sew-matura-hilfszettel

Inhaltsverzeichnis

Angular

Bindings

```
<p>{{ text }}</p>
<button [disabled]="text == 'Hello World!'" (click)="handleClick()">btn</button>
<input [(ngModel)]="name" />
```

```
export class AppComponent {
  text = "Welcome to demo1";
  name = "unknown";

  handleClick() {
    this.text = "Hello World!";
  }
}
```

Direktiven

```
<p *ngIf="name !== ''">Hello {{ name }}</p>
<p [ngStyle]="color: getColor()" [ngClass]="{type: getTextType()}"></p>
<ul>
  <li *ngFor="let user of users; let i = index">{{ i }} - {{ user }}</li>
</ul>

<div [ngSwitch]="day">
  <p *ngSwitchCase="days.MONDAY">Montag</p>
  <p *ngSwitchCase="days.TUESDAY">Dienstag</p>
  <p *ngSwitchDefault>Sonstiger Tag</p>
</div>
```

```
export class AppComponent {
  name = "unknown";
  color = "red";
  type = "bold";
  users = ["user1", "user2", "user3"];
  day = Days.MONDAY;
  days = Days;

  getColor() {
    return this.color;
  }

  getTextType() {
    return this.type;
  }
}

export enum Days {
  MONDAY,
  TUESDAY,
  WEDNESDAY,
}
```

Components Databinding

```
@Input('firstname') firstname!: string;
@Input('lastname') lastname!: string;
@Output() removeEvent = new EventEmitter();
@ViewChild('nicknameInput', {static: false}) nicknameInput: ElementRef;
nickname = '';

login() {
  this.nickname = this.nicknameInput.nativeElement.value;
}

remove() {
this.removeEvent.emit();
}
```

```
<input type="text" #nicknameInput />
<button (click)="login()">Login</button>

<p>Hello {{nickname}}</p>

<p #pTag>gets removed</p>
<some-component
  [firstname]="'Max'"
  lastname="Muster"
  (removeEvent)="handleRemoveEvent(pTag)"
></some-component>
```

```
handleRemoveEvent(pTag: HTMLElement) {
console.log(pTag.innerHTML);
}
```

Services

```
export class LoggingService() {
    public logStatus(status: string) {
        console.log(status);
    }
}

 getMembersByClub(clubId: number){
    return this.http.get<Person[]>("http://localhost:8081/api/club/members/" + clubId);
  }

  addMember(membershipDTO:MembershipDTO){
    return this.http.post(this.url + "/membership/add", membershipDTO);
  }
```

```
constructor(private service: LoggingService) {}

onStatusChange(status: string) {
    this.service.logStatus(status);
}
```

Routing

Konfiguration

```
const appRoutes: Routes = [
    { path: '', component: HomeComponent },
    { path: 'gallery', component: GalleryComponent },
    { path: 'gallery/:index', component: DetailComponent},
    { path: '**', component: TableComponent},
    { path: '', redirectTo: '/chat-list', pathMatch: 'full' },
];

...
imports: [
    BrowserModule,
    FormsModule,
    RouterModule.forRoot(appRoutes)
]
...
```

Verwendung

```
<a routerLink="/gallery">Gallery</a>
<a routerLink="/gallery" [queryParams]="{category: 'Nature'}">
  Gallery for Nature Pictures
</a>
<router-outlet></router-outlet>
```

```
constructor(private router: Router, private route: ActivatedRoute) {}

this.router.navigate(['gallery']);
this.router.navigate(['gallery'], {relativeTo: this.route});
this.router.navigate(['gallery', index]);
this.router.navigate(['gallery'], {queryParams: {category: 'Nature'}});
```

```
export class TextComponent implements OnInit {
  index: string;
  category: string;

  constructor(private route: ActivatedRoute) {}

  ngOnInit() {
    this.route.params.subscribe((params) => {
      this.id = params["id"] == null ? "" : params["id"];
    });
    this.route.params.subscribe((params) => {
      this.category = params["category"] == null ? "" : params["category"];
    });
  }
}
```

Observable

```
export class AppComponent implements OnInit, OnDestroy {
  myObsSubscription: Subscription;

  ngOnInit() {
    const myObservable = Observable.create((observer: Observer<string>) => {
      setTimeout(() => {
        observer.next("first package");
      }, 2000);
      setTimeout(() => {
        observer.next("second package");
      }, 4000);
      setTimeout(() => {
        observer.error("this does not work");
      }, 5000);
    });

    this.myObsSubscription = myObservable.subscribe(
      (data: string) => {
        console.log(data);
      },
      (error: string) => {
        console.log(error);
      },
      () => {
        console.log("completed");
      }
    );
  }

  ngOnDestroy() {
    this.myObsSubscription.unsubscribe();
  }
}
```

