

## 1 Билет

### 1. Java EE CDI Beans: именованние бинов и @Alternative

Универсальные компоненты уровня бизнес логики. Общая идея – отвязаться от конкретного фреймворка при создании бизнес-логики внутри приложения. CDI бины поддерживают внедрение зависимостей. Жизненным циклом бинов управляет CDI контейнер. Реализация происходит через аннотации: @Named("name"); указывается скоуп (@SessionScoped). Обычно, в качестве типа внедряемого объекта используется интерфейс, а CDI контейнер сам определяет, какую реализацию нужно выбрать. Этот интерфейс говорит что у нас есть какой-то механизм, который выполняет ожидаемое действие, при этом как он это делает нас не интересует. Нам достаточно знать интерфейс, а о реализации уже думает за нас CDI контейнер. Возникает ситуация, когда появляется несколько реализаций одного интерфейса и чтобы не возникало ошибки, нам нужно пометить альтернативные реализации аннотацией @Alternative. Это значит, что пока мы явно не скажем, что нужно использовать именно эту альтернативу, она не будет выбрана. Так же надо добавить альтернативу в beans.xml и использование будет следующее `initializer.selectAlternatives()`. @Priority - приоритетность альтернатив.

### 2. Angular DI

Dependency Injection - паттерн проектирования, который позволяет создавать объект, используя другие объекты. При этом поля объекта настраиваются внешней сущностью. В компоненты внедряют сервисы, в которых реализуется бизнес логика, не связанная с представлением. Например логирование, общение с API. (с помощью DI легче тестировать). Для того, чтобы класс можно было использовать с помощью DI, он должен содержать декоратор @Injectable(). Зависимости компонентов указываются в качестве параметров их конструкторов. Принципы: приложение содержит хотя бы один глобальный инжектор (root) который занимается DI; Injector создает зависимости и передает их экземпляры контейнеру; Provider - объект, который сообщает Injectory как получить или создать экземпляр зависимости; Обычно провайдер сервиса - сам его класс;

### 3. Конфигурация Spring Web MVC для запуска на сервере приложений без использования Spring Boot. Описать web.xml, необходимый для запуска такого приложения

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                             http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0">
  <servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>/WEB-INF/applicationContext.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</web-app>
```

## 2 Билет

### 1. Концепция Location Transparency. Реализация в Java/Jakarta EE

Возникает из CDI и JNDI. означает, что при использовании CDI механики мы можем добиться того, что нам станет не важно, где физически находится компонент, к которому мы обращаемся. С помощью LocTransp мы можем одинаково обращаться как к локальному объекту, так и удаленному. Получением удаленного объекта занимается сервер приложений. RMI – API в Java для вызова методов удаленных объектов и помогает реализовать принцип Location Transparency. Позволяет Client Host вызвать метод, находящийся на Server Host так, будто он локальный. У серверного объекта, чтобы его можно было вызвать, должен быть публичный интерфейс. Stub - Заглушка, Skeleton - каркас. Server Stub - имплементация серверного интерфейса, которая живет на клиентской машине и делает вид, что она и есть серверный объект. При этом серверной логики внутри Stub нет. Запросы, полученные Stub, Stub отправляет на сервер в Server Skeleton, который вызывает методы уже физического объекта. После Skeleton возвращает результат работы метода Stub, а тот передает его клиенту в месте вызова. За физический удаленный вызов отвечают Server Stub и Server Skeleton. Регистрируем серверный

объект в RMI Registry; Поиск объекта сервера ; Возвращение серверного Stub клиенту ; Обмен данными

## 2. Реализация уровня представления в Spring Web MVC

Фреймворк не специфицирует жёстко технологию, на которой должно быть построено представление. По-умолчанию – JSP - технология, позволяющая создавать динамические веб-страницы с помощью Java. Можно использовать Thymeleaf, FreeMarker, Velocity etc. Можно реализовать представление вне контекста Spring – целиком на JS. Представление отвечает за то, как будут визуализироваться данные в браузере пользователя. За поиск представления по имени отвечает ViewResolver - интерфейс, при помощи которого DispatcherServlet определяет какое представление нужно использовать на основании имени. Принципы работы: Контроллеры обрабатывают запросы, подготавливают данные и помещают их в модель, которая передается в представление. В представлениях используется механизм привязки данных для доступа к информации из модели и ее отображения. После обработки данных из модели, представление формирует ответ пользователю в нужном формате, будь то HTML-страница, JSON-объект или другой тип данных.

**3. React компонент, реализующий "карусель"(вбросов) изображений: отображается 1 изображение из n. Можно переключать кнопками "вперед"и "назад". Изображения хранятся на клиентском устройстве.**

