**Технически университет – София**

**Факултет по приложна математика и информатика**

**Архитектурно описание**

**на тема**

„Система за организация на библиотека“

**Изготвили**: Проектна група 9

Васил Николов Янакиев, 77 гр, 471219008

Даяна Димитрова Димитрова, 77 гр, 471219010

Джан Айдънов Язаджиев, 77 гр, 471219016

**Дата:** 30.10.2021 г. **Възложил:** Александър Петков

**Съдържание**

[**I.** **Въведение** 3](#_Toc86507550)

[**II.** **Предназначение** 3](#_Toc86507551)

[ **Обхват** 3](#_Toc86507552)

[ **Актьори** 3](#_Toc86507553)

[**III.** **Използвания понятие и абревиатури** 4](#_Toc86507554)

[**IV.** **Източници** 4](#_Toc86507555)

[**V.** **Архитектурен обзор** 4](#_Toc86507556)

[**1.** **Use-case изглед** 4](#_Toc86507557)

[**2.** **Логически изглед** 5](#_Toc86507558)

[**3.** **Процесен изглед** 5](#_Toc86507559)

[**4.** **Изглед на данните** 7](#_Toc86507560)

[**5.** **Изглед на внедряването** 8](#_Toc86507561)

[**VI.** **Нефункционални изисквания** 9](#_Toc86507562)

[ **Разширяемост** 9](#_Toc86507563)

[ **Използваемост** 9](#_Toc86507564)

[ **Преносимост** 9](#_Toc86507565)

[ **Сигурност** 9](#_Toc86507566)

[ **Надеждност** 9](#_Toc86507567)

[ **Производителност** 9](#_Toc86507568)

# **Въведение**

Документът цели да запознае заинтересованите лица за структурата и основните принципи на проектиране, които ще включва софтуерът. Архитектурното описание не навлиза в детайли относно начина на осъществяване на функционалностите, а по-скоро дава структура и изглед на уеб приложението за организация на библиотека. Представени са софтуерните елементи, отношенията между тях и свойствата им. За онагледяване на тази информация са предоставени UML диаграми, съобразени с изгледа, който представят. Участниците в проекта са изпълнителите: Васил Янакиев, Даяна Димитрова, Джан Язаджиев, Иван Пеовски и Лилия Костова.

# **Предназначение**

## **Обхват**

Документът минава през няколко етапа от развитието на уеб приложението за организация на библиотека, които са следните:

* изискванията (нуждите) на крайните потребители на системата като се разглеждат в детайли в следващото ниво на проектиране на системата
* функционални изисквания, структура на системата и основни компоненти в нея
* поведението на системата по време на изпълнение, системните процеси и комуникацията по между им
* разполагането на софтуерните компоненти върху хардуера и физическите връзки (отново в хардуера)
* логическия модел на данните в системата
* нефункционални изисквания, на които системата „държи“ и отговаря

## **Актьори**

Актьорите, които са включени в диаграмите по-долу, са Библиотекар – човекът с най-пряка връзка със системата и Читател – човекът, който играе косвена роля в приложението. Заинтересованите лица от документа са няколко, като всяко едно от тях има по-голяма тежест от останалите в конкретен за него изглед. Разпределението на изгледите и участниците в тяхното осъществяване са, както следва:

