# Постановка целей

1. Выбрать тему проекта
2. Определить стек технологий(язык, фреймворки, вспомогательные инструменты)
3. Разбить проект на задачи
4. Выполнить сами задачи

# Тема проекта

Варианты :

1. Приложение «Домашняя бухгалтерия»
2. Веб-приложение «Домашняя бухгалтерия»
3. Программа для обменника валют
4. Калькулятор бюджета
5. Приложение для учета товара в магазине

# Стек технологий

* 1. Язык программирования – Java.
  2. Хранение данных – база данных(MySQL, PostgreSQL ).
  3. Фреймворки(Spring Boot(DI(dependency injection)), Spock(testing), JDBC(DB connection), SWING(UI design)).

# Задачи

* Скачать IntelidgeIDEA Enterprise (Сережа найти установочный файл на раб ноуте и не забыть сделать настройки по инструкции)
* Скачать MySQL. Установить и настроить + создать таблицы и практика SQL для запросов
* Сделать репозиторий на GitHub для проекта + пригласить Пашу(скинуть свой логин)
* Подключить фреймворки к проекту. Подключить Git к проекту
* Подготовить теорию об спринге, свинге и продумать архітектуру приложения.
* Сделать DAO уровень, Service уровень, уровень отображений, базу данных, логин, подключение к БД.

# Сбор требований

Models:

User (login, Fisrt Nmae/last name(optional), password, last activity Date)

Client (id, Fisrt Nmae/last name, last activity Date, Orders, telephone number, bonuses, email, birthday )

Products(id, Name, price, quantity, delivery date)

Orders(id, date, total price, client\_id, products\_id

Services:

UserService(create, update, get, delete, getByLogin)

ClientService(create, update, get, delete, countBonus)

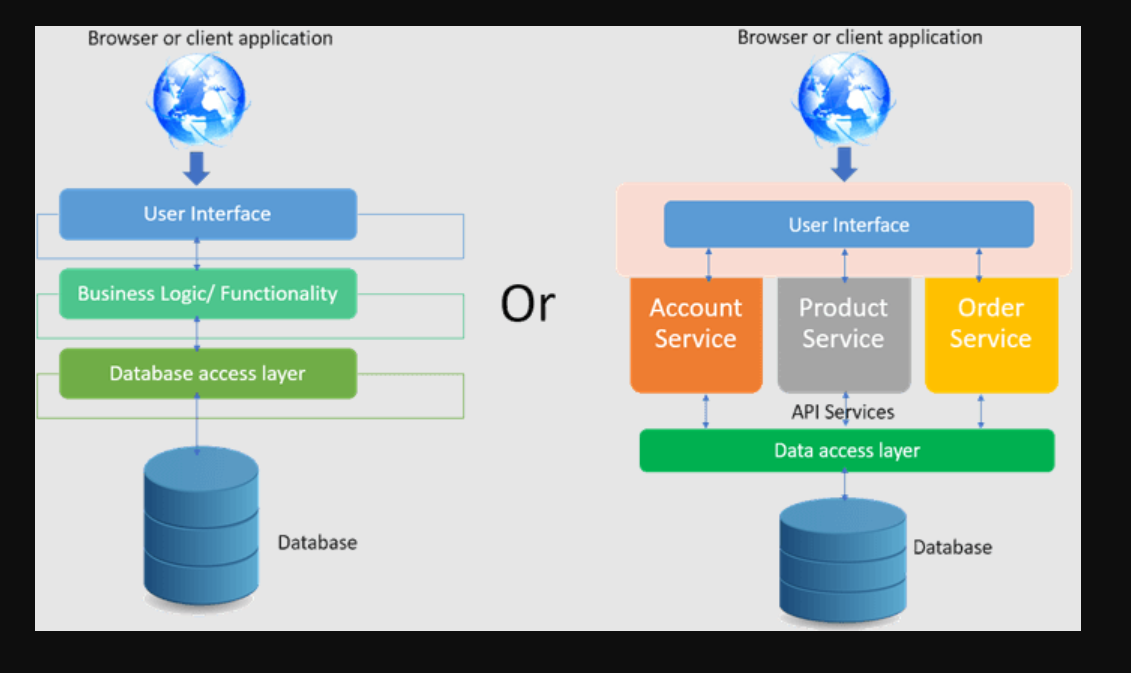
Products(create, update, get, delete,)