```
const Carousel = () => {
  const [currentImageIndex, setCurrentImageIndex] = useState(0);
  const images = [
    '/images/image1.jpg'...
  ];
  const prevImage = () => {
    setCurrentImageIndex((prevIndex) =>
      prevIndex === 0 ? images.length - 1 : prevIndex - 1);};
  const nextImage = () => {
    setCurrentImageIndex((prevIndex) =>
      prevIndex === images.length - 1 ? 0 : prevIndex + 1);};
  return (
    <div>
      <img src={images[currentImageIndex]} alt={`Image ${currentImageIndex + 1}`} />
      <div> <button onClick={prevImage}>back</button>
        <button onClick={nextImage}>next</button> </div>
    </div>);};
```

## 3 Билет

### 1. Технология RMI. Использование RMI в Java EE

Система RMI позволяет объекту, запущенному на одной виртуальной машине Java, вызывать методы объекта, запущенного на другой виртуальной машине Java. Работает поверх TCP. В общем случае, объекты передаются по значению, передаваемые объекты должны быть Serializable. RMI основана на более ранней технологии удаленного вызова процедур Remote Procedure Call (RPC). Использование: Серверное приложение, как правило, создает удаленные объекты (remote objects), делает доступные ссылки на эти объекты и находится в ожидании вызова методов этих объектов. Клиентское приложение получает у сервера ссылку на удаленные объекты, после чего вызывает его методы. Технология RMI, обеспечивающая механизм взаимодействия клиента и сервера передачей между ними соответствующей информацией, реализована в виде java.rmi пакета, содержащего целый ряд вложенных подпакетов; один из наиболее важных подпакетов java.rmi.server реализует функции сервера RMI. RMI обеспечивает процесс преобразования информации данных по сети и позволяет java приложениям передавать объекты с помощью механизма сериализации объектов.

### 2. Управление состоянием в React. Redux

Хук useState() предназначен для управления локальным состоянием компонента. Хук «useContext()» позволяет извлекать значения из контекста в любом компоненте, обернутом в провайдер (provider). Flux — архитектура для создания приложений на React, в которой описывается, как хранить, изменять и отображать глобальные данные. Основные концепции: Dispatcher принимает события от представления и отправляет их на обработку хранилищу данных. Store знает, как менять данные. Напрямую из React-компонента их изменить нельзя. После изменения данных Store посылает события представлению, и оно перерисовывается. Redux — небольшая библиотека, реализующая упрощенный паттерн Flux. В Redux есть store — синглтон, хранилище состояние всего приложения. Изменения состояния производятся при помощи чистых функций. Они принимают на вход state и действия и возвращают либо неизмененный state либо копию.

**3. Написать конфигурацию Application Context без Spring Boot, чтобы он выполнял поиск бинов в org.itmo.web, при этом исключая бины, которые реализуют с класс RubezhkaPassService**

```
@Configuration
@ComponentScan(basePackages = "org.itmo.web", excludeFilters =
  ↳ @ComponentScan.Filter(type = FilterType.ASSIGNABLE_TYPE, classes =
  ↳ RubezhkaPassService.class))
public class AppConfig {
  @Bean
```

```

    public SomeBean someBean() {
        return new SomeBean("Hello from SomeBean!");
    }
}
public class Main {
    public static void main(String[] args) {
        AnnotationConfigApplicationContext context = new
            ↪ AnnotationConfigApplicationContext();
        context.register(AppConfig.class);
        context.refresh();
    }
}

```

## 4 Билет

### 1. Управляемые бины (Managed bean) - назначение, способы конфигурации, доступ из jsp/xhtmll

Managed beans - обычные JAVA классы управляемые JSF. Хранят состояние JSF-приложения. Содержат параметры и методы для обработки данных, получаемых из компонентов. Занимаются обработкой событий. Настройка происходит в faces-config.xml или при помощи аннотаций (@ManagedBean). В faces-config.xml конфигурация происходит следующим образом: указывается имя, класс бина, скоуп У Managed Beans есть скоуп – время, в которое бин будет создан и будет доступен. Скоупы: NoneScoped - жизненным циклом управля- ют другие бины; RequestScoped - контекст - запрос; ViewScoped - контекст-страница(компонент создается один раз при обращении к странице); SessionScoped - контекст - сессия; ApplicationScoped - контекст - приложение; CustomScoped - компонент создается и сохраняется в коллекции типа Map. Областью жизни управляет программист. В JSF обращаться к ManagedBean можно через EL:#{myBean.property}

### 2. Архитектура и состав Spring Web MVC

Model инкапсулирует данные приложения для формирования представления. View формирует HTML страницу. Фр-эймворк не специфицирует жестко технологию на которой будет построено представление. Можно использовать Thymyleaf, Freemaker, реализовывать представление вне спринга на JS. Controller обрабатывает запрос пользователя, связывает модель с представлением, управляет состоянием модели. В шаблоне мы можем читать свойства модели и отображать их на странице. Класс и его методы могут быть помечены аннотациями привязывающими его к HTTP методам или URL. DispatcherServlet - сервлет, который принимает все запросы и передает управление контроллерам, написанными программистом. HandlerMapper — интерфейс для поиска подходящего контроллера. Контроллер — класс с аннотацией @Controller, который занимается обработкой запросов. В нем реализуется некая бизнес логика для подготовки данных. ViewResolver — интерфейс для поиска подходящего представление.

**3. Интерфейс на angular, проверяющий, аутентифицирован ли пользователь(по наличию куки jsessionid), и, если нет, позволяющий ему аутентифицироваться посредством ввода логина и пароля.**

```

@Component({
    selector: 'app-login',
    template: `
        <form (ngSubmit)="login()">
            <input type="text" placeholder="login" [(ngModel)]="username">
            <input type="password" placeholder="pass" [(ngModel)]="password">
            <button type="submit">log</button>
        </form>`
})
export class LoginComponent {
    username: string = '';
    password: string = '';
    constructor() {}
    login(): void {
        // request on server for auth user
    }
}
@Component({
    selector: 'app-root',
    template: '<router-outlet></router-outlet>'
})
export class AppComponent implements OnInit {
    constructor(private router: Router) {}
    ngOnInit(): void {
        if (!document.cookie.split(';').some(cookie =>
            ↪ cookie.trim().startsWith('jsessionid='))) {

```

