1 Билет

1. Java EE CDI Beans: именование 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 помечать альтернативные реализации аннотацией @Alternative. Это значит, что пока мы явно не скажем, что нужно использовать именно эту альтушку, 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, использующий 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:xsi="url"
    xmlns="url"
        xmlns:web="url"
    xsi:schemaLocation="url"
    id="WebApp_ID" version="3.0">
    <display-name>MVC</display-name>
    <servlet>
        <servlet-name>SpringController</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/spring-mvc.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>SpringController</servlet-name>
        <url-pattern>*.htm</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({ images }) {
    const [currentIndex, setCurrentIndex] = useState(0);
    const handleNext = () => {
        setCurrentIndex((prevIndex) => (prevIndex + 1) % images.length);
    };
    const handlePrevious = () => {
        setCurrentIndex((prevIndex) => prevIndex - 1 + images.length);
    };
    return (
    <div>
        <button onClick={handlePrevious}>back</button>
        <button onClick={handleNext}>next</button>
        <img src={images[currentIndex]} alt="Image" />
    </div>
    );
}
export default Carousel;
```

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 действиеи 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 {
    public static void main(String[] args) {
        AnnotationConfigApplicationContext context = new
            ↪ AnnotationConfigApplicationContext(AppConfig.class);
        // using
        context.close();
    }
}
```

4 Билет

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

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 создает- ся 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. Фрэймворк не специфицирует жестко технологию на которой будет построено представление. Можно использовать Thmyleaf, 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 аутентифицировться посредством ввода логина и пароля.

```
@Component({
    selector: 'app-login',
    templateUrl: '
    <form (ngSubmit)="onSubmit()" >
        <input type="text" v-model="username">
        <input type="password" v-model="password">
        <button type="submit">Login</button>
    </form>'
})
export class LoginComponent {
    username: string;
    password: string;
    constructor(private authService: AuthService) { }
    onSubmit() {
        this.authService.login(this.username, this.password);
    }
}

@Injectable({
  providedIn: 'root'
})
export class AuthService {
  constructor(private router: Router) { }
```

```
  login(username: string, password: string) {
    if (!Cookies.get('jsessionid')) {
      //logic
      Cookies.set('jsessionid', 'your-session-id');
      this.router.navigate(['/home']);
    } else {
      this.router.navigate(['/home']);
    }
  }
}

@Injectable({
  providedIn: 'root'
})
export class AuthGuard implements CanActivate {
  constructor(private authService: AuthService) { }
  canActivate(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot
  ): Observable<boolean> {
    return this.authService.isAuthenticated();
  }
}

const routes: Routes = [
  { path: 'home', canActivate: [AuthGuard] },
  { path: 'login', component: LoginComponent },
  { path: '**', redirectTo: 'login' }
];
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule],
  providers: [AuthGuard],
  bootstrap: AppComponent
})
export class AppRoutingModule { }
```

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 столь- ко, 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 штуковина, 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 UIViewRoo#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 DisptacherSevlet 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 являет 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 Babelom 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 внутрь там, где это нужно const todos = ['1', '2', '3']; return (

{todos.map((message) => <Item key={message} message={message} />)});

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("org.itmo.web");
        MyBean myBean = context.getBean(MyBean.class);
        // using bean
        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(passForm)">
    <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 тексовым полем проблемы

```
<html lang="en" xmlns:th="http://thymeleaf.com">
<body>
    <form th:action="@{/submit}" th:object="${supportForm}" method="post">
        <div>
            <select id="reason" name="reason" th:field="*{reason}">
                <option value="pivo">OutOfPivoException</option>
                <option value="general">Other</option>
            </select>
            <input type="email" id="email" name="email" th:field="*{email}">
            <textarea id="issue" name="issue" th:field="*{issue}"></textarea>
            <button type="submit">send</button>
        </div>
    </form>
</body>
</html>

@Controller
public class SupportController {
    @GetMapping("/support")
    public String showSupportForm(Model model) {
        model.addAttribute("supportForm", new SupportForm());
        return "support-form";
    }
    @PostMapping("/submit")
    public String submitSupportForm(SupportForm supportForm) {
        return "thank-you";
    }
}
@Lombok
public class SupportForm {
```

```
    private String reason;
    private String email;
    private String issue;
}
```

