(1) INTERNET EXPLAINED

1. **Каква е целта на транспортния слой?**

   Целта му е да пренесе едно съобщение по мрежата, така че крайният получател да го получи в същия вид, в който е изпратено.

2. **Опишете разликата между TCP и UDP.**

   Разликата е в надеждността на TCP, който е по-бавен, но за сметка на това предава едно съобщение без загуби за разлика от UDP протоколът, който може да пропусне някои части от съобщението.

3. **Каква е ролята на IP протокола?**

   Ролята му е да адресира изпращача и получателя, така че да може пакетът да премине през цялата мрежа и да достигне до целта.

4. **IPv4 и IPv6 - какво е новото и защо искаме IPv6?**

   IPv4 са недостатъчни за броя устройства, които са налични.

5. **IPv4 класове срещу CIDR. Какво е новото и защо се е наложило?**

   За да се удължи живота на IPv4 е вкаран така наречената CIDR нотация, като идеята там е, че вече IPv4 адресите не се делят на класове, а по-скоро се делят на адрес и маска.

6. **Какво е TTL в IP?**

   Time To Leave пакетите в IP са едно число (брояч), което намалява с преминаването от node към node и когато стане 0 в някой node - пакетът се drop-ва. Основната цел на този флаг е била да не зациклят някакви пакети в мрежата завинаги. Например инструментът traceroot използва TTL.

7. **Какво представлява OSI моделът? Какво е TCP/IP модел?**

   Това е концептуален модел, който описва мрежата и я разделя на няколко слоя и тези слоеве се използват предимно с теоретична стойност, а TCP/IP е една малко по-опростена версия на този модел.

(2) HTTP PROTOCOL

1. **Обяснете статус кодовете на response-ите, които започват с 1xx, 2xx, 3xx, 4xx и 5xx?**

   - **1xx** - Информативни кодове, които например може да се използват за смяна на протокол или например статус 100 continue, което дава на клиента информация, че може да продължи с операцията която прави. С тази група кодове, сървърът потвърждава заявката, инициирана от браузъра (клиента), и че тя се обработва;

   - **2xx** - Успешни кодове. Получена, разбрана, обработена и очаквана информация, предадена на браузъра;

   - **3xx** - Кодове за пренасочване. Поисканият ресурс е заменен с друга дестинация. Може да са необходими допълнителни действия от браузъра;

   - **4xx** - Клиентска грешка. Уебсайтът или страницата не са достигнати. Страницата е недостъпна или е имало технически проблем със заявката;

   - **5xx** - Сървърна грешка. Заявката е приета но поради грешка на сървъра не се е изпълнила.

2. **Избройте някои по-често срещани HTTP статус кодове.**

   - **1xx**
     - 100 Continue. Всичко до тук е наред и клиентът трябва да продължи със заявката или да я игнорира, ако вече е завършена.
     - 102 Processing. Получена е пълната заявка и сървърът работи по нея.
     - 103 Early Hints. Изпраща се от сървъра, докато той все още подготвя желания отговор, като подсказки за ресурсите, които сървърът ще върне в окончателният отговор.

   - **2xx**
     - 200 OK. Заявката е изпълнена успешно;
     - 201 Created. Заявката е успешна и е довела до създаване на ресурс на сървъра;
     - 202 Accepted. Заявката е приета за обработка, но обработката не е завършена;
     - 203 Non-Authoritative Information. Заявката беше успешна, но приложеният полезен товар беше променен от трансформиращо прокси от първоначалния с отговор 200 (OK) на първоначалния сървър.
     - 204 No Content. Заявката е успешна и клиентът не е нужно да напуска текущата си страница. Много често се използва, когато сме изтрили някакъв ресурс на сървъра успешно.

   - **3xx**
     - 300 Multiple Choices. Заявката има повече от един възможен отговор. Потребителският агент или потребителят трябва да избере един от тях. Тъй като няма стандартизиран начин за избор на един от отговорите, този код на отговор се използва много рядко.
     - 301 Moved Permanently. Исканият ресурс е окончателно преместен към URL адреса, даден в location property-то от header-ите. Браузърът по подразбиране пренасочва към новия URL адрес и търсачките актуализират своите връзки към ресурса.

