Wer rustet, der rostet nicht

Über den unkomplizierten Einsatz von Rust in typischen Backendszenarien.

27.09.2024

JUG Saxony Day 2024

sciota

About Us

karol.manterys@sciota.io

tobias.nebel@sciota.io

sciota

IT-Architektur &
Softwareentwicklung

Full Stack Lösungen

Schnittstellen & Protokolle

Smart Products & (Industrial) Internet of Things

Agenda

1. Intro
2. Demo App Konzept & Spring Boot Beispiel
3. Aufbau des Rust Projekts
4. Vergleich & Fazit

Up next

1. Intro
2. Demo App Konzept & Spring Boot Beispiel
3. Aufbau des Rust Projekts
4. Vergleich & Fazit

Intro

- Web App Projekte bisher oft mit Spring Boot Ökosystem (warum auch nicht)
- Beyond Java: Geht da noch was? **Rust als Alternative**?

→ Haben wir ausprobiert, waren positiv überrascht

und fühlen uns stark motiviert, unsere Erkenntnisse zu teilen

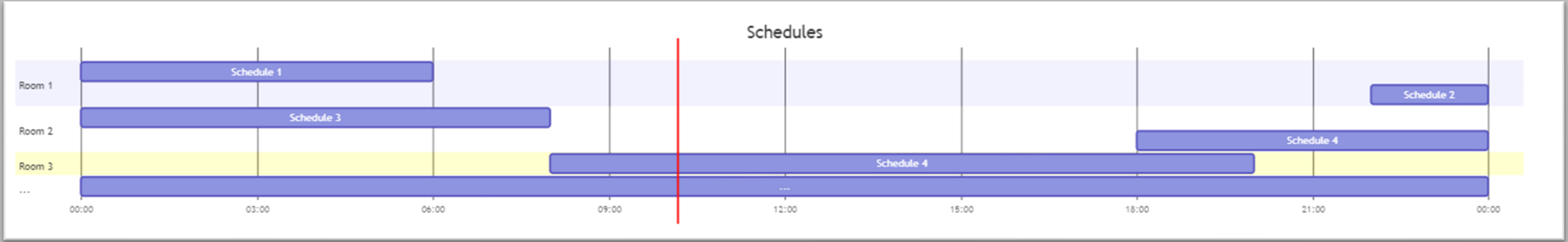
Was euch erwartet

- Wir zeigen euch den Aufbau einer einfachen **Demo-Webanwendung im Rust Kosmos**
    - Zum besseren Verständnis im Vergleich zu Spring Boot
- Wir gehen auf **keine tiefen Sprachspezifika** von Rust ein
- Wir wollen **motivieren**, nicht missionieren
- Es ist **keinerlei Spring-Bashing** vorgesehen ;-)

Up next

Demo App: Alarmservice

sciota



<<Event>>
MOTION_DETECTED

→ **Alarm** in **Room 3**

Schedules

- Schedules definieren, <u>in welchem Raum</u> und <u>zu welchen Zeiten</u> eine Alarmanlage „scharf geschalten ist“
- Eingehende Events („Bewegung erkannt“) werden durch den Server auf Alarmierungsbedarf geprüft; ggf. werden Alarme erzeugt und in der DB persistiert

WebApp: Was kann unser Service?

sciota

CRUD

- Endpunkte für Room, Schedule, Alarm
  - DB Queries
  - Entity ←→ DTO Mapping

technisch

Alarming

- Endpunkt für Events
  - Prüfung der Schedules; ggf. Persistieren eines Alarms in der DB