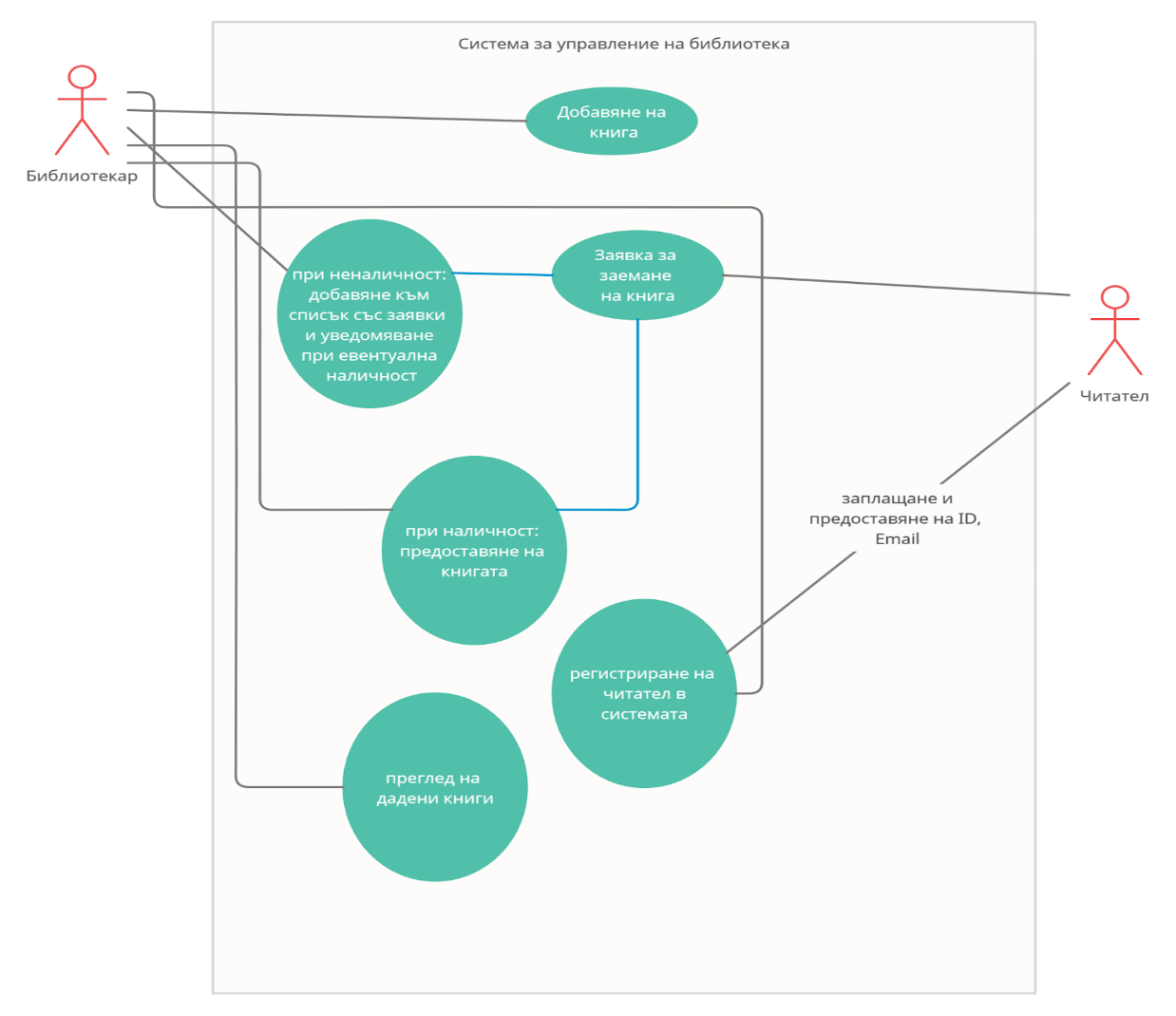
* *Use-case изглед* – за всички заинтересовани лица
* *Логически изглед* – за проектантите на системата, разработчиците
* *Процесен* *изглед* – за системните интегратори
* *Изглед на данните* – за разработчика на БД
* *Изглед на внедряването* – за системните инженери

# **Използвания понятие и абревиатури**

* UML - Unified Modeling Language
* Primary key – първичен ключ
* Foreign key – Външен ключ

# **Източници**

1. Bachmann F., Bass L., Klein M. and Shelton C., „Designing software architectures to achieve quality attribute requirements “, q IEE, 2005
2. Bosch, Jan. "Software architecture: The next step." Software architecture.Springer Berlin Heidelberg, 2004.194-199.
3. **Архитектурен обзор**
4. **Use-case изглед**

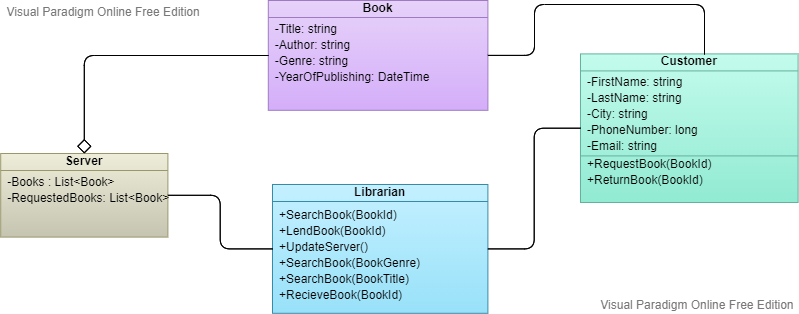
****

Use-case изгледът е показан чрез use-case диаграма. Диаграмата показва взаимодействията между читател и библиотекар. Също така са показани действията, които могат да бъдат извършени от библиотекаря - преглеждане на всички дадени книги, както и добавяне на нови книги.

## **Логически изглед**

Логическият изглед е представен чрез Class диаграма. Диаграмата е съставена от 4 класа, които са: клиент, книга, сървър и библиотекар. Показано е какви атрибути или операции изпълняват всяка една от класовете. В основата стои библиотекарят, който си взаимодейства със клиента, както и с сървъра. Той ще има възможност да потърси дадена книга по критерии и да я даде назаем на клиент. Както и да си взаимодейства със сървъра и да го променя атрибутите в него. От друга страна клиентът, който физически ще посети библиотеката ще може да направи заявка за книга и ако тя е налична в библиотеката, ще я получи.