   - **4xx**
     - 400 Bad Request. Сървърът няма да обработи заявката поради нещо, което се възприема като клиентска грешка (например неправилен синтаксис на заявка, невалидно рамкиране на съобщение на заявка или измамно маршрутизиране на заявка).

- 401 Unauthorized. Код за състояние на неупълномощен отговор показва, че клиентската заявка не е завършена, защото липсват валидни идентификационни данни за заявения ресурс.
- 402 Payment Required. Код, който индикира, че исканото съдържание не е налично, докато клиентът не извърши плащане.
- 403 Forbidden (Unauthenticated). Сървърът разбира заявката, но отказва да я упълномощи.
- 404 Not Found. Сървърът не може да намери исканият ресурс. Връзките, които водят до страница 404, често се наричат прекъснати или мъртви връзки.
- 405 Method Not Allowed. Сървърът знае методът на заявката, но целевият ресурс не поддържа този метод.
- 406 Not Acceptable. Сървърът не може да произведе отговор, съответстващ на списъка с приемливи стойности, дефинирани в header-ите за проактивно договаряне на съдържанието на заявката, и че сървърът не желае да предостави представяне по подразбиране.
- 407 Proxy Authentication Required. Заявката не е приложена, тъй като липсват валидни идентификационни данни за прокси сървър, който е между браузъра и сървъра, който има достъп до искания ресурс.
- 409 Conflict. Конфликт на заявка с текущото състояние на целевия ресурс.
- 410 Gone. Достъпът до целевия ресурс вече не е наличен на първоначалния сървър и това състояние вероятно ще бъде постоянно.
- 415 Unsupported Media Type. Сървърът отказва да приеме заявката, тъй като форматът на полезния товар е в неподдържан формат.
- 417 Expectation Failed. Очакването, дадено в полето Expect от header-а, не може да бъде изпълнено.
- 428 Precondition Required. Сървърът изисква заявката да бъде условна.
- 429 Too Many Requests. Потребителят е изпратил твърде много заявки за даден период от време („ограничаване на скоростта“).

- **5xx**
  - 500 Internal Server Error. Сървърът е срещнал неочаквано състояние, което му е попречило да изпълни заявката.
  - 501 Not Implemented. Сървърът не поддържа функционалността, необходима за изпълнение на заявката.
  - 502 Bad Getaway. Сървърът докато е действал като свръзка или като прокси, е получил невалиден отговор от сървъра нагоре по веригата.
  - 503 Service Unavailable. Сървърът не е готов да обработи заявката.
  - 504 Gateway Timeout. Сървърът докато е действал като свръзка или като прокси, е не е успял да получи навременен отговор от сървъра нагоре по веригата, за да обработи своя резултат, който да върне
  - 511 Network Authentication Required. Изисква се мрежова автентикация за да се получи достъп до мрежата.

3. **Какво е HTTP/2 и с какво подобрява HTTP/1.1?**

HTTP/2 е следващо поколение протокол след HTTP/1, който е напълно съвместим с него. HTTP/2 подобрява HTTP/1.1 чрез въвеждането на:

- **Multiplexing** - възможността да се пратят паралелно много заявки по една единствена връзка (по една единствена TCP конекция). Това отстранява overhead-а при създаването на множество TCP конекции със сървъра.
- **Server push** - позволява на сървъра да връща проактивно (без да са поискани от потребителя) множество файлове наведнъж.

4. **Какви недостатъци има HTTP/2?**

Недостатъкът на HTTP/2 е че този протокол все още лежи и разчита на TCP протокола, което налага появата на HTTP/3. TCP протокола има недостатъци като :

- голяма тежест за клиента и сървъра и съответно по-бавно тръгва;

- congestion control алгоритъма е вътре в него и много трудно може да се експериментира с него;
- Three-Way-Hand-Shake, който бави допълнително и разхищава мрежата допълнително като се използва с TLS;
- Проблемна смяна на мрежите.

