[1. АНАЛІЗ ЗАДАЧІ АВТОМАТИЗАЦІЇ ГРУПОВИХ ПЛАТЕЖІВ ТА ОБҐРУНТУВАННЯ ТЕМИ ДИПЛОМНОГО ПРОЕКТУ. 1](#_Toc485690223)

[1.1 Основні задачі автоматизації групових платежів 1](#_Toc485690224)

[1.2 Існуючі рішення автоматизації групових платіжних операції. 3](#_Toc485690225)

[1.2 Обгрунтування теми 3](#_Toc485690226)

[2.ТЕХНІЧНІ ЗАСОБИ РОЗРОБЛЕННЯ веб-системи 5](#_Toc485690227)

[2.1 Вибір технологій та середовища розробки 5](#_Toc485690228)

[2.2 Супроводні бібліотеки. 9](#_Toc485690229)

[2.3 Засоби розробки інтерфейсу. 11](#_Toc485690230)

[2.4 Засоби тестування продукту. 13](#_Toc485690231)

[2.5 Вибір ORM 14](#_Toc485690232)

[2.6 Вибір бази даних. 16](#_Toc485690233)

[2.7 Динамічне відображення даних 19](#_Toc485690234)

[2.8 Вибір засобів розробки серверної частини 20](#_Toc485690235)

# 1. АНАЛІЗ ЗАДАЧІ АВТОМАТИЗАЦІЇ ГРУПОВИХ ПЛАТЕЖІВ ТА ОБҐРУНТУВАННЯ ТЕМИ ДИПЛОМНОГО ПРОЕКТУ.

## 1.1 Основні задачі автоматизації групових платежів

Існують два погляди на цю проблему: бізнес-процес підприємств-постачальників послуг та бізнес-процес споживачів цих послуг.

Бізнес-процеси підприємств громадського харчування добре піддаються формалізації і, відповідно, комп'ютерній автоматизації. Всі схеми використовують бізнес-операції з єдиного набору: отримання замовлення, приготування страв, розрахунок з клієнтом. В цей набір вписується багато підприємств індустрії розваг: більярдні зали, боулінг-клуби і подібні заклади з тимчасовою тарифікацією розважальних послуг і сполученням з ресторанними послугами.

Еволюційний шлях комп'ютеризації цих закладів починався від окремих універсальних програм бухгалтерського обліку, складських операцій і управління контрольно-касовим обладнанням. Наступний етап розвитку полягав у автоматизації усіх бізнес-процесів, пов'язаних з кліентом, за допомогою єдиного програмного забезпечення.

Розглянемо це питання на прикладі ресторанного бізнесу.

Сучасне обладнання дозволяє більш просте спілкування офіціанта з клієнтами і кухнею. Тепер замовлення можна приймати через спеціальні термінали, після яких вони йтимуть прямо на кухню, де для кухарів друкуватимуться на сервісному принтері або будуть показуватися на спеціальному моніторі замовлень. При цьому, власник завжди буде в курсі останніх подій і, в разі потреби, зможе внести деякі правки. Проте, цей підхід передбачає використання стороннього обладнання. Реалізація розроблюваної системи зможе автоматизувати всі потрібні функції за допомогою єдиного веб-додатку.

З іншого боку, дане питання розглядається зі сторони споживача, який користується системою. Споживач або група споживачів може замовляти послуги у зареєстрованих постачальників або сплачувати між собою послуги за спільним рахунком.

Крім зручності в постачанні та обслуговуванні, система надає докладну інформацію про замовлення, суми і багато іншого.

Користувач зможе бачити всі його операції на його електронній поштовій скриньці, підключати платіжну картку або переглядати підключених постачальників послуг.

## 1.2 Існуючі рішення автоматизації групових платіжних операції.

На сьогоднішній день є багато систем, які намагаються вирішити питання автоматизації в сфері надання послуг.

Кожен постачальник послуг зацікавлений в тому, щоб автоматизувати всі можливі аспекти, що виключить людський фактор і зменшить витрати. Доставка послуг навивоз і різнобічні системи дуже розповсюджені, але кожен постачальник створює свою систему, і споживачеві доводиться звикати і користуватися тим, що максимально орієнтовано під потреби конкретного постачальника.

У більшості закладів громадського харчування є свої, вузькоспеціалізовані системи, які використовуються тільки в їх мережі.

Сервіси доставки їжі використовують свої додатки, які розробляються під свої потреби, ця практика дуже розповсюджена і використовується повсякденно.

Головна проблема в цьому, що споживачу потрібно мати спеціалізований додаток для кожного заклада та реєструватися в кожному з них. Споживачу набагато зручніше мати єдиний додаток, в якому будуть представленні всі заклади.

