Самостоятельная групповая работа

Medical Startup Prototype — PatientCare

Формат задания

- **Тип работы:** групповая
- **Размер группы:** ~8+ человек
- **Время выполнения:**
  - **Занятие 1:** 90 минут
  - **Занятие 2:** 90 минут
- **Результат:** работающий backend-прототип + демонстрация

Ролевая игра

На время выполнения задания преподаватель выступает **в роли заказчика** — представителя частной медицинской клиники.

Заказчик:

- хорошо понимает **бизнес-задачи**,
- **не разбирается в технологиях**,
- **не принимает технических решений**.

Все технические решения принимает **ваша команда**.

Описание проекта (от заказчика)

Клиника хочет получить **внутренний IT-сервис** для учёта пациентов.

Сервис должен позволять:

- регистрировать пациентов,
- хранить базовую медицинскую информацию,
- просматривать список пациентов,
- получать данные пациента по идентификатору,
- защищаться от некорректных данных.

❌ Проект **не предполагает** UI, мобильного приложения, email-уведомлений, авторизации.

Цель задания

1. Создать **backend-прототип** медицинского сервиса.
2. Отработать:

- совместную работу,
- распределение ролей,
- принятие решений,
- базовую архитектуру backend-приложения.

3. Понять, как выглядит **реальный командный процесс**, а не только написание кода.

Основная сущность

Entity: `Patient`

Минимальный набор полей:

Поле	Описание
id	уникальный идентификатор
firstName	имя пациента
lastName	фамилия пациента
dateOfBirth	дата рождения
gender	пол
insuranceNumber	номер страховки
bloodType	группа крови
createdAt	дата регистрации

Enum’ы (пример)

```
Gender: MALE, FEMALE, OTHER
BloodType: A_POS, A_NEG, B_POS, B_NEG, AB_POS, AB_NEG, O_POS, O_NEG
```

Валидация данных (обязательно)

Заказчик ожидает, что:

- имя и фамилия **не пустые**,
- дата рождения **в прошлом**,
- номер страховки **обязателен**,
- gender и bloodType **обязательны**,
- при невалидных данных:
    - пациент **не сохраняется**,
    - возвращается **HTTP 400**,
    - событие логируется.

Использовать:

- @NotBlank
- @NotNull
- @Past
- @Valid

** REST API (минимум)**

Обязательные endpoint’ы

1. GET /api/patients

   Получить список пациентов
2. GET /api/patients/{id}

   Получить пациента по id

3. POST /api/patients

   Создать пациента (через @RequestBody)
4. DELETE /api/patients/{id}

   Удалить пациента

   (допускается soft-delete)

Дополнительно

- Query Params:
    - ?gender=MALE
    - ?bloodType=O_POS
    - ?ageFrom=18&ageTo=65

База данных и Liquibase

Требования

- Все изменения БД **только через Liquibase**
- Формат changelog — **XML**

- Минимум:

    1. changeset создания таблицы
    2. changeset добавления ограничений / колонок

- Seed-данные:

    - **только для test-профиля**
    - используются в интеграционных тестах

Docker (обязательно)

Проект должен:

- иметь Dockerfile,
- собираться через docker build,
- запускаться через docker,
- быть доступен по HTTP после запуска.

База данных:

- H2

Логирование (обязательно)

Логировать:

- успешное создание пациента (INFO),
- попытку создать невалидного пациента (WARN).

Тестирование

Обязательно

- **1 unit-тест** Entity (валидация)
- **1 integration-тест** на каждый Controller:
    - @SpringBootTest
    - @ActiveProfiles("test")
    - реальная БД (H2)
    - проверка:
        - успешного POST → 201
        - неуспешного POST → 400

Работа в команде (важная часть задания)

Группа **сама распределяет роли**.

Пример:

- 1–2 — Entity + Liquibase
- 1–2 — Controller + Validation
- 1 — Docker
- 1 — Integration tests
- остальные — тестирование, Postman, помощь, фиксация проблем

⚠️ Важно: мы будем обсуждать **процесс работы**, а не только код.

Что нужно показать в конце

1. Кратко:
    - что за сервис,
    - какую проблему решает.
2. Код:
    - Entity
    - Controller
    - Liquibase changelog

3. API:
    - успешный POST
    - неуспешный POST
4. Docker:
    - сборка образа
    - запуск контейнера
5. Тест:
    - запуск одного integration-теста

Обсуждение после демонстрации

Мы будем обсуждать:

- как вы распределяли задачи,
- кто принимал решения,
- где было сложно,
- что пошло не так,
- что бы вы сделали иначе,
- как Docker и Liquibase помогли или помешали.

✅ Критерии успешного выполнения

Задание считается выполненным, если:

- приложение запускается,
- REST API работает,
- данные валидируются,
- Liquibase реально применяется,
- Docker реально используется,
- есть хотя бы один осмысленный интеграционный тест,
- команда может объяснить что и почему она сделала.