5. **Какво представлява HTTP/3?**

   HTTP/3 е нов стандарт в разработка, който ще повлияе на начина, по който уеб браузърите и сървърите комуникират, със значителни подобрения за потребителското изживяване, включително производителност, надеждност и сигурност. Някои от нещата, които ще се подобрят са:

   - По-бързо установяване на връзка: алгоритъмът QUIC позволява договарянето на TLS версията да се случи едновременно с криптографските и транспортни ръкостискания в слоя на приложението;
   - Нулево време за двупосочно пътуване (0-RTT): За сървъри, към които вече са се свързали, клиентите могат да пропуснат изискването за ръкостискане (процесът на взаимно потвърждаване и проверка, за да се определи как ще комуникират);
   - По-всеобхватно криптиране: Новият подход на QUIC към ръкостисканията ще осигури криптиране по подразбиране - огромно надграждане от HTTP/2 - и ще помогне за намаляване на риска от атаки.

3. **Как работи TLS/SSL?**

   Това е криптиран канал от 6-то ниво (Presentation) на OSI модела. Обменя се ключ между клиента и сървъра, който се използва за криптиране на връзката, за да не може трето лице да се намеси по средата и да злоупотреби с информацията, която се предава.

4. **Какви HTTP глаголи/операции познавате?**

   GET - за извличане на ресурс;
   POST - за създаване на ресурс;
   PUT - за обновяване на ресурс;
   PATCH - за частично обновяване на ресурс;
   DELETE - за изтриване на ресурс;
   OPTIONS - за връщане на някаква информация, като например какви други методи поддържа приложението.

5. **Опишете структурата на едно URL .**

   Унифициран локатор на ресурси (URL) – стандартизиран адрес на даден мрежов ресурс (документ или страница)

   http://en.wikipedia.org:80/w/index.php?search=Students&dd=true#last

   схема, FQDN или хост, порт, път, низ на заявката, фрагмент идентификатор (anchor)

6. **Каква е разликата между URI и URL?**

   URI - Uniform Resource Identifier; **URL** - Uniform Resource Locator.
   URL-ите са подмножество на URI-те и съдържат в себе си механизъм за вземане на ресурс.

7. **Каква е разликата между GET/PUT/POST/PATCH HTTP методите?**

   GET - достъп до ресурс, PUT - цялостно обновяване на ресурс, POST - създаване на

ресурс и PATCH - частична актуализация на ресурс.

8. **Какво представлява моделът POST-redirect-GET и защо се използва?**

   Когато направим POST заявка за създаване на някакъв ресурс, той се създава на сървъра и обикновено не е много удачно да върнем директно response-а, тъй като при refresh на страницата - браузърът ще се опита отново да създаде ресурса. За това е удачно да върнем статус REDIRECT и да пренасочим към URL-а на новия request и браузърът да направи GET заявка.

(3) SPRING BOOT

1. **Какво е IoC и DI? Какво е Spring IoC контейнер?**

   Inversion of Control (IoC) имаме, когато някакъв framework изземва контрола от нас, като например когато създаваме някакъв обект и в конструктора само инициализираме зависимостите. Dependency Injection е една от имплементациите на IoC. Друга такава имплементация за IoC е например Template Method дизайн шаблона.
   Spring IoC контейнерът е нещото което осъществява DI в Spring. Spring Bean-овете работят в имплементациите на Application Context.

2. **Какво е Spring Boot?**

   Spring Boot е начин, по който може да си конфигурираме нашето Spring приложение с помощта на удобен dependency management, auto configuration и т.н. Както и името му подсказва - да си BOOTstrap-нем нашето Spring приложение.

3. **Има ли разлика между Spring Boot и Spring MVC?**

   Да. Spring MVC е така наречения Model-View-Controller framework. С помощта на Spring MVC може да правим приложения в Servlet stack-а и там да създаваме Controller-и да връщаме response-и и т.н. Докато Spring Boot е начин, по който може да си set-up-нем някакво приложение в това число и Spring MVC приложение.

4. **Какво е Spring Boot Starter?**

   Това е едно dependency, което влачи след себе си други dependency-та. Идеята е когато създаваме някакво Spring приложение, тези стартери да обединяват зависимостите между различните dependency-та, тъй като всяко dependency разчита на различна (по някой път и конкретна) версия на друго dependency и ние няма как да помним всичката тази информация на изуст, а и не е нужно, тъй като стартерите правят това вместо нас. Това е една голяма кутия с компоненти или обединение на различни компоненти.