Orders(create, update, get, delete, )

DBConnectionUtil

Exception

\*\*\*HashUtil(for login)



**Архитектура приложений в разработке программного обеспечения**  
  
**1.** Введение в архитектуру приложений:  
- Архитектура приложения - это организация его компонентов и их взаимодействия для достижения целей проекта.  
  
2. Основные виды архитектур:  
  
a. **Монолитная архитектура:**  
- Одно приложение с одним исполняемым файлом.  
- Проста в разработке и развертывании.  
- Ограничена масштабируемостью.  
  
b. **Микросервисная архитектура:**  
- Разделение приложения на небольшие независимые сервисы.  
- Повышает масштабируемость и гибкость.  
- Требует сложной инфраструктуры.  
  
c. **Слоистая архитектура:**  
- Разделение приложения на логические уровни (представление, бизнес-логика, данные).  
- Облегчает сопровождение и тестирование.  
  
d. **Client-Server архитектура:**  
- Разделение приложения на клиентскую и серверную части.  
- Позволяет масштабировать сервер и клиенты независимо.  
  
**3. Как выбрать подходящую архитектуру:**  
  
- Выбор зависит от конкретных требований проекта, бюджета и ожиданий по масштабируемости и гибкости.  
  
**4. Основные принципы архитектуры приложений:**  
  
a. **Разделение ответственности (Separation of Concerns):** Каждый компонент должен заниматься только своей задачей.  
  
b. **Модульность (Modularity):** Разбиение приложения на независимые модули упрощает сопровождение и масштабирование.  
  
c. **Инкапсуляция (Encapsulation):** Скрытие деталей реализации для предотвращения ненамеренных изменений.  
  
d. **Масштабируемость (Scalability):** Возможность расширения приложения для обработки более высоких нагрузок.  
  
e. **Гибкость (Flexibility):** Способность приложения адаптироваться к изменяющимся требованиям.  
  
f. **Тестирование (Testability):** Облегчение тестирования каждого компонента независимо.  
  
g. **Производительность (Performance):** Обеспечение быстродействия и оптимизация приложения.  
  
h. **Безопасность (Security):** Защита данных и предотвращение уязвимостей.  
  
**5. Примеры популярных технологий и фреймворков:**  
- Монолит: Spring Boot, Django.  
- Микросервисы: Kubernetes, Docker.  
- Слоистая: MVC (Model-View-Controller).  
- Client-Server: REST, GraphQL.  
  
Архитектура приложений - это ключевой аспект в разработке программного обеспечения, и правильный выбор архитектуры может существенно повлиять на успех проекта.

**Фреймворки при разработке на Java: Краткий конспект**  
  
**1. Что такое фреймворк:**  
- Фреймворк - это набор предопределенных классов, библиотек и шаблонов, предназначенных для упрощения разработки приложений.  
  
**2. Виды фреймворков в разработке на Java:**  
  
a. **Веб-фреймворки:**  
- Применяются для разработки веб-приложений.  
- Примеры: Spring, JavaServer Faces (JSF), Apache Struts.  
  
b. **Мобильные фреймворки:**  
- Используются для создания мобильных приложений под Android и iOS.  
- Примеры: Android SDK, Flutter.  
  
c. **IoC/DI фреймворки:**  
- Облегчают управление зависимостями и инверсию управления.  
- Пример: Spring Framework (Spring IoC/DI).  
  
d. **ORM фреймворки:**  
- Позволяют взаимодействовать с базами данных через объектно-реляционное отображение.  
- Примеры: Hibernate, Java Persistence API (JPA), JDBC.  
  
e. **Тестирование фреймворки:**  
- Предоставляют инструменты для автоматизированного тестирования Java-приложений.  
- Пример: JUnit.  
  