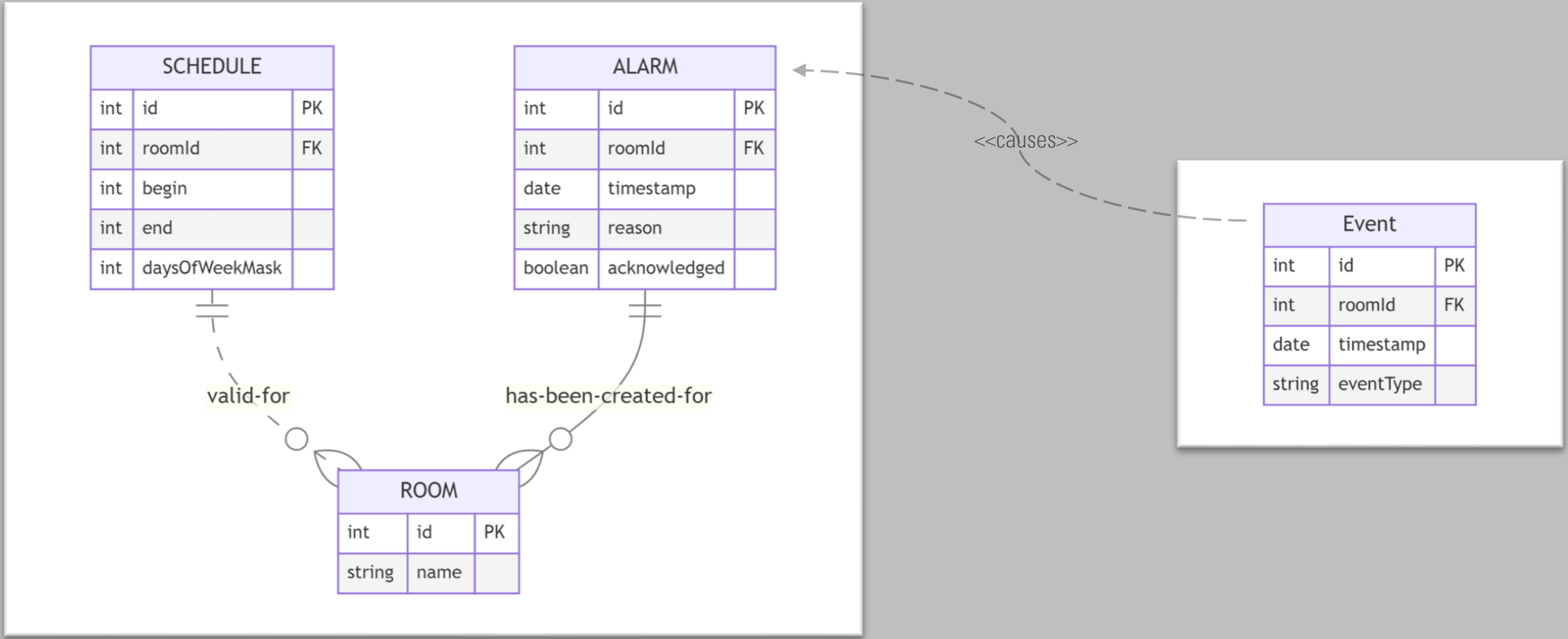
funktional

Ops

- Config File für Ports, DB Credentials, …
- Logging (trace/debug/info/…)
- Metriken
- DB Versioning