10 Билет

1. Профили плаформы 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 ее, 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 ис- пользуя 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. Интерфей на Angular, формирующий две страницы URL - «Главную» (/home) и «Новости» (/news). Переход между страницами должен осуществляться посредством гиперссылок.

```
const appRoutes: Routes = [
    { path: 'home', component: HomeComponent },
    { path: 'news', component: NewsComponent },
    { path: '', redirectTo: '/home', pathMatch: 'full' }];
    @NgModule({
    declarations: [
        AppComponent,
        HomeComponent,
        NewsComponent
    ],
    imports: [
        BrowserModule,
        RouterModule.forRoot(appRoutes)
    ],
    providers: [],
    bootstrap: [AppComponent]
    })
export class AppModule { }

@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css']
})
export class HomeComponent implements OnInit {
  constructor() { }
  ngOnInit(): void {
  }
}

@Component({
```

```
  selector: 'app-news',
  templateUrl: './news.component.html',
  styleUrls: ['./news.component.css']
})
export class NewsComponent implements OnInit {
  constructor() { }
  ngOnInit(): void {
  }
}

app.component.html:
<nav>
    <ul>
        <li><a routerLink="/home">General</a></li>
        <li><a routerLink="/news">News</a></li>
    </ul>
</nav>
<router-outlet></router-outlet>
```

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

```
@Component({
    selector: 'app-input-form',
    templateUrl: '
    <div>
        <input type="text" id="name" [(ngModel)]="userName" />
        <input type="date" id="date" [(ngModel)]="date" />
        <button (click)="generatePSG()">ok</button>
    </div>'
})
export class InputFormComponent {
    userName: string = '';
    date: string = '';
    generatedPSG: boolean = false;
    generatePSG() {
    const psg = new PSG(this.userName, this.date);
    this.generatedPSG = true;
    }
}
```

```
@Component({
    selector: 'app-psg-blank',
    templateUrl: '
    <div *ngIf="generatedPSG">
    <h2>PSG</h2>
        <p>{{ userName }}</p>
        <p>{{ date }}</p>
    </div>'
})
export class PSGBlankComponent {
}
```

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 под- ключаться 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.

3. Конфигурация, чтобы JSF обрабатывал все запросы приходящие с .xhtml и со всех URL, начинающихся с /faces/

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="url"
            xmlns:xsi="url"
            xsi:schemaLocation="url"
            version="4.0">
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.xhtml</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>/faces/*</url-pattern>
    </servlet-mapping>
</web-app>
```

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.

```
const UserTable = ({ users }) => {
    return (
    <table>
        <thead>
        <tr>
            <th>Screen Name</th>
            <th>Full Name</th>
        </tr>
        </thead>
        <tbody>
        {users.map((user, index) => (
            <tr>
                <td>{user.screenName}</td>
                <td>{user.fullName}</td>
            </tr>
        ))}
        </tbody>
    </table>
    );
};

const UserListPage = () => {
    const initialUsers = [
    { screenName: 'user1', fullName: 'leo' },
    // ...
    ];
    const [users, setUsers] = useState(initialUsers);
    const [searchTerm, setSearchTerm] = useState('');
    const handleSearch = (event) => {
    setSearchTerm(event.target.value);
    };
    const filteredUsers = users.filter((user) =>
    user.fullName.toLowerCase().includes(searchTerm.toLowerCase())
    );
    return (
    <div>
```

```
        <input
        type="text"
        placeholder="Search by Full Name"
        value={searchTerm}
        onChange={handleSearch}
        />
        <UserTable users={filteredUsers} />
    </div>
    );
};
export default UserListPage;
```

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 их них требуют зависимость, то они ждут своей очереди, когда это 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 бинам 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

```
<html xmlns="url"
        xmlns:h="html"
        xmlns:f="core">
<h:head>
    <h:outputScript library="js" name="jquery.min.js" target="head" />
    <script>
        function loadMoreNumbers() {
            var lastNumber = document.getElementById('lastNumber').value;
            $.ajax({
                type: 'GET',
                url: 'loadNumbers.xhtml',
                data: {
                    lastNumber: lastNumber
                },
                success: function (data) {
                    $('#numberList').append(data);
                }
            });
        }
    </script>
</h:head>
```

