат от тип ActionResult със стойност успех или провал в зависимост от това дали обновяването на студент е успешна или не

private boolean validateStudent(Student student)

Вход: обект от тип Student съдържащ данните за студента

Изход: истина или лъжа в зависимост от това дали подадените данни да валидни или не

public ActionResult deleteProfile (String studentName)

Вход: име на студента

Изход: резултат от тип ActionResult със стойност успех или провал в зависимост от това дали изтриването на студент е успешно или не

private isStudentAlreadyExist(String studentName)

Вход: името на студент

Изход: истина или лъжа в зависимост от това дали студент с такова име съществува или не

2.3.3.7.3 **Get Information –** получаване на информация за студентите

public List<Student> getAllStudentsInfo()

Вход: нямаме входно данни

Изход: списък съдържащ данните за всички студенти

2.3.3.7.4 **Update student book -** обновяване на студентска книжка на студент

public ActionResult updateStudentBook(StudentBook stBook)

Вход: обект от тип *StudentBook*

Изход: резултат от тип ActionResult със стойност успех или провал в зависимост от това дали обновяването на студентската книжка е успешно или не

2.3.3.7.5 **Assign curriculum –** добавяне на курс към студент

public ActionResult assignCarriculum(String cariculumName, String studentName)

Вход: име на курс и име на студент

Изход: резултат от тип ActionResult със стойност успех или провал в зависимост от това дали добавяне на курс към студент е успешно или не

2.3.3.7.6 Messaging manager – отговаря за създаването, изпращането и преглеждането на съобщения.

public ActionResult createMessage(Message message)

Вход: обект от ти Message

Изход: резултат от тип ActionResult със стойност успех или провал в зависимост от това дали създаването на съобщение е успешно или не

public ActionResult sendMessage(Message message)

Вход: обект от ти Message

Изход: резултат от тип ActionResult със стойност успех или провал в зависимост от това дали изпращането на съобщение е успешно или не

public List<Messages> getMessages(User user)

Вход: обект от тип User

Изход: списък от обекти от тип Message които са асоциирани с този потребителя

2.3.4 Други

2.3.4.1 Грешки и изключения

При възникване на грешка по време на работата на сървъра, тя да се връща като грешен резултат от заявка на съответното приложение, след което сървъра продължава нормално работата си.

2.3.4.2 Зависимости от други елементи

Модулът зависи от Базата от данни. При грешка със свързването с нея, приложението връща подходяща грешка на съответното приложение и известява системния администратор.

2.4 Report API

2.4.1 Описание и предназначение на модула.

Модула експоузва публичен интерфейс за достъп до генерираните официални справки и публични събития.