Сьогодні поєднання швидкості і якості стало реальним завдяки можливості автоматизації, і цим користуються в міру можливості. Але після огляду ринку систем-аналогів, переважна більшість, як згадано вище, створює системи тільки під свої потреби. Виявлені системи, яка задовольняють потрібну абстракцію.

1.2 Обгрунтування теми

Автоматизація платежів і замовлень дозволяє спростити і прискорити повсякденному процеси населення. Як приклад, ми можемо уявити, що група людей вирішила відвідати заклади громадського харчування. Для їх обслуговування потребуется оффіціан, меню та інше. При автоматизації можна уникнути цього, якщо заклад і споживачі підключені до додатка, то все це можна зробити за допомогою нього. Від замовлення до розрахунку, все що потрібно - регестрации закладу в додатку і розміщення своїх послуг. Як одна людина так і група може зробити потрібний їм замовлення і розплатитися наприкінці за допомогою унікального номера. Так само можна зробити автоматизовану доставку інших послуг.

Основними перевагами, які отримає заклад від впровадження автоматизації операцій, є швидке обслуговування і якісний сервіс; контролювання всіх фінансових операцій; недопущення помилок при оформленні замовлення, оскільки тут відсутній людський фактор; точність інформації.

Основними переваги користувача, які він отримає від автоматизації платежів, є зручність замовлення та розрахунків, отримання інформації про постачальника і перелік його послуг, відгуки щодо якості послуг.

Розроблювана веб-орієнтована система повинна враховує переваги та недоліки вже існуючих засобів автоматизації рішень. Система повинна задовольняти наступним вимогам:

* безкоштовність програмного продукту;
* чітке розподілення обов’язків кожного користувача в системі;
* наявність робочого функціоналу для всіх членів робочого процесу;
* простота користування та інтуїтивно зрозумілий інтерфейс;
* розроблений функціонал для реалізації розрахункових платіжних операцій;
* створення груп користувачів;
* створення особистих кабінетів користувачів;
* верифікація почтової адреси і розсилання результатів платіжних операцій;
* емуляція підключення до платіжної системі;
* підключення до геолокаційного сервісу який дозволить відображати інформацію на карті;

Автоматизація групових платіжних операцій спростить повсякденні платіжні операції і виключить людський фактор. Відсутність на ринку достатньої і не вузькоспеціалізованого програмного забезпечення дозволить легше вийти на спеціалізований ринок і впровадити. В Україні дана сфера дуже затребувана, оскільки розвиток бізнесу йде величезними темпами і це спростить популяризацію. Автоматизація стала тут стандартом, необхідною умовою конкурентоспроможності бізнесу.

Автоматизація споживчої сфери спростить життя споживачам і сприятимуть розвитоку даної сфери. Спрощення процесу отримання послуг групами людей дозволить уникнути ситуацій, коли постачальник послуг має фіксовані розрахунки і не дозволяє розділити рахунок.

2.ТЕХНІЧНІ ЗАСОБИ РОЗРОБЛЕННЯ веб-системи

## 2.1 Вибір технологій та середовища розробки

Дана веб-орієнтована система є безкоштовною, отже всі технології та середовище розробки повинні бути вільними для користування, тобто не потребувати фінансових витрат. Для цього було вирішено використовувати мову програмування Java, адже вона не потребує придбання ліцензій і надає потужний механізм для розробки веб-додатків.

Java - об'єктно-орієнтована мова програмування, випущена 1995 року компанією «Sun Microsystems» як основний компонент платформи Java. З 2009 року мовою займається компанія «Oracle», яка того року придбала “Sun Microsystems”. В офіційній реалізації Java-програми компілюються у байт-код, який при виконанні інтерпретується віртуальною машиною для конкретної платформи.

Мова значно запозичила синтаксис із C і C++. Зокрема, взято за основу об'єктну модель С++, проте її модифіковано. Усунуто можливість появи деяких конфліктних ситуацій, що могли виникнути через помилки програміста та полегшено сам процес розробки об'єктно-орієнтованих програм. Ряд дій, які в С/C++ повинні здійснювати програмісти, доручено віртуальній машині. Передусім, Java розроблялась як платформо-незалежна мова, тому вона має менше низькорівневих можливостей для роботи з апаратним забезпеченням, що в порівнянні, наприклад, з C++ зменшує швидкість роботи програм. За необхідності таких дій Java дозволяє викликати підпрограми, написані іншими мовами програмування.

Головним мотивом створення Java була потреба в мові програмування, яка б не залежала від платформи (тобто від архітектури) і яку можна було б використовувати для створення програмного забезпечення, що вбудовується в різноманітні побутові електронні прилади, такі як мобільні засоби зв'язку, пристрої дистанційного керування тощо.