```
<h:body>
    <ul id="numberList">
        <ui:repeat value="#{numberBean.firstTenPrimeNumbers}" var="number">
            <li>#{number}</li>
        </ui:repeat>
    </ul>
    <h:form>
        <input type="hidden" id="lastNumber" value="#{numberBean.lastNumber}" />
        <h:commandButton value="load" onclick="loadMoreNumbers(); return false;" />
    </h:form>
</h:body>
</html>
```

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.springfra mework.web.servlet.ViewResolver. В одном приложении можно использовать несколько ViewResolver'ов <bean id="viewRe solver"class="o.s.w.s.v.UrlBasedViewReso - lver» <property na me="viewClass"value="o.s.w.s.v.Jstl - View"/> <property name="prefix"value="/WEB-INF/jsp/"/> <property name="suffix«/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. На против каждого студента должна быть кнопка «очистить». По нажатию на неё студень уделяется и обновляется список. Взаимодействие должно быть с помощью Ajax

```
<!DOCTYPE html>
<html xmlns="url"
        xmlns:h="html"
        xmlns:f="core"
        xmlns:ui="facelets">
<h:head>
    <h:outputScript library="javax.faces" name="jsf.js" />
</h:head>
<h:body>
    <h:form>
        <h:dataTable value="#{studentBean.students}" var="student">
            <h:column>
                #{student.name}
            </h:column>
            <h:column>
                <h:commandButton value="Clear"
                    ↪ action="#{studentBean.dismissStudent(student)}">
                     <f:ajax render="@form" />
                </h:commandButton>
            </h:column>
        </h:dataTable>
    </h:form>
</h:body>
```

```
</html>

@Named
@SessionScoped
public class StudentBean implements Serializable {
    private List<Student> students;
    public StudentBean() {
        students = new ArrayList<>();
        students.add(new Student("user"));
    }
    public List<Student> getStudents() {
        return students;
    }
    public void dismissStudent(Student student) {
        students.remove(student);
    }
}
```

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.DateTime}"/>; с помощью вложенного тега <f:converter converterId="#{javax.faces.DateT 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 = 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"

```
const StudentsTable = () => {
    const [students, setStudents] = useState([...]);
    const [currentPage, setCurrentPage] = useState(1);
    const [searchQuery, setSearchQuery] = useState('');
    const nextPage = () => {
    setCurrentPage(prevPage => prevPage + 1);
    };
    const prevPage = () => {
    setCurrentPage(prevPage => Math.max(prevPage - 1, 1));
    };
    const handleSearch = (e) => {
    const query = e.target.value.toLowerCase();
    setSearchQuery(query);
    const filteredStudents = students.filter(student =>
        student.fullName.toLowerCase().includes(query)
    );
    setStudents(filteredStudents);
    setCurrentPage(1);
```

```
    };
    const renderStudents = () => {
        const startIndex = (currentPage - 1) * 10;
        const endIndex = startIndex + 10;
        const currentStudents = students.slice(startIndex, endIndex);
        return (
            <table>
            <tbody>
                {currentStudents.map(student => (
                <tr key={student.id}>
                    <td>{student.id}</td>
                    <td>{student.fullName}</td>
                    <td>{student.group}</td>
                </tr>
                ))}
            </tbody>
            </table>
        );
    };
    return (
    <div>
        <input
        type="text"
        value={searchQuery}
        onChange={handleSearch}
        />
        {renderStudents()}
        <div>
        <button onClick={prevPage} disabled={currentPage === 1}>
            back
        </button>
        <button onClick={nextPage}>
            next
        </button>
        </div>
    </div>
    );
};
export default StudentsTable;
```

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

```
@Component({
    selector: 'chat',
    template: `
    <div class="chat-container">
        <div *ngFor="let message of messages">
        <p><strong>{{ message.author }}</strong> - {{ message.date }}</p>
        <p>{{ message.text }}</p>
        </div>
        <input type="text" [(ngModel)]="userInput" (keyup.enter)="sendMessage()"/>
    </div>
    `
})
export class ChatBotComponent {
    messages: { author: string; text: string; date: string }[] = [];
    userInput: string = '';
    sendMessage() {
        const currentDate = new Date();
        const formattedDate = currentDate.toLocaleString();
        this.messages.push({
            author: 'bot',
            text: 'sam durak',
            date: formattedDate,
        });
        this.userInput = '';
    }
}
@NgModule({
  declarations: [AppComponent, ChatBotComponent],
  imports: [BrowserModule, FormsModule],
  providers: [],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

18 Билет

1. Java Server Faces. Что это, преимущества и недостатки (порядок вопросов может быть другим)

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 инжектить друг в друга.

