**Система сборки** – это инструмент, преобразующий исходный код в программу, которую можно запускать. Существует несколько систем сборок:

* ***Apache Ant*** – использует императивный подход с составлением алгоритма действий в *XML*-файле.
* ***Apache Maven*** – использует декларативный подход с описанием конечного результата.
* ***Gradle*** – комбинирует императивный (сценарии сборки) и декларативный (дерево зависимостей) подходы.

Преимущества *Maven*:

* Декларативное описание проекта. Для этого используется файл *pom.xml*, где объявляется, что должно получиться в результате (не требует конкретных команд).
* Автоматическое управление зависимостями. Система сама скачивает зависимости и разрешает конфликты версий.
* Фиксированные циклы сборки проекта, состоящие из определенной последовательности фаз, с достижением конкретных целей в каждой.
* Четкая структура каталогов, где располагаются исходный код, тесты и другие ресурсы.
* Модульная структура, позволяющая легко расширить функциональность проекта.

**Архетип** – это шаблон, который можно использовать в качестве основы для своего проекта. Он помогает стандартизировать разработку и получить доступ к уже готовым решениям.

**Артефакт** – это то, что создано в процессе разработки программного обеспечения. Им может быть любая библиотека, зависимость или плагин.

**Артефакт *Maven*** – это файл *JAR* формата, который появляется при сборке проекта. У него есть своя группа настроек:

* ***groupId*** – уникальный идентификатор группы, которые создали проект. Как правило он основан на полном доменном имени организации (например, *ru.yandex.practicum*).
* ***artifactId*** – уникальное имя артефакта, генерируемого этим проектом.
* ***version*** – версия артефакта.

Ссылка на гайд по основам *XML* - <https://doka.guide/tools/xml/>

Для оформления *pom.xml* также могут использоваться следующие **теги**:

* *<build> </build>* - блок настройки процесса сборки
* *<plugins> </plugins>* - блок настройки плагинов
* *<resources> </ resources>* - блок настройки доп. ресурсов
* *<dependencies> </ dependencies>* - блок настройки зависимостей
* *<dependencyManagement> </ dependencyManagement>* - блок настройки зависимостей в *parent-POM*
* *<goals> </ goals>* - блок настройки задач/целей
* *<profiles> </ profiles>* - блок настройки профилей создания
* *<developers> </ developers>* - блок настройки разработчиков
* *<mailingLists> </ mailingLists>* - блок настройки списков рассылки
* *<parent> </ parent>* - блок настройки родительского проекта
* *<scope> </scope>* - модификатор, указывает на каком этапе сборки используется данный артефакт

Ссылка на документацию Maven - <https://maven.apache.org/pom.html#Maven_Coordinates>

**Репозиторий *Maven*** – это директория, где хранятся все файлы проекта, библиотеки, плагины и любые другие артефакты. Существуют локальный репозиторий (папка с расширением *.m2*), удаленные репозитории сообществ (через протоколы *https://*, *file://* и другие) и центральный репозиторий Maven Central (<https://repo.maven.apache.org/maven2/>). Инструкции по подключению можно найти на <https://search.maven.org/> и <https://mvnrepository.com/>. Пример подключения:

*<repositories>*

*<repository>*

*<id>additional-local-dir</id>*

*<url>file://path/to/directory</url>*

*</repository>*

*<repository>*

*<id>org-remote-repository</id>*

*<url>https://maindomain.org/maven2</url>*

*</repository>*

*<repositories>*

Приоритеты поиска зависимостей в репозиториях локальный -> центральный -> удаленный.

Процесс сборки включает несколько жизненных циклов. Всего их три:

1. ***clean*** – в ходе этого цикла происходит очистка проекта.
2. ***default*** – основной цикл сборки. В ходе него происходит компиляция кода, генерация тестов, сборка дополнительных артефактов, различного рода проверки и т.д. Включает 23 фазы.
3. ***site*** – посвящен созданию проектной документации.

Каждый цикл включает набор фаз. Есть подготовительные (*pre-*), основные и завершающие (*post-*) фазы. Ссылка на фазы циклов - <https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html>.

Следующий плагин с помощью команды ***surefire-report:report***создает HTML-страницу с отчетом о пройденных тестах в удобном для чтения виде.

*<plugin>*

*<groupId>org.apache.maven.plugins</groupId>*

*<artifactId>maven-surefire-report-plugin</artifactId>*

*<version>2.22.2</version>*

*</plugin>*

**Дерево зависимостей** (***dependency tree***) – совокупность зависимостей всех компонентов, используемых в проекте. Посмотреть все зависимости можно с помощью команды ***dependency:tree***. Зависимости, которые появились в проекте неявно, называются **транзитивными** (***transitive***) или переходными.

Когда для работы пакетов требуются разные версии одной и той же библиотеки, проект попадает в **ад зависимостей** (***dependency hell***). Из-за этого проект может не запуститься. В качестве решения предлагается файл ***BOM*** (***Bill of Material***) или **список материалов**.

В *maven* реализована возможность **наследования** от *parent*-проекта с помощью тега *<parent></parent>*. Для создания проекта-предка используется строка *<packaging>pom</packaging>*. Изменения в проекте-наследнике имеют приоритет над проектом-предком. Благодаря наследования можно переиспользовать все связанные зависимости. Файл *BOM* является предком для *POM*-файла. Например, с помощью наследования *BOM*-файла можно указать все версии зависимостей и не указывать их (или переопределить версию нужной зависимости в блоке *<properties>*) в *POM*-файле.