```

        this.router.navigate(['/login']);
    }
}
}

```

## 5 Билет

### 1. Область видимости управляемых бинов. Способы конфигурации управляемых бинов

Контекст определяет, к чему будет привязан бин и его время жизни. Конфигурировать можно через аннотации (@ManagedBean(name = "myBean") @RequestScoped), либо через faces-config.xml: <managed-bean-name>myBean</managed-bean-name> <managed-bean-class>com.example.MyBean</managed-bean-class> <managed-bean-scope>request</managed-bean-scope>. NoneScoped - контекст не определён, жизненным циклом управляют другие бины; по умолчанию RequestScoped - контекст - запрос; ViewScoped - контекст-страница(компонент создается один раз при обращении к странице, и используется ровно столько, сколько пользователь находится на странице); SessionScoped - контекст - сессия; ApplicationScoped - контекст - приложение; CustomScoped - компонент создается и сохраняется в коллекции типа Map. Областью жизни управляет программист.

### 2. Принципиальные отличия подходов, реализуемых Spring и Java/Jakarta EE

Java EE — для легко масштабируемого монолитного приложения, Spring — для совсем маленьких приложений с GUI на Front-end или для микросервисной архитектуры. Java EE может работать в общем случае только в рамках Enterprise Application Server'a, а приложение на Spring стеке может работать на чем угодно. Spring beans — обычные джава бины. EJB — достаточно мощная штука, в которую встроена поддержка распределенного исполнения, включая распределенный сборщик мусора, аутентификацию, поддержку транзакций. Java EE: преимущественно основан на аннотациях и CDI; JFC MVC framework для веб-разработки; JPA спецификация для обработки операции DB; JTA API с внедрением; EJB контейнерная и POJO основанная реализация; Oracle лицензия. Spring: На основе IOC и AOP; Использует фреймворк Spring DAO для подключения к базе данных; Предоставляет уровень абстракции для поддержки различных реализаций JTA; Лицензия с открытым исходным кодом

### 3. Компонент для React, формирующий строку с автодополнением. Массив значений для автодополнения должен получаться с сервера посредством запроса к REST API

```

const AutoComplete = () => {
  const [inputValue, setInputValue] = useState('');
  const [suggestions, setSuggestions] = useState([]);
  useEffect(() => {
    const fetchData = async () => {
      try {
        const response = await axios.get('REST_API_ENDPOINT');
        setSuggestions(response.data);
      } catch (error) {
        console.error(error);
      }
    };
    fetchData();
  }, []);
  const handleInputChange = (e) => {
    setInputValue(e.target.value);
  };
  return (
    <div>
      <input
        type="text"
        value={inputValue}
        onChange={handleInputChange}/>
      <ul>
        {suggestions.filter((suggestion) =>
          suggestion.toLowerCase().includes(inputValue.toLowerCase()))
          .map((suggestion, index) => (<li key={index}>{suggestion}</li>))}
      </ul>
    </div>
  );
};
export default AutoComplete;

```

## 6 Билет

### 1. Фазы Invoke Application и Render Response

При поступлении запроса необходимо выполнить определенную цепочку действий, чтобы проанализировать запрос и подготовить ответ. За программиста это делает фреймворк(JSF) Invoke Application Phase: Управление передаётся слушателям событий, JSF обрабатывает события также решается вопрос навигации; Формируются новые значения компонентов. (Вызывается метод `UIViewRoot#processApplication()` для обработки событий.) Render Response Phase: JSF Runtime обновляет представление в соответствии с результатами обработки запроса; Если это первый запрос к странице, то компоненты помещаются в иерархию представления; Формируется ответ сервера на запрос; На стороне клиента происходит обновление страницы.

## 2. Какие задачи решает Spring Boot

Spring Boot предоставляет разработчикам программного обеспечения гибкую настройку, надежную пакетную обработку, эффективный рабочий процесс и большое количество инструментов, помогая разрабатывать надежные и масштабируемые приложения на базе Spring. Он обладает автоконфигурацией, что позволяет ускорить настройку конфигурации. Включает в себя встроенные HTTP-серверы Jetty и Tomcat для тестирования веб-приложений. Позволяет легко подключаться к базам данных и сервисам очередей, таким как Oracle, PostgreSQL, MySQL, Rabbit MQ и многим другим. Spring Boot оснащен встроенным контейнером сервлетов. Это позволяет избавиться от потребности в его отдельной настройке и развертывании на нем приложения. Благодаря встроенной в Spring Boot поддержке контейнеров сервлетов, разработанные решения могут легко запускаться на встроенном сервере, таком как Tomcat. В то же время разработчики программного обеспечения могут просто переключиться на другие контейнеры, включая Jetty или Wildfly.

## 3. Компонент для React, реализующий форум, с возможностью писать комментарии и ставить лайки, минимум два React компонента, использовать мнимую ReactAPI.