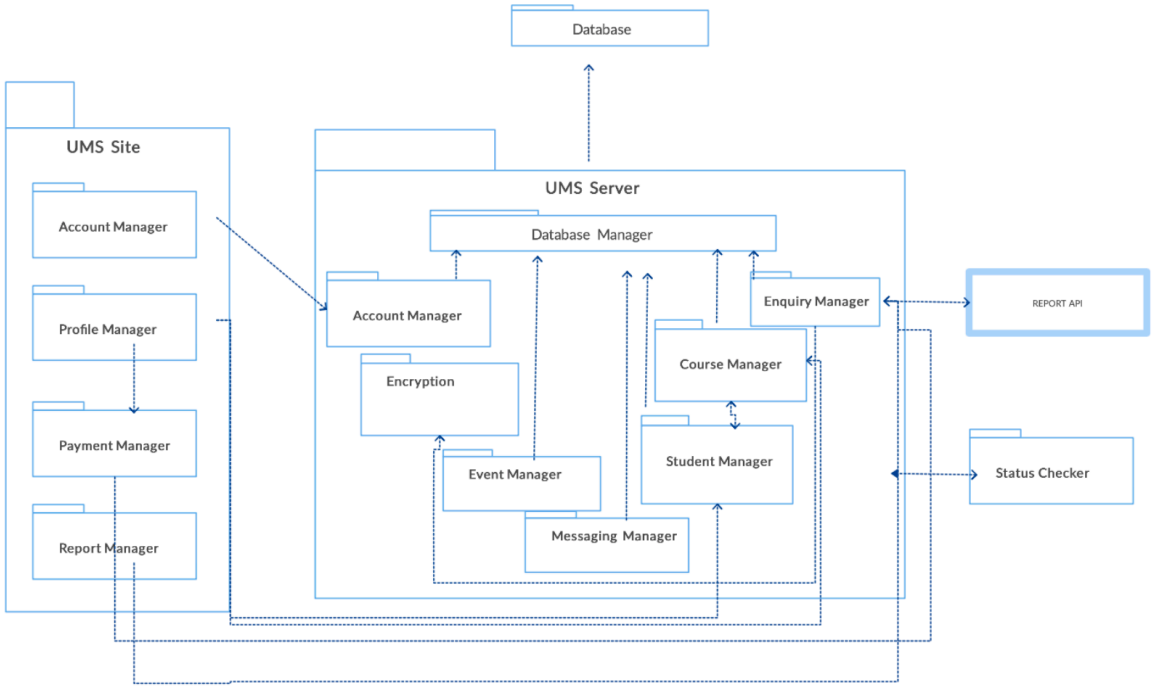
3 Описание на допълнителните структури

3.1 Употреба на модули

3.1.1 Мотивация за избор

В тази глава ще опишем връзката между модулите, които представихме в структурата Декомпозиция. Структурата точно и ясно проследява част от най-важните процеси в системата и това е мотивацията да изберем точно нея.

3.1.2 Първично представяне



3.1.3 Описание на елементите и връзките

Структурата се състои от елементи – модули, и връзки между тях – кой модул кой използва. Елементите в тази структура са модулите, които са представени в секция Декомпозиция на модулите.

3.1.3.1 Вътрешни връзки на UMSServer

- Enquire Manager използва модула Encryption при заявка за официална справка за да защити от опит за фалшификация. Справката се криптира като се издава и частен ключ за декриптиране.
- Модулите Account Manager, Event Manager, Messaging Manager, Student Manager, Course Manager, Enquire Manager използват Database Manager при нужда от обръщение към базата от данни.
- Модулите Student Manager и Course Manager си обменят данни при записване на студент за даден курс или обратното.

3.1.3.2 Вътрешни връзки на UMSSite

- Модулите Profile Manager и Payment Manager комуникират в случаите когато някой от потребителите иска да му бъде издадена справка, за която той трябва да плати.

3.1.3.2 Връзки на UMSSite и UMSServer

- Account Manager в UMSServer предоставя подходящ интерфейс за автентикация и авторизация на крайния потребител – форми за попълване на данни и подходящи съобщения за грешка или успех на желаната операция. Account Manager на UMSServer се грижи да поеме данните от UMSSite, да извърши желаната операция и да съобщи дали тя е успешна или не.
- Payment Manager отговаря за връзката с външна система за разплащане – ePay. След успешно свързване и получаване на потвърждение за плащането от външната система, модулът изпраща необходимата информация на Enquire Manager в UMSServer, за да се запише в базата от данни на системата информацията за заплащането.
- Profile Manager се свързва със Course Manager и Student Manager на UMSServer, за да извърши исканите от потребителя действия, свързани с курсовете и студентите. Спрямо типа на потребителя само част от действията са разрешени. Проверката за авторизация се извършва в UMSSite и в UMSServer с цел по-висока сигурност.
- Report Manager се връзва с Enquire Manager за да предоставят информация за репорти, която да бъде изложена като публичен интерфейс.

3.1.3.3 Връзка на UMSServer и Status Checker

Status Checker изпраща заявка през определен интервал от време заявка за проверка дали сървъра е изправен. Ако той е паднал се задейства Recovery на сървъра.

3.1.4 Описание на обкръжението

Payment Manager (UMSSite) си взаимодейства с външна система за плащания – ePay. За всяка отделна система се обособява отделен подмодул в Payment Manager с цел останалата част от системата да работи с абстракция на плащанията.

3.1.5 Възможни вариации

Към Payment Manager могат да се добавят и за други външни системи.

3.2 ВНЕДРЯВАНЕ

3.2.1 Мотивация за избор

Структурата на внедряване предоставя възможност да се представи връзката между хардуера и софтуера, както и връзките с външни системи по много добър начин.

