1. **JDBC.**

Технология JDBC (Java Database Connectivity) – это стандартный прикладной интерфейс (API) языка Java для организации взаимодействия между приложением и СУБД. JDBC позволяет получать соединения с БД по специально описанному URL. Драйверы JDBC обычно создаются поставщиками СУБД.

Есть следующие типы драйверов:

- *Bridge драйвер*
- *Драйвер использующий native API*
- *Драйвер использует сетевой протокол и middleware*
- *Драйвер работает с Java и работает напрямую с БД*

Преимущества:

- *Лёгкость разработки: разработчик может не знать специфики базы данных, с которой работает*
- *Код не меняется, если компания переходит на другую базу данных*
- *Не нужно устанавливать громоздкую клиентскую программу*
- *К любой базе можно подсоединиться через легко описываемый URL*

Компоненты JDBC

- Driver Manager (предоставляет средства для управления набором драйверов баз данных, нужен для выбора БД и создания соединений)
- Драйвер (обеспечивает реализацию общих интерфейсов для СУБД)
- Соединение (сессия между приложением и драйвером БД)
- Запрос (запрос к бд на выборку или изменение данных)

- Результат (мн-во строк и столбцов бд)
- Метаданные (сведения о полученном результате и об используемой бд)

Использование JDBC.

- Последовательность действий:
- Загрузка класса драйвера базы данных.
- Установка соединения с БД.
- Создание объекта для передачи запросов.
- Выполнение запроса.
- Обработка результатов выполнения запроса.
- Закрытие соединения.

Загрузка класса драйвера базы данных:

Class.forName([location of driver]);

Получение соединения:

Connection cn = DriverManager.getConnection("jdbc:mysql://localhost/my_db", "login", "password"); // логин пароль урл можно задавать в файле датабейс пропертис и брать оттуда

Объект Statement

– используется для выполнения запросов и команд SQL, а также для установки некоторых ограничений на запросы;

– один и тот же объект Statement может быть использован многократно для различных запросов.

executeQuery выполняет предварительный запрос на выборку (SELECT).

Для INSERT, UPDATE, DELETE используется executeUpdate()

По окончании использования необходимо последовательно вызвать метод close() для объектов ResultSet, Statement и Connection для освобождения ресурсов.

DAO

DAO используется для абстрагирования и инкапсулирования доступа к источнику данных, управляет соединением. Для каждой таблицы можно создавать свое DAO, которое сокроет логику открытия соединения, отправки запроса и т.д., что облегчает работу с БД.

PreparedStatement : предварительно готовится и хранится, позволяет ускорить обмен информацией с БД.

PreparedStatement-оператор предварительно откомпилирован, поэтому он выполняется быстрее обычных операторов ему соответствующих. Он используется для часто вызываемых запросов, но с разными параметрами, и таким образом улучшает производительность.

PreparedStatement ps = cn.prepareStatement(sql); // компиляция подготовленного запроса

ps.setInt(1, id);

ps.setString(2, name);

(можно подставить для каждой позиции в PreparedStatement нужное значение).

выполнение компилированного запроса

ps.executeUpdate()

CallableStatement

В терминологии JDBC, хранимая процедура - последовательность команд SQL, хранимых в БД и доступных любому пользователю этой СУБД. Механизм создания и настройки хранимых процедур зависит от конкретной базы данных.

• Интерфейс CallableStatement обеспечивает выполнение хранимых процедур • Объект CallableStatement содержит команду вызова хранимой процедуры, а не саму хранимую процедуру.

• CallableStatement способен обрабатывать не только входные (IN) параметры, но и выходные (OUT) и смешанные (INOUT) параметры. Тип выходного параметра должен быть зарегистрирован методом registerOutParameter().

• После установки входных и выходных параметров вызываются методы execute(), executeQuery() или executeUpdate().

(то есть в БД есть какая-то процедура и ее можно вызвать с помощью этой штуки)

Пул соединений

Пул соединений реализуется согласно шаблону Singleton. В нем необходимо создать свойство -коллекцию соединений, в котором будут хранится все свободные соединения с СУБД.

