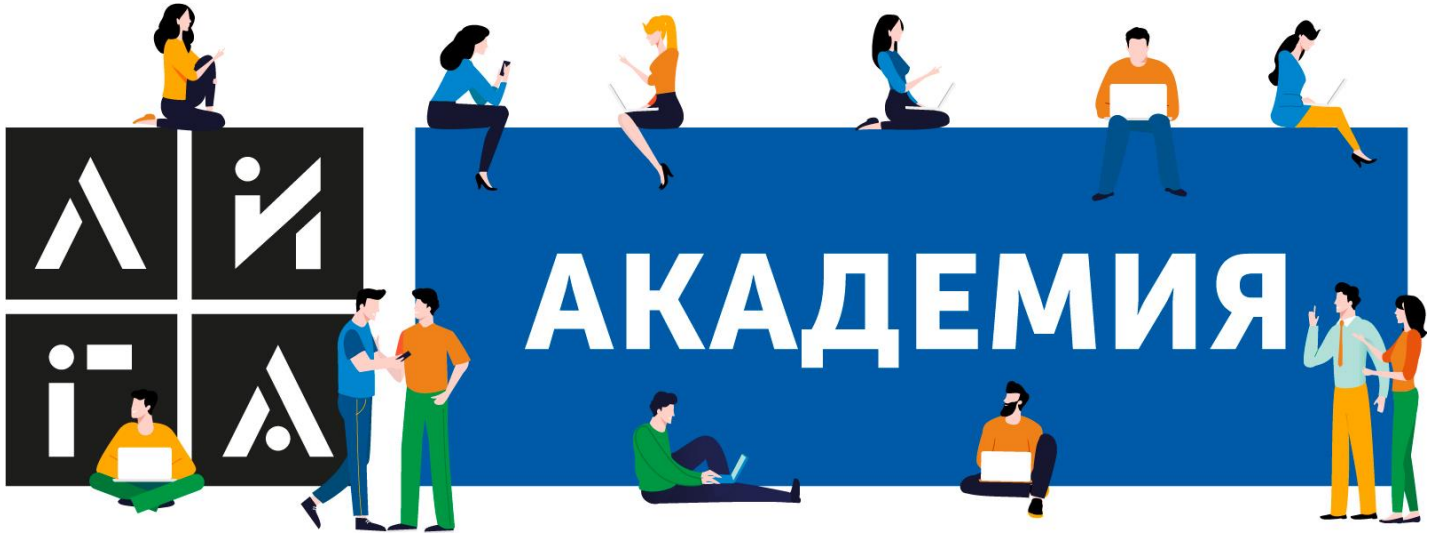

АКАДЕМИЯ

Spring Framework

IoC, DI, MVC, REST, JPA

01.

Spring Framework и Spring Boot

02.

Управление зависимостями: DI для IoC

03.

Прорубаем выход наружу: веб-контейнер MVC + REST

04.

Боремся с забывчивостью: ORM и Spring Data JPA

Spring Framework и Spring Boot

Начало начал, альфа и омега.

АКАДЕМИЯ

Spring Framework и Spring Boot

Универсальный фреймворк

- **Spring Framework** – собирательное название для совместимых между собой фреймворков, реализующих различные типовые задачи.
- Устанавливает общие правила построение и взаимодействия между собой различных модулей.
- Ядром фреймворка можно считать контейнер IoC, управляющий бинами и внедрением зависимостей.

Ускоритель разработки

- **Spring Boot** – набор библиотек и конфигураций, позволяющих резко ускорить разработку приложений, использующих Spring Framework
- Состоит из набора стартеров(starter), которые ускоряют подключение и начальную конфигурацию дополнительных модулей

Spring Framework и Spring Boot

Основные модули

- **Spring MVC** – фреймворк, с помощью которого строятся веб-приложения, основанные на HTTP и сервлетах. Обеспечивает обработку HTTP-запросов и сопутствующие функции.
- **Spring Data** – обеспечивает работу с базами данных, как через JDBC, так и через ORM.
- **Spring Security** – фреймворк, реализующий основные протоколы аутентификации и авторизации.
- **Spring Cloud** – множество функций, полезных при развертывании и функционировании веб-приложений в облаке, особенно если веб-приложение выполнено в микросервисной архитектуре**.**
- **Spring AOP** – реализует аспектно-ориентированное программирование, .
- **Spring Remoting** – фреймворк для работы с удаленными вызовами для распределенных приложений.