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

```
@Component({
    selector: 'app-register',
```

```
    template: `
    <form (ngSubmit)="register()">
        <input type="email" name="email" [(ngModel)]="email" required />
        <input type="password" name="password" [(ngModel)]="password" required />
        <button type="submit">Register</button>
    </form>
    `,
})
export class RegisterComponent {
    email: string = '';
    password: string = '';
    constructor(private authService: AuthService) {}
    register() {
    const currentTime = new Date().toLocaleString();
    this.authService.register(this.email, this.password, currentTime);
    }
}
@Component({
  selector: 'app-login',
  template: `
    // same register page
  `,
})
export class LoginComponent {
  email: string = '';
  password: string = '';
  constructor(private authService: AuthService) {}
  login() {
    this.authService.login(this.email, this.password);
  }
}
@Injectable({
  providedIn: 'root',
})
export class AuthService {
  register(email: string, password: string, time: string) {
    // logic}
  login(email: string, password: string) {
    // logic}
}
const routes: Routes = [
  { path: 'login', component: LoginComponent },
  { path: 'register', component: RegisterComponent },
  { path: '', redirectTo: '/login', pathMatch: 'full' },
];
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule],
})
export class AppRoutingModule {}
@NgModule({
  declarations: [AppComponent, RegisterComponent, LoginComponent],
  imports: [BrowserModule, FormsModule],
  providers: [AuthService],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

19 Билет

1. REST контроллеры в Spring. Сериализация и десериализация

Для обработки запросов написать контроллер с аннотацией @RestController в котором будет обработка клиентских запросов. Специальными аннотация- ми Get/Post/Put/Delete Mapping помечаются методы для обработки запросов. В аргументах аннотации можно указать path. Десериализация: если присутствует тело запроса, прописан заголовок ContentType, и обработчик запроса принимает аргумент, помеченный аннотацией @RequestBody, Spring автоматически десериализует данные, используя Jackson. Из коробки доступен формат JSON, но можно установить поддержку XML. Помимо тела запроса, данные могут приходить как часть URL: их можно вытащить через @PathVariable, или как часть GET-параметра, используя @RequestParam. Сериализация: из обработчика возвращается объект, а Spring его автоматически сериализует. Формат выбирается исходя из HTTP заголовка Accept. Формат можно прописать вручную в свойстве produces аннотации @RequestMapping.

2. Архитектура Angular приложения. Модули, компоненты, представление, сервисы

Приложение разбивается на модули, которые могут импортировать друг в друга. В модулях определяются компоненты — строительные блоки интерфейса, инкапсулирующие верстку, стили, и логику приложения, можно переиспользовать внутри других компонентов. Компонент состоит из TS класса, помеченного декоратором @Component, HTML шаблона, CSS стилей. Компонент вместе с шаблоном образуют представление, которое образуют иерархию. Существует двухсторонняя связь между классом компонента и представлением — при изменении данных в компоненте обновляется представление и наоборот. Задачи приложения, которые не касаются представ- ления, выносятся в сервисы (загрузка данных с сервера, валидация данных, фоновые процессы, логирование). Angular поддерживает внедрение сервисов в компоненты - @Injectable.

3. Rest контроллер для MВидео на Spring. Должен по запросу возвращать товар; возвращать страницу по переданному в jquery номеру страницы и кол-ву товаров на странице