• По завершении работы с БД следует вернуть соединение в пул, вызвав соответствующий метод

Стандартная реализация пула выполнена в виде класса, который дает возможность:

-загрузить необходимые драйвера для конкретной базы данных;
- получить ссылку на объект типа DataSource;
- получить доступное соединение типа Connection из хранилища;
- возвратить соединение обратно в хранилище;
- уничтожить все ресурсы и закрыть все соединения из хранилища;

Транзакции

Транзакции нужны для таких ситуаций, когда есть несколько операций, и если хотя бы одна из них не была успешной, то все операции не должны выполняться (т.е. либо выполняются все операции, либо не выполняется ни одной). В таком случае этот набор трактуют как одну операцию и называют транзакцией.

Транзакции должны удовлетворять услвоиям:

- Атомарность – (либо все, либо ни одна). Успешно завершенные транзакции фиксируются, в случае неудачи происходит откат к начальному состоянию
- Согласованность (если произошел сбой, то система возвращается в состояние до начала неудавшейся транзакции)
- Изолированность – во время выполнения транзакции все объекты сущности участвующие в ней должны быть синхронизированы
- Долговечность – все изменения произведенные с данными во время транзакции записываются в систему данных

По умолчанию в JDBC включен режим setAutoCommit, поэтому коммит выполняется после команд executeQuery() и executeUpdate(). Для подтверждения транзакции используется метод commit, для отката к последнему коммиту (либо savepoint) используется команда rollback().

```
Savepoint svpt = cn.setSavepoint("NewEmp");

cn.rollback(svpt);
```

Уровни изоляции транзакций определены в виде констант интерфейса Connection (по возрастанию уровня ограничения):

• TRANSACTION_NONE – информирует о том, что драйвер не поддерживает транзакции;

• TRANSACTION_READ_UNCOMMITTED – позволяет транзакциям видеть несохраненные изменения данных, что разрешает грязное, непроверяющееся и фантомное чтения;

• TRANSACTION_READ_COMMITTED – означает, что любое изменение, сделанное в транзакции, не видно вне неё, пока она не сохранена. Это предотвращает грязное чтение, но разрешает непроверяющееся и фантомное;

• TRANSACTION_REPEATABLE_READ – запрещает грязное и непроверяющееся, но фантомное чтение разрешено;

• TRANSACTION_SERIALIZABLE – определяет, что грязное, непроверяющееся и фантомное чтения запрещены.

2. Технология Java Persistence API

ORM (Object-Relational Mapping) – технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных»

Java Persistence API предоставляет:

• стандартный механизм для ORM,

• EntityManager API для создания, обновления и удаления объектов,

• язык запросов JPA-QL для извлечения сущностей.

EJB (Enterprise Java Beans) – это фреймворк для построение бизнес-логики приложения.

В EJB3 JPA используются аннотации для определения объектов и отношений.

- логические аннотации (позволяют описать объектную модель, связи между классами)

- физические аннотации (описывающие физическую схему, таблицы, столбцы, индексы и т.д.).

Сущность (Entity) – это простой объект (POJO), который нужно сохранить в реляционной базе данных. Сущность может быть как абстрактным, так и конкретным классом, а также может наследовать другой POJO. Для того чтобы объявить POJO сущностью, его нужно отметить аннотацией @Entity.

Аннотация @Table указывается на уровне класса. Она позволяет указать таблицу, каталог и схему, которые соответствуют сущности. Если эта аннотация не указана, в качестве имени таблицы воспринимается имя класса. @Entity @Table(name="table_name")

EJB3 поддерживает три типа наследования

- - Стратегия «по таблице на каждый класс».
- - Стратегия «одна таблица на иерархию классов».
- - Стратегия «составной подкласс»

Стратегия «по таблице на каждый класс» • Каждому классу соответствует своя таблица в базе данных. Преимущества: для работы не нужно соединять несколько таблиц, если подклассы также являются сущностями. Недостатки: база становится не нормализованной, эта стратегия слабо поддерживает объединение таблиц, соответствующих классам одного уровня, а так же полиморфизм.

Стратегия «одна таблица на иерархию классов» • Преимущества: простота и отсутствие необходимости соединять таблицы. Недостатки: база становится ненормализованной. При этом столбцы, соответствующие полям подклассов, должны допускать нулевые значения, большое количество столбцов в таблице.