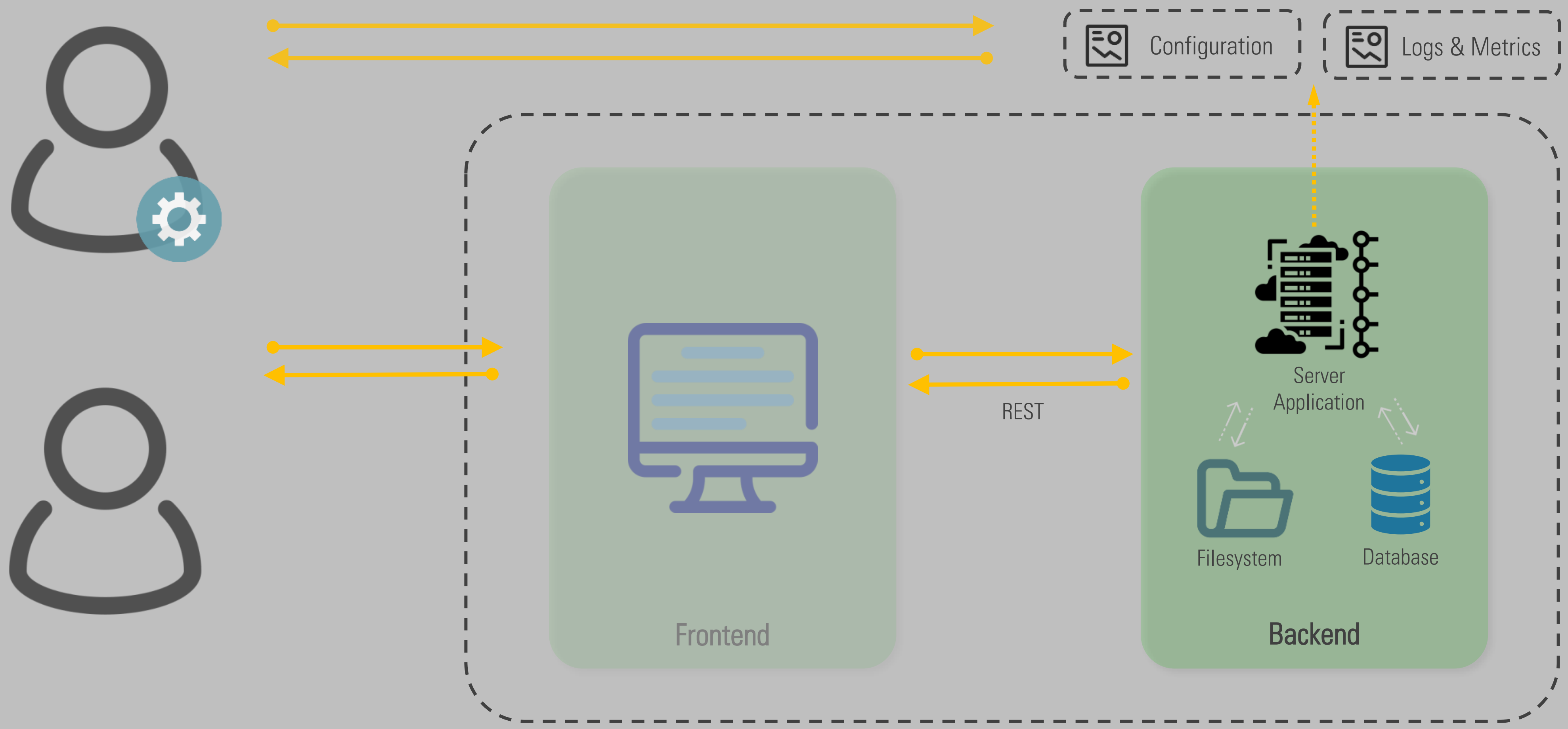
nicht-funktional

DB Entities


SCHEDULE
int id PK
int roomId FK
int begin
int end
int daysOfWeekMask
ALARM
int id PK
int roomId FK
date timestamp
string reason
boolean acknowledged
valid-for
has-been-created-for
ROOM
int id PK
string name
<<causes>>
Event
int id PK
int roomId FK