5. **Какво е Spring Boot Auto-configuration?**

   Auto-configuration са един от основните механизми, с които Spring Boot конфигурира нашето приложение. Обикновено в една auto-configuration се крие някакъв клас, който е анотиран с @Autoconfiguration, която от своя страна е анотирана с @Configuration анотация. В зависимост от определени условия (например дали някой Bean е наличен в Application Context-а или не), ние може да expose-ваме нови Bean-ове, които променят целия Application Context и вкарват нови конфигурирани играчи вътре. Auto-configuration е един вид възможност да се даде така наречената opinionated конфигурация. Тоест нещо конфигурирано по такъв начин, както създателите на самата Auto-configuration или на самия Spring Boot смятат за удобно. Но идеята е също така да всеки да има възможност да преконфигурира както той смята за удобно.

6. **Какво е Spring Boot Actuator?**

   Actuator-а е проект, който ако го използваме/добавим към нашето приложение ще ни даде така наречените non-functional features. Това са feature-и, които не са свързани непосредствено с нашия проект. Например, ако правим web-shop, то един functional feature ще представлява добавянето на продукт в количката, докато non-functional feature ще е например някаква метрика като размер на паметта, която се използва или друго като например liveness и health-check проби, колко Bean-а се използват в Application Context-а и т.н. Освен това ние сами може да си създадем свой собствени Spring Boot Actuator end point-ове.

7. **Как може да създадем един Spring Boot проект?**

   По най различни начини, като един от най-популярните е да използваме Spring Initializr, който е достъпен като отделен application и в web формат. Има го вградено и в IntelliJ Ultimate версията. Може да го създадем и чрез Spring CLI (app, който се използва от командния ред).

8. **Дайте примери за няколко Spring Boot Starter-а.**

   - spring-boot-starter-data-jpa
   - spring-boot -starter-thymeleaf
   - spring-boot-starter-web
   - spring-boot-starter-validation
   - spring-boot-starter-test
   - spring-boot-starter-actuator
   - spring-boot-starter-security

9. **Какво представляват Spring Beans?**

   Това са различни обекти, между които общото е, че се менажират от Spring. Те живеят в така наречения Application Context. Spring ги инициализира, менажира и унищожава. Целия жизнен цикъл на тези обекти се управлява от Spring. Казано с други думи - не използваме оператора new когато искаме да създадем даден Bean, а вместо това, за да получим достъп до тях използваме IoC и DI (Inversion of Control & Dependency Injection).

10. **За какво служи @PostConstruct анотацията?**

    @PostConstruct анотация се слага върху метод, който трябва да бъде извикан след като е инициализиран един Spring Bean.

11. **За какво служи PreDestroy анотацията?**

    PreDestroy анотация се слага върху метод, които се извиква от контейнера, когато се унищожава някак Bean.

12. **Има ли разлика между предназначението на application.properties и application.yml файловете.**

    Не. И двата файла са с еднакво предназначение, но структурата на информацията в тях е различна. Форматът в yml или yaml файловете е по-удобен и по-четим за хора, поради йерархичната си структура, която също така и редуцира текста във файла, но разчита на отстъпа (indent-а) на редовете.

13. **Какви Bean scopes познавате? Дайте примери и посочете какви са разликите между тях.**

    Основните Bean scope-ове са "singleton" и "prototype". Singleton се инициализира един път и играе ролята на singleton в application context-а. Prototype се създава всеки път когато има нужда се използва. Другите scope-ове са "request", "session", "websocket" и други custom scope-ове.

14. **Само на Java ли се пишат Spring приложения?**

    Не. Пишат се и на други JVM-based езици като Kotlin, Scala, Groovy, Clojure, Fantom. Ceylon, Jython.

(4) Spring MVC

1. **Какво е MVC?**

   Съкращението идва от Model-View-Controller и представлява един Design Pattern. В този дизайн шаблон имаме един клас, който е контролер и приема някакви команди, създава модел и като резултат се връща едно view. В MVC: Controller-а е класът, който обработва заявката, Model-а е клас който замества (наследява в случая на ModelMap) HashMap, а View-то е например Thymeleaf View, което представлява обект, в който се попълват някакви стойности, които се визуализират накрая под формата на html, json и т.н.

2. **Какво разбирате под "Frontend Controller"?**

   Frontend Controller в едно Web приложение е класът, който приема всички заявки, които идват към дадения Web App.

3. **Какво е DispatcherServlet?**

   DispatcherServlet е т.н. Frontend Controller в SpringMVC и през него минават всичките MVC заявки. Съответния handler за заявката, който я изпълнява се намира в DispatcherServlet-а.