```
const Comment = ({ comment, onLike }) => {
  const [isLiked, setIsLiked] = useState(false);
  const handleLike = () => {
    setIsLiked(!isLiked);
    onLike(comment.id, !isLiked);
  };
  return (
    <div>
      <div>{comment.author}</div>
      <div>{comment.content}</div>
      <div>
        <button onClick={handleLike}>
          {isLiked ? 'Unlike' : 'Like'} ({comment.likes})
        </button>
      </div>
    </div>
  );
};

export default Comment;

const Forum = () => {
  const [comments, setComments] = useState([
    { id: 1, author: '1', content: '2', likes: 3 },
  ]);
  const handleLike = (id, isLiked) => {
    const updatedComments = comments.map((comment) => {
      if (comment.id === id) {
        return { ...comment, likes: isLiked ? comment.likes - 1 : comment.likes + 1 };
      }
      return comment;
    });
    setComments(updatedComments);
  };
  return (
    <div>
      {comments.map((comment) => (
        <Comment
          key={comment.id}
          comment={comment}
          onLike={handleLike}
        />
      ))}
    </div>
  );
};

export default Forum;
```

## 7 Билет

### 1. Handler Mapping в Spring Web MVC

Диспетчер сервлетов DispatcherServlet Spring с помощью Handler Mapping определяет какой контроллер он должен использовать для определенного запроса request. HandlerMapping — интерфейс, который реализуется объектами, которые определяют отображение между запросами и объектами обработчиков. Помимо «основного» Handler, в обработке запроса могут участвовать один или несколько «перехватчиков» (реализаций интерфейса HandlerInterceptor). Обработчик будет всегда обернут в экземпляре HandlerExecutionChain, возможно в сопровождении некоторых экземпляров HandlerInterceptor. DispatcherServlet сначала вызывает метод preHandle каждого HandlerInterceptor в заданном порядке, и в конце, внедряет обработчик, если все методы preHandle вернули true. По умолчанию интерфейс HandlerMapping в Spring MVC реализуется классом RequestMappingHandlerMapping. В Spring MVC вы можете встретить реализацию интерфейса, когда применяете аннотацию @RequestMapping.

### 2. JSX. Особенности синтаксиса. Применение в React. Пример синтаксиса

React представляет собой дерево из компонентов. Точкой входа является index.js который определяет компоненты. Каждый компонент включает в себя другие компоненты. В React приложениях разметка пишется в JSX файлах. JSX - это синтаксическое расширение для JS, позволяющее легко делать верстку компонентов в React, используя XML-подобный синтаксис. const element = <h1>123</h1>; Под капотом компилируется Babelом в JS Применяется для переиспользования компонентов, условного рендеринга, циклов. Встраивание любых JS выражений через фигурные скобки: const name = '1'; const element = <h1>{name}</h1>; Экранирование выражений (нельзя заинжектить код извне) Поддержка spread оператора и возможность инлайнить JS код внутри там, где это нужно const todos = ['1', '2', '3']; return ( <ul> {todos.map((message) => <Item key={message} message={message} />)} </ul>);

### 3. Написать Java класс с методом main, инициализирующий и конфигурирующий Spring ApplicationContext. Поиск бинов в пакете org.itmo.web

```
public class SprcInit {
    public static void main(String[] args) {
        AnnotationConfigApplicationContext context = new
            ↪ AnnotationConfigApplicationContext();
        context.scan("org.itmo.web");
        context.refresh();
        SomeBean someBean = context.getBean(SomeBean.class);
        someBean.doSomething();
        context.close();
    }
}
```

## 8 Билет

### 1. Spring MVC: обработка запроса, dispatcher servlet

Вся логика работы Spring MVC построена вокруг DispatcherServlet, который принимает и обрабатывает все HTTP-запросы и ответы на них. При получении запроса, происходит следующая цепочка событий: DispatcherServlet обращается к интерфейсу HandlerMapping, который определяет какой контроллер должен быть вызван. Контроллер принимает запрос и вызывает соответствующий служебный метод (GET, POST), который возвращает в диспатчер имя View. При помощи ViewResolver диспатчер определяет, какой View надо использовать на основании полученного имени. После того, как View создан, диспатчер отправляет данные модели в виде атрибутов, которые уже в конечном итоге отображаются в браузере.

### 2. SPA особенности, плюсы и минусы

Веб-приложение, использующее единственный HTML-документ как оболочку для всех веб-страниц и динамически подгружает HTML, CSS, JS, обычно посредством AJAX. За навигацию отвечает JS. Клиент и сервер реализуются независимо и взаимодействуют по REST(обычно JSON). Простое кэширование данных; Скорость работы, основная часть ресурсов уже загружена, на страничку подгружаются только необходимые данные. Уменьшение нагрузки на сервер приложения. Тяжелые клиентские фреймворки; Без JS невозможно пользоваться полным функционалом приложения; Недоступна SEO оптимизация. Загрузка больших объемов JavaScript- кода может отрицательно сказаться на производительности и использовании ресурсов устройства клиента, особенно на мобильных устройствах.

### 3. Написать интерфейс на Angular: ввод паспортных данных, серия (числа, 4 цифры), номер (числа, 6 цифр), дата выдачи (дата), место выдачи(строка)

```
@Component({
  selector: 'app-pass',
  templateUrl: './pass.html'
})
export class PassComponent {
  passData = {
    series: '',

```