date timestamp
string eventType

Technisches Scenario

sciota



Techstack: Essentials

sciota

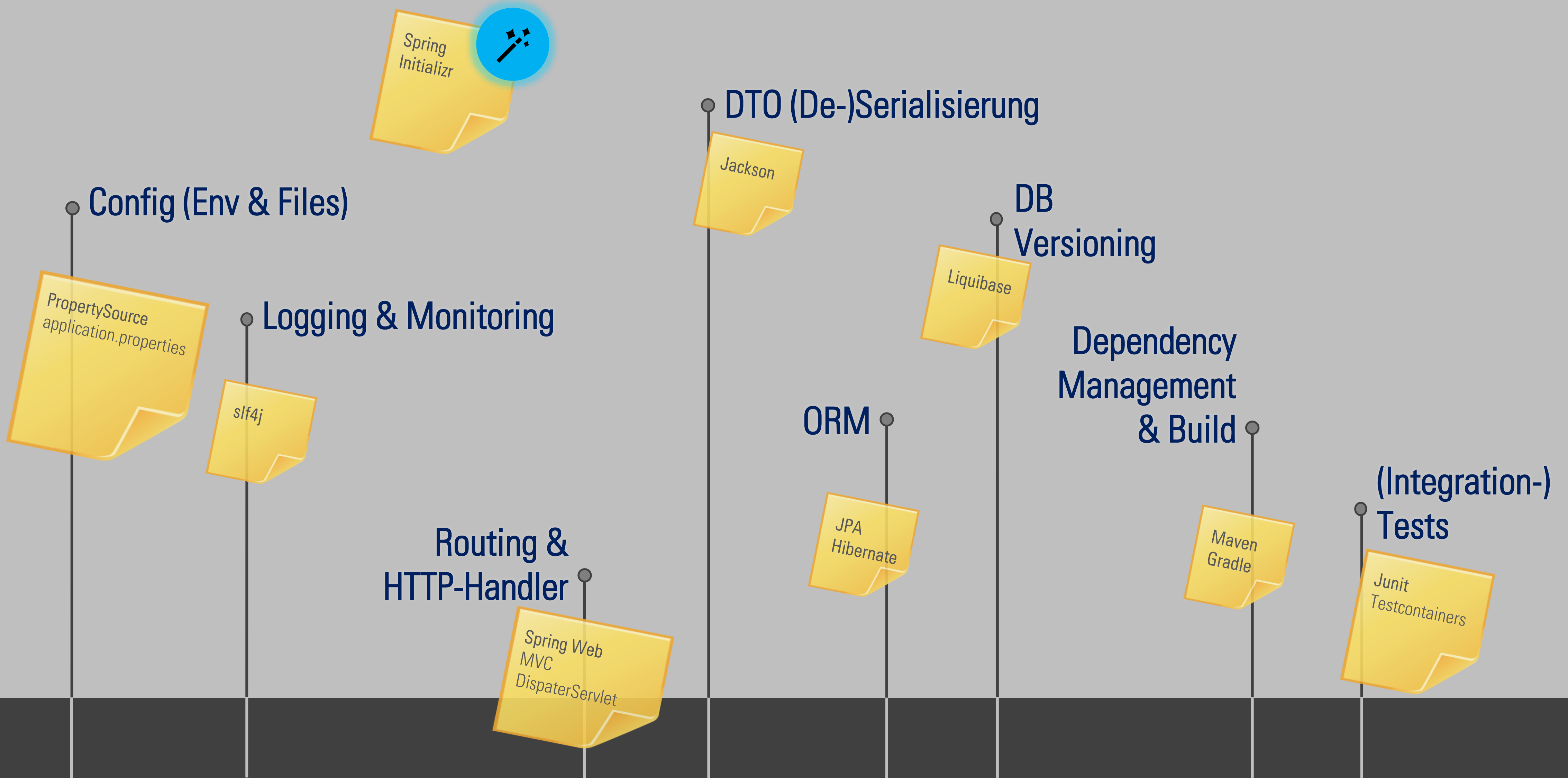
- Config (Env & Files)
- Logging & Monitoring
- Routing & HTTP-Handler
- DTO (De-)Serialisierung
- ORM
- DB Versioning
- Dependency Management & Build
- (Integration-) Tests