Subject

```
export class SearchService {
  searchSubject = new Subject<string>();

  constructor() {}

  search(term: string) {
    this.searchSubject.next(term);
  }
}
```

```
export class Comp1Component implements OnInit {
  constructor(private searchService: SearchService) {}

  ngOnInit() {}

  search(term: HTMLInputElement) {
    this.searchService.search(term.value);
  }
}
```

```
export class Comp2Component implements OnInit {
  searchSubscription: Subscription;

  constructor(private searchService: SearchService) {}

  ngOnInit() {
    this.searchSubscription = this.searchService.searchSubject.subscribe(
      (data: string) => {
        console.log(data);
      }
    );
  }

  ngOnDestroy() {
    this.searchSubscription.unsubscribe();
  }
}
```

Forms

Template Driven

```
<form (ngSubmit)="onSubmit(f)" #f="ngForm">
  <div class="form-group">
    <label for="username">Username</label>
    <input
      type="text"
      id="username"
      class="form-control"
      name="username"
      ngModel
      required
    />
  </div>
  <div class="form-group">
    <label for="email">Email</label>
    <input
      type="email"
      id="email"
      class="form-control"
      name="email"
      ngModel
      required
      email
      #email="ngModel"
    />
    <span class="help-block" *ngIf="!email.valid && email.touched">
      Please enter a valid email address!
    </span>
  </div>

  <button type="submit" class="btn btn-primary" [disabled]="!f.valid">
    Submit
  </button>
</form>
```

```
export class AppComponent {
  onSubmit(form: NgForm) {
    console.log(form.value.username);
  }
}
```

Reactive

// ng generate @angular/material:table/menu/address-form/navigation/dashboard name

```
<form [formGroup]="regForm">
  <div class="form-group">
    <label for="username">Username</label>
    <input
      type="text"
      id="username"
      class="form-control"
      formControlName="username"
    />
    <span
      *ngIf="!regForm.get('username').valid && regForm.get('username').touched"
      class="help-block"
    >
      Please enter a valid username!
    </span>
  </div>
  <div class="form-group">
    <label for="email">Email</label>
    <input
      type="email"
      id="email"
      class="form-control"
      formControlName="email"
    />
    <span
      *ngIf="!regForm.get('email').valid && regForm.get('email').touched"
      class="help-block"
    >
      Please enter a valid email address!
    </span>
  </div>

  <button
    type="submit"
    class="btn btn-primary"
    (click)="onSubmit()"
    [disabled]="!regForm.valid"
  >
    Submit
  </button>
</form>
```

```
export class AppComponent implements OnInit {
  regForm: FormGroup;

  ngOnInit() {
    this.regForm = new FormGroup({
      username: new FormControl(null, Validators.required),
      email: new FormControl(null, [Validators.required, Validators.email]),
    });
  }

  onSubmit() {
    console.log(this.regForm.value);
  }
}
```

Pipes

```
 <td mat-cell *matCellDef="let element">{{element.endTs |date: 'dd.MM.yyyy
hh:mm:ss' }}</td>
{{auction.startingPrice | currency}}
{{auction.startingPrice | currency:'EUR'}}
```

Angular-Material

```
 ng add @angular/material
ng generate @angular/material:table Table

weitere generierbare Materials:
menu
address-form
navigation
dashboard
```

MatInput

```
<mat-form-field appearance="fill">
  <mat-label>Enter your email</mat-label>
  <input
    matInput
    placeholder="pat@example.com"
    [formControl]="email"
    required
  />
  <mat-error *ngIf="email.invalid">{{getErrorMessage()}}</mat-error>
  <mat-icon matSuffix>sentiment_very_satisfied</mat-icon>
  <mat-hint>Hint</mat-hint>
</mat-form-field>
<mat-form-field appearance="fill">
  <mat-label>Select</mat-label>
  <mat-select>
    <mat-option value="one">First option</mat-option>
    <mat-option value="two">Second option</mat-option>
  </mat-select>
</mat-form-field>
<mat-form-field appearance="fill">
  <mat-label>Textarea</mat-label>
  <textarea matInput></textarea>
</mat-form-field>
```

```
  email = new FormControl('', [Validators.required, Validators.email]);

  getErrorMessage() {
    if (this.email.hasError('required')) {
      return 'You must enter a value';
    }

    return this.email.hasError('email') ? 'Not a valid email' : '';
  }
```