Стратегия «составной подкласс»

Преимущества: база данных уже нормализована. При этом представления (view) базы данных имеет ту же структуру, что и объектная модель и поэтому последнюю можно легко выделить. Недостатки: снижение производительности в глубоких иерархиях, полиморфных запросах и отношениях.

Persistence Contexts

• это множество управляемых сущностей, в котором каждому уникальному идентификатору соответствует единственная сущность. В persistence context сущности и их жизненный цикл управляются объектом EntityManager.

• EntityManager может быть управляемым контейнером. Его жизненный цикл управляется контейнером Java EE. управляемым приложением.

Интерфейс Query позволяет работать с запросами (статическими и динамическими).

Запросы должны быть написаны на языке JPA-QL. Основное его отличие от стандартного языка заключается в том, что вместо имен таблиц и столбов указывают имена сущностей и их свойств.

Оптимистическое блокирование: Перед тем, как подтвердить изменение данных , провайдер контролирует, что сущность не изменилась с тех пор как она была считана из БД другими транзакциями.

Пессимистическое блокирование: Провайдер создает транзакцию, которая получает длительную блокировку на данные (до завершения транзакции). Другие транзакции не могут модифицировать или удалить данные пока блокировка не будет снята.

JSP

JSP (JavaServer Pages) — технология, позволяющая вебразработчикам динамически генерировать HTML, XML и другие веб-страницы. Является составной частью единой технологии создания бизнес-приложений JEE. Технология позволяет внедрять Java-код, а также EL (expression language) в статичное содержимое страницы

- Есть разделение динамического и статического содержания
- Независимость от платформы (т.к. основана на Java)
- Многократное использование компонентов
- Поддерживает java script и html-теги

Смысл разделения динамического и статического содержания в том, что статические ресурсы могут находиться под управлением HTTP-сервера, в то время как динамические нуждаются в движке (Servlet Engine) и в большинстве случаев в доступе к уровню данных

динамические ресурсы, изменяемые в процессе работы: сервлеты, JSP, а также java-код;

статические ресурсы, не изменяемые в процессе работы – HTML, JavaScript, изображения и т.д.

JSP страница содержит статический HTML называемый текстом шаблона. JavaServer Pages (JSP) позволяют отделить динамическую часть страниц от

статического HTML. Динамическую часть кода заключается в специальные тэги, большинство которых начинаются с "<%" и завершаются “%>”

JSP может включать в себя следующие конструкции:

- Элементы скриптов
- Директивы
- Действия

Элементы скриптов JSP позволяют вставлять код на Java в сервлет, создаваемый из текущей JSP страницы. Существуют три формы:

- Выражения, имеющие форму <%= выражение %>, которые обрабатываются и направляются на вывод
- Скриплеты, имеющие форму <% код %> , которые вставляются в метод service() сервлета
- Объявления, имеющие форму <%! код %>, которые вставляются в тело класса сервлета, вне существующих методов

Выражения JSP

Предопределенные объекты, которые можно использовать в выражениях:

- request объект типа HttpServletRequest;
- response объект типа HttpServletResponse;
- session объект типа HttpSession ассоциируется с запросом (если таковой имеется);
- out объект типа JspWriter (буферизированный вариант типа PrintWriter) используется для отсылки выводимых данных клиенту.

Директивы JSP воздействуют на всю структуру класса сервлета. Обычно они имеют следующую форму: <%@ директива фтрибут = “значение”%>

Можно объединить установку нескольких атрибутов для одной директивы.

Существуют три основных типа директив: page - позволяет совершать такие операции, как импорт классов, изменение суперкласса сервлета, и т.п.; include - дает возможность вставить файл в класс сервлета при

трансляции JSP файла в сервлет; taglib - для подключения библиотеки пользовательских тегов.

Директива JSP page

isThreadSafe="true|false"

- true задает нормальный режим выполнения сервлета, когда множественные запросы обрабатываются одновременно с использованием одного экземпляра сервлета.
- false сигнализирует о том, что сервлет должен наследовать SingleThreadModel (однопоточную модель) при которой последовательные или одновременные запросы обрабатываются отдельными экземплярами сервлета.

session="true|false". Определяет, будут ли использоваться сессии.