Up next

1. Intro
2. Demo App Konzept & Spring Boot Beispiel
3. Aufbau des Rust Projekts
4. Vergleich & Fazit

sciota

Techstack: Spring Boot Web Starter



alarmservice-spring

Demo …

Up next

1. Intro
2. Demo App Konzept & Spring Boot Beispiel
3. Aufbau des Rust Projekts
4. Vergleich & Fazit

sciota

Techstack: Rust

- Config (Env & Files)
- Logging & Monitoring
- Routing & HTTP-Handler
- DTO (De-)Serialisierung
- ORM
- DB Versioning
- Dependency Management & Build
- (Integration-) Tests

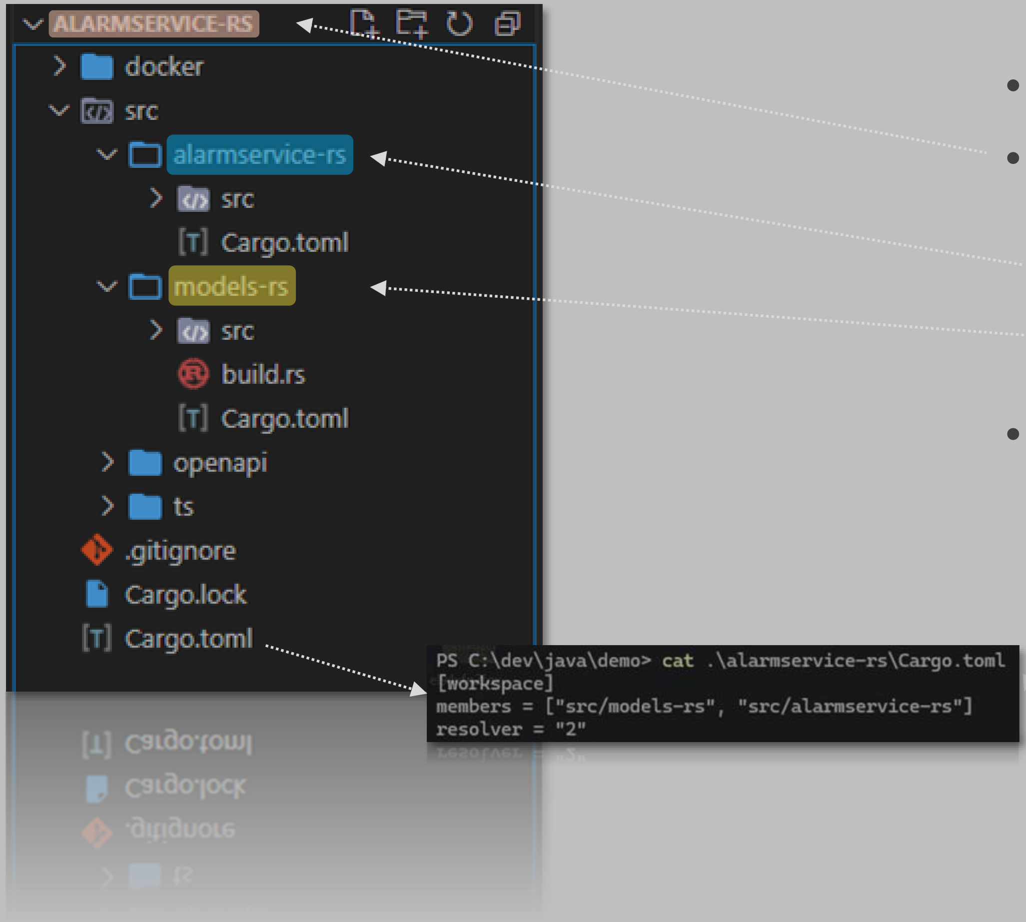
Exkurs: Glossar

sciota

Rust	Java
crate	project
mod / module	package
struct	class
trait	interface
macro	-
derive-macro	annotation, aspect

Projektstruktur (Root)

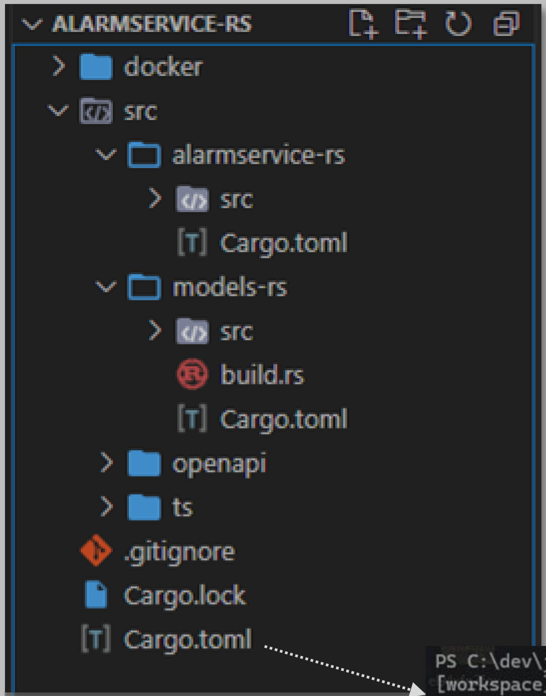
sciota



- „Workspace“ crate als Project Root `wrksp. crate`
- Cargo: Rusts package manager
- `Cargo.toml` referenziert lediglich Sub-crates
  - `alarmservice-rs` (unser Service) `bin crate`
  - `model-rs` (generierte DTOs) `lib crate`
- Cargo sucht nach `Cargo.toml` files in Sub-crates und baut diese individuell

Projektstruktur anlegen

sciota




```
PS C:\dev\java\demo> cat .\alarmservice-rs\Cargo.toml
[workspace]
members = ["src/models-rs", "src/alarmservice-rs"]
resolver = "2"
```

```
user@machine:/demo$ mkdir alarmservice-rs && cd alarmservice-rs
user@machine:/demo/alarmservice-rs$ echo -e "[workspace] \n\
members = ["src/models-rs", "src/alarmservice-rs"] \n\
resolver = \"2\""
user@machine:/demo/alarmservice-rs$ mkdir src && cd src
user@machine:/demo/alarmservice-rs/src$ cargo new alarmservice-rs
user@machine:/demo/alarmservice-rs/src$ cargo new models-rs
```

Techstack: Rust

sciota

Cargo

DTO (De-)Serialisierung

serde

Config (Env & Files)

clap (args)
config-rs (env & files)
serde

DB Versioning

Logging & Monitoring

Dependency Management & Build

ORM

(Integration-) Tests

Cargo

Routing & HTTP-Handler

Apropos Dependencies: cargo add!

sciota

```
user@machine:/demo/alarmservice-rs$ cargo add --package alarmservice-rs clap
user@machine:/demo/alarmservice-rs$ cargo add --package alarmservice-rs config
user@machine:/demo/alarmservice-rs$ cargo add --package alarmservice-rs serde
```

crates.io

```
user@machine:/demo/alarmservice-rs$ cat src/alarmservice-rs/Cargo.toml
[package]
name = "alarmservice-rs"
version = "0.1.0"
edition = "2021"

[dependencies]
clap = { version = "4.5.17", features = ["derive"] }
config = { version = "0.14", features = ["yaml"] }
serde = { version = "1.0.197", features = ["derive"] }
```

Crate ‘clap’

sciota

```rust
mod config;

#[derive(Parser)]
struct Args {
    #[arg(short, long, default_value_t = String::from("config/default"))]
    config: String,
}

fn main() {
    // parse our args and init 'config' (from default or provided arg)
    let args = Args::parse();

    // parse our app config
    // …
}
```