3.2.2 Първично представяне

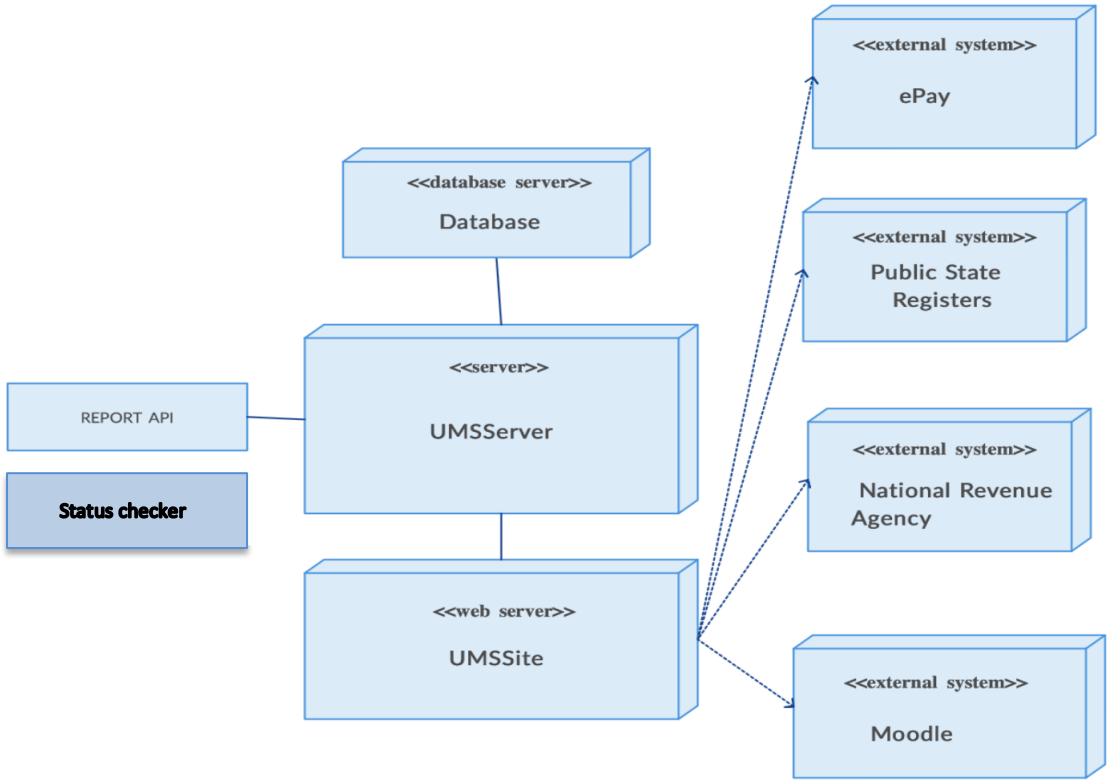
Основни елементи в структурата са сървъри, уеб апликейшън и външни системи.

Сървър е стартирана инстанция на софтуер, която може да приема заявки от клиент и да връща подходящи отговори.

Уеб апликацията бива използвана от крайния потребител на системата, като той я достъпва през браузър. Апликейшъна може да се обръща към сървърите чрез заявки за изпълняване на искането действие.

Външните системи представляват софтуер, който не е физически свързан с нашата система, но тя го използва за реализиране на дадена функционалност.

Системата излага публичен интерфейс чрез който може да бъде достъпвана информация за заявки, репорти и други.



3.2.3 Описание на елементите и връзките

Database модулът се намира върху Database Server. Обособен е на отделна машина с цел предпазване на данните от неуторизиран достъп, външни действия или евентуални откази на UMSServer.

UMSServer модулът е разположен на Application Server. Той държи главната бизнес логика и клиентските приложения, които се обръщат към него чрез заявки.

UMSSite е Web Server, който си комуникира с UMSServer за частта по бизнес логиката чрез заявки и с Web Browser за визуализирането на потребителския интерфейс чрез HTML страници.

Web Browser е външна система, чрез която потребителите достъпват функционалността, предоставена от UMSSite.

ePay е външна система, която осъществява плащанията в системата.

Public State Registers е външна система, към която периодично се изпраща информация за статуса на студентите. Изпращаната информация се контролира от потребителите от учебен и административен отдел.