4. **Как може да предадем атрибути към едно view?**

   Атрибутите във view-то могат да се инжектират вътре в модела, като всеки атрибут има име и стойност. С помощта на името може да получим достъп до стойността в един Thymeleaf Template например.

5. **Какви слоеве има в типичния Spring Application?**

   В един Spring App е прието да имаме Controller-и (или Web слой), където се намират класовете, които обработват Http заявката. След това имаме Service слой, с които Controller-ите работят и където се намира бизнес логиката на приложението. След това имаме Repository-та, с които Service-ите работят и чрез които се осъществяват достъп до обектите в базата данни. Самите обекти в базата данни представляват Entity-та, които са POJO класове, анотирани по подходящ начин, така че Hibernate да може да работи с тях. Също така имаме и View-та, които са крайния продукт на нашите Controller-и.

6. **Как може да реализираме обработката на GET/POST и други подобни заявки в контролер?**

   За тази цел има подходящи анотации например @GetMappin и @PostMapping и съответно за други Http методи има подобни анотации. Освен това може да го реализираме с @RequestMapping анотация и подаване на подходящия метод вътре в тялото ѝ. Трети начин за реализация е чрез конфигурация в клас анотиран с @Configuration, където в метод анотиран с @Bean връщаме типизиран обект **RouterFunction<ServerResponse>** и му подаваме TestHandlers.

7. **Какво ще стане ако изпратим POST заявка към endpoint, който може да обработва само GET заявки?**

   Spring автоматично ще върне грешка със статус код 405 (Method not allowed).

8. **Как анотираме контролер класовете?**

   Анотираме ги с @Controller анотацията, а когато имплементираме REST API използваме

@RestController, което е @Controller + @ResponseBody.

9. **Как използваме @RequestMapping анотацията?**

   Използваме я на ниво клас и ни дава префикс на всички методи в класа на контролера. @RequestMapping може да се използва и на ниво метод, като обикновено в такъв случай се задава освен пътя и метода (GET, POST, etc.).

10. **Има ли разлики между @GetMapping и @RequestMapping с метод GET върху метод?**

    Не, няма. Едно и също нещо е. @GetMapping е alias път към @RequestMapping с метод GET.
    Следните са еквивалентни:
    @RequestMapping(value = "/ex/foo", headers = "key=val", method = GET)
    и
    @GetMapping(value = "/ex/foo", headers = "key=val")

11. **Как може да вземем POST параметър?**

    С помощта на DTO binding model и с @RequestParam анотацията.

12. **Как предаваме обект към Thymeleaf view?**

    С помощта на обектите, които Spring ни позволява да инжектираме в методите на контролерите. Това са Model, ModelMap и ModelAndView.

13. **Как може да вземем optional параметър?**

    В @RequestParam анотацията има параметър required, който е с булева стойност и приема true или false.

14. **Как може да зададем (default) стойност по подразбиране на параметър?**

    В @RequestParam анотацията има параметър defaultValue, който приема стойността по подразбиране като String.

15. **Каква е разликата между Path variable и Query parameter?**

    Path variable е променлива, която се кодира в пътя на URL-а (localhost:8080/orders/3/details), а Query parameter-а е променлива, която се кодира в query string-а (localhost:8080/orders?id=3)

(5) State Management

1. **Какво означава Cookie?**

   Cookie-то е малък текстов файл, който се създава от сървъра (а могат да се създадат и чрез JavaScript) и се изпраща от сървъра към клиента (например уеб браузър). Този текстов файл не съдържа изпълним код, а ключ-стойност параметри и браузърът го записва локално. Cookie-тата съществуват, за да се преодолее факта, че Http протокола е stateless (не пази състояние, т.е. сървърът не прави разлика между два отделни Http Request-а - от къде са дошли и кой е потребителят, който стой зад тях). Това е твърде ограничаващо и за това в началото на 90-те години се измислят т.нар. cookie-та.

2. **Какво е Scope на едно Cookie?**

   От къде е дошло cookie-то и кой сървър би могъл да го получи при Http Request.