Досить скоро майже всі найпопулярніші тогочасні веб-оглядачі отримали можливість запускати «безпечні» для системи Java-аплети всередині веб-сторінок. “Sun Microsystems” випустила Java 2 (спершу під назвою J2SE 1.2), де було реалізовано декілька конфігурацій для різних типів платформ. Наприклад, J2EE призначалася для створення додатків застосунків, а значно урізана J2ME для приладів з обмеженими ресурсами, таких як мобільні телефони.

Період становлення Java збігся у часі з розквітом міжнародної інформаційної служби World Wide Web. Ця обставина відіграла вирішальну роль у майбутньому Java, оскільки Web теж вимагала платформо-незалежних програм. Як наслідок, були зміщені акценти в розробці Sun з побутової електроніки на програмування для Інтернет.

Початкові цілі:

* Синтаксис мови повинен бути "простим, об'єктно-орієнтовним та звичним";
* Реалізація має бути "безвідмовною та безпечною";
* Повинна зберегтися "незалежність від архітектури та переносимість";
* Висока продуктивність виконання;
* Мова має бути "інтерпретованою, багатонитєвою, із динамічним зв’язуванням модулів";

Під “незалежністю від архітектури” мається на увазі можливість програми, на мові Java, працювати на будь-якій підтримуваній апаратній чи системній платформі без змін у початковому коді та перекомпіляції.

Стандартні бібліотеки забезпечують загальний спосіб доступу до таких платформозалежних особливостей, як обробка графіки, багатопотоковість та роботу з мережами. У деяких версіях задля збільшення продуктивності JVM байт-код можна компілювати у машинний код до або під час виконання програми.

Основна перевага використання байт-коду — це портативність. Тим не менш, додаткові витрати на інтерпретацію означають, що інтерпретовані програми будуть майже завжди працювати повільніше, ніж скомпільовані у машинний код, і саме тому Java одержала репутацію «повільної»[джерело?] мови.

Проте, цей розрив суттєво скоротився після введення декількох методів оптимізації у сучасних реалізаціях JVM.

Одним із таких методів є just-in-time компіляція (JIT, що перетворює байт-код Java у машинний під час першого запуску програми, а потім кешує його). Як результаті, така програма запускається і виконується швидше, ніж простий інтерпретований код, але ціною додаткових витрат на компіляцію під час виконання. Складніші віртуальні машини також використовують динамічну рекомпіляцію, яка полягає в тому, що віртуальна машина аналізує поведінку запущеної програми й вибірково рекомпілює та оптимізує певні її частини. З використанням динамічної рекомпіляції можна досягти більшого рівня оптимізації, ніж за статичної компіляції, оскільки динамічний компілятор може робити оптимізації на базі знань про оточення виконання та про завантажені класи. До того ж, він може виявляти так звані гарячі точки (англ. hot spots) — частини програми, найчастіше внутрішні цикли, які займають найбільше часу при виконанні. JIT-компіляція та динамічна рекомпіляція збільшує швидкість Java-програм, не втрачаючи при цьому портативності.

Існує ще одна технологія оптимізації байткоду, широко відома як статична компіляція, або компіляція ahead-of-time (AOT). Цей метод передбачає, як і традиційні компілятори, безпосередню компіляцію у машинний код. Це забезпечує хороші показники в порівнянні з інтерпретацією, але за рахунок втрати переносимості: скомпільовану таким способом програму можна запустити тільки на одній, цільовій платформі.

Швидкість офіційної віртуальної машини Java значно покращилася з моменту випуску ранніх версій, до того ж, деякі випробування показали, що продуктивність JIT-компіляторів у порівнянні зі звичайними компіляторами у машинний код майже однакова. Проте ефективність компіляторів не завжди свідчить про швидкість виконання скомпільованого коду, тільки ретельне тестування може виявити справжню ефективність у даній системі.

2.2 Супроводні бібліотеки.

Apache Commons – це проектом Apache зосереджений на всі аспекти повторного використовування компонентів Java.

Вихідний код репозиторія Apache Commons можуть перезаписати всі Apache Software Foundation учасники. Всі учасники повинні прочитати керівні принципи, що сприяють розробці. Утиліти проекту Apache Commons лежать в основі таких проектів як Apache Tomcat, Struts, Hibernate та ін.

Згідно зі статутом проекту, проект повинен «створювати і підтримувати пакети, написані на мові Java, призначені для використання в розробці серверних додатків, і спроектовані таким чином, щоб їх можна було використовувати незалежно в будь-якому більшому продукті або фреймворку.

