SEW Cheatsheet 5BHITM

Backend

Model

Entity

```
@Entity
// OR PanacheEntity (No definition of ID requiered)
public class Model1 extends PanacheEntityBase {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "model_seq")
    @SequenceGenerator(name = "model_seq", sequenceName = "model_seq", initialValue =
100) // OPTIONAL
    public Long id;

    @Enumerated(EnumType.STRING) //OPTIONAL
    public Enum enumValue;

    @ManyToOne
    @JoinColumn(name = "otherModel_id") //OPTIONAL
    public OtherModel otherModel;

    @OneToMany(mappedBy = "model", fetch = FetchType.EAGER, cascade = CascadeType.ALL)
    public List<Model2> model2List;

    @ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.PERSIST)
    @JoinTable(name = "model1_model2", joinColumns = @JoinColumn(name = "model1_id"),
inverseJoinColumns = @JoinColumn(name = "model2_id")) //OPTIONAL
    public List<Model2> otherModel2List;

    // CONSTRUCTOR etc.
}

public class Model2 extends PanacheEntity {
    //does not need id because of PanacheEntity

    //bi directional relationships with Model1

    @ManyToOne
    public Model1 model;

    @ManyToMany(mappedBy = "otherModel2List")
    public List<Model1> model1List;
```

```
}
```

Inheritance

```
@Entity
@Inheritance(strategy = InheritanceType.JOINED) // or .SINGLE_TABLE or
.TABLE_PER_CLASS
public class ParentModel ...
```

DTO

```
public record ModelDTO(DataType param1, ...) { }
```

Repository

```
@ApplicationScoped
// or PanacheRepository<Model> but then findById etc. only works if the ID of the
model is a Long
public class ModelRepository implements PanacheRepositoryBase<Model, ID_DataType> {
    @Inject
    OtherRepo otherRepo;

    public void panacheFunction() {
        // Panache Functions:
        find("param", param);
        findById(id); // only working if ID is a Long or PanacheRepositoryBase is used
        list();
        listAll();
        delete("param", param);
        deleteById(id); // only working if ID is a Long or PanacheRepositoryBase is
used
        persist(model);
        ...

        // Sort Results
        listAll(Sort.by("attribute").ascending())
        listAll(Sort.by("lastName").and("firstName"));
        ...
    }

    public List<Model> entityManagerFunction() {
        return getEntityManager().createQuery("SELECT m FROM Model m WHERE m.param =
:param", Sport.class)
                .setParameter("param", "Value")
                .getResultList();
    }
```

```
    public List<Model> empPerLocation() {
        TypedQuery<Model> query = getEntityManager().createQuery("select new
at.htl.model.dto.DTO_Name(params, ...) " +
                "from Table t " +
                "left join Table2 t2 on (t2.paramName = t1.paramName)" +
                "...", Model.class);
        return query.getResultList();
    }
}
```

Resource

```
@Path("/path")
public class CourseResource {

    @Inject
    RepoClass Repo

    @GET // or POST, PUT, DELETE
    @Path("/{param}")
    public DataType function(@PathParam("param") DataType param) {
        ...
    }

    @POST
    @Path("/")
    @Transactional
    public Response postFunction(DataType bodyData) {
        ...
        return Response.status(statusCode).entity(data).build();

        // Return a Created Respone with a URI to the object in the location header
        return Response.created(new URI("link/to/object").build();
    }
}
```

Websockets

```
@ServerEndpoint("/socket")
@ApplicationScoped
public class CalenderSocket {
    @Inject
    ObjectMapper objectMapper;

    Set<Session> sessions = new HashSet<>();
```

```
    @OnOpen
    public void onOpen(Session session) {
        sessions.add(session);
        ...
    }

    // Same for OnClose as OnOpen but remove session

    @OnError
    public void onError(Session session, Throwable throwable) {
        ...
    }

    @OnMessage
    public void onMessage(String message, @PathParam("name") String name) {
        ...
    }

    public void broadcast(DataType data) {
        try{
            String data = objectMapper.writeValueAsString(data);
            sessions.forEach(s -> {
                s.getAsyncRemote().sendObject(data, result -> {
                    // check for Exeption -> result.getExeption
                    ...
                });
            });
        }catch (Exception e){
            ...
        }
    }
}
```