3. **Какво е съдържанието на едно Cookie?**

   Основното нещо което се намира в едно cookie са key-value-pair стойности. Освен ключ и стойност двойки, cookie-то съдържа и т.нар. атрибути. Атрибутите например биха могли да са: secure (т.е. cookie-то да бъде изпратено по HTTPS, когато имаме някакъв TLS), http-only (т.е. cookie-то не може да бъде пипано от JavaScript), same-site (атрибут, който определя кога да се изпраща едно cookie при заявка към сървъра, например ако ползваме директен линк към браузъра или браузвате директно в сайта), expires (определя времеви интервал до когато е валидно cookie-то)

4. **За какво служат Cookies?**

   Служат за персонализация, за user tracking и най-често се използват за session management. В Cookie се държи SESSION-ID-то, което отговаря на HTTP сесията, която се пази на сървъра.

5. **Какво е 3rd Party Cookie?**

   3rd party cookie-тата се използват за user tracking и за реклами. Когато отворим сайт A и там се изпълнява скрипт от сайт B и той сложи свое cookie там. Тоест отварям сайт A, а всъщност се появява cookie от сайт B на нашия браузър. Когато отвориме сайт C, отново се зарежда cookie-то от сайт B и сайт C може да прочете cookie-то на база на него да препоръча нещо специфично (сайт C ще знае къде и какво сме предпочитали, което е в разрез с нарушаване на личните данни и за това тези third party cookies ще се използват до края на 2023).

6. **Какво представлява HTTP сесията? Как се създава и унищожава?**

   Това е една област в паметта на сървъра (или в базата данни, до която има достъп сървъра), където се пази информация за даден user. Може да слагаме най-различни неща в сесията, но тя обикновено е свързана с някакъв идентификационен номер, който се предава между клиента и сървъра, като най-често това става чрез cookie. Http сесия може да се инжектира от Spring IoC контейнера и може да се унищожи като се инвалидира чрез метода ѝ invalidate( ).

7. **С помощта на кои класове и анотации работим със сесии в Spring MVC?**

   Когато имаме някакъв контролен бихме могли да обявим аргумент, който е от тип HttpSession, като инстанция на HttpSession се инжектира от Spring IoC контейнера.

8. **С помощта на кои класове и анотации работим със cookies в Spring MVC?**

ResponseCookie - клас, който дава възможност да използваме build pattern-а, за да направим cookie, което по-късно може да се set-не в HttpResponse-а. Стойността на cookie може да прочетем чрез анотацията @CookieValue.

9. **Как може в Spring да добавим атрибут в HTTP Session?**

   Като инжектираме HttpSession обекта като аргумент и го използваме за да му добавим атрибут с метода му addAttribute.

10. **Как може сървърът да изтрие едно Cookie?**

    Като сетне същото cookie със същото име, със същите property-та но със max-age=0, което ще доведе до изтриването на cookie-то.

(6) Spring Essentials

1. **Какво представлява Thymeleaf Engine?**

   Thymeleaf е модерна сървърна машина (програма) за шаблони на Java както за уеб, така и за самостоятелни среди. Основната цел на Thymeleaf е да внесе елегантни шаблони в процеса на разработка - HTML, който може да се показва правилно в браузърите и също така да работи като статични прототипи, което позволява по-силно сътрудничество в екипите за разработка.

   С модули за Spring Framework, множество интеграции с любимите ви инструменти и възможност за включване на ваша собствена функционалност, Thymeleaf е идеален за съвременната HTML5 JVM уеб разработка – въпреки че може да направи много повече.

2. **Какви алтернативи на Thymeleaf познавате?**

   JSP, Groovy Templates.

3. **Какво добавя Spring към Thymeleaf?**

   Thymeleaf engine-а е много лесен да бъде extend-ван и за това Spring се възползва от това нещо и добавя някой неща като например да се parse-ват някакви SpEL expressions, както и някой екстри като например th:field за интеграция със Spring MVC и т.н.

4. **Как се реализират if-else statement в Thymeleaf?**

   Това става с помощта на th:if и th:unless, еквивалентни на програмните if и else съответно.

5. **Как се реализират цикли в Thymeleaf?**

   Това става с помощта на th:each поставен на някой елемент, който искаме да се повтаря.

6. **Какво е elvis оператор и поддържа ли се от Thymeleaf?**

   Да, поддържа се такъв оператор и целта му е да провери дали една стойност е различна от null и ако да - да върне същата стойност, а ако не да върне нещо друго.