SnackBar

```
export class SnackBarOverviewExample {
  constructor(private snackBar: MatSnackBar) {}

  openSnackBar(message: string, action: string) {
    // message and action with config => action can be undefined if none is required
    this.snackBar.open(message, action, {
      duration: 3000,
    });

    // snackbar with own component providing data
    let snackBarRef = this.snackBar.openFromComponent(
      MessageArchivedComponent,
      {
        data: "some data",
      }
    );
  }
}
```

```
export class MessageArchivedComponent {
  constructor(@Inject(MAT_SNACK_BAR_DATA) public data: string) {}
}
```

Table

```
<div class="mat-elevation-z8">
  <table mat-table [dataSource]="dataSource">
    <!-- Position Column -->
    <ng-container matColumnDef="position">
      <th mat-header-cell *matHeaderCellDef>No.</th>
      <td mat-cell *matCellDef="let element">{{element.position}}</td>
    </ng-container>

    <!-- Name Column -->
    <ng-container matColumnDef="name">
      <th mat-header-cell *matHeaderCellDef>Name</th>
      <td mat-cell *matCellDef="let element">{{element.name}}</td>
    </ng-container>

    <!-- Weight Column -->
    <ng-container matColumnDef="weight">
      <th mat-header-cell *matHeaderCellDef>Weight</th>
      <td mat-cell *matCellDef="let element">{{element.weight}}</td>
    </ng-container>

    <!-- Symbol Column -->
    <ng-container matColumnDef="symbol">
      <th mat-header-cell *matHeaderCellDef>Symbol</th>
      <td mat-cell *matCellDef="let element">{{element.symbol}}</td>
    </ng-container>

    <tr mat-header-row *matHeaderRowDef="displayedColumns"></tr>
    <tr mat-row *matRowDef="let row; columns: displayedColumns;"></tr>
  </table>

  <mat-paginator
    [pageSizeOptions]="[5, 10, 20]"
    showFirstLastButtons
    aria-label="Select page of periodic elements"
  >
  </mat-paginator>
</div>
```

```
export class TablePaginationExample implements AfterViewInit {
  displayedColumns: string[] = ["position", "name", "weight", "symbol"];
  dataSource = new MatTableDataSource<PeriodicElement>(ELEMENT_DATA);

  @ViewChild(MatPaginator) paginator: MatPaginator;

  ngAfterViewInit() {
    this.dataSource.paginator = this.paginator;
  }
}

export interface PeriodicElement {
  name: string;
  position: number;
  weight: number;
  symbol: string;
}

const ELEMENT_DATA: PeriodicElement[] = [
  { position: 1, name: "Hydrogen", weight: 1.0079, symbol: "H" },
  { position: 2, name: "Helium", weight: 4.0026, symbol: "He" },
  { position: 3, name: "Lithium", weight: 6.941, symbol: "Li" },
];
```

```
<!-- Sorting -->
<table
  mat-table
  [dataSource]="dataSource"
  matSort
  (matSortChange)="announceSortChange($event)"
>
  <ng-container matColumnDef="position">
    <th mat-header-cell *matHeaderCellDef mat-sort-header>Name</th>
    <td mat-cell *matCellDef="let element">{{element.position}}</td>
  </ng-container>

  <tr mat-header-row *matHeaderRowDef="displayedColumns"></tr>
  <tr mat-row *matRowDef="let row; columns: displayedColumns;"></tr>
</table>
```

List

```
<mat-list>
  <mat-list-item>
    <span matListItemTitle>Pepper</span>
    <span matListItemLine>Produced by a plant</span>
  </mat-list-item>
  <mat-list-item>
    <span matListItemTitle>Salt</span>
    <span matListItemLine>Extracted from sea water</span>
  </mat-list-item>
  <mat-list-item>
    <span matListItemTitle>Paprika</span>
    <span matListItemLine>Produced by dried and ground red peppers</span>
  </mat-list-item>
</mat-list>
```

Date-API

LocalDate

```
LocalDate localDate=LocalDate.now();
LocalDate.of(2015,02,20);
LocalDate.parse("2015-02-20");
LocalDate tomorrow=LocalDate.now().plusDays(1);
LocalDate previousMonthSameDay=LocalDate.now().minus(1,ChronoUnit.MONTHS);
DayOfWeek sunday = LocalDate.parse("2016-06-12").getDayOfWeek();
int twelve = LocalDate.parse("2016-06-12").getDayOfMonth();
boolean leapYear=LocalDate.now().isLeapYear();
boolean notBefore = LocalDate.parse("2016-06-12").isBefore(LocalDate.parse("2016-06-11"));
boolean isAfter = LocalDate.parse("2016-06-12").isAfter(LocalDate.parse("2016-06-11"));
LocalDateTime beginningOfDay = LocalDate.parse("2016-06-12").atStartOfDay();
LocalDate firstDayOfMonth = LocalDate.parse("2016-06-12").with(TemporalAdjusters.firstDayOfMonth());
```