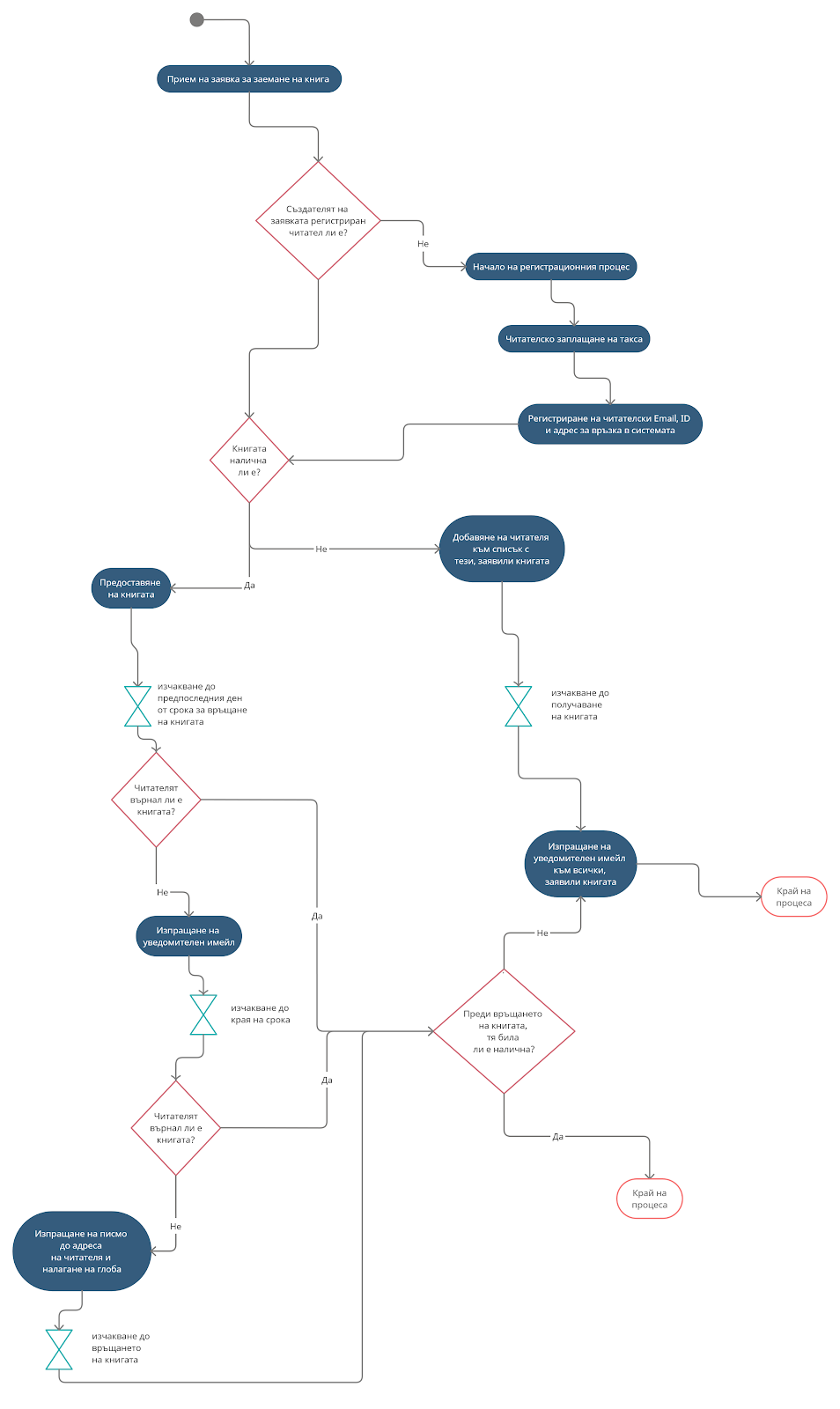
Книгата е свързана с класа на сървъра, в който имаме 2 листа с object book. Те ще носят информацията за всички книги. Отделният клас на книгата е съставен от 4 атрибута: име, автор, жанр и година на издаване.

****

## **Процесен изглед**

Процесният изглед е представен чрез Activity диаграма. Диаграмата показва процеса, иницииран от приема на заявка за заемане на книга. Първата проверка, която се извършва, е тази свързана със статуса на този създал заявката - дали е регистриран читател, или не. Ако не е регистриран читател, се налага да се регистрира.

Следващата проверка, която се извършва, е тази свързана с наличността на книгата. Ако книгата не е налична, всеки създал заявка за заемането й получава известие, когато стане налична. Ако книгата е налична, тя се предоставя на читателя. Един ден преди крайния срок се извършва проверка - ако читателят не е върнал книгата, му се изпраща известие. Ако читателят просрочи връщането на книгата, му се изпраща писмо в пощата и му се налага глоба.