Управление зависимостями

DI для IoC

АКАДЕМИЯ

Контейнер Inversion of Controls

Dependency Inversion Principle

- **Инверсия зависимостей** – это та самая буква D в аббревиатуре SOLID.
- Позволяет избавить высокоуровневые абстракции от деталей реализации низкоуровневых абстракций
- Контейнер IoC позволяет отказаться от самостоятельного управления жизненным циклом объектов

Dependency Injection

- **Внедрение зависимостей** – механизм, с помощью которого Spring реализует принцип инверсии зависимостей
- Контейнер IoC самостоятельно создает и наполняет объекты, находящиеся под его управлением. Эти объекты принято называть «бины»(Beans).
- Бины, находящиеся под управлением контейнера IoC могут содержать другие бины, содержимое которых наполняется автоматически в зависимости от контекста.
- Для обозначения объектов, которые управляются контейнером IoC, применяются специализированные **аннотации** Spring.

Основные аннотации

- **@Component** – помечает класс, экземпляры которого будут управляться контейнером IoC. Имеет множество наследников, такие как @Controller, @Repository и другие.
- **@Configuration** – помечает класс, который является источником данных о конфигурации приложения
- **@Bean** – помечает метод класса, который будет непосредственно создавать экземпляр бина. Такой метод может находиться как внутри класса помеченного @Configuration, так и внутри класса, помеченного @Component.
- **@Autowired** – помечает переменную член класса, конструктор или метод-сеттер. Служит для указание, что контейнер должен заполнить помеченную переменную, либо передать в конструктор/сеттер соответствующий бин.
- **@Scope** – помечает бин и служит для указания области видимости объекта:

  singleton - По умолчанию. Spring IoC контейнер создает единственный экземпляр бина. Как правило, используется для бинов без сохранения состояния(stateless)

  prototype - Spring IoC контейнер создает любое количество экземпляров бина. Новый экземпляр бина создается каждый раз, когда бин необходим в качестве зависимости, либо через вызов getBean(). Как правило, используется для бинов с сохранением состояния(stateful)

  request - Жизненный цикл экземпляра ограничен единственным HTTP запросом; для каждого нового HTTP запроса создается новый экземпляр бина. Действует, только если вы используете web-aware ApplicationContext

  session - Жизненный цикл экземпляра ограничен в пределах одной и той же HTTP Session. Действует, только если вы используете web-aware ApplicationContext

  global session - Жизненный цикл экземпляра ограничен в пределах глобальной HTTP Session(обычно при использовании portlet контекста). Действует, только если вы используете web-aware ApplicationContext

  application - Жизненный цикл экземпляра ограничен в пределах ServletContext. Действует, только если вы используете web-aware ApplicationContext

Прорубаем выход наружу

веб-контейнер MVC + REST

Фреймворк Spring MVC

Spring MVC

- Построен на технологии сервлетов(Servlets).
- Реализует архитектурный паттерн Model View Controller
- Позволяет реализовать веб-API для фронтенда на любом фреймворке(React, Angular, Vue)
- Совместим с множеством шаблонных движков, например Thymeleaf, Freemaker, Mustache