LocalDateTime

```
LocalDateTime.now();
LocalDateTime.of(2015,Month.FEBRUARY,20,06,30);
LocalDateTime.parse("2015-02-20T06:30:00");
localDateTime.plusDays(1);
localDateTime.minusHours(2);
localDateTime.getMonth();
```

DateFormat

```
@JsonbDateFormat(value = "yyyy-MM-dd")
@Column(name = "BOOKINGDATE")
private LocalDate bookingDate;
@JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss")
private LocalDateTime lastUpdate;
```

HTML-Zusatz

Select

```
 <select  [(ngModel)]="selectedLevel" (change)="selected()" id="startSelect">
    <option *ngFor="let i of listOfObjects" value="{{i.start}}">{{i.start}}</option>
</select>
```

DTO in .ts

```
let coursePlanPersonDTO: CoursePlanPersonDTO = {
     planId: parseInt(this.inputString),
     firstname: this.firstName,
     lastname: this.lastName
   }
```

HTTP

```
@NgModule({
 declarations: [AppComponent],
 imports: [
   BrowserModule,
   FormsModule,
   HttpClientModule
 ],
 providers: [],
 bootstrap: [AppComponent]
})
```

```
export class PostService {
  constructor(private http: HttpClient) {}
  url = "http://localhost:8081/api/";

  getBySearch(searchWord: string): Observable<Course[]> {
    return this.http.get<Course[]>(this.url + "course/search/" + searchWord);
  }

  getCourseById(id: string): Observable<Course> {
    return this.http.get<Course>(this.url + "course/" + id);
  }

  getCoursePlanByCourseId(courseId: string): Observable<CoursePlan[]> {
    return this.http.get<CoursePlan[]>(this.url + "course/plan/" + courseId);
  }

  register(coursePlanPersonDTO: CoursePlanPersonDTO): Observable<any> {
    console.log(coursePlanPersonDTO);
    return this.http.post(this.url + "course/plan1", coursePlanPersonDTO);
  }

  createAndStorePost(title: string, content: string) {
    const postData: Post = { title: title, content: content };
    this.http
      .post<{ name: string }>(
        "https://ng-complete-guide-6f4f5.firebaseio.com/posts.json",
        postData
      )
      .subscribe((responseData) => {
        console.log(responseData);
      });
  }

  fetchPosts() {
    this.http.get<Post>(
      "https://ng-complete-guide-6f4f5.firebaseio.com/posts.json"
    );
  }

  fetchPostsWithHeaders() {
    return this.http.get<Post>(
      "https://ng-complete-guide-6f4f5.firebaseio.com/posts.json",
      {
        headers: new HttpHeaders({ "Custom-Header": "Hello" }),
        params: new HttpParams().set("print", "pretty"),
        observe: "body", // oder 'response', 'events'
      }
    );
  }

  deletePosts() {
    return this.http.delete(
      "https://ng-complete-guide-6f4f5.firebaseio.com/posts.json"
    );
  }
}
```

Usage

```
export class AppComponent implements OnInit {
  loadedPosts: Post[] = [];
  error = null;

  constructor(private postService: PostService, private http: HttpService) {}

  search() {
    this.http.getBySearch(this.searchInput).subscribe(value => {
      this.coursesBySearch = value;
    });

  ngOnInit() {
    this.postService.fetchPosts().subscribe(
      (posts) => {
        this.loadedPosts = posts;
      },
      (error) => {
        this.error = error.message;
        console.log(error);
      }
    );
  }

  onCreatePost(postData: Post) {
    this.postService.createAndStorePost(postData.title, postData.content);
  }

  onFetchPosts() {
    this.postService.fetchPosts();
  }

  onClearPosts() {
    this.postService.deletePosts().subscribe(() => {
      this.loadedPosts = [];
    });
  }
}
```

Websockets

```
export class WebSocketService implements OnInit {
  myWebSocket!: WebSocketSubject<any>;
  public data: Survey = new Survey();

  ngOnInit() {
    this.myWebSocket = new WebSocketSubject("ws://localhost:8080/ws");
    this.myWebSocket.asObservable().subscribe(
      (msg: Message) => console.log("message received: " + msg),
      (err: Event) => console.log("error: " + err),
      () => console.log("complete")
    );

    // andere Version
    this.myWebSocket = webSocket({
      url: "ws://localhost:8081/survey",
      deserializer: (msg) => msg.data,
    });

    this.myWebSocket.subscribe((value) => {
      let json = JSON.parse(value.toString());
      console.log(json);
      this.data.text = json.text;
      this.data.result = new Map(
        Object.keys(json.result).map((key) => [key, json.result[key]])
      );

      // oder für einfache Messages ohne JSON
      this.messages.push(msg); //messages: string[] = []
    });
  }
  public vote(option: string) {
    this.httpClient
      .post("http://localhost:8081/survey/vote/" + option + "/true", {})
      .subscribe();
  }

  sendMessage(msg: Message) {
    this.myWebSocket.next(msg);
  }

  close() {
    this.myWebSocket.complete();
  }
}
```