```
@RestController
public class MVideoController {
    private List<String> products = new ArrayList<>();
    @GetMapping("/products")
    public List<String> getProducts(
            @RequestParam(name = "page", defaultValue = "1") int page,
            @RequestParam(name = "pageSize", defaultValue = "10") int pageSize) {
        int startIndex = (page - 1) * pageSize;
        int endIndex = Math.min(startIndex + pageSize, products.size());
        return products.subList(startIndex, endIndex);
    }
    public MVideoController() {
        for (int i = 1; i <= 50; i++) {
            products.add("Box " + i);
        }
    }
}
```

20 Билет

1. FacesServlet - назначение и конфигурация

FacesServlet — главный сервлет, который занимается жизненным циклом обработки запросов в приложениях, построенных на JSF. Обрабатывает запросы с браузера. Формирует объекты события и вызывает методы-слушатели. Он занимается синхронизацией состояний DOM и view (компонентов). Пришел запрос; получил; понял какую view хочет получить юзер; проинициализировал бины, выгрузил дефолт данные; сформировал html; отдал клиенту; браузер сформировал DOM; получили две иерархии; любое действие клиента; отправился запрос; сервлет модифицировал представление; сформировал новую версию html. Конфигурация через файл web.xml. Там настраиваются правила навигации, регистрируются ManagedBeans, конвертеры, валидаторы, компоненты пользовательского интерфейса. Многие из этих настроек также доступны через аннотации. В разделе <servlet> регистрируется сам серверлет и имя, а в разделе <servlet-mapping> правила url навигации.

2. React - особенности, плюсы и минусы использования

React - это библиотека для разработки графических интерфейсов. Построен на компонентном подходе: существует корневой компонент App, встроен в index.html, внутри которого вложены другие компоненты, образуя дерево компонентов. За отображение React интерфейса в DOM-дерево отвечает React DOM. При этом React реализует архитектуру SPA - подход, при котором вместо множества html-страниц, замапленных на разные URL, есть JS-скрипт, который на

лету управляет DOM-деревом - изменяет его, модифицирует, меняет вьюшки и так далее. Перед непосредственной отрисовкой (Render) React-приложение проходит стадию согласования (Reconciliation) строится новое дерево компонентов с измененным состоянием и сравнивается с предыдущим через хитрый, быстрый алгоритм. По итогу не требуется отрисовывать заново всё дерево, а только измененные его части. Каждый компонент представляет собой независимый блок кода, который отвечает за определенную часть пользовательского интерфейса. Декларативность. JSX разметка: под капотом превращается в JS, компоненты должны возвращать всегда 1 элемент, все в camelCase, закрытые теги. Можно писать нативные и фулстек приложения.

3. Написать REST-контроллер на Spring MVC, предоставляющий CRUD-интерфейс(Creat, Read, Update, Delete) к таблице со списком покемонов. Read должно получать из бд покемона по уникальному номеру, а также получать страницу с покемонами по номеру страницы.

```
@RestController
@RequestMapping("/api/pokemons")
public class PokemonController {
    private final PokemonRepository pokemonRepository;
    @Autowired
    public PokemonController(PokemonRepository pokemonRepository) {
        this.pokemonRepository = pokemonRepository;
    }
    @GetMapping("/{id}")
    public Pokemon getPokemonById(@PathVariable Long id) {
        return pokemonRepository.findById(id);
    }
    @GetMapping
    public Page<Pokemon> getPageOfPokemons(
            @RequestParam(name = "page", defaultValue = "0") int page,
            @RequestParam(name = "size", defaultValue = "10") int size) {
        PageRequest pageRequest = PageRequest.of(page, size);
        return pokemonRepository.findAll(pageRequest);
    }
    @PostMapping
    public Pokemon createPokemon(@RequestBody Pokemon pokemon) {
        return pokemonRepository.save(pokemon);
    }
    @PutMapping("/{id}")
    public Pokemon updatePokemon(@PathVariable Long id, @RequestBody Pokemon
       ↪ updatedPokemon) {
        Optional<Pokemon> optionalPokemon = pokemonRepository.findById(id);
        if (optionalPokemon.isPresent()) {
            Pokemon pokemon = optionalPokemon.get();
            pokemon.setName(updatedPokemon.getName());
            return pokemonRepository.save(pokemon);
        }
        return null;
    }
    @DeleteMapping("/{id}")
    public void deletePokemon(@PathVariable Long id) {
        pokemonRepository.deleteById(id);
    }
}
```

21 Билет

1. Структура приложения JSF

JSP или XHTML - страницы содержащие компо- ненты GUI. JSP или XHTML представляют из себя обычный HTML, но со своими тэгами и префиксами. Для этого в стандартной структуре jsf есть: Библиотека тэгов - они описывают эти дополнительные тэги jsf; Управляемые бины - бины управляемые рантаймом jsf (контейнером), чем является faces Servlet; Дополнительные объекты (компоненты, конверторы и валидаторы); Дополнительные тэги; web.xml - Файл развёртывания, где определяются параметры и настройки контейнера сервлетов, а также связи сервлетов JSF; faces-config.xml - Конфигурационный файл JSF, где определяются бины, навигация, управляемые бины, ресурсы и другие параметры при-

ложения JSF.

2. Spring Web MVC: особенности, область применения, интеграция с веб-сервером Java/Jakarta EE

Spring Web MVC – фреймворк в составе Spring для разработки веб-приложений. Основан на паттерне, который делит предложение компоненты на модель (Model), представление (View) и контроллер (Controller). Model инкапсулирует данные приложения, в целом они будут состоять из POJO. View отвечает за отображение данных фрэймворк не специфицирует жестко технологию на которой будет построено представление. По умолчанию JSP. Controller обрабатывает запрос пользователя, создаёт соответствующую Модель и передаёт её для отображения на View. Back-end; универсальный, удобен для разработки REST API. На клиентской стороне интегрируется с популярными JS-фреймворками. Удобно интегрируется с Thymeleaf. Можно настраивать Spring MVC как через XML-конфигурации, так и через Java-конфигурацию с использованием классов конфигурации @Configuration. Spring MVC использует множество аннотаций, таких как @Controller, @RequestMapping, @RequestParam, @ResponseBody и другие, для обозначения компонентов и определения маппингов URL.

3. Spring Rest контроллер, реализующий калькулятор int-чисел с 4 операциями (+, -, *, /). Должна быть реализована валидация передаваемых значений