7. **Как се изкарва повтаряем markup в Thymeleaf?**

   Чрез фрагменти.

8. **Какво представляват Thymeleaf фрагментите?**

   HTML фрагменти за повторно използване на някои общи части на сайт.

9. **Защо използваме link expressions?**

   За да свържем адрес с даден HTML препратка. Релативен път се свързва с контекста на нашия уеб сървър. Можем лесно да навигираме през нашите шаблонни файлове. Абсолютен път се свърза със посочения ресурс дори и извън нашия сървър.

10. **Обяснете за какво служи @Primary анотацията?**

    @Primary за задаване на приоритет на Bean, когато има множество Bean-ове от един и същ тип. Когато Bean е анотиран с @Primary, той ще се инстанцира, ако не е зададен

уточняваш @Qualifier на инстанцирания обект от еквивалентен тип.

11. **Какво е Same Origin и CORS?**

    Политиката за същия произход е критичен механизъм за сигурност, който ограничава как документ или скрипт, зареден от един източник, може да взаимодейства с ресурс от друг произход.
    CORS е механизъм за сигурност, който позволява на уеб страница от един домейн или Origin да има достъп до ресурс с различен домейн (заявка между домейни). CORS е облекчаване на политиката за същия произход, внедрена в съвременните браузъри.

12. **С коя анотация управляваме CORS?**

    @CrossOrigin

13. **Дайте пример за употреба на CORS.**

```
@CrossOrigin
@GetMapping("/{id}")
public Account retrieve(@PathVariable Long id) {
    ...
}
```

14. **За какво служи @ModelAttribute анотацията върху метод?**

    Създава Bean на ниво контролер, който може да се достъпва в Thymeleaf.

(7) Thymeleaf and Validations

1. **За какво служи стартера spring-boot-starter-validation?**

   В този стартер има две основни неща. Първото е dependency към javax/jakarta validation API, който съдържа анотации като @Size, @NotNull, @NotBlank и т.н. Този API обаче не е достатъчен за да сработят нещата, а е необходимо да имаме някаква имплементация на валидатор и по подразбиране Spring използва hibernate валидатора, който може да разпознава тези анотации, които използваме и съответно когато поставим @Valid на някой аргумент – да стартира валидацията.

2. **Как се използва анотацията @Valid?**

   Анотацията @Valid може да се сложи на аргумент в нашия контролер. Когато Spring види, че има такава анотация, той взима имплементацията и му дава BindingModel-а, който е попълнен, за да извърши някаква валидация. Тази валидация се основава на анотациите, които слагаме в модела, като @NotNull, @NotEmpty, @NotBlank и т.н. както и custom валидаторите.

3. **Как се имплементира custom валидатор?**

   За да имплементираме custom валидатор е необходимо да имаме анотация, която е анотирана с мета-анотация @Constraint и вътре в тази мета-анотация да дадем самия клас, който върши работата по самата валидация. Тоест, дефинираме анотация и я анотираме с мета-анотация @Constraint, даваме класа на валидатора, който прави истинската валидация и след това ползваме този custom валидатор като анотираме в нашите BindingModel-и field-ове или целия клас (ако на валидацията са и необходими няколко field-а от класа, който валидира).

4. **За какво се използва BindingResult?**

   BindingResult-а може да се използва като контейнер за грешките, които са настъпили в процеса на binding/валидация. Съответно може да видим какви грешки са настъпили докато е текла валидацията. Например имаме атрибут, който трябва да е @NotNull, а от клиента не е пристигнало нищо като резултат за този атрибут и в BindiungResult се съдържа грешката.

5. **Какво ще се случи, ако имаме аргумент на метод анотиран с @Valid и следващия аргумент НЕ Е от тип BindingResult?**

   Ще се случи грешка със статус код 400, която се връща на клиента. Тази грешка означава invalid client request, т.е. сървъра сигнализира на клиента че request-а му не е валиден.

6. **Обяснете за какво служи POST-redirect-GET в контекста на валидацията.**

   Идеята е, че когато имаме или нямаме грешка при валидацията при POST-ване на нещо за да създадем обект, ако върнем потребителя на същата страница и потребителя натисне refresh бутон, то същите стойности ще бъдат submit-нати отново, което не е желано поведение.