AuthGuard

```
@Injectable({
  providedIn: "root",
})
export class AuthGuardService {
  constructor(private authService: AuthService, private router: Router) {}

  canActivate(): boolean {
    if (!this.authService.isLoggedIn()) {
      this.router.navigate(["auth"]);
      return false;
    }
    return true;
  }
}
```

`routing`

```
const routes: Routes = [
  { path: "home", component: HomeComponent, canActivate: [AuthGuardService] },
  { path: "auth", component: AuthComponent },
  { path: "**", redirectTo: "home" },
];
```

Interceptors

```
@Injectable()
export class AuthInterceptor implements HttpInterceptor {
  constructor() {}

  intercept(
    req: HttpRequest<any>,
    next: HttpHandler
  ): Observable<HttpEvent<any>> {
    const idToken = sessionStorage.getItem("id_token");
    if (idToken) {
      const cloned = req.clone({
        headers: req.headers.set("Authorization", "Bearer " + idToken),
      });
      return next.handle(cloned);
    } else {
      return next.handle(req);
    }
  }
}
```

`app.module.ts`

```
providers: [
  {
    provide: HTTP_INTERCEPTORS,
    useClass: AuthInterceptor,
    multi: true,
  },
];
```

Quarkus

Entities and ID Generation

Autoincrement

```
@Entity
public class Address {
  @Id
  @GeneratedValue()
  public Long id; // Für alle Entities: entweder public oder private mit Getter & Setter

  public String street;
}
```

Table

```
@Entity
@TableGenerator(name = "addressGen", initialValue = 1000, allocationSize = 50)
public class Address {
  @Id
  @GeneratedValue(strategy = GenerationType.TABLE, generator = "addressGen")
  public Long id;

  public String street;
}
```

Sequence

```
@Entity
@SequenceGenerator(name = "addressSeq", initialValue = 1000, allocationSize = 50)
public class Address {
  @Id
  @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "addressSeq")
  public Long id;

  public String street;
}
```

Embedded ID / Composite Key

```
@Entity
public class Address {
  @EmbeddedId
  public AddressId id;

  public String street;
}
```

```
@Embeddable
public class AddressId implements Serializable {
//  @ManyToOne wenn 2 Tabellen
  public Long id;

  public String city;
}
```

With Relations

```
@Embeddable
public class MembershipId implements Serializable {
    @Column(name="ssid")
    public long ssid;

    @Column(name="club_id")
    public long clubId;
}
```

```
@Entity
public class Membership extends PanacheEntityBase {

    @EmbeddedId
    public MembershipId membershipId;

    @ManyToOne
    @MapsId("ssid")
    @JoinColumn(name = "ssid")
    public Person person;

    @ManyToOne
    @MapsId("clubId")
    @JoinColumn(name="club_Id")
    public Club club;

    @Column(name = "join_date")
    public LocalDate joinDate;

    @Column(name = "exit_date")
    public LocalDate exitDate;
}
```

```
@Entity
public class Person extends PanacheEntityBase {
    @Id
    @GeneratedValue
    public long ssid;

    @Column(name = "first_name")
    public String firstname;

    @Column(name = "last_name")
    public String lastname;

    @OneToMany(mappedBy = "person")
    @JsonIgnoreProperties({"person"})
    List<Membership> memberships;
}
```

```
@Entity
public class Club extends PanacheEntityBase {
    @Id
    @GeneratedValue
    public long id;

    public String name;

    @OneToMany(mappedBy = "club")
    @JsonIgnoreProperties({"club"})
    public List<Membership> memberships;
}
```

Column

```
@Entity
public class Address {
  @Id
  @GeneratedValue()
  @Column(name = "address_id")
  public Long id;

  @ManyToOne
  Street street;
}
```

UriInfo

```
@PUT
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces(MediaType.APPLICATION_JSON)
    @Transactional
    public Response putEmployee(Employee employee, @Context UriInfo context) {
        try {
            Employee emp = repo.getEntityManager().merge(employee);
            URI uri = context.getAbsolutePathBuilder().path(Long.toString(emp.id)).build();
            return Response.created(uri).build();
        }catch (Exception e) {
            e.printStackTrace();
            return Response.status(Response.Status.BAD_REQUEST).build();
        }
    }
```