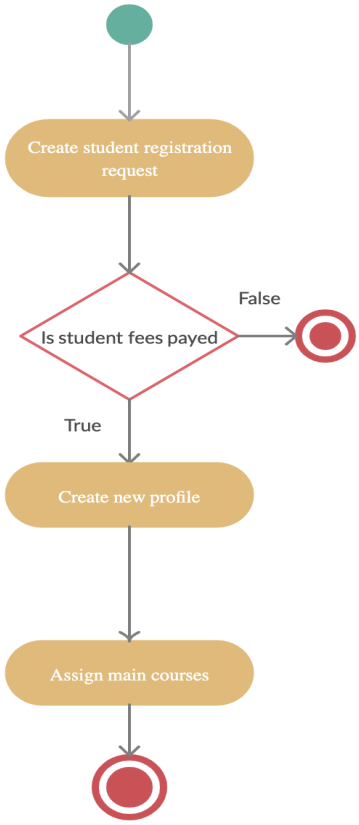
National Revenue Agency е външна система с която системата се отчита при правенето на парични преводи или други парични транзакции.

Moodle е външна система, която контролира учебното съдържание.

3.3 СТРУКТУРА НА ПРОЦЕСИТЕ

3.3.1 Процес на регистриране на нов студент

3.3.1.1 Първично представяне



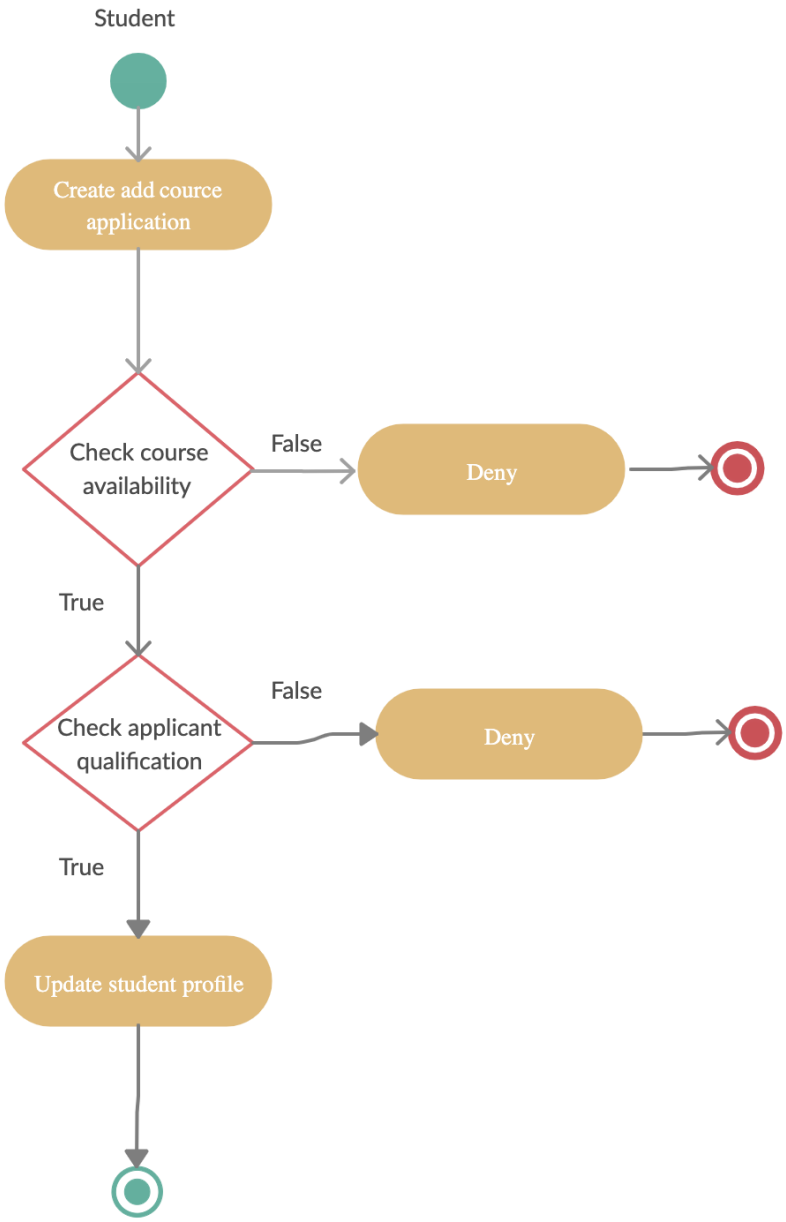
3.3.1.2 Описание на елементите и връзките