При наследовании может быть только один проект-предок и нельзя подключить второй *BOM*-файл. Обойти это ограничение можно, подключив *BOM* как зависимость с использованием строк *<type>pom</type>* и *<scope>import</scope>*.

<https://docs.oracle.com/middleware/1212/core/MAVEN/maven_version.htm#MAVEN8855>

<https://medium.com/javarevisited/10-important-maven-plugins-for-java-developers-330b98b71720>

<https://habr.com/ru/post/205118/>

<https://github.com/eirslett/frontend-maven-plugin>

<https://www.baeldung.com/maven-polyglot>

<https://spring.io/>

<https://start.spring.io/>

***Spring Boot*** – упрощенная версия ***Sping Framework*** с максимально упрощенной настройкой запуска.

Аннотация ***@SpringBootApplication*** показывает фреймворку *Spring Boot*, что это основной класс с конфигурацией приложения.

Файл ***application.properties*** хранит настройки проекта (стандартные настройки *Spring Boot*, пути к сторонним приложениям и т.д.). Например, *server.port=8081*.

<https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html>

В результате разработки подхода ***API First*** (когда разработка строится вокруг *API*) появился архитектурный стиль ***REST*** (*Representation State Transfer*) или **передача репрезентативного состояния**. Он состоит из ключевых принципов проектирования распределенных систем, заложенных Филдингом:

1. Клиент-серверная архитектура с разделением ответственности (***Client-Server***).
2. Сервер не хранит информацию о состоянии клиента (***Stateless***). Каждый запрос независим, как будто сделан в первый раз.
3. Единообразие интерфейса обращения к серверу с любого типа устройства (***Uniform Interface***).
4. Многоуровневость архитектуры (***Layered system***). Все внутренние взаимодействия также работают по принципу клиент-сервер.
5. Кешируемость (***Cache***). При идентичном запросе данные берутся из кеша, а не повторно запрашиваются у сервера.
6. Код по запросу (***Code on demand***). Функциональность клиента может быть расширена кодом, приходящим с сервера (сценарии оживления страниц).

**Преимущества использования *REST*:**

1. Производительность
2. Масштабируемость
3. Гибкость к изменениям
4. Отказоустойчивость
5. Простота поддержки

Ключевое понятие в *REST* – это **ресурс**, т.е. данные любого типа и формата. Для указания местоположения ресурса используется ***URL***.

**Основные правила именования ресурсов:**

* Использовать существительные
* Слеш указывает на иерархию
* Слеш в конце *URL* лучше не использовать
* Для разделения слов использовать дефисы (пробелы, слитное написание и нижние подчеркивания под запретом)
* Не включать в название ресурса имя *HTTP*-метода

Правила именования ресурсов в *REST* <https://restfulapi.net/resource-naming/>

В большинстве случаев в качестве протокола реализации *REST* используется *HTTP* и его методы:

* ***GET*** получает ресурсы
* ***POST*** создает новый ресурс
* ***PUT*** заменяет существующие данные или, при их отсутствии, создает новый ресурс
* ***PATCH*** используется для частичного обновления данных ресурса
* ***DELETE*** удаляет ресурс
* ***HEAD*** используется для получения заголовков ответа
* ***OPTIONS*** используется для получения списка *HTTP*-методов, которые поддерживает сервер

Метод считается **небезопасным**, если может изменить ресурс. Метод считается **идемпотентным**, если всегда даст один и тот же результат независимо от текущего состояния. Метод *POST* не идемпотентен, т.к. каждый раз создает новый ресурс.

Приложение на базе *Spring Framework* настраивается через **аннотации**. При запуске фреймворк проверяет их наличие и связывает помеченный ими код с определенной функциональностью. Это называется **конфигурацией, основанной на аннотациях**.

**Контроллер** – это специальный класс в приложении, который предназначен для обработки *HTTP*-запросов от клиента и возвращения результатов (т.е. класс для создания эндпоинтов).

Аннотация ***@RestController*** обозначает классы-контроллеры с методами обработки запросов к эндпоинтам *API*.

Аннотация ***@RequestMapping*** дает фреймворку понять, по какому пути можно будет отправить запрос к этому методу контроллера. Пути запроса строятся относительно базового пути приложения. Позволяет аннотировать весь класс – в этом случае путь строится путем сложения общей аннотации класса и аннотации конкретного метода.

Начиная со Spring Framework 4.3, на основе аннотации *@RequestMapping* появились новые аннотации ***@GetMapping***, ***@PostMapping***, ***@PutMapping***, ***@DeleteMapping*** и ***@PatchMapping***. Они позволяют указать конкретный метод запроса и отказаться от свойства *method*.

*@RequestMapping(value = "/home", method = RequestMethod.GET) // можно так*

*@GetMapping("/home") // но так проще и удобнее*

Аннотация ***@RequesrBody*** означает, что значение аргумента нужно взять из тела запроса. При этом объект, который пришел в теле запроса, будет автоматически сконвертирован в *Java*-объект.

*public Post create(@RequestBody Post post) {*

*posts.add(post);*

*return post; }*

Готовый проект [catsgram](https://start.spring.io/#!type=maven-project&language=java&packaging=jar&jvmVersion=11&groupId=ru.yandex.practicum&artifactId=catsgram&name=catsgram&description=%D0%9C%D0%B8%D0%BD%D0%B8-%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D1%8F%20%D0%BF%D0%BE%D0%BF%D1%83%D0%BB%D1%8F%D1%80%D0%BD%D0%BE%)

Гайды на postman <https://habr.com/ru/company/maxilect/blog/596789/>

<https://learning.postman.com/docs/getting-started/introduction/>