В ієрархії проектів Apache Software Foundation Apache Commons є проектом "верхнього рівня". У свою чергу, Commons розбивається на безліч проектів «другого рівня». Кожен проект другого рівня має свій сайт, каталог в загальному репозиторії вихідного коду, каталог в загальній системі відслідковування помилок і має на меті розробку одного однойменного компонента (бібліотеки) Java. Активні проекти мають кілька постійних учасників, неактивні проекти можуть взагалі не мати постійних учасників.

JavaMail - це Java API, який використовується для відправки та отримання електронної пошти через SMTP, POP3 та IMAP. JavaMail вбудований в платформу Java EE, але також надає додатковий пакет для використання в Java SE.

Spring Security - надає комплексні служби безпеки для корпоративних додатків на базі Java EE. Особлива увага приділяється підтримці проектів, створених з використанням Spring Framework, який є провідним рішенням Java EE для розробки корпоративного програмного забезпечення.

Люди використовують Spring Security з багатьох причин, але більшість з них залучаються до проекту після виявлення функцій безпеки специфікації сервлетів Java EE або специфікації EJB не вистачає глибини, необхідної для типових сценаріїв корпоративних додатків. Незважаючи на присутність цих стандартів, важливо визнати, що вони не стерпні на рівні WAR або EAR. Тому, якщо ви переключаєте серверні середовища, зазвичай потрібна велика робота по налаштуванню ТВ-тюнерів безпеки вашого застосування в нову цільову середу. Використання Spring Security дозволяє подолати ці проблеми, а також приносить вам десятки інших корисних настроюються функцій безпеки.

Існує дві основні галузі безпеки додатків: «аутентифікація» і «авторизація» (або «контроль доступу»). Це дві основні області, на які націлена Spring Security. «Аутентифікація» - це процес встановлення принципала, який, за їх твердженням, є «головним», це означає, що користувач, пристрій або будь-яка інша система може виконувати дію в вашому додатку). «Авторизація» відноситься до процесу прийняття рішення може принципалу дозволено виконувати дію в вашому додатку. Щоб досягти моменту, коли потрібен дозвіл на авторизацію, ідентифікатор принципала вже встановлено процесом аутентифікації.

## 2.3 Засоби розробки інтерфейсу.

Для розробки веб-додатку на Java було обрано використовувати технологію JavaServer Faces. JavaServer Faces (JSF) — це MVC каркас програмування, технологія для [веб-додатків](https://uk.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%B7%D0%B0%D1%81%D1%82%D0%BE%D1%81%D1%83%D0%BD%D0%BE%D0%BA), що написані на [Java](https://uk.wikipedia.org/wiki/Java). Він слугує для того, щоб полегшувати розробку користувацьких інтерфейсів для Java

EE [додатків](https://uk.wikipedia.org/wiki/%D0%97%D0%B0%D1%81%D1%82%D0%BE%D1%81%D1%83%D0%BD%D0%BE%D0%BA). На відміну від більшості MVC фреймворків, які керуються запитами, підхід JSF ґрунтується на використанні [компонентів](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%BE%D0%BD%D0%B5%D0%BD%D1%82). Стан компонентів користувацького інтерфейсу зберігається, коли користувач запитує нову сторінку й потім відновлюється, якщо запит повторюється. Користь технології JSF обумовлена, в першу чергу, наявністю специфікації JSF. Специфікація дозволяє розробляти JSF фреймворки з різним призначенням та різною внутрішньою структурою. Вона лиш гарантує, що фреймворк буде підпорядкований певній структурі. Гарними сферами застосування еталонної реалізації JSF є невеликі веб-сайти. Модель-вид-контролер ([англ.](https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0) Model-View-Controller, MVC) — [архітектурний шаблон](https://uk.wikipedia.org/wiki/%D0%90%D1%80%D1%85%D1%96%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%BD%D1%96_%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D0%B8_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F), який використовується під час проектування та розробки [програмного забезпечення](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%B5_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F).

Цей шаблон поділяє систему на три частини: [модель даних](https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85), представлення даних та [керування](https://uk.wikipedia.org/wiki/%D0%9A%D0%B5%D1%80%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F). Застосовується для відокремлення даних (модель) від [інтерфейсу користувача](https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BA%D0%BE%D1%80%D0%B8%D1%81%D1%82%D1%83%D0%B2%D0%B0%D1%87%D0%B0) (представлення) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу користувача. Мета шаблону — гнучкий дизайн програмного забезпечення, який повинен полегшувати подальші зміни чи розширення програм, а також надавати можливість повторного використання окремих компонентів програми. Крім того використання цього шаблону у великих системах призводить до певної впорядкованості їх структури і робить їх зрозумілішими завдяки зменшенню складності. Архітектурний шаблон MVC поділяє програму на три частини. У тріаді до обов'язків компоненту Модель (Model) входить зберігання даних і забезпечення інтерфейсу до них. Представлення (View) відповідальний за представлення цих даних користувачеві. Контролер (Controller) керує компонентами, отримує сигнали у вигляді реакції на дії користувача, і повідомляє про зміни компоненту Модель. Така внутрішня структура в цілому поділяє систему на самостійні частини і розподіляє відповідальність між різними компонентами.