- `main.rs` benötigt CLI-Argument für die Angabe eines individuellen Config Files
- `#[derive(Parser)]` stattet das einfache `struct Args` mit `parse()` Funktionalität aus

```
user@machine:/demo/alarmservice-rs$ cargo run --package alarmservice-rs \
-- --config config/custom.yml

user@machine:/demo/alarmservice-rs$ ./target/release/alarmservice-rs \
--config config/custom.yml
```

Crate ‘config’

sciota

main.rs

```
fn main() {
    // …

    // parse our app config
    let config = AppConfig::parse(args.config).expect("Parsing config failed!");

    // …
```

config.rs

```
use serde::Deserialize;

#[derive(Deserialize)]
pub struct Server {
    pub port: u16,
}

#[derive(Deserialize)]
pub struct Postgres {
    // …
}

#[derive(Deserialize)]
pub struct AppConfig {
    pub server: Server,
    pub postgres: Postgres,
    pub logging: Logging,
}
```

\+

config.rs

```
use config::{Config, ConfigError, File};

impl AppConfig {
    pub fn parse(config_file_path: String)
    -> Result<Self, ConfigError> {
        Config::builder()
            .add_source(File::with_name(
                config_file_path.as_str())
            .required(true))
            .build()?
            .try_deserialize()
    }
}
```

- `main.rs` benötigt eine Konfiguration
- Definition von `AppConfig` mit nested `structs`
- Implementieren der `parse()` Funktion (Glue-Code für config crate)

Crate 'serde'

sciota

```
#[derive(Serialize, Deserialize)]
pub struct MyStruct {
    pub myInt: i32,
    pub myString: String,
}

pub fn do_it() {
    let data = MyStruct {
        myInt: 42,
        myString: "sciota.io".to_string(),
    };
    let serialized = serde_json::to_string(&data).unwrap();
    let deserialized: MyStruct = serde_json::from_str(&serialized).unwrap();
}
```

serde_yml

- (De)Serialisierung von `structs`
- `serde` ist Quasi-Standard
- Wird in vielen anderen crates genutzt bzw. erwartet

Techstack: Rust

sciota

- Cargo
- DTO (De-)Serialisierung
  - serde
- Config (Env & Files)
  - clap (args)
  - config-rs (env & files)
  - (serde)
- DB Versioning
- Logging & Monitoring
  - tracing
  - metrics-prometheus
- Dependency Management & Build
  - Cargo
- ORM
- (Integration-) Tests
- Routing & HTTP-Handler

Crate ‘tracing’

sciota

Initialisierung

```
// generate formatting layer
let pretty_formatting_layer = fmt::Layer::new()
    .pretty()
    .with_target(true)
    .with_line_number(true)
    .with_thread_names(true)
    // …
;

// generate subsciber and set it globally
let subscriber = Registry::default()
    .with(pretty_formatting_layer);
set_global_default(subscriber).expect("Failed to set subscriber!");
```

- Einfaches Logging Framework
- Unterstützt Spans

Logging

```
trace!("Trace Message.");
debug!("Debug Message.");
info!("Info Message.");
warn!("Warn Message.");
error!("Error Message.");
```

Spans

```
let span = span!(Level::TRACE, "my_span").entered();

// … do some heavy work and log about it …

let span = span.exit();
```

Crate ‘axum-prometheus’

sciota

Initialisierung

```
// initialize axum-prometheus
let metric_handle = PrometheusBuilder::new()
        .add_global_label("app", "alertservice-rs")
        .install_recorder()
        .expect("Failed to initialize prometheus metric handle");

// initialize a metric
let events_counter = counter!("events_counter");

// …

events_counter.increment(1);
```

HTTP-Endpunkt

```
let routes: Router = Router::new()

        // … some routes …

        .route("/metrics", get(|| async move { metric_handle.render() }))
        .layer(PrometheusMetricLayer::new());
```

- Einfaches Metrics Framework
- Nutzung ähnlich zu 'tracing‘
- Bringt Renderer für Prometheus mit
- `PrometheusMetricLayer` sammelt Metriken über Endpunkte