```

        number: '',
        issueDate: '',
        placeOfIssue: ''
    };
    constructor(private http: HttpClient) {}
    onSubmit(form: any) {
        if (form.valid) {
            this.http.post('URL', this.passData);
        } else {
            console.log('Error');
        }
    }
}
}
./pass.html
<form #passportForm="ngForm" (ngSubmit)="onSubmit(passportForm)">
    <input type="number" id="series" name="series" [(ngModel)]="passData.series"
        ↪ required pattern="[0-9]{4}">
    <input type="number" id="number" name="number" [(ngModel)]="passData.number"
        ↪ required pattern="[0-9]{6}">
    <input type="date" id="issueDate" name="issueDate" [(ngModel)]="passData.issueDate"
        ↪ required>
    <input type="text" id="placeOfIssue" name="placeOfIssue"
        ↪ [(ngModel)]="passData.placeOfIssue" required>
    <button type="submit">send</button>
</form>

```

## 9 Билет

### 1. JNDI. JNDI в Java EE. Способы взаимодействия с JNDI. Их преимущества и недостатки.

JNDI — API для доступа к объектам и ресурсам по их именам. Приложение знает только JNDI-имя, а сами детали подключения описываются администратором в веб-контейнере. JNDI поддерживает разные реализации сервиса служб имен и каталогов. (DNS, RMI, LDAP, COBRA) Naming Manager некий драйвер службы каталогов. JNDI SPI — обертка над конкретной службой каталогов, которая может быть написана даже не на джаве. Варианты использования: CDI аннотации, работает только в managed компонентах; прямой вызов API, работает везде. `new InitialContext().lookup("res")`; Преимущества JNDI: пароли к бд лежат отдельно от приложения; при изменении бд не нужно пересобирать приложение. Недостатки: зависимость от контейнера; при использовании старой версии log4j есть уязвимость (log4shell) основанная на jndi.

### 2. React. Особенности. Архитектура

React — JS библиотека для разработки пользовательского интерфейса (SPA). Позволяет создавать свои собственные компоненты, с пропсами и стейтом. Компоненты рендерятся в HTML. Передача данных от родителя к детям. Виртуальный DOM. При изменении state происходит ререндер компонента с обновлением вложенных компонентов/тэгов. Важная особенность React — использование JSX. Это расширение синтаксиса JavaScript, которое удобно использовать для описания интерфейса. Кроме того, React содержит виртуальный DOM — облегченное представление в памяти реального DOM. При написании сложных приложений, работающих с большим количеством данных, часто применяется архитектура Flux и библиотека Redux. Хуки — функции, которые позволяют использовать состояние и другие возможности React без создания класса. `useState` и `useEffect` являются наиболее распространенными хуками.

### 3. Форма для отправки сообщения в поддержку на Thymeleaf и Spring MVC. С выбором причины(select), полем для email и текстовым полем проблемы

```
1234
```

## 10 Билет

### 1. Профили платформы Java/Jakarta EE.

Web Profile — содержит в себе только те компоненты, которые нужны для работы веб-приложения, это Servlet, JSP, JSF, JPA, CDI, EJB. Full Profile — полный сборник джавы ee, в нем есть еще JAX-RS, JAX-WS, JAXB, JNDI, JAVA MAIL. Платформы: JME — представляет из себя API и минимально требовательную VM для разработки и старта приложения на смартфоне/планшете. JSE — занимается обеспечением основными стандартными функциями самой Java, и она определяет: базовые типы и объекты языка, классы более высокого уровня, производительность приложения в сети и обеспечение защищенности. JEE — для разработки Enterprise приложений. Она строится на основе платформы JSE, а еще дает возможность разработки более крупно масштабируемых, сложно уровневых и безопасных программ. Содержит: WebSocket, JSF,

Unified EL, API для веб- служб RESTful, DI, EJB, JPA, и Java Transaction API.

## 2. Способы реализации DI в Spring. @Qualifier.

Constructor-based DI - контейнер вызовет конструктор с аргументами бинов, которые потом заинджектятся в класс; Setter-based DI - сначала контейнер вызовет конструктор бина без аргументов, после вызовет помеченные аннотациями @Autowired сеттеры и впихнет туда нужные зависимости. До сих пор остается местом споров. Это связано с другой особенностью Spring Boot — IoC. Программист сам решает когда вызывать ту или иную процедуру, делать DI и т.п. В Spring Boot — это делает IoC — инициализация и вызовы процедур в Runtime. Получается, что используя DI с помощью «сеттера» вы не можете знать в какой именно момент вы зависимость будет внедрена; Field-based DI - контейнер через рефлексию будет в поля класса пропихивать зависимости. Используется редко по причине нарушения инкапсуляции, ведь внедряемое поле должно быть помечено как public. @Qualifier - позволяет определить пользовательские аннотации для уточнения, какой именно бин должен быть инжектирован, если имеется несколько кандидатов.

3. Интерфейс на Angular, формирующий две страницы URL - «Главную» (/home) и «Новости» (/news). Переход между страницами должен осуществляться посредством гиперссылок.

123

## 11 Билет

### 1. Построение пользовательского интерфейса в JSF. Иерархия компонентов JSF

Интерфейсы веб приложений на JSF описываются в XHTML файлах. Там могут быть как обычные HTML-элементы (div, p, h1, img), так и JSF компоненты. э Компоненты — это классы наследники UIComponent, образуют иерархию. Корень - UIViewRoot. У каждого компонента (кроме UIViewRoot) есть родитель, а также могут быть дети. Плюс компонентов в том, что они могут инкапсулировать сложную верстку и логику на JS за одним XHTML тегом. Существуют сторонние библиотеки компонентов, такие как PrimeFaces и IceFaces, которые упрощают построение интерфейса обширным набором готовых компонентов. Компоненты расположены на Facelets-шаблонах или страницах JSP Все компоненты реализуют базовый интерфейс javax.faces.component.UIComponent, вследствие чего можно делать собственные компоненты (но сложно) Компоненты на странице объединены в древовидную структуру - представление – корневым элементом является экземпляр класса javax.faces.component.UIViewRoot

### 2. CDI Beans: принципы инъекции. Способы разрешения ситуации неоднозначных зависимостей (2 бина реализуют 1 интерфейс)

Для того чтобы внедрить бины можно использовать аннотацию @Inject, тогда контейнер найдет у себя подходящий бин и сам создаст его. Так же если подходит несколько бинов, то будет выброшено исключение, чтобы избежать этого можно использовать аннотацию @Alternative Внедрение зависимостей работает с полями класса и с конструктором. Механизм выбора подходящего бина учитывает запрашиваемый класс или интерфейс, название бина (@Named) и альтернативы (@Alternative). @Qualifier используется когда надо конкретизировать какой именно бин внедрить и это и является решением ситуации неоднозначных зависимостей. В бины внедряется не оригинальный класс бина-зависимости, а класс-прокси, который создается на лету самим контейнером и позволяет реализовывать перехватчики.

### 3. Angular: приложение, принимающее имя и дату и формирующее бланк ПСЖ на клиенте

123

## 12 Билет

### 1. Angular: шаблоны, представление

Представление компонента задается с помощью шаблонов. Шаблоны похожи на обычный html, взаимодействуют с классом компонента через data binding. Представления группируются иерархически. Компонент может содержать иерархию представлений, которая содержит встроенные представления из других компонентов. Поддерживается интерполяция: {{ value }}. [attr]="value" — одностороннее связывание, [(attr)]="value" — двухстороннее связывание, @event="handler" — обработчик событий. Для условной отрисовки используется директива \*ngIf, для циклов — \*ngFor. Шаблоны могут содержать фильтры и директивы. Директивы - инструкции по преобразованию DOM. date | date:'shortDate' преобразует объект Date в короткую дату (pipe). Фильтры - могут преобразовывать данные в нужный формат (можно объединять в последовательности (pipe chains)).

### 2. Spring Boot, зачем он нужен. Стартеры

Spring Boot предоставляет разработчикам программного обеспечения гибкую настройку, надежную пакетную обработку, эффективный рабочий процесс и большое количество инструментов, помогая разрабатывать надежные и масштабируемые приложения на базе Spring. Он обладает автоконфигурацией, что позволяет ускорить настройку конфигурации. Включает в себя встроенные HTTP-серверы Jetty и Tomcat для тестирования веб-приложений. Позволяет подключаться к базам данных. Spring Boot оснащен встроенным контейнером сервлетов. Разработчики программного обеспечения могут переключиться на другие контейнеры, включая Jetty или Wildfly. Стартеры - это набор удобных дескрипторов зависимостей, которые вы можете включить в свое приложение. Вы получаете универсальный набор для всех необходимых вам Spring и связанных с ними технологий без необходимости искать примеры кода и копировать и вставлять множество дескрипторов зависимостей.



## 13 Билет

### 1. MVC модель JSF

JSF реализует MVC модель, которая разделяет уровень представления с уровнем бизнес логики, реализует RPC. В качестве контроллера выступает специальный FacesServlet, представление - Facelet или JSP-страница, модель - набор управляемых бинов, реализующих логику на стороне сервера. Controller в JSF – FacesServlet, главный сервлет в JSF, обрабатывает запросы с браузера, формирует объекты-события и вызывает методы-слушатели. Конфигурируется в web.xml View: JSF-страницы (XHTML или JSP). Интерфейс строится из компонентов. Компоненты расположены на Facelets-шаблонах или страницах JSP, реализуют интерфейс javax.faces.component.UIComponent. Можно создавать собственные компоненты. Компоненты на странице объединены в древовидную структуру — представление. Model: Managed beans. Вместо них можно использовать CDI, EJB или Spring бины. Конфиг с помощью аннотаций или устаревшего xml. Содержат параметры и методы для обработки данных с компонентов. Используются для обработки событий UI и валидации данных. Жизненным циклом управляет JSF Runtime Environment.

### 2. Java Jakarta EE CDI Beans: основные аннотации.

@Named для именования бинов; @Inject для внедрения зависимостей @Alternative позволяет указывать конкретную реализацию интерфейса/абстрактного класса на лету для использования разных реализаций в разных ситуациях; @Produces, @Disposes – методы для создания и уничтожения бинов соответственно; @Observes – используется в методе-наблюдателе для обработки события; @Stereotype - аннотации, включающие в себя другие аннотации. Используются в больших и сложных приложениях, где есть много бинов, выполняющих схожие действия. Стереотип может задавать: Контекст (scope) по-умолчанию; Любое число назначенных перехватчиков; Опционально – аннотацию @Named, специфицирующую имя, под которым бин будет виден из EL; Опционально – аннотацию @Alternative, специфицирующую то, что бины внутри этого стереотипа являются альтернативами друг другу. Interceptors – классы, реагирующие на определенные события ЖЦ бинов, например @PostConstruct, @PreDestroy, @AroundInvoke и т.д.

3. Страница, построенная с помощью React, реализующая таблицу со списком пользователей системы (два поля — screenName и fullName). Таблица должна поддерживать поиск по fullName (правило поиска - «содержит, с учетом регистра»). Нужно использовать минимум 2 React компонента.

## 14 Билет

### 1. Платформы Java. Особенности, сходства и различия

Java Micro Edition представляет из себя API и минимально требовательную VM для разработки и старта приложения на смартфоне. Основана на более ранней версии JSE, поэтому некоторые функции не работают Java Standard Edition занимается обеспечением основными стандартными функциями самой Java, и она определяет абсолютно все: базовые типы и объекты языка, классы более высокого уровня, производительность приложения в сети и обеспечение защищенности. Java Enterprise Edition для разработки Enterprise приложений. Она строится на основе платформы JSE, а еще дает возможность разработки более крупно масштабируемых, сложно уровневых и безопасных программ. Содержит: WebSocket, JSF, Unified EL, API для RESTful, DI, EJB, JPA, и Java Transaction API. Все платформы Java поддерживают полный функционал языка Java и отличаются лишь наличием или отсутствием определенных API.

### 2. Двухфазные и Трёхфазные конструкторы в Spring и Java EE

В момент когда спринг вызовет конструктор класса которым спринг управляет как бином, мы еще не можем использовать поля, которые спринг должен внедрить. Это происходит потому что спринг делает двухпроходный алгоритм: сначала он у всех объектов вызывает их конструкторы, если какие-то из них требуют зависимость, то они ждут своей очереди, когда эта зависимость будет создана, затем спринг начинает обрабатывать аннотации. Вторым проходом по объектам выполняет все аннотации, которые являются аннотациями настройки. Если нам надо для конструирования бина нужно как-то использовать поля, то можно использовать вторую фазу конструктора. То есть вторая фаза - метод, который размечается PostConstruct, который будет вызываться спрингом после того, когда второй цикл когда спринг пробежится по всем бином будет завершен. То есть настройка закончена, бин пост процессоры вызвал, теперь можно повызывать вторые фазы конструкторов если они у кого-то есть. В методе пост констракт уже можно будет использовать поля. Третья фаза - после конструирования объекта с обработанными зависимостями подразумевается добавление срезов (аспектов). Аспекты - вкрапления, позволяющие изменить поведение так же как данные при настройке бина. Добавить поведение до после вызова оригинала.

3. JSF страница, которая выводит 10 простых чисел, а затем ajax'ом динамически подгружает остальные пачками по 10

## 15 Билет

### 1. Spring Web MVC: View Resolvers

View Resolver — интерфейс, реализуемый объектами, которые способны находить представление по его имени. С помощью него Dispatcher Servlet находит нужный View. Представление в Spring Web MVC может быть построено на разных технологиях. С каждым представлением сопоставляется его символическое имя. Преобразованием символических имён в ссылки на конкретные представления занимается специальный класс, реализующий интерфейс `org.springframework.web.servlet.ViewResolver`. В одном приложении можно использовать несколько ViewResolver'ов

```
<bean id="viewResolver" class="org.springframework.web.servlet.view.UrlBasedViewResolver"
    <property name="viewClass" value="org.springframework.web.servlet.view.JstlView"/>
    <property name="prefix" value="/WEB-INF/jsp/" />
    <property name="suffix" value=".jsp" />
</bean>
```

### 2. Хуки в React. Что это. Для чего нужны

Хуки — механизм в React, который облегчает повторное использование кода для решения общих задач. С их помощью можно использовать состояние и другие возможности React без создания класса. К основным относятся те, что повторяют функциональность таких классовых компонентов, как работа с состоянием, побочными эффектами (жизненный цикл), контекстом и прямым доступом к DOM. `useState` и `useEffect` являются наиболее распространенными хуками. `useState`: вызываем его, чтобы наделить наш функциональный компонент внутренним состоянием. React будет хранить это состояние между рендерами. Вызов `useState` возвращает массив с двумя элементами: текущее значение состояния и функцию для его обновления. Эту функцию можно использовать где угодно, например, в обработчике событий. С помощью хука эффекта `useEffect` мы можем выполнять побочные эффекты из функционального компонента. Хук `useContext()` позволяет извлекать значения из контекста в любом компоненте, обернутом в провайдер (`provider`).

**3. JSF. Написать xhtml + CDI-бин для странички, на которой будет список отчисленных студентов. Напротив каждого студента должна быть кнопка «очистить». По нажатию на неё студень удаляется и обновляется список. Взаимодействие должно быть с помощью Ajax**

123

## 16 Билет

### 1. Конвертация данных JSF-компонентов. Создание и назначение конвертеров.

JSF предоставляет набор конвертеров для преобразования значений компонентов `UIComponent` с присвоением значения свойству `managed bean` объекта. (`@FacesConverter("URLConverter")`) Конвертеры реализуют интерфейс `javax.faces.convert.Converter` и служат чтобы конвертировать строковое представление html-данных в нужный тип. Существует несколько типов конвертеров: автоматическое `<h:inputText value="#{user.age}"/>`; с помощью атрибута `converter` `<h:inputText converter="#{javax.faces.convert.IntegerConverter}"/>`; с помощью вложенного тега `<f:converter converterId="#{javax.faces.convert.DateTimeConverter}"/>`. Исполняется после фазы формирования представления.

### 2. Реализация уровня доступа к данным при помощи Spring Data JPA. Концепция, конфигурация.

Цель Spring Data JPA — автоматизировать типовые операции по работе с базой данных (автогенерация sql запросов). Механизм делится на две части: Модель — класс, который соотносится с таблицей в базе данных. Репозиторий — интерфейс, отвечающий за CRUD-операции над сущностью и ее коллекциями. (`@Repository`) Spring Data JPA автоматически генерирует репозитории для модели на основе того, как модель проаннотирована. В свою очередь, аннотирование модели опирается на структуру таблицы в базе данных. `@Entity` — модель является сущностью. `@Table` — задает имя таблицы. `@Column` говорит о том что это поле является атрибутом сущности бд. `@Id` — идентификатор. `@GeneratedValue(strategy = GenerationType.IDENTITY)`, которая определяет стратегию создания идентификатора. `IDENTITY` указывает, что идентификатор генерируется автоматически. При отсутствии необходимых методов можно создавать свои. Это производится с помощью аннотации `@Query`.

**3. Интерфейс на React, реализующий «постраничный» просмотр таблицы с информацией о студентах (табельный номер, ФИО, группа). Таблица должна быть снабжена переключателями страниц «вперед» и «назад». Также должна быть возможность поиска студентов по ФИО по правилу «содержит, без учета регистра». "содержит" case-insensitive**

123

## 17 Билет

### 1. Валидация JSF. Создание, назначение и виды валидации.

Валидаторы в JSF — реализации интерфейса `Validator`. Метод `validate` принимает `FacesContext`, `UiComponent` и значение. Осуществляется перед обновлением значения компонента на уровне модели. Класс, осуществляющий валидацию, должен реализовывать интерфейс `javax.faces.validator.Validator`. Существуют стандартные валидаторы для основных типов данных. `DoubleRangeValidator`, `LengthValidator`, `RegexValidator`, `RequiredValidator`. Собственные валидаторы с помощью аннотации `@FacesValidator`. Внедрение: с помощью аргументов компонента `required="true"`; `<f:validateLongRange minimum="1"/>`; с помощью логики на уровне управляемого бина. Валидация происходит после фазы конвертации и

если обе эти фазы были выполнены успешно данные записываются из локальных переменных компонентов в поля.

## 2. Назначение и реализация контроллера Spring MVC, Spring Web MVC

Контроллер отвечает за обработку пользовательских запросов и построение соответствующей модели и передает ее в представление для визуализации. DispatcherServlet делегирует запрос контроллерам для выполнения специфической для него функциональности. Аннотация @Controller указывает, что определенный класс выполняет роль контроллера. Аннотация @RequestMapping используется для сопоставления URL либо с целым классом, либо с конкретным методом-обработчиком. @Controller @RequestMapping("/hello") public class HelloController{ @RequestMapping(method = RequestMethod.GET) public String printHello(ModelMap model) { model.addAttribute("message "Hello"); return "hello";}}

3. Чат- бот на Ангуляре. На каждое сообщение от пользователя бот должен отвечать «Сам дурак». Нужно указать автора, дату

123

## 18 Билет

### 1. Java Server Faces. Что это, преимущества и недостатки

JSF - фреймворк для разработки веб-приложений. Входит в состав платформы Java EE (его спецификация). Особенности: Компонентно-ориентированная структура. Интерфейс строится из компонентов, которые могут быть вложены друг в друга. Рендерятся в HTML элементы. Для отображения данных используются JSP или XML-шаблоны (facelets). Бизнес логика выносится в Java бины. Написан поверх Servlet API. Входит в JAVA EE Преимущества: разделение бизнес логики от представления (реализует MVC); Управление обменом данными на уровне компонент; программисту нужно писать меньше JS кода; простота реализации AJAX; работа с событиями на стороне сервера; расширяемость(доп наборы компонент, можно определять свои); под- держка в IDE Недостатки: Плохо масштабируется. Сложно реализовывать не предусмотренную авторами функциональность и компоненты; Не подходит для высокопроизводительных приложений; learning curve.

### 2. Java EE/Jakarta CDI бины. Что это и зачем нужны, если есть ManagedBean и EJB

CDI в Java EE представляет собой мощный инструмент для управления зависимостями и контекстами в приложениях. CDI позволяет более безопасно и удобно инжектировать зависимости, используя аннотации типа @Inject. CDI предоставляет @Qualifier и @Produces, позволяя более точно управлять инъекцией зависимостей. CDI предлагает механизмы для декорирования бинов и использования перехватчиков, что облегчает добавление перехрестной функциональности, такой как логирование и транзакции. В отличие от MB, CDI бины намного мощнее и гибче, они могут использовать перехватчики, стереотипы, декораторы и многое другое, а также смешиваться с другими бинами. EJB же обладают некоторыми особенностями, недоступными для CDI (например, транзакционные функции, таймеры, асинхронность, удаленность). Однако, в целом, EJB и CDI схожи, и их можно даже инжектировать друг в друга.

3. Angular. Приложение для регистрации и авторизации. При логине нужен email и пароль, при регистрации email, пароль и время

123