Регистрацията на нов студент се осъществява от потребител с администраторски права чрез UMSSite. Системата се свързва с Payment Manager, а той извършва проверка в базата данни като се търси дали има регистрирано плащане на този студент. Ако е налично плащане, то се създава профил на студента в системата чрез Profile Mangear-а и чрез Course Manager-а се залагат задължителните курсове, които ще трябва да посещава студента. Ако пък не,

системата връща съобщение, че не е открила направено плащане и операцията по регистрацията не може да продължи.

3.3.2 Процес на записване на студент за курс

3.3.2.1 Първично представяне

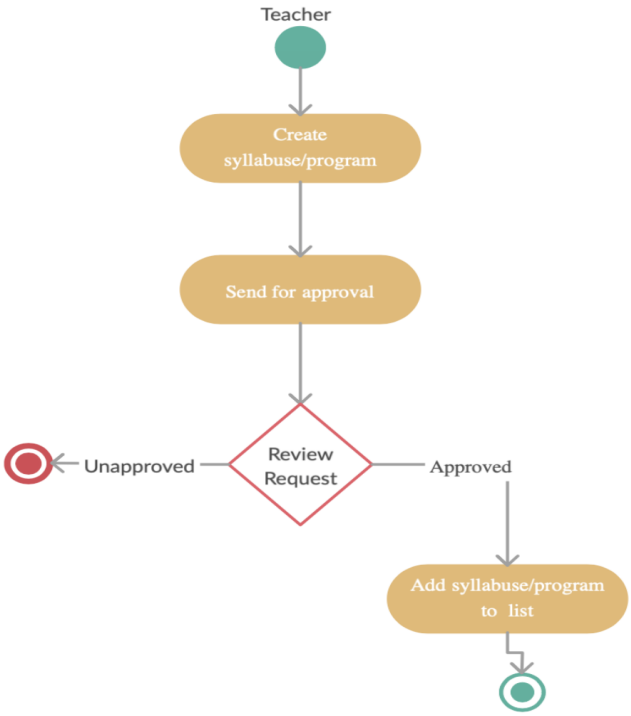


3.3.2.1 Описание на елементите и връзките

Студента се записва за даден курс в системата чрез UMSSite ProfileManager, комуникиращ с Course Manager. Студента изпраща заявка да бъде добавен в даден курс от каталога. Course Manager-а ревюира заявката и ако в курса няма свободни места връща съобщение за невъзможност за запис. Ако пък има свободни места се проверява дали този курс е в рамките на специалността на този студент и при условие, че профилът му отговаря на входните изисквания за компетентности за съответния курс. Ако студентът не притежава нужните компетенции се връща съобщение на потребителя, че курсът не може да бъде записан. Ако студентът отговаря на изискванията, той бива записван и Student Manager-а ъпдейтва профила на студента да отразява курса и прави нов запис в базата. Студента получава съобщение, че е бил записан за желания курс.