Crate ‘axum-prometheus’

sciota

Initialisierung

GET /metrics

```
# TYPE axum_http_requests_total counter
axum_http_requests_total{app="alertservice-rs",method="GET",status="200",endpoint="/metrics"} 5

# TYPE events_counter counter
events_counter{app="alertservice-rs"} 0

# TYPE axum_http_requests_pending gauge
axum_http_requests_pending{app="alertservice-rs",method="GET",endpoint="/metrics"} 1

# TYPE axum_http_requests_duration_seconds summary
axum_http_requests_duration_seconds{app="alertservice-rs",method="GET",status="200",endpoint="/metrics",quantile="0"} 0
axum_http_requests_duration_seconds{app="alertservice-rs",method="GET",status="200",endpoint="/metrics",quantile="0.5"} 0
axum_http_requests_duration_seconds{app="alertservice-rs",method="GET",status="200",endpoint="/metrics",quantile="0.9"} 0
axum_http_requests_duration_seconds{app="alertservice-rs",method="GET",status="200",endpoint="/metrics",quantile="0.95"} 0
axum_http_requests_duration_seconds{app="alertservice-rs",method="GET",status="200",endpoint="/metrics",quantile="0.99"} 0
axum_http_requests_duration_seconds{app="alertservice-rs",method="GET",status="200",endpoint="/metrics",quantile="0.999"} 0
axum_http_requests_duration_seconds{app="alertservice-rs",method="GET",status="200",endpoint="/metrics",quantile="1"} 0
axum_http_requests_duration_seconds_sum{app="alertservice-rs",method="GET",status="200",endpoint="/metrics"} 0.0023729000000000003
axum_http_requests_duration_seconds_count{app="alertservice-rs",method="GET",status="200",endpoint="/metrics"} 5
```

Techstack: Rust

sciota

Cargo

DTO (De-)Serialisierung

serde

Config (Env & Files)

clap (args)
config-rs (env & files)
(serde)

DB Versioning

Logging & Monitoring

tracing
metrics-prometheus

Dependency Management & Build

Cargo

ORM

Routing & HTTP-Handler

axum

(Integration-) Tests

Crate ‘tokio’

sciota

Regular

```
fn main() {
    hello();
}

fn hello() {
    print!("Hello");
}
```

Async

```
#[tokio::main]
async fn main() {
    hello2().await;
}

async fn hello() {
    print!("Hello");
}
```

- Tokio ist eine `async` Runtime
- `async` Funktionen können nur aus async gerufen werden

→ auch `main()` muss `async` sein

Crate 'axum'

sciota

Initialisierung

```
#[tokio::main]
async fn main() {
    // routes
    let app = Router::new()
        .route("/hello", get(|| async { "Hello, World!" }))
        .route("/room/:id", post(post_room_handler))
        .with_state(AppState { db_conn: ... });

    // server
    let listener = tokio::net::TcpListener::bind("0.0.0.0:9001").await.unwrap();
    axum::serve(listener, app).await.unwrap();
}
```

State

```
pub struct AppState {
    pub config:  AppConfig,
    pub db_conn: DatabaseConnection,
    pub metrics: AppMetrics,
    // …
}
```

DTO structs

```
#[derive(Serialize, Deserialize)]
pub struct RoomDto {
    #[serde(rename = "roomId")]
    pub room_id: Option<i64>,
    #[serde(rename = "name")]
    pub name: Option<String>,
}
```

HTTP-Endpunkt

```
pub async fn post_room_handler(
    Path(room_id): Path<i64>,
    Query(q): Query<QueryParams>,
    Json(room_dto): Json<RoomDto>,
    state: State<AppState>,
) -> Result<StatusCode, CustomError> {

    // …

}
```

- Web App Framework
- async: axum nutzt tokio

- `AppState`: einfaches `struct` mit Config und Services aus Initialisierung
- Path- & Query-Params
- Deserialisierter Body

Intermezzo: OpenAPI Specs

sciota

```
user@machine:/demo/alarmservice-rs/src$ java -jar openapi/generator-cli/openapi-generator-cli-7.2.0.jar generate \
--package-name alarmservice \
-g rust \
--model-package dtos \
--model-name-suffix dto \
-o src/models-rs/ \
-i src/openapi/api.yml
```

build.rs

```rust
fn main() {
    let mut openapi_generate_rs_cmd = Command::new("java");
    let result = openapi_generate_rs_cmd
        .arg("-jar")
        .arg(openapi_generator_path)
        .arg("generate")
        .args(["--package-name", "alarmservice"])
        .args(["-g", "rust"])
        .args(["--model-package", "dtos"])
        .args(["--model-name-suffix", "dto"])
        .args(["-o", generated_src_path_rs])
        .args(["-i", openapi_spec_path])
        .stdout(Stdio::piped())
        .output()
        .expect("Cannot run command")
        .status;
    // …

}
```

- Modellgenerierung mittels `openapi-generate-cli`
  - Generator bringt Templates für Rust mit!
- Aufruf der CLI mittels custom build via `build.rs` in der Sub-crate `models-rs`
  - Cargo findet `build.rs` file automatisch und führt dieses aus