- true сигнализирует о том, что заранее определенная переменная session (тип HttpSession) должна быть привязана к существующей сессии;
- false определяет что сессии не будут использоваться, и попытки обращения к переменной session приведут к возникновению ошибки при трансляции JSP страницы в сервлет

buffer="размерkb|none". Задает размер буфера для JspWriter out. Значение принимаемое по умолчанию зависит от настроек сервера, но должно превышать 8kb.

• autoflush="true|false". Определяет, будет ли автоматически очищаться буфер при переполнении.

• extends="пакет.class". Задает суперкласс для генерируемого сервлета.

Неявные объекты: request, response, out, session, application, config, pageContext и page. Дополнительно: exeption

request: Это объект HttpServletRequest, связанный с запросом, который позволяет обращаться к параметрам запроса (через метод getParameter), типу запроса (GET, POST, HEAD, и т.д..), и входящим HTTP заголовкам (cookies, Referer, и т.д..).

response Это объект типа HttpServletResponse, связанный с ответом на запрос клиента.

session Это объект типа HttpSession, связанный с запросом. Сессии создаются автоматически, и эта переменная существует даже если нет ссылок на входящие сессии.

application Это объект типа ServletContext полученный через использование метода getServletConfig().getContext().

config Это объект типа ServletConfig для текущей страницы.

pageContext В JSP представлен новый класс PageContext. Определяет контекст jsp страницы и предоставляет доступ к неявным объектам.

page По сути является синонимом для this, и не нужен при работе с Java.

out Это объект типа PrintWriter, используемый для отправки вывода клиенту.

Java Servlet

Сервлеты – это компоненты приложений Java Enterprise Edition, выполняющиеся на стороне сервера, способные обрабатывать клиентские запросы и динамически генерировать ответы на них. Наибольшее распространение получили сервлеты, обрабатывающие клиентские запросы по протоколу HTTP. Сервлеты можно внедрять в различные сервера, так как API сервлета, который вы используете для его написания, ничего не «знает» ни о среде сервера, ни о его протоколе. Множество Web-серверов поддерживает технологию Java Servlet

Пакет javax.servlet обеспечивает интерфейсы и классы для написания сервлетов. Центральной абстракцией API сервлета является интерфейс Servlet. Все сервлеты реализуют данный интерфейс напрямую, но более распространено расширение класса, реализующего его, как HttpServlet. Интерфейс Servlet объявляет, но не реализует методы, которые управляют сервлетом и его взаимодействием с клиентами

Принимая запрос от клиента, сервлет получает два объекта: 1. ServletRequest, который инкапсулирует связь клиента с сервером. 2. ServletResponse, который инкапсулирует обратную связь сервлета с клиентом. ServletRequest и ServletResponse – это интерфейсы, определенные пакетом javax.servlet.

Интерфейс ServletRequest дает сервлету доступ к: 1. Информации, такой как имена параметров, переданных клиентом, протоколы (схемы), используемые клиентом и имена удаленного хоста, создавшего запрос и сервера который их получает. 2. Входному потоку ServletInputStream.

Сервлеты используют входной поток для получения данных от клиентов, которые используют протоколы приложений, такие как HTTP POST и PUT методы. Интерфейсы, которые расширяют интерфейс ServletRequest, позволяют сервлету получать больше данных, характерных для протокола. К примеру, интерфейс HttpServletRequest содержит методы для доступа к главной информации по протоколу HTTP.

Интерфейс ServletResponse дает сервлету методы для ответа на запросы клиента. Он: 1. Позволяет сервлету устанавливать длину содержания и тип MIME ответа (метод setContentType(String type)). 2. Обеспечивает исходящий поток ServletOutputStream и Writer, через которые сервлет может отправлять ответные данные. Интерфейсы, которые расширяют интерфейс ServletResponse, дают сервлету больше возможнойстей для работы с протоколами. Например, интерфейс HttpServletResponse содержит методы, которые позволяют сервлету манипулировать информацией заголовка HTTP.

В HTTP сервлетах есть возможность управления сессией (session-tracking). Автор сервлета может использовать этот API для поддержания состояния между сервлетом и клиентом, которое сохраняется на нескольких соединениях в течении определенного времени. В HTTP сервлетах также есть объекты, которые обеспечивают создание файлов-cookie. Автор сервлета использует файлысookie API для сохранения данных, ассоциированных с клиентом и для получения этих данных.