3.3.3 Процес по създаване на програма, курс от преподавател

3.3.3.1 Първично представяне

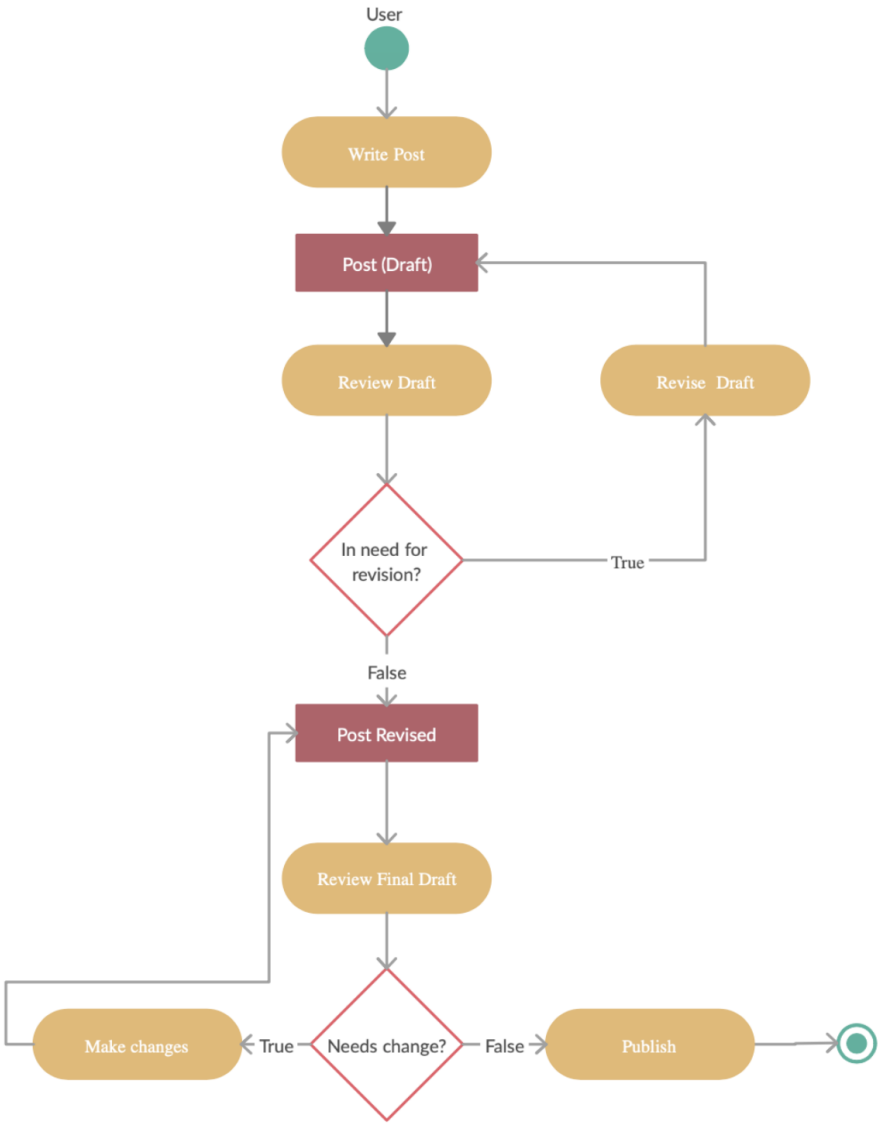


3.3.3.2 Описание на елементите и връзките

Създаването на програма или курс се извършва като се използва Profile Manager на UMSSite. Той прави заявка към UMSServer-а, който валидира данните за курса и проверява дали такъв курс вече съществува. Ако всичко е наред записва курсът в базата данни и връща отговор на UMSSite. UMSSite от своя страна показва съобщение за успех. Ако изходът от валидацията на заявката е негативен отново се връща резултат на UMSSite, който този път показва съобщение за грешка.