```
@RestController
@RequestMapping("/calculator")
public class CalculatorController {
    @PostMapping("/add")
    public int add(@Valid @RequestBody Numbers n) {
        return n.getFirst() + n.getSecond();
    }
    @PostMapping("/subtract")
    public int subtract(@Valid @RequestBody Numbers n) {
        return n.getFirst() - n.getSecond();
    }
    @PostMapping("/multiply")
    public int multiply(@Valid @RequestBody Numbers n) {
        return n.getFirst() * n.getSecond();
    }
    @PostMapping("/divide")
    public int divide(@Valid @RequestBody Numbers n) {
        if (n.getSecondNumber() == 0) {
            throw new IllegalArgumentException("Cannot divide by zero!");
        }
        return n.getFirstNumber() / n.getSecondNumber();
    }
    @Getter
    @Setter
    public static class Numbers {
        @NotNull
        private Integer f;

        @NotNull
        private Integer s;
    }
}
```

22 Билет

1. Spring бины. @Component. Стереотипы

В Spring бином называется объект, который управляется, создается и настраивается Spring-контейнером. Эти объекты создаются на базе конфигурации, которая задается с помощью аннотаций. Только эти объекты участвуют в инъекции зависимостей при сборке Spring-приложения. @Component – это общий стереотип для любого Spring бина. @Repository, @Service и @Controller – это специализированные формы @Component для более конкретных случаев использования (на уровнях хранения, сервисном и представления, соответственно). Поэтому мы можем аннотировать свои компонентные классы с помощью @Component, но, если вместо этого аннотировать их @Repository, @Service или @Controller, наши классы будут больше подходить для обработки инструментами или связи с аспектами. Стереотипные аннотации - идеальные цели для срезов. Компоненты Spring могут также вносить метаданные определения бинов в контейнер. Это можно

сделать с помощью той же аннотации @Bean, которая используется для определения метаданных бина в классах.

2. Разметка в React. JSX. Структура приложения React

JSX — надстройка над JS, которая позволяет вкраплять HTML-синтаксис в код. Можно использовать стандартные HTML элементы (такие как div, span, h1, input) так и кастомные React компоненты. JSX код: text компилируется в вызов функции React.createElement(“div”, className: “foo”, “text”) React исходит из принципа, что логика рендеринга неразрывно связана с прочей логикой UI: с тем, как обрабатываются события, как состояние изменяется во времени и как данные готовятся к отображению. Разделяет ответственность с помощью слабо связанных компонентов, которые содержат и разметку, и логику. Из компонентов строится приложение. React построен на компонентном подходе: существует корневой компонент App, вмонтированный в index.html, внутри которого вложены другие компоненты, образуя дерево компонентов. За отображение React интерфейса в DOM-дерево отвечает React DOM.

3. Managed bean, который после инициализации HTTP сессии, обращается в таблице Н_УЧЕБНЫЕ_ПЛАНЫ и заполняет коллекцию данными этой таблицы. Работать с БД нужно с помощью JDBC драйвера "jbdc/Orbi sPool"