Entity Relations

One to One

```
@Entity
public class Address {
  @Id
  @GeneratedValue()
  public Long id;

  public String street;

  @OneToOne(mappedBy = "address")
  public Person person;
}
```

```
@Entity
public class Person {
  @Id
  @GeneratedValue()
  public Long id;

  public String name;

  @OneToOne
  public Address address;
}
```

One to Many

```
@Entity
public class Address {
  @Id
  @GeneratedValue()
  public Long id;

  public String street;

  @OneToMany(mappedBy = "address", cascade = CascadeType.ALL)
  @JoinColumn(name = "address_id")

  @Column(nullable = false)
  public List<Person> persons;
}
```

JsonIgnore

```
@Entity
public class Address {
  @Id
  @GeneratedValue()
  public Long id;

  public String street;

  @OneToMany(mappedBy = "address", cascade = CascadeType.ALL)
  @JoinColumn(name = "address_id")

  //if bidirectional use either @JsonIdentityInfo & JsonIdentityReference or @JsonIgnore
  //Ergebnis:
  // "person": 1

  @Column(nullable = false)
  @JsonIdentityInfo(
  generator = ObjectIdGenerators.PropertyGenerator.class,
  property = "id")
  @JsonIdentityReference(alwaysAsId = true)
  public List<Person> persons;

  //oder

  //Ergebnis:
  //"person": {
  //    "id": 1
  // }

  @JsonIgnoreProperties({"name"})
  public List<Person> persons;

}
```

```
@Entity
public class Person {
  @Id
  @GeneratedValue()
  public Long id;

  public String name;

  @ManyToOne
  public Address address;
}
```

Self reference

```
@Entity
public class Person extends PanacheEntityBase {

    @Id
    @GeneratedValue
    public Long ssid;

    @Column(name = "first_name")
    public String firstName;

    @Column(name = "last_name")
    public String lastName;

    @ManyToOne
    @JsonIgnoreProperties({"employees"})
    public Person boss;

    @OneToMany(mappedBy = "boss")
    @JsonIdentityReference(alwaysAsId = true)
    @JsonIdentityInfo(generator = ObjectIdGenerators.PropertyGenerator.class, property = "ssid")
    @JsonIgnoreProperties({"boss"})
    public List<Person> employees;
}
```

Many to Many

```
@Entity
public class Address {
  @Id
  @GeneratedValue()
  public Long id;

  public String street;

  @ManyToMany(mappedBy = "addresses")
  public List<Person> persons;
}
```

```
@Entity
public class Person {
  @Id
  @GeneratedValue()
  public Long id;

  public String name;

  @ManyToMany
  @JoinTable(name="address_person"
  public List<Address> addresses;
}
```

Fetching

```
@Entity
public class Address {
  @Id
  @GeneratedValue()
  public Long id;

  public String street;

  @OneToMany(mappedBy = "address", cascade = CascadeType.ALL, fetch = FetchType.LAZY) // default oder EAGER
  @JoinColumn(name = "address_id")
  public List<Person> persons;
}
```

Repository

```
@ApplicationScoped
public class AddressRepository {

  @Inject
  EntityManager em;

  public Address save(Address address) {
    if (address.getId() == null) {
      em.persist(address);
      return address;
    } else {
      return em.merge(address);
    }
  }

  public Address findById(Long id) {
    return em.find(Address.class, id);
  }

  public void deleteById(Long id) {
    Address address = findById(id);
    em.remove(address);
  }
}
```

Queries

```
@ApplicationScoped
public class AddressRepository {

  @Inject
  EntityManager em;

  public List<Person> getMembersByClub(Long clubId){
        String sql = "Select person from Membership ms where ms.id = :clubId";
        return getEntityManager().createQuery(sql, Person.class).setParameter("clubId", clubId).getResultList();
    // .get(0) oder .getSingleResult für nur 1
    }

  public void deleteMembershipByPersonId(Long id){
        String sql = "Delete from Membership ms where ms.id = :id";
        getEntityManager().createQuery(sql).setParameter("id", id).executeUpdate();
    }

  //Join-Query mit Record
  public AddressDTO getAdressPerson() {
    TypedQuery<AdressDTO> query = em.createQuery(
            "SELECT new com.example.AddressDTO(a.address, p.name) " +
            "FROM Address a JOIN p.name p", AdressDTO.class);
    return query.getResultList();
  }

  //Query mit Aggregate Funktion
  public Double getAdressPerson() {
    TypedQuery<Double> query = em.createQuery("SELECT SUM(p.income) FROM Person p", Double.class);
    return query.getSingleResult();
  }

  TypedQuery<Membership> query = em.createQuery(
                "SELECT m from Membership m " +
                        "where m.id.club.id = :club_id " +
                        "and m.id.person.ssid = :ssid", Membership.class);
  query.setParameter("club_id", membershipDTO.club_id);
  query.setParameter("ssid", membershipDTO.ssid);
  return query.getSingleResult();


}
```