3.3.4 Процес на създаване на събитие

3.3.4.1 Първично представяне

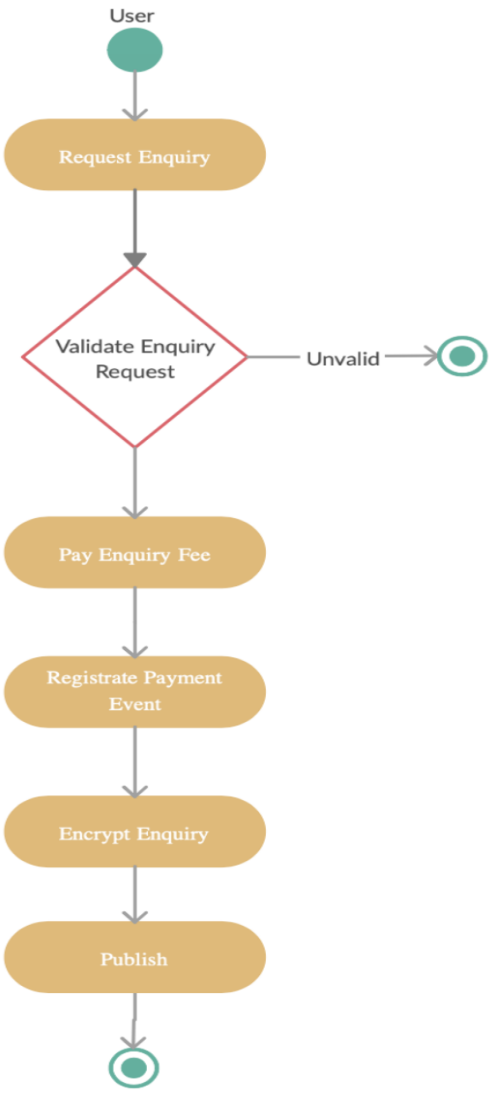


3.3.4.2 Описание на елементите и връзките

Всеки един от потребителите на системата има право да създава събития. Profile Manager-а се грижи за изпълнението на тази функционалност като изпраща данните то UMSServer където се обработват от съответния мениджър. Потребителя създава чернова на своето събитие, което бива ревюирано. В случай, че той иска да направи промени, бива върнат отново в черновата си. Ако пък не, се прави финален ревижън на събитието, което ако няма нужда от промяна се публикува на главната страница на системата или на друго специфично място и съобщение за успешно създадено събитие бива върнато на потребителя. Ако има нужда от промяна то стъпките се повтарят.

3.3.5 Процес на заявяване на заявка за справка

3.3.5.1 Първично представяне

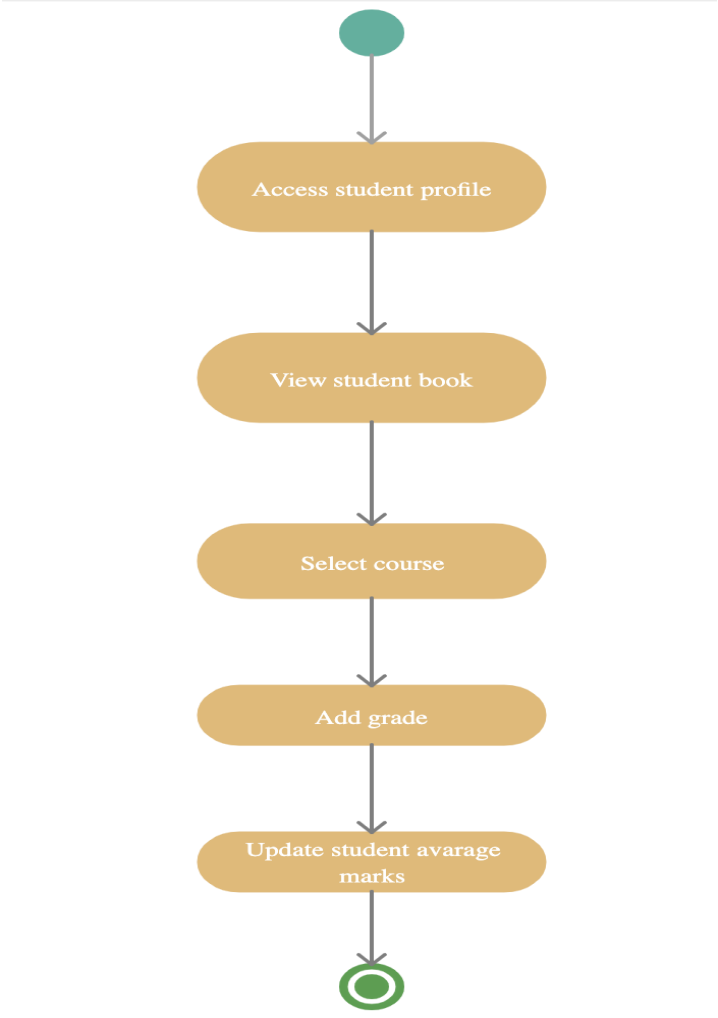


3.3.5.2 Определяне на елементите и връзките

Всеки потребител има правото да създава заявка за издаването на справки от различен тип. За тази цел той използва Profile Manager компонентът на UMSSite. След създаването на заявка за справка, тя се изпраща към UMSServer-а, където се валидира. Ако валидацията е неуспешна сървърът изпраща отговор със съобщение за грешка на UMSSite. UMSSite визуализира съобщението. Ако валидацията е успешна се изпраща заявка за плащане на таксата за издаването на справка към Payment manager. Регистрира се плащането. Криптират се данните за заявката. След това UMSServer изпраща заявката е отговор на UMSSite.