## **Изглед на данните**

Изгледът на данните е представен чрез Entity relationship diagram. Диаграмата е съставена от шест таблици и два изгледа. Структурата на базата е от голямо значение за едно приложение, тъй като тя е в ядрото на неговото ефективно представяне. Таблиците Book, Readers и Genre са номенклатури. В *Book* са всички необходими данни за една книга, в *Readers* са данните на читателите, които някога са заемали книги от библиотеката, а в *Genre* са видовете жанрове, с които книгите в библиотеката се асоциират.

В таблицата *Out\_Of\_Stock* ще стоят записи за читатели, които са поискали да вземат съответна книга, но тя не е била в наличност, също така ще се пази и самата книга заедно с датата и времето, в което е направена заявка за изчерпана книга. От друга страна в *Stock* ще са всички книги от таблицата Book заедно с тяхната настояща наличност (бройка). В таблицата *Borrowed\_Book* ще са книгите, които са дадени, читатели, които са ги заели, както и датата на даване и съответната дата за връщане.

Базата ще разполага и с още една таблица под името *Options*. В нея ще са включени общо приети глобални променливи като максимален брой заети книги от читател, допустим период за задържане на книга у читателя и други. Тъй като тези стойности и променливи ще бъдат ясни в процеса на разработка, то сега тази таблица е оставена празна. Но нейната идея е много ключова, тъй като тя ще подобри четимостта на кода и при евентуална промяна на тези стойности, няма да има нужда да се търсят всички места, на които те са използвани. Достатъчно би било те да се променят в базата.

Ще бъдат създадени и два изгледа. *Book\_Inspection\_View* и *Reader\_Borrow\_View* ще служат за по-лесно боравене с данните, които ще се визуализират в потребителския интерфейс. Te ще са извадки от таблиците само с информация, която е полезна за ползвателите на приложението, а именно служителите на библиотеката. Важно уточнение към диаграмата: колоните, които са в bold представляват primary key, а тези, които са в italic са foreign key.

## **Изглед на внедряването**

Изгледът на внедряването е представен чрез Deployment diagram. Като главно действащо лице в конкретната диаграма е служителят в библиотеката, т.е. ползвателят на приложението. Той се свързва с приложението посредством монитор и клавиатура. За целта той има нужда от компютър, който да разполага с браузър, през който да може да достъпи интерфейсът на приложението. За да работи с функционалността на приложението трябва да има и добра Интернет връзка. В уеб сървъра са компонентите на software design pattern-а, на чиято база приложението е създадено, а именно Model, View и Controller. За да се зареждат и данните, а не само структурата на софтуера, има нужда и от Базата данни, която също е изобразена на диаграмата.

# **Нефункционални изисквания**

* **Разширяемост** – приложението подлежи на разширяемост, тъй като неговата логика е добре разпределена. Frontend-а и backend-а са разделени. Използва се комуникация чрез REST. Нещо повече, то лежи на архитектурния шаблон MVC, което значи, че е разбито в три основни направления – динамична структура на данните, изгледа в потребителската част и главното командване (обработването на заявките към системата). В този ред на мисли добавянето на нови функционалности и модули не би повлияло на работата на останалите.
* **Използваемост** – приложението залага много на това изискване и предоставя достъпен интерфейс, тъй като то ще бъде използвано от хора с ниска до средна компютърна грамотност. Изгледите на всяка страница ще са с подходящи цветове, които сами да говорят за функционалността си. Там, където има нужда, ще бъдат добавени упътвания. Местата на, които е разположена обемна информация за следене, ще бъдат снабдени с филтри по определени критерии.
* **Преносимост** - Преносимостта на високо ниво е използваемостта на един и същ софтуер в различни среди. Нашата система ще работи само на PC.
* **Сигурност** - атрибут, свързан със степента на защита на данните от неправомерен достъп до тях, осигуряване на идентификация на потребители. Сигурността не е от най-важните нефункционални изисквания за нашето приложение, поради факта, че ние нямаме потребители. Имаме един служител, който ще има достъп до базата данни, в която са записани личните данните (име, фамилия, град, телефонен номер, email) на отделните читатели. Но сигурността е гарантирана и чрез минимизирането на данните, т.е. по никакъв начин няма да се събират данни за читателите, които не са нужни за обработка.
* **Надеждност** - Способността на системата да продължи да функционира с течение на времето. В нашата архитектура има малко на брой актьори, което води до малък шанс за проблеми.
* **Производителност** - тъй като данните, които трябва да се обработят във всеки един момент са сравнително малки на големина - изцяло текстови заявки - не би имало забавяне на системата дори при максимално натоварване.