Panache

Entity

```
@Entity
public class Address extends PanacheEntity {
  private String street;
}
```

Entity without automatic ID

```
@Entity
public class Address extends PanacheEntityBase {
  @Id
  @GeneratedValue()
  private Long id;

  private String street;
}
```

Named Query

```
@Entity
@NamedQueries({
  @NamedQuery(name = "Address.findAll", query = "from Address"),
  @NamedQuery(name = "Address.findForStreet", query = "from Address where street = ?1")
})
public class Address extends PanacheEntity {
  private String street;
}

return Address.find("#Address.findForStreet", street);
```

Repository

```
@ApplicationScoped
public class AddressRepository implements PanacheRepository<Address> {

  public Address save(Address address) {
    if (address.getId() == null) {
      persist(address);
      return address;
    } else {
      return getEntityManager().merge(address);
    }
  }

  public Address findById(Long id) {
    return find("id", id).firstResult();
  }

  public void deleteById(Long id) {
    Address address = findById(id);
    delete(address);
  }
}
```

Resource

```
@Path("/api")
public class AdressResource {
  @Inject
  AddressRepository addressRepository;

  @GET
  @Path("/addresses")
  @Produces(MediaType.APPLICATION_JSON)
  public List<Address> getAddresses() {
    return addressRepository.listAll();
  }

  // mit Response
  @GET
  @Produces(MediaType.APPLICATION_JSON)
  @Path("/{id}")
  public Response getClubById(@PathParam("id") Long id){
        return Response.ok(repo.methodenNameVomRepo(id)).build();
  }

  @POST
  @Path("/add")
  @Consumes(MediaType.APPLICATION_JSON)
  @Produces(MediaType.APPLICATION_JSON)
  @Transactional
  public Response addPerson(Person person, @Context UriInfo uriInfo){
      repo.persist(person);
      URI uri = uriInfo.getAbsolutePathBuilder().path(person.getId().toString()).build();
      return Response.created(uri).entity(person).build();
  }

  @PUT
  @Consumes(MediaType.APPLICATION_JSON)
  @Produces(MediaType.APPLICATION_JSON)
  @Transactional
  @Path("/endMembership")
  public Response endMembership(MembershipDTO msDTO){
    repo.getEntityManager().merge(msDTO);
    return Response.ok().entity(msDTO).build();
  }

  @DELETE
  @Path("/addresses/{id}")
  @Produces(MediaType.APPLICATION_JSON)
  public Response deleteMembership(@PathParam("id") Long id){
    repo.deleteByPersonId(id);
    return Response.ok().build();
  }

  //with query param
  @GET
  @Path("/addresses")
  @Produces(MediaType.APPLICATION_JSON)
  public List<Address> getAddresses(@QueryParam("street") String street) {
    return addressRepository.list("street", street);
  }
}
```

Websocket

appication.properties `quarkus.websocket.dispatch-to-worker=true quarkus.http.cors.origins=*`

WebSocketServer

```
@ServerEndpoint("/websocket/{name}")
public class newWebsocketServer {

    Map<String, Session> sessionMap = new ConcurrentHashMap<>();

    @Inject
    SurveyController surveyController;

    @OnOpen
    public void onOpen(Session session,@PathParam("name") String name) {
        System.out.println("Connected ... " + session.getId());
        sessionMap.put(session.getId(), session);
        session.getAsyncRemote().sendObject("Hello "+ name);

    }

    @OnMessage
    public void onMessage(String message, Session session, @PathParam("name") String name) {
        System.out.println("Message from " + session.getId() + ": " + message);
    broadcast(name + ": " + message);

    // in DB speichern
        messageRepo.persist(new Message(message));
    // webSocket.broadcast(message); kann auch zB beim GET in Resource aufgerufen werden



    // Animal JSON decode Beispiel
    Animal animal =  Json.decodeValue(message, Animal.class);
    }

    @OnClose
    public void onClose(Session session, CloseReason closeReason, @PathParam("name") String name) {
        System.out.println(String.format("Session %s close because of %s", session.getId(), closeReason));
        sessionMap.remove(session.getId());
    }

    @OnError
    public void onError(Session session, @PathParam("name") String name,Throwable throwable) {
        System.out.println("Error: " + throwable.getMessage());
    }

    public void broadcast(String message) {
        // Nachricht schicken
        for (Session session : sessionMap.values()) {
            session.getAsyncRemote().sendObject(message);
        }

        // JSON Objekt schicken
        Survey survey = surveyController.getSurvey();
        for (Session session : sessionMap.values()) {
            session.getAsyncRemote().sendObject(Json.encode(survey));
        }
    }
}
```