Frontend

Angular Commands

```
// Start Project
ng serve

// Generate Component
ng g c component-name

// Generate Service
ng g s service-name

// Generate Interface
```

```
ng g i interface-name
```

Angular Material

```
ng add @angular/material

// Example for Angular Material Component
ng g @angular/material:navigation menu
```

General

Binding

```
// One-Way Binding
{{ value }}

// Two-Way Binding
[(ngModel)]="value"

// Event Binding
(click)="function()"

// Property Binding
[disabled]="isDisabled"
```

onInit

```
class Component implements OnInit {
    ngOnInit() {
        // Code
    }
}
```

ngFor

```
<div *ngFor="let value of values">
...
</div>
```

ngIf

```
<div *ngIf="...">
Please select a school class.
```

```
</div>
```

ngStyle

```
[ngStyle]="{'background-color': isBlue() ? 'blue' : 'green'}"
```

ngClass

```
[ngClass]="{cssClass: someFunction()}"
```

Model

```
export interface Model {
    id: number;
    name: string;
    ...
}
```

Import / Output

parent.component.html

```
<app-children-component [inputName]="data" (outputName)="onEmit($event)"> // $event
sends parameters to parent function
```

children.component.ts

```
// name is optional and <Model> after EventEmitter is optional
@Input("name") parameter: Model = {} as Model;
@Output("name") parameterOutput = new EventEmitter<Model>();
```

Routing

app.routes.ts

```
{path: 'route/:param', component: RouteComponent} // Without / in front of route
{path: '**', component: NotFoundComponent} // Wildcard route
```

navigation

```
// RouterLink with routerLinkActive -> IMPORT in .component.ts
```

```
<div routerLink="/" class="link" routerLinkActive="link-active"
[routerLinkActiveOptions]="{exact: true}">
      HOME
</div>

// RouterOutlet -> IMPORT in .component.html
<router-outlet></router-outlet>
```

params

component.ts

```
route = inject(ActivatedRoute);

// Subscribe to param changes -> in the ngOnInit function
this.route.params.subscribe(params => {
      this.value = params['param'];
})

// Get current param
this.router.snapshot.paramMap.get('param')
```

HttpClient

app.config.ts

```
provideHttpClient() // add to providers
```

http.service.ts

```
constructor(private http: HttpClient) { }

getData() {
    return this.http.get<Model>(API_URL)
}

postData(data) {
    return this.http.post(API_URL, data);
}
```

component.ts

```
httpService = inject(HttpService);

// In load funtion
this.httpService.getData().subscribe((value) => {
    this.data = value;
```

```
});
```

Forms

component.html

```
<form [formGroup]="formName" (ngSubmit)="onSubmit()">
      <div>
        <label for="name">Name:</label>
        <input id="name" formControlName="name" type="text">
        <div *ngIf="studentForm.get('name')?.invalid &&
studentForm.get('name')?.touched">
          Value is invalid.
        </div>
      </div>

      <div>
        <label for="dateValue">DatePicker:</label>
        <input id="dateValue" formControlName="dateValue" type="date">
      </div>

      <div>
        <label for="selectValue">Select:</label>
        <select id="selectValue" formControlName="selectValue">
          <option value="" disabled>Option 1</option>
            // Multiple Options -> *ngFor
        </select>
      </div>

      <button type="submit" [disabled]="formName.invalid">Submit</button>
</form>
```

component.ts

```
formName: FormGroup;

  constructor(private fb: FormBuilder) {
    this.studentForm = this.fb.group({
      name: ['', [Validators.required, Validators.minLength(2)]],
      dateValue: ['', Validators.required],
      selectPicker: ['', Validators.required]
    });
  }

// Get Values
this.formName.value.name

// Reset Form
```

```
this.formName.reset();
```

Websockets

component.ts

```
ngOnInit() {
    const socket = new WebSocket("ws://localhost:8080/socket");
    socket.onmessage = (event: MessageEvent) => {
      const data = JSON.parse(event.data);
      console.log("Socket message: " + data);
      this.courses = data;
    };
    // socket.onopen or .onmessage ...
  }
```

Frontend Functionality

Filter array

component.ts

```
array.filter(value => {
        // check and return true/false
    })
```

Disabled Button

component.html

```
<button [disabled]="!isValid">Speichern</button>
```

component.ts

```
isValid: boolean = false;

checkIfValid() {
    // check and set true/false
    this.isValid = true;
}
```