Model View Controller

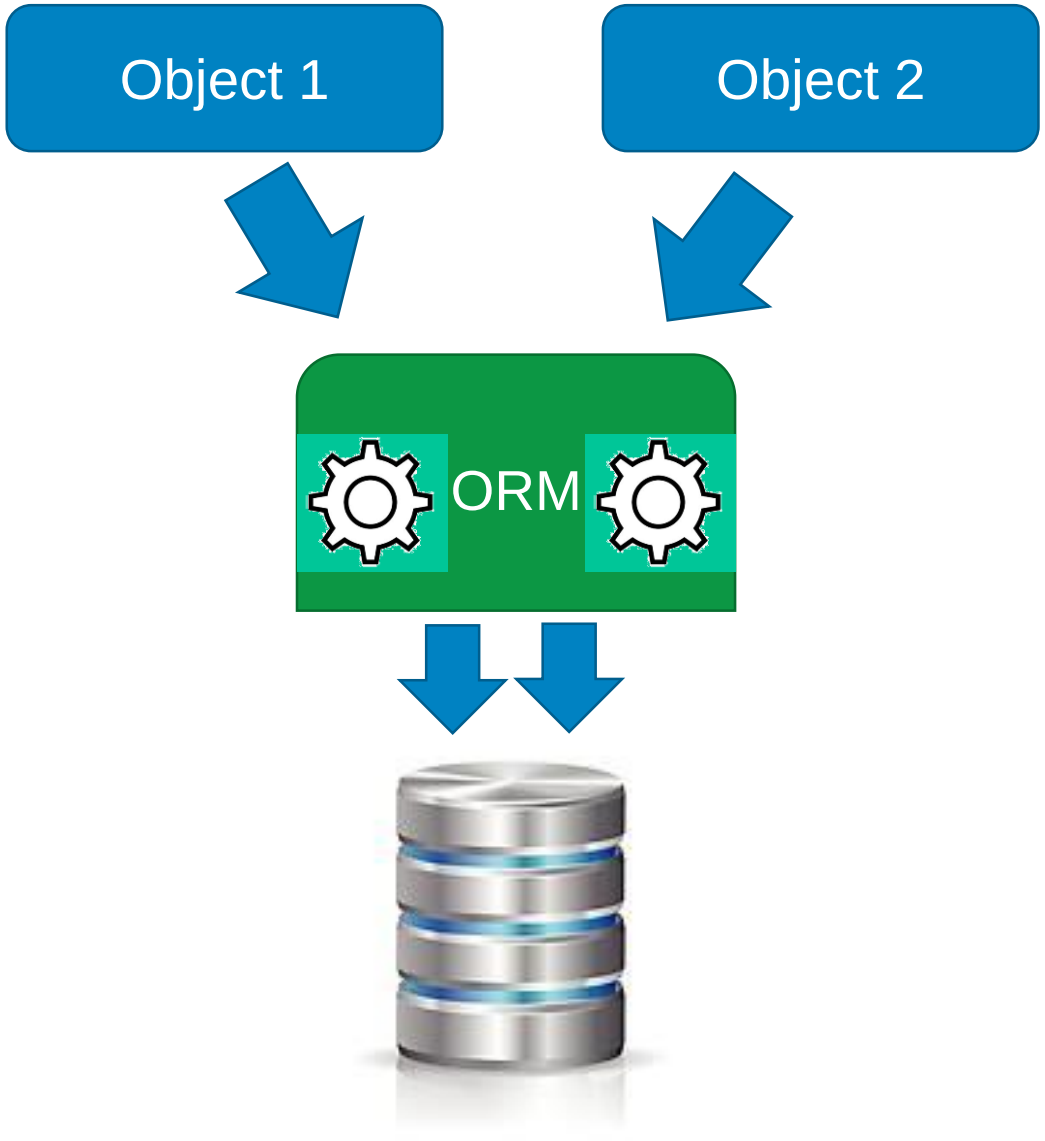
- **Model** — этот компонент отвечает за структуру данных, логику взаимосвязей между объектами данных приложения.
- **View** — этот компонент отвечает за непосредственное взаимодействие с пользователем. Определяет внешний вид приложения, способы его использования, реализует получение данных от пользователя.
- **Controller** — этот компонент отвечает за связь между Model и View. Код компонента controller определяет первоначальную реакцию на действия пользователя – возврат ошибки во View, либо обрабатывает данные для передачи в Model и другие варианты.

Основные аннотации

- **@Controller** – помечает класс-контроллер Spring MVC.
- **@RequestMapping** - помечает методы обработчика запросов внутри @Controller классов;
- **@RequestBody** – отображает тело HTTP-запроса на объект, который является входящим параметром для метода, который обрабатывает запрос
- **@RequestParam** - используется для доступа к отдельным параметрам HTTP-запроса
- **@ResponseBody** – указывает фреймворку, что результат метода-обработчик запроса необходимо отобразить в HTTP-ответ
- **@RestController** - объединяет @Controller и @ResponseBody .


Боремся с забывчивостью
ORM и Spring Data JPA
АКАДЕМИЯ

Что такое ORM



Object-Relational Mapping

Объектно-реляционное отображение

- ORM – это концепция работы с реляционными базами данных, применение которой позволяет работать с записями в БД как с объектами
- Объект ORM может состоять как из записи в одной таблице, так и собираться из нескольких таблиц одновременно
- Объект ORM может содержать ссылки на другие объекты ORM, как единичные, так и множественные(коллекции)