3.3.6 Процес на нанасяне на оценка за дадена дисциплина

3.3.6.1 Първично представяне



3.3.6.2 Описание на елементите и връзките

Нанасянето на оценка по дадена дисциплина се извършва от Преподавател. Чрез UMSSIte преподавателя достъпва профила на търсения от него студент и неговата студентска книжка. Той намира курса и въвежда оценката. StudentManager-а ще се погрижи новата информация да бъде записана в базата и средния успех на студента да бъде преизчислен като се има в предвид новите промени. В случай, че всички стъпки преминат успешно съобщение за успешно въведена оценка ще бъде изведено на потребителя. Ако пък не ще бъде изведено съобщение за грешка.

4. Архитектурна обосновка

В тази секция ще представим основните архитектурни драйвери и кои тактики сме използвали за реализирането им в системата

4.1. Функционалността на системата да бъде разделена в три приложения:

- UMSSite – уеб сайт, чрез който да се осъществява регистриране и управление на профилите на крайни потребители, курсове, справки
- UMSServer – сървър приложение, което осъществява същинското управление на потребители, курсове, издаване на справки, плащания
- ReportAPI - програмен интерфейс, който служи за създаване и изпращане на репорти към външните системи
- Status checker- извършва проверка за състоянието на сървъра. Ако бъде установена неговата неизправност, се задейства процес на Recovery
- Database - отговаря за съхранението на данните

Реализацията на това изискване е показана в Структурата на внедряване.

4.2. Системата трябва да защитава справките от опити за фалшифициране. Това се реализира посредством кодиране на справките.

Кодирането се осъществява от Encryption Manager (UMSServer). При издаване на справка се издава декриптиращ ключ. Този ключ работи само за съответния потребител.

4.3. Системата трябва да е лесно изменяема относно графичния интерфейс на приложенията, използваните бази данни или използваните алгоритми за криптиране.

За графичният интерфейс отговаря UMSSite. Графичните изображения използвани в юзър интерфейсът не се запазват в базата а се получават от външни източници. На страната на

UMSSite се извършва кеширане на тези изображения с цел да се избегнат проблеми със забавяне.

Цялата бизнес логика е изнесена в UMSServer. Той си комуникира Database и UMSSite.

Report API е програмен интерфейс отговорен за издаването и изпращането на рипорти. Той си комуникира UMSite и Database.

4.4. UMSServer трябва да криптира изпратената информация с асиметричен ключ, като криптирането ще се осъществява с публичен ключ, а декриптирането ще се осъществява с частен – известен само на потребителя за който се издава справката.. За криптирането отговаря Encryption Manager (UMSServer).

4.5 UMSSite и Report API само при наличие на връзка с интернет.

4.6 Системата трябва да има high availability, един от начините с който се постига това е използването на PostgreSQL като база от данни, която се слави със своето high availability и fault tolerance

4.7 Системата има изискване за поддръжка на защита на всички лични и финансови данни от неоторизиран достъп. Това се осъществява от Account Manager и Encryption.

4.8 Поради факта, че правим проверка за статуса на сървъра можем да гарантираме Allways ON на нашата система.

4.9. PostgreSQL поддържа **Continuous Archiving and Point-in-Time Recovery (PITR).** Поради това нашата система няма нужда от ресурси за допълнителна база данни, която да служи като back-up. По всяко време PostgreSQL поддържа лог на запис (WAL) в pg_xlog / поддиректория на директорията с данни на клъстера. Лога записва всяка промяна, направена във файловете с данни на базата данни. Този лог съществува предимно за целите на безопасността при сривове: ако системата се срине, базата данни може да бъде възстановена до съгласуваност чрез „преиграване“ на записите в дневника, направени след последната контролна точка.

4.10 При много голямо натоварване в пиковете часове, за да може системата да издържи е имплементиран **Throttling.** Системата може да издържи натоварване от 100 рикуеста в секунда. Всеки рекуест над тези 100 бива автоматично отрязан без да се заема от ресурсите на системата.