**3. Когда следует использовать фреймворки:**  
  
a. **Ускорение разработки:** Фреймворки предоставляют готовые компоненты и структуры, что сокращает время разработки.  
  
b. **Упрощение сложных задач:** Они облегчают решение типичных задач, таких как взаимодействие с базой данных или обработка HTTP-запросов.  
  
c. **Соответствие стандартам:** Фреймворки могут соблюдать стандарты и лучшие практики, что способствует созданию более надежных и безопасных приложений.  
  
d. **Масштабируемость:** Фреймворки помогают создавать масштабируемые приложения, что важно для роста и развития бизнеса.  
  
e. **Тестирование и сопровождение:** Фреймворки способствуют созданию более тестируемого и сопровождаемого кода.  
  
Фреймворки - это мощный инструмент для разработчиков Java, который помогает сделать код более структурированным, эффективным и удобным для разработки разнообразных видов приложений.

**База данных:**  
  
**1. Что такое база данных (БД):**  
- База данных - это структурированное хранилище данных, организованное таким образом, чтобы обеспечивать эффективное добавление, извлечение и обновление информации.  
  
**2. Виды баз данных:**  
  
a. **Реляционные базы данных (RDBMS):**  
- Основаны на модели данных, использующей таблицы и отношения между ними.  
- Примеры: MySQL, PostgreSQL, Oracle.  
  
b. **Нереляционные базы данных (NoSQL):**  
- Используются для хранения и обработки неструктурированных данных.  
- Примеры: MongoDB, Cassandra, Redis.  
  
**3. Основные различия:**  
  
- Реляционные БД используют таблицы с жестко определенной схемой данных, в то время как NoSQL БД могут хранить данные в более гибкой форме (документы, ключ-значение и т. д.).  
- Реляционные БД обычно обеспечивают жесткую согласованность данных, тогда как NoSQL БД чаще поддерживают гибкость и доступность данных.  
  
**4. Сферы применения:**  
  
a. **Реляционные БД:**  
- Широко используются в корпоративных приложениях для хранения структурированных данных (например, финансовых данных).  
- Часто используются в системах управления содержимым (CMS) и системах управления ресурсами предприятия (ERP).  
  
b. **NoSQL БД:**  
- Применяются в больших объемах данных и для хранения неструктурированных данных (например, в социальных сетях и аналитических приложениях).  
- Хорошо подходят для задач, где необходима горизонтальная масштабируемость и высокая доступность (например, в кеш-системах).  
  
**5. Нормализация таблиц:**  
  
- Нормализация - это процесс организации данных в реляционных БД с целью устранения избыточности и обеспечения целостности данных. Нормализация выполняется с помощью набора нормальных форм (от первой до пятой).  
  
**6. Как достигают нормализации:**  
  
a. **Первая нормальная форма (1NF):** Убирают повторяющиеся группы данных, выделяя их в отдельные таблицы.  
  
b. **Вторая нормальная форма (2NF):** Разделяют данные на таблицы так, чтобы каждый столбец зависел только от первичного ключа.  
  
c. **Третья нормальная форма (3NF):** Убирают транзитивные зависимости между столбцами.  
  
d. **Другие нормальные формы (4NF, 5NF):** Применяются при необходимости дополнительной оптимизации и устранения аномалий данных.  
  
**7. Когда использовать нормализацию:**  
  
- Нормализация используется в реляционных БД для обеспечения эффективности и целостности данных. Она полезна, когда необходимо управлять сложными структурами данных и обеспечить минимизацию избыточности и аномалий данных.  
  
Нормализация является важной частью проектирования баз данных и помогает создавать структуры данных, которые легко обновлять и поддерживать в будущем.