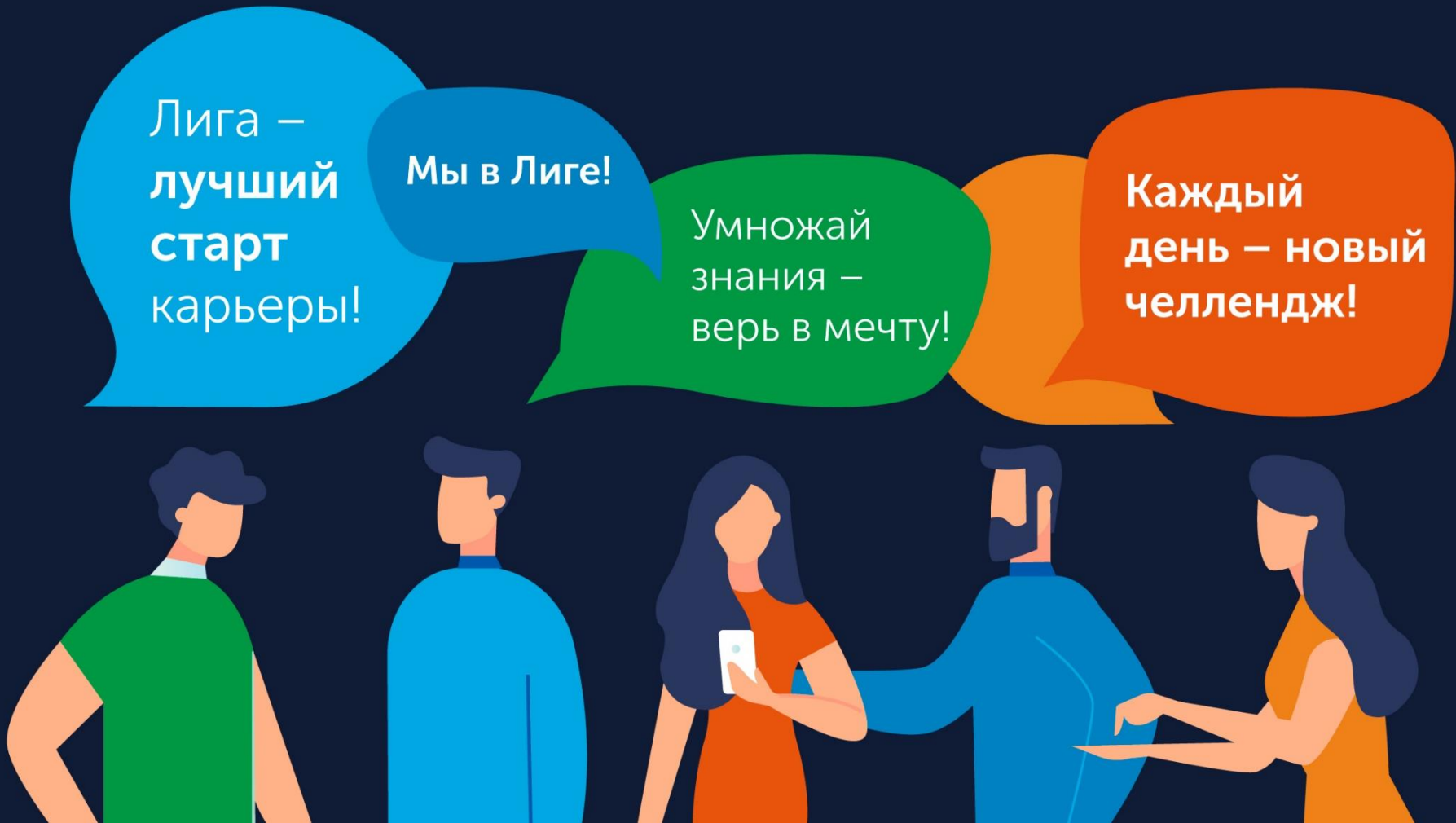
Spring Data JPA

- Spring Data – фреймворк в составе Spring, который позволяет работать с данными в БД и включает в себя реализацию концепции ORM
- Реализация концепции ORM в Spring Data называется Spring Data JPA(Java Persistence API)
- Существует большое количество библиотек, которые совместимы с JPA. Наиболее популярной является Hibernate.

Основные аннотации

- **@Repository** – помечает класс, который будет управлять данными.
- **@Entity** — аннотация используется, чтобы указать, что текущий класс представляет тип сущности - Entity класса.
- **@Table** — аннотация используется для указания первичной таблицы текущего аннотированного объекта Entity класса.
- **@Id** – помечает переменную как идентификатор сущности
- **@PersistenceContext —** аннотация используется для указания EntityManager, который необходимо ввести как зависимость.
- **@OneToOne, @ManyToOne, @OneToMany, @ManyToMany** — аннотации, указывающие соотношение связанных объектов – «один к одному», «многие к одному», «один ко многим», «многие ко многим»

Всем спасибо!



Андрей Лабазин

Разработчик

АКАДЕМИЯ