MVC поділяє цю частину системи на три самостійні частини: введення даних, компонент обробки даних і виведення інформації. Модель, як вже було відмічено, інкапсулює ядро даних і основний функціонал з їх обробки. Також компонент Модель не залежить від процесу введення або виведення даних. Компонент виводу представлення може мати декілька взаємопов'язаних областей, наприклад, різні таблиці і поля форм, в яких відображається інформація. У функції Контролера входить моніторинг за подіями, що виникають в результаті дій користувача (зміна положення курсора миші, натиснення кнопки або введення даних в текстове поле).

Зареєстровані події транслюються в різні запити, що спрямовуються компонентам Моделі або об'єктам, відповідальним за відображення даних. Відокремлення моделі від вигляду даних дозволяє незалежно використовувати різні компоненти для відображення інформації. Таким чином, якщо користувач через Контролер внесе зміни до Моделі даних, то інформація, подана одним або декількома візуальними компонентами, буде автоматично відкоригована відповідно до змін, що відбулися.

2.4 Засоби тестування продукту.

При розробці моделі було використано технологію розробки через тестування. Розробка через тестування Test-driven development (TDD) — [технологія розробки програмного забезпечення](https://uk.wikipedia.org/w/index.php?title=%D0%A2%D0%B5%D1%85%D0%BD%D0%BE%D0%BB%D0%BE%D0%B3%D1%96%D1%8F_%D1%80%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B8_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F&action=edit&redlink=1), яка використовує короткі ітерації розробки, що починаються з попереднього написання тестів, які визначають необхідні покращення або нові функції. Кожна ітерація має на меті розробити код, який пройде ці тести. Нарешті, програміст або група вдосконалюють код для погодження змін. Один із ключових моментів TDD полягає у тому, що підготовка тестів перед написанням самого коду пришвидшує процес внесення змін. Керована тестами розробка вимагає від розробника створення автоматизованих модульних тестів, які визначають вимоги до коду безпосередньо перед написанням самого коду. Тест містить перевірки умов, які можуть або виконуватися, або ні. Коли вони виконуються, кажуть, що тест пройдено. Проходження тесту підтверджує поведінка, передбачувана програмістом. На практиці модульні тести покривають критичні та нетривіальні ділянки коду. Це може бути код, схильний до частих змін, код, від роботи якого залежить працездатність великої кількості іншого коду, або код з великою кількістю залежностей. Середовище розробки повинно швидко реагувати на невеликі модифікації коду. Архітектура програми повинна базуватися на використанні багатьох [сильно пов’язаних](https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B2%27%D1%8F%D0%B7%D0%B0%D0%BD%D1%96%D1%81%D1%82%D1%8C_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)) компонентів, які [не](https://uk.wikipedia.org/wiki/%D0%97%D0%B2%27%D1%8F%D0%B7%D0%BD%D1%96%D1%81%D1%82%D1%8C_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)) залежать або майже не залежать один від одного , завдяки чому тестування коду спрощується. TDD не тільки пропонує перевірку коректності, а й впливає на дизайн програми. Спираючись на тести, розробники можуть швидше уявити, яка функціональність необхідна користувачеві. Таким чином, деталі інтерфейсу з'являються задовго до остаточної реалізації рішення. Зрозуміло, що до тестів застосовуються такі самі вимоги щодо якості коду, як і до основного коду.

2.5 Вибір ORM

ORM (англ. Object-relational mapping, Об'єктно-реляційна проекція) — технологія програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування, створюючи "віртуальну об'єктну базу даних".

При об'єктно-орієнтованому програмуванні об'єкти в програмі представляють об'єкти з реального світу. Як приклад можна розглянути адресну книгу, яка містить список людей разом з кількома телефонами і кількома адресами. В термінах об'єктно-орієнтованого програмування вони представлятимуться об'єктами класу “Людина”, які міститимуть наступний список полів: ім'я, список (або масив) телефонів і список адрес.

Суть проблеми полягає в перетворенні таких об'єктів у форму, в якій вони можуть бути збережені у файлах або базах даних, і які легко можуть бути витягнуті в подальшому, зі збереженням властивостей об'єктів і відносин між ними. Ці об'єкти називають “постійними” (англ. persistent). Історично існує кілька підходів до рішення цієї задачі.

Розроблено безліч пакетів, що знімають необхідність в перетворенні об'єктів для зберігання в реляційних базах даних.