```
@ManagedBean
@SessionScoped
public class UchebnyePlanyBilya {
    private List<UchebnyyPlan> uchebnyePlany;
    @PostConstruct
    public void init() {
        uchebnyePlany = new ArrayList<>();
        try {
            String jdbcUrl = "jdbc:Orbi sPool://url";
            Connection connection = DriverManager.getConnection(jdbcUrl, "admin228",
                ↪ "admin1337");
            String sql = "SELECT * FROM N_UCHEB_PLANS";
            PreparedStatement statement = connection.prepareStatement(sql);
            ResultSet resultSet = statement.executeQuery();
            while (resultSet.next()) {
                UchebnyyPlan plan = new UchebnyyPlan();
                plan.setId(resultSet.getInt("id"));
                plan.setName(resultSet.getString("name"));
                uchebnyePlany.add(plan);
            }
            resultSet.close();
            statement.close();
            connection.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

23 Билет

1. Фаза получения значений компонентов (Apply Request Values Phase)

Данная фаза идёт после фазы формирования представления и до фазы валидации значений компонентов. На стороне клиента все значения хранятся в строковом формате, поэтому нужна проверка их корректности: Вызывается конвертер в соответствии с типом данных значения. Если конвертация - успешно, значение сохраняется в локальной переменной компонента. Если конвертация заканчивается неудачно, создаётся сообщение об ошибке, которое помещается в FacesContext.

2. Реализация IoC и DI в Spring

Инверсия управления — принцип для уменьшения связности кода. Самописным кодом управляет фреймворк, занимающийся жизненным циклом компонентов и коммуникацией между ними. Реализуется посредством внедрения зависимостей. Внедрение зависимостей - паттерн проектирования, который позволяет создавать объект, использующий другие объекты. При этом поля объекта настраиваются внешней сущностью. Позволяет убрать зависимость компонента от контейнера. ApplicationContext контейнер, который тоже управляет жизненным циклом компонентов и коммуникацией между ними и также реализуется посредством внедрения зависимостей, которые реализуется через аннотации. @Autowired - внедрение. Spring IoC контейнеру требуются метаданные для конфигурации. Для этого классы помечаются аннотацией @Component, а также её наследниками @Repository, @Service и @Controller. @Configuration, @Bean - позволяют определить бины и их конфигурацию программно. В старых версиях Spring DI можно было настроить с использованием XML-файлов, опреде-

ляя бины и их зависимости в контексте приложения. Без XML с помощью кода с использованием классов, помеченных @Configuration, и методов @Bean, которые возвращают экземпляры бинов.

3. Написать на React интерфейс интернет-магазина "НВидио показывающий список товаров с возможностью сортировать их по категории

```
const Shop = ({ productsData }) => {
    const [selectedC, setSelectedC] = useState('All');
    const handleChange = (category) => {
        setSelectedC(category);
    };
    const filteredProducts = selectedCategory === 'all'
    ? productsData
    : productsData.filter(product => {
        if (selectedCategory === 'in shop') {
          return product.availability === 'in shop';
        } else if (selectedCategory === 'delivery') {
          return product.availability === 'delivery';
        } else {
          return product.category === selectedCategory;
        }
      });
  return (
    <div>
      <div>
        <select onChange={(e) => handleChangeCategory(e.target.value)}>
          <option value="all">all</option>
          <option value="in shop">in shop</option>
          <option value="delivery">delivery</option>
        </select>
      </div>
      <ul>
        {filteredProducts.map(product => (
          <li key={product.id}>{product.name}</li>
        ))}
      </ul>
    </div>
  );
};
export default Shop;
```

24 Билет

1. Компоненты Angular: особенности разработки, взаимодействие с представлениями и сервисами.

В модулях определяются компоненты — строительные блоки интерфейса, инкапсулирующие верстку, стили, и логику приложения, которые можно переиспользовать внутри других компонентов. Любой компонент состоит из TS класса, помеченного декоратором @Component, HTML шаблона, CSS стилей. Компонент вместе с шаблоном образуют представление. Представления образуют иерархию. Существует двухсторонняя связь между классом компонента и представлением — при изменении данных в компоненте обновляется представление и наоборот. Задачи приложения, которые не касаются представления, выносятся в сервисы. Для того чтобы созданный сервис мог быть использован компонентом или другим сервисом, нужно пометить @Injectable() сервис. Так как они создаются именно для стороннего использования, то рекомендуется всегда использовать декоратор. Все сервисы регистрируются Injector-ом, который является частью механизма DI в Angular.

2. Инициализация Spring Beans. Роль ApplicationContext-а в процессе создания бинов.