Intermezzo: OpenAPI Specs

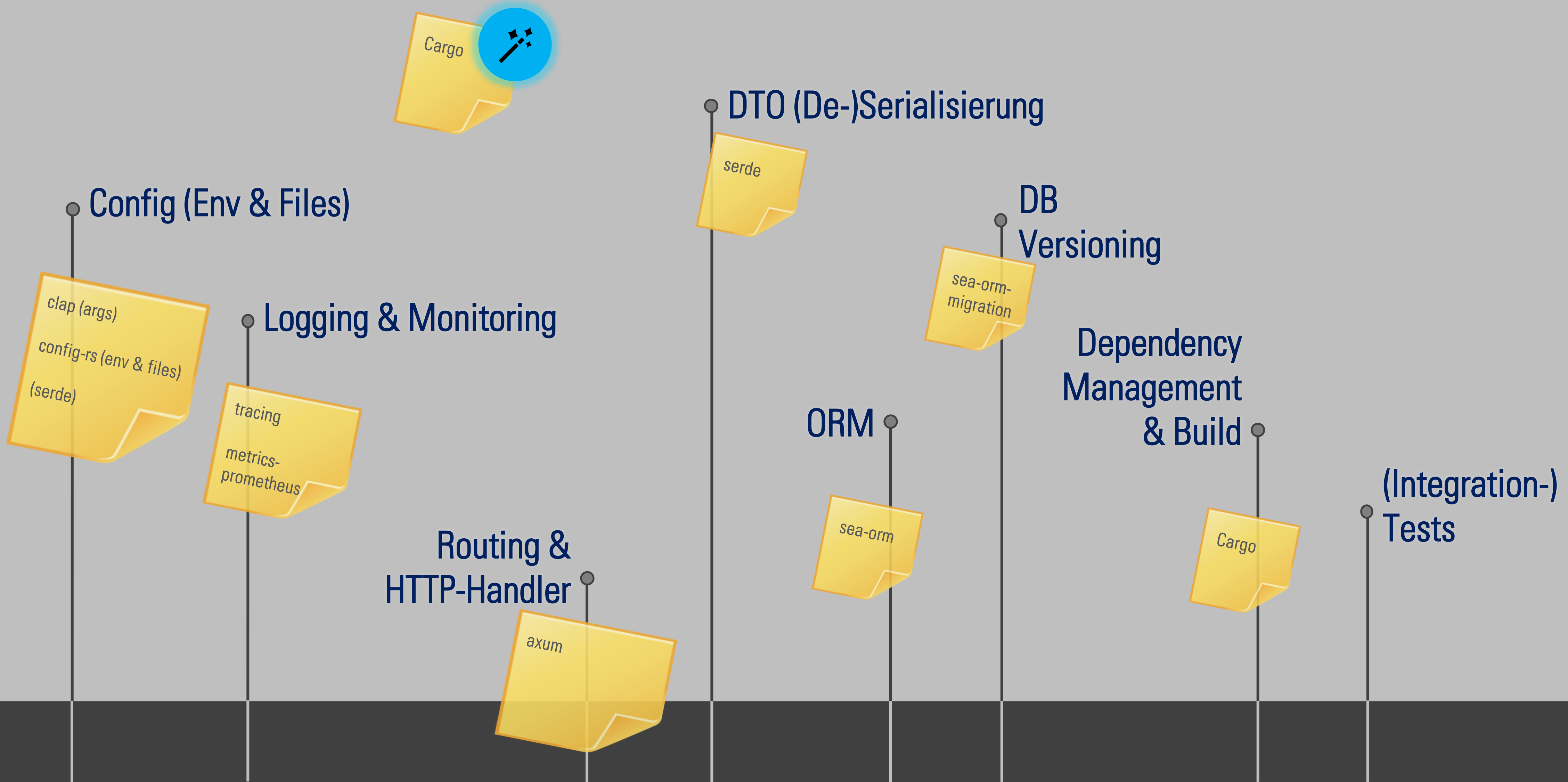
sciota

src/models-rs/src/models/room_dto.rs

```rust
/*
 * Alarmservice Demo
 *
 * No description provided (generated by Openapi Generator https://github.com/openapitools/openapi-generator)
 *
 * The version of the OpenAPI document: 1.0.0
 *
 * Generated by: https://openapi-generator.tech
 */

use crate::models;

use serde_derive::{Deserialize, Serialize};

#[derive(Clone, Default, Debug, PartialEq, Serialize, Deserialize)]
pub struct RoomDto {
    #[serde(rename = "roomId", skip_serializing_if = "Option::is_none")]

    pub room_id: Option<i64>,
    #[serde(rename = "name", skip_serializing_