Жизненный цикл сервлета

• Пользователь вводит URL в браузере. Конфигурационный файл Webсервера указывает, что этот URL предназначен для сервлета, управляемого контейнером сервлетов на сервере. Если экземпляр сервлета еще не был создан (существует только один экземпляр сервлета для приложения), контейнер загружает класс и создает экземпляр объекта.

• Контейнер вызывает метод init() сервлета. Он дает сервлету возможность инициализировать данные и подготовиться для обработки запросов.

• Контейнер вызывает метод service() сервлета и передает HttpServletRequest и HttpServletResponse. Для каждого нового клиента при обращении к сервлету создается независимый поток, в котором

производится вызов метода service(). Метод service() предназначен для одновременной обработки множества запросов.

• При необходимости, когда сервлет выполнил полезную работу, контейнер вызывает метод destroy() сервлета, в теле которого следует помещать код освобождения занятых сервлетом ресурсов.

JavaScript. Работа с окном браузера и с документом. События.

Библиотека RxJS.

Реактивное программирование — это парадигма асинхронного программирования, связанная с потоками данных (streams) и распространением изменений. Эта парадигма полезна при обработке больших объемов асинхронных данных.

Преимущества:

- Приложение реагирует своевременно, если это вообще возможно;
- Приложение реагирует на сбои;
- Приложение остается отзывчивым при различной рабочей нагрузке;
- Реактивные системы полагаются на асинхронную передачу сообщений.

RxJS (Reactive Extensions for JavaScript) — это библиотека для реактивного программирования с использованием наблюдаемых объектов (observables), которая упрощает создание асинхронного кода или callback-based кода.

Webpack — это инструмент, позволяющий скомпилировать JavaScript модули в единый JS-файл. Webpack также известен как сборщик модулей.

Конфигурационный файл Webpack - webpack.config.js хранится в корневой директории проекта. Загрузчики - принимают содержимое файлов, преобразуют его необходимым образом и включают результат преобразования в общую сборку. Загрузчик ts-loader позволяет работать с ts-файлами.

Плагины. Плагин HTMLWebpackPlugin автоматически создает HTMLфайл с уже подключенным скриптом

Зависимости проекта

Express - фреймворк, используемый при создании вебприложений для node. server.js запускает HTTP-сервер на порту 8080 и отправляет страницу index.html на запросы, поступающие по маршруту GET /*. Модуль TypeScript - компилирует TypeScript код в удобочитаемый, основанный на стандартах JavaScript.

Stream — это последовательность текущих событий, упорядоченных во времени. Stream производит (emits) 3 типа объекта: типизированное значение (data value), Ошибку (error), Сигнал завершения (termination mark). Эти генерируемые события фиксируются только асинхронно. Для этого определяются функции, которые вызываются и им передается одно из трех соответствующих выходных событий в качестве параметра.

Observable: поток значений или событий.

Observer: это набор функций, которые знают, как прослушивать значения, предоставленные Observable.

Subscription: представляет выполнение Observable (полезно для отмены выполнения)

Operators: это чистые функции, которые обеспечивают функциональный стиль программирования для работы с коллекциями (map, filter, concat, reduce и т. д).

Observables – это lazy push коллекции. Протоколы push и pull описывают как производитель данных (Producer) взаимодействует с потребителем данных (Consumer). В pull системах Consumer определяет, когда он получает данные от Producer. Сам Producer не знает, когда данные будут доставлены Consumer.

В системах push Producer определяет, когда отправлять данные Consumer. Consumer не знает, когда он получит эти данные. Promise — самый распространенный тип системы push в JavaScript. Promise (Producer)

предоставляет значение зарегистрированным callbacks (Consumer); Promise отвечает за точное определение того, когда это значение «проталкивается» в callbacks. Observables - новая push система в JavaScript. Observable — это производитель нескольких значений, «проталкивающий» их к наблюдателям (потребителям).