Деякі пакети вирішують цю проблему, надаючи бібліотеки класів, здатних виконувати такі перетворення автоматично. Маючи список таблиць в базі даних і об'єктів в програмі, вони автоматично перетворять запити з одного вигляду в іншій. В результаті запиту об'єкта “Людина” (з прикладу з адресною книгою) необхідний SQL-запит буде сформований і виконаний, а результати “магічним” чином перетворені в об'єкти “Номер телефону” всередині програми.

З погляду програміста система повинна виглядати як постійне сховище об'єктів. Він може просто створювати об'єкти і працювати з ними як завжди, а вони автоматично зберігатимуться в реляційній базі даних.

На практиці все не так просто і очевидно. Всі системи ORM звичайно проявляють себе в тому або іншому вигляді, зменшуючи в деякому роді можливість ігнорування бази даних. Більш того, шар транзакцій може бути повільним і неефективним (особливо в термінах згенерованого SQL). Все це може привести до того, що програми працюватимуть повільніше і використовувати більше пам'яті, ніж програми, написані «вручну».

Але ORM позбавляє програміста від написання великої кількості коду, часто одноманітного і схильного до помилок, тим самим значно підвищуючи швидкість розробки. Крім того, більшість сучасних реалізацій ORM дозволяють програмістові при необхідності жорстко задати код SQL-запитів, який використовуватиметься при тих або інших діях (збереження в базу даних, завантаження, пошук тощо) з постійним об'єктом.

Java Persistence API — стандартизований інтерфейс для Java ORM фреймворків. Є частиною EJB 3 та J2EE 5, хоча може використовуватись незалежно від них. Виник через популярність вільного ORM фреймворку Hibernate, та бажання мати незалежний від конкретної реалізації стандарт.

Hibernate — засіб відображення між об'єктами та реляційними структурами (object-relational mapping, ORM) для платформи Java. Hibernate є вільним програмним забезпеченням. Hibernate надає легкий для використання каркас (фреймворк) для відображення між об'єктно-орієнтованою моделлю даних і традиційною реляційною базою даних.

Для взаємодії з базою даних був вибраний ORM [фреймвор](https://uk.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA)к Hibernate.

Метою Hibernate є звільнення розробника від значних типових завдань із програмування взаємодії з базою даних. Розробник може використовувати Hibernate як при розробці з нуля, так і для вже існуючої бази даних.

Hibernate піклується про зв'язок класів з таблицями бази даних (і типів даних мови програмування із типами даних SQL), і надає засоби автоматичної побудови SQL запитів й зчитування/запису даних, і може значно зменшити час розробки, який зазвичай витрачається на ручне написання типового SQL і JDBC коду. Hibernate генерує SQL виклики і звільняє розробника від ручної обробки результуючого набору даних, конвертації об'єктів і забезпечення сумісності із різними базами даних.

Hibernate забезпечує прозору підтримку збереження даних, тобто їхньої персистентності (англ. persistence) для «POJO»-об'єктів, тобто для звичайних Java-об'єктів; єдина сувора вимога до класу, що зберігається — конструктор за замовчуванням .

Hibernate забезпечує використання SQL-подібної мови Hibernate Query Language (HQL), яка дозволяє виконувати SQL-подібні запити, записані поряд з об'єктами даних Hibernate. Запити критеріїв надаються як об'єктно-орієнтована альтернатива до HQL.

## 2.6 Вибір бази даних.

База даних – сукупність даних, організованих відповідно до концепції, яка описує характеристику цих даних і взаємозв'язки між їх елементами; ця сукупність підтримує щонайменше одну з областей застосування. У загальному випадку база даних містить схеми, таблиці, подання, збережені процедури та інші об'єкти. Дані у базі організовують відповідно до моделі організації даних. Таким чином, сучасна база даних, крім саме даних, містить їх опис та може містити засоби для їх обробки.

Існують реляційні бази даних і так звані NoSQL (Not Only SQL) бази даних.

NoSQL - база даних, що забезпечує інший механізм зберігання та видобування даних, ніж звичний підхід таблиць-відношень в реляційних базах даних.

NoSQL бази даних все більше і більше використовуються в задачах із застосуванням big data та real-time веб-застосунках. NoSQL системи також називають "Not only SQL" для підкреслення того, що вони можуть підтримувати SQL-подібну структуру та мову запитів.

Види нереляційних баз даних:

MongoDB - це документо-орієнтована система керування базами даних (СКБД) з відкритим кодом, яка не потребує опису схеми таблиць. MongoDB займає нішу між швидкими і масштабованими системами, що оперують даними у форматі ключ/значення, і реляційними СКБД, функціональними і зручними у формуванні запитів.