Resource

```
 @Path("websocketServer/{filialeName}")
    @GET
    @Consumes(MediaType.APPLICATION_JSON)
    public Response webSocketServer(@PathParam("filialeName") String name){
        webSocketServer.broadcast(name);
        return Response.ok().build();
    }
```

Eventbus

```
 @Inject
EventBus eventBus;

eventBus.send("eventName", survey);

// Funktion in ressource
@ConsumeEvent("greeting")
public String consume(String name){
    // do something with name
}
```

ChronoUnit

```
ChronoUnit.DAYS.between(startDate, endDate)
```

Encoder and Decoder

GameWebsocket

```
@ServerEndpoint(value = "/quiz-game-websocket/{gameId}/{name}",
        encoders = {GameEncoder.class})
public class GameWebSocket {
    public void onOpen(Session session, @PathParam("gameId") Long gameId, @PathParam("name") String name) {
        ...

        // Example for sending an object
        session.getAsyncRemote().sendObject(GameMapper.INSTANCE.gameFromEntity(game));
    }

    @OnClose
    public void onClose(Session session, @PathParam("gameId") Long gameId, @PathParam("name") String name) {
        ...
    }

    @OnError
    public void onError(Session session, @PathParam("gameId") Long gameId, @PathParam("name") String name, Throwable throwable) {
        // Error Report
        ...
    }

    @OnMessage
    @Transactional
    public void onMessage(String message,
                          @PathParam("gameId") Long gameId,
                          @PathParam("name") String name,
                          Session session) {
        GameDecoder decoder = new GameDecoder();
        if (decoder.willDecode(message)) {
            try {
                handleAdmin(decoder.decode(message), gameId);
            } catch (DecodeException e) {
                throw new RuntimeException(e);
            }
            return;
        }
    }
}
```

Mapstruct

```
@Mapper
public interface GameMapper {
    GameMapper INSTANCE = Mappers.getMapper(GameMapper.class);

    @Mapping(source = "users", target = "users", qualifiedByName = "usersListFromEntity")
    @Mapping(source = "quiz", target = "quiz", qualifiedByName = "quizFromEntity")
    Game gameFromEntity(GameEntity ge);

    @Mapping(source = "users", target = "users", qualifiedByName = "usersListToEntity")
    @Mapping(source = "quiz", target = "quiz", qualifiedByName = "quizToEntity")
    GameEntity gameToEntity(Game g);


    @Named("usersListFromEntity")
    static List<User> usersListFromEntity(List<UserEntity> users) {
        return users.stream().map(UserMapper.INSTANCE::userFromEntity).collect(Collectors.toList());
    }

    @Named("quizFromEntity")
    static Quiz quizFromEntity(QuizEntity qe) {
        return QuizMapper.INSTANCE.quizFromEntity(qe);
    }

    @Named("usersListToEntity")
    static List<UserEntity> usersListToEntity(List<User> users) {
        return users.stream().map(UserMapper.INSTANCE::userToEntity).collect(Collectors.toList());
    }

    @Named("quizToEntity")
    static QuizEntity quizToEntity(Quiz q) {
        return QuizMapper.INSTANCE.quizToEntity(q);
    }
}
```

Encoder

```
public class GameEncoder implements Encoder.Text<Game> {

    ObjectMapper objectMapper = new ObjectMapper();

    @Override
    public String encode(Game game) throws EncodeException {
        try {
            return objectMapper.writeValueAsString(game);
        } catch (JsonProcessingException e) {
            throw new EncodeException(game, e.getMessage());
        }
    }

    @Override
    public void init(EndpointConfig endpointConfig) {}

    @Override
    public void destroy() {}
}
```

Decoder

```
public class GameDecoder implements Decoder.Text<Game> {
    ObjectMapper objectMapper = new ObjectMapper();

    @Override
    public Game decode(String s) throws DecodeException {
        try {
            return objectMapper.readValue(s, Game.class);
        } catch (JsonProcessingException e) {
            throw new DecodeException(s, e.getMessage());
        }
    }

    @Override
    public boolean willDecode(String s) {
        try {
            objectMapper.readValue(s, Game.class);
        } catch (JsonProcessingException e) {
            return false;
        }
        return true;
    }

    @Override
    public void init(EndpointConfig endpointConfig) {}

    @Override
    public void destroy() {}
}
```