В среде реактивного программирования наблюдаемый объект (Observable) отвечает за отправку данных подписчикам (push) . Observable можно создавать с помощью new Observable.

const observable = new Observable(function subscribe(subscriber) { });

На наблюдаемый объект нужно подписаться: observable.subscribe(observer);

Observer — это потребитель значений, предоставляемых Observable. Observer — это набор функций, по одной для каждого типа уведомления, доставляемого Observable: next, error, complete.

```
const observer = {
        next: x => console.log('Observer got a next value: ' + x),
        error: err => console.error('Observer got an error: ' + err),
        complete: () => console.log('Observer got a complete notification'), };

observable.subscribe(observer)
```

Фреймворк Angular

Angular – это фреймворк для разработки приложений на клиентской стороне с использованием архитектуры model– view–controller (MVC) and model–view–view-model (MVVM).

Одностраничные приложения (SPA) - это приложения, у которых есть одна точка входа - HTML-страница (может быть index.html); все содержимое приложения динамически добавляется и удаляется с этой страницы. В одностраничных web-приложениях исходный HTMLдокумент отправляется в браузер. Действия пользователя приводят к Ajax запросам небольших фрагментов HTML или данных, вставленных в существующий набор элементов, которые отображается пользователю.

TypeScript – язык, используемый в разработке Angular2 (>)

Angular CLI - используется для создания проектов, генерации кода приложений и библиотек, выполнения текущих задач разработки, таких как тестирование, связывание, развертывание.

Angular для организации кода использует модульную систему под названием NgModule. Каждое Angular приложение имеет как минимум один класс с декоратором @NgModule. Это корневой модуль, обычно называемый AppModule.

Компонент является основным строительным блоком пользовательского интерфейса (UI). Angular application - это дерево компонентов. Каждый компонент сопоставляется с шаблоном (template). Компонент содержит свойства, методы, конструктор, а также события ввода, события вывода и методы жизненного цикла, такие как ngOnInit, ngOnDestroy и т. Д.

Шаблон (template) это HTML код , определяющий как отображать компонент на экране. Шаблон помимо обычного HTML содержит директивы (directives), события (events), интерполяции (interpolation), привязку данных (data binding) и теги других компонентов.

Angular как фреймворк стал популярным выбором для фронтенд-разработки, написан с использованием TypeScript. Синтаксис языка имеет некоторое сходство с Java, что упрощает работу Java-разработчиков. Особенности языка: • types • classes • decorators • imports • language utilities

tsconfig.json - конфигурационный файл для TypeScript компилятора. Он определяет как преобразовать (transpile) TypeScript файл в Javascript.

compilerOptions – настройки компилятора

"target":"es5" - версия JavaScript для компилятора. Компилятор преобразует TypeScript в простой код JavaScript, использующий только возможности заданной версии. Значение es5 соответствует стандарту ES5, поддерживаемому большинством браузеров;

"module":"commonjs" - формат создания модулей JavaScript; значение должно соответствовать загрузчику, используемому в проекте;

"moduleResolution": "node" - режим обработки команд import компилятором. Со значением node пакеты ищутся в папке node_modules, где их размещает NPM;

"emitDecoratorMetadata": true - Со значением true компилятор включает информацию о декораторе, к которой можно обратиться при помощи пакета reflect-matadata; "experimentalDecorators": true },

"exclude": ["node_modules"] - параметр сообщает компилятору, какие каталоги следует игнорировать }

Преимущества TypeScript перед JavaScript - обеспечивает лучшую типизацию данных, структуру и читабельность кода, а также обладет обратной совместимостью с JS. Недостатки - типы контролируются только до этапа компиляции, в результате которого весь код превращается в JavaScript. Это означает, что если с сервера будут переданы аргументы неверного типа — код об этом не сообщит.

TypeScript

Примитивные типы: string, number, boolean, bigint, undefined, null, symbol. Явная аннотация типа: let firstname : string = 'Danny' Избыточно, т.к. TypeScript автоматически присваивает тип переменной (вывод типа).: let firstname = 'Danny'

TypeScript не имеет доступа к DOM. Это означает, что при обращаении к DOM-элементам TypeScript не может быть уверен в том, что они существуют. С оператором ненулевого подтверждения ! можно сказать компилятору, что выражение не равно null или undefined: const link = document.querySelector('a')!;

Если надо найти DOM-элемент по его классу или id, надо сообщить TypeScript, что этот элемент существует, и что он имеет тип HTMLFormElement. Для этого используется приведение типов (ключевое слово as): const form = document.getElementById('signup-form') as HTMLFormElement;