MongoDB підтримує зберігання документів в JSON-подібному форматі, має досить гнучку мову для формування запитів, може створювати індекси для різних збережених атрибутів, ефективно забезпечує зберігання великих бінарних об'єктів, підтримує журналювання операцій зі зміни і додавання даних в БД, може працювати відповідно до парадигми Map/Reduce, підтримує реплікацію і побудову відмовостійких конфігурацій.

Apache Cassandra — розподілена система керування базами даних, що відноситься до класу noSQL-систем і розрахована на створення високомасштабованих і надійних сховищ величезних масивів даних, представлених у вигляді хеша.

Спочатку проект був розроблений в надрах Facebook і в 2009 році переданий під оруду фонду Apache Software Foundation. Промислові рішення на базі Cassandra розгорнуті для забезпечення сервісів таких компаній, як Cisco, IBM, Cloudkick, Reddit, Digg, Rackspace і Twitter.

Реляційна база даних — база даних, заснована на реляційній моделі даних. Слово "реляційний" походить від англ. relation. Для роботи з реляційними БД застосовують реляційні СКБД. Інакше кажучи, реляційна база даних — це база даних, яка сприймається користувачем як набір нормалізованих відношень різного ступеня.

Види реляційних баз даних:

MySQL – була розроблений компанією «ТсХ» для підвищення швидкодії обробки великих баз даних. Ця система керування базами даних з відкритим кодом була створена як альтернатива комерційним системам. MySQL з самого початку була дуже схожою на mSQL, проте з часом вона все розширювалася і зараз MySQL — одна з найпоширеніших систем керування базами даних. Вона використовується, в першу чергу, для створення динамічних веб-сторінок, оскільки має чудову підтримку з боку різноманітних мов програмування.

PostgreSQL – об'єктно-реляційна система керування базами даних. Є альтернативою як комерційним СКБД (Oracle Database, Microsoft SQL Server, IBM DB2 та інші), так і СКБД з відкритим кодом (MySQL, Firebird, SQLite).

Порівняно з іншими проектами з відкритим кодом, такими як Apache, FreeBSD або MySQL, PostgreSQL не контролюється якоюсь однією компанією, її розробка можлива завдяки співпраці багатьох людей та компаній, які хочуть використовувати цю систему керування базами даних та впроваджувати у неї найновіші досягнення.

Сервер PostgreSQL написаний мовою C. Звичайно розповсюджується у вигляді набору текстових файлів із початковий кодом. Для інсталяції необхідно відкомпілювати файли на своєму комп'ютері і скопіювати в деякий каталог. Весь процес детально описаний в документації.

Oracle – об'єктно-реляційна система керування базами даних від Oracle Corporation.

В якості бази даних була вибрана MySQL, у зв’язку з можливістю безкоштовного користування та легкістю встановлення та налаштування.

MySQL – реляційна система керування базами даних, призначена для вбудовування в Java-додатку або обробки транзакцій в реальному часі.

## 2.7 Динамічне відображення даних

AJAX (Asynchronous Javascript And Xml) – технологія для взаємодії з сервером без перезавантаження сторінок. За рахунок цього зменшується час відгуку і веб-додаток за інтерактивністю більше нагадує десктоп. Наприклад, при натисканні кнопки голосувати - з браузера на сервер буде відправлено повідомлення, а сервер відповість браузеру, що голос прийнятий.

Технологія AJAX, як вказує перша буква A в її назві - асинхронна, тобто браузер, відіславши запит, може робити що завгодно, наприклад, показати повідомлення про очікування відповіді, прокручувати сторінку, і т.п.

Технологія AJAX використовує комбінацію:

* (X) HTML, CSS для подачі і стилізації інформації
* DOM-модель, операції над якою виробляються JavaScript на стороні клієнта, щоб забезпечити динамічне відображення і взаємодія з інформацією
* XMLHttpRequest для асинхронного обміну даними з веб-сервером. У деяких AJAX-фреймворк і в деяких ситуаціях, замість XMLHttpRequest використовується IFrame, SCRIPT-тег або інший аналогічний транспорт.
* JSON часто використовується для обміну даними, однак будь-який формат підійде, включаючи форматований HTML, текст, XML і навіть який-небудь EBML

Типовий AJAX-додаток складається як мінімум з двох частин. Перша виконується в браузері і написана, як правило, на JavaScript, а друга –знаходиться на сервері і написана, наприклад, на Ruby, Java або PHP.

У синхронної моделі браузер відправляє запит на сервер і чекає, поки той зробить всю необхідну роботу. Сервер виконує запити до бази даних, загортає відповідь у необхідний формат і виводить його. Браузер, отримавши відповідь, викликає функцію показу. Всі процеси виконуються послідовно, один за одним. Мережеві затримки включені від час очікування. Користувач не може займатися чимось іншим на цій самий сторінці, поки відбувається синхронний обмін даними.

Браузер тримає постійне з'єднання з сервером (або робить час від часу коригувальні запити), і потрібні зміни відсилаються цим каналом. Взагалі, проблема застарілого контексту безпосередньо відноситься до задачі цілісності даних. За кінцеву перевірку цілісності, як і при валідації форм, в будь-якому випадку несе відповідальність сервер.

## 2.8 Вибір засобів розробки серверної частини

Також у розробці був використаний Spring Framework, в першу чергу як Inversion of Control-контейнер.

Spring Framework забезпечує вирішення багатьох завдань, з якими стикаються Java-розробники і організації, які хочуть створити інформаційну систему, засновану на платформі Java. Через широку функціональності важко визначити найбільш значущі структурні елементи, з яких він складається. Spring Framework не повністю повязаний з платформою Java Enterprise, незважаючи на його масштабну інтеграцію з нею, що є важливою причиною його популярності.

Spring Framework, ймовірно, найбільш відомий як джерело розширень (features), потрібних для ефективної розробки складних бізнес-додатків поза великовагових програмних моделей, які історично були домінуючими в промисловості. Ще одне його достоїнство в тому, що він ввів раніше невикористовувані функціональні можливості в сьогоднішні панівні методи розробки, навіть поза платформи Java.

Цей фреймворк пропонує послідовну модель і робить її придатною до більшості типів додатків, які вже створені на основі платформи Java. Вважається, що Spring Framework реалізує модель розробки, засновану на кращих стандартах індустрії, і робить її доступною в багатьох областях Java.

Центральною частиною Spring Framework є контейнер Inversion of Control, який надає засоби конфігурації і управління об'єктами Java за допомогою відображення. Контейнер відповідає за управління життєвим циклом об'єкта: створення об'єктів, виклик методів ініціалізації і конфігурація об'єктів шляхом зв'язування їх між собою.

Об'єкти, які створюються контейнером, також називаються керованими об'єктами (beans). Звичайно конфігурація контейнера здійснюється шляхом завантаження XML-файлів, що містять визначення bean і надають інформацію, необхідну для створення bean.

Об'єкти можуть бути отримані одним з двох способів:

* Пошук залежності – шаблон проектування, в якому викликає об'єкт запитує у об'єкта-контейнера екземпляр об'єкта з певним ім'ям або певного типу.
* Впровадження залежності – шаблон проектування, в якому контейнер передає екземпляри об'єктів за їх іменем іншим об'єктам за допомогою конструктора, властивості або фабричного методу.

В данній системі було використане саме впровадження залежності - Dependency injection.

Впровадження залежності (DI) - процес надання зовнішньої залежності програмного компоненту. Є специфічною формою «інверсії управління» (англ. Inversion of control, IoC), коли вона застосовується для управління залежностями. У повній відповідності з принципом єдиної обов'язки об'єкт віддає піклуватися про побудову необхідних йому залежностей зовнішньому, спеціально призначеному для цього спільного механізму.

Робота фреймворку, що забезпечує впровадження залежності, описується наступним чином. Додаток, незалежно від оформлення, виконується всередині контейнера IoC, що надається фреймворком. Частина об'єктів в програмі, як і раніше, створюється звичайним способом мови програмування, частина створюється контейнером на основі наданої йому конфігурації.

Умовно, якщо об'єкту потрібно отримати доступ до певного сервісу, об'єкт бере на себе відповідальність за доступ до цього сервісу: він або отримує пряме посилання на місцезнаходження сервісу, або звертається до відомого “сервіс-локатора” і запитує посилання на реалізацію певного типу сервісу. Використовуючи ж впровадження залежності, об'єкт просто надає властивість, яка в змозі зберігати посилання на потрібний тип сервісу. I коли об'єкт створюється, посилання на реалізацію потрібного типу сервісу автоматично вставляється в цю властивість (поле), використовуючи засоби середовища.

Впровадження залежності більш гнучке, тому оскільки легше створювати альтернативні реалізації даного типу сервісу, а потім вказувати, яка саме реалізація повинна бути використана в, наприклад, файлі конфігурації, без змін в об'єктах, які цей сервіс використовують. Це особливо корисно в юніт-тестування, тому що вставити реалізацію “заглушки” сервісу в тестований об'єкт дуже просто.

З іншого боку, зайве використання впровадження залежностей може зробити програми більш складними і важкими у супроводі: оскільки для розуміння поведінки програми програмісту необхідно дивитися не тільки в вихідний код, а ще й в конфігурацію, а конфігурація, як правило, невидима для IDE, які підтримують аналіз посилань і рефакторинг, якщо явно не вказана підтримка фреймворків з впровадженнями залежностей.