Подходы: Метод бина с аннотацией @PostConstruct, не позволяет вводить параметры; Метод afterPropertiesSet() бина реализующего интерфейс InitializingBean; @Bean(initmethod="somInitMethod"). Этот подход не допускает входных параметров. В xml-конфигурации можно установить для всех бинов сразу, с помощью default-init-method. Могут быть использованы в JSF путём конфигурации в faces-config.xml. Подход Spring Boot: Есть интерфейсы CommandLineRunner и ApplicationRunner, оба из которых будут работать после создания ApplicationContext, оба из них позволяют вводить бины в качестве входных параметров. ApplicationListener позволяет прослушивать стандартные события, связанные с жизненным циклом контекста, а также кастомные события.

3. Компонент для React, реализующий интерфейс ввода данных банковской карты — номер (16 цифр), имя держателя (строка, только латинские символы) срок действия (в формате MM/YY) и защитный код (3 цифры).

```
const CreditCardForm = () => {
    const [cardNumber, setCardNumber] = useState('');
    const [cardHolderName, setCardHolderName] = useState('');
    const [expiry, setExpiry] = useState('');
    const [cvv, setCVV] = useState('');
    const handleCardNumberChange = (event) => {
        if (event.target.value.length <= 16) {
            setCardNumber(event.target.value);
        }
    };
    const handleCardHolderNameChange = (event) => {
        const regex = /^[a-zA-Z\s]+$/;
        if (regex.test(event.target.value)) {
            setCardHolderName(event.target.value);
        }
    };
    const handleExpiryChange = (event) => {
        const regex = /^(0[1-9]|1[0-2])\/?([0-9]{2})$/;
        if (regex.test(event.target.value)) {
            setExpiry(event.target.value);
        }
    };
    const handleCVVChange = (event) => {
        if (event.target.value.length <= 3) {
            setCVV(event.target.value);
        }
    };
    const handleSubmit = (event) => {
        event.preventDefault();
        sendCard();
    };
    return (
    <form onSubmit={handleSubmit}>
        <input type="number" value={cardNumber} onChange={handleCardNumberChange}
        ↪ maxLength={16} />
        <input type="text" value={cardHolderName} onChange={handleCardHolderNameChange}
        ↪ />
        <input type="date" value={expiry} onChange={handleExpiryChange} maxLength={5} />
        <input type="number" value={cvv} onChange={handleCVVChange} maxLength={3} />
        <button type="submit">Submit</button>
    </form>
    );
};
export default CreditCardForm;
```

25 Билет

1. Профили запуска Spring. Как определить, какой сейчас активен?

Профили Spring позволяют кастомизировать приложение для работы в различном окружении, инстанционируя различные реализации одного и того же бина и присваивая различные значения свойствам приложения в зависимости от активного профиля. Список активных профилей задается для приложения следующими способами: В servlet context параметре spring.profiles.active в файле web.xml; В системном свойстве spring.profiles.active; В Spring Boot, профили могут быть активированы через аннотацию @Profile на классах или методах, а также через командную строку с помощью параметра spring.profiles.active. Для определения активного профиля в приложении Spring можно использовать интерфейс Environment, который предоставляет информацию о среде приложения, включая активные профили. Метод getActiveProfiles() возвращает массив строк с активными профилями, которые затем можно использовать по своему усмот-

рению

2. Process validations phase и Update model values phase

Process Validation Phase: На этом этапе реализация JavaServer Faces обрабатывает все валидаторы, зарегистрированные в компонентах в дереве, используя свой метод validate (processValidators). Он изучает атрибуты компонента, которые определяют правила проверки, и сравнивает эти правила с локальным значением, хранящимся для компонента. Если значение компонента не проходит валидацию, формируется сообщение об ошибке, которое сохраняется в FacesContext. Update Model Values Phase: Реализация JavaServer Faces устанавливает соответствующие свойства объекта на стороне сервера для локальных значений компонентов. В этот момент, если приложению необходимо перенаправить на другой ресурс веб-приложения или сгенерировать ответ, который не содержит никаких компонентов JavaServer Faces, оно может вызвать метод FacesContext.responseComplete.

3. Часы на ангуляре которые показывают локальное время и обновляются каждую секунду.

```
@Component({
    selector: 'app-clock',
    template: '<p>{ currentTime | date: 'HH:mm:ss' }</p>'
})
export class ClockComponent implements OnInit {
    currentTime: Date = new Date();
    constructor() { }
    ngOnInit(): void {
            setInterval(() => {
                this.updateTime();
        }, 1000);
    }
    updateTime(): void {
        this.currentTime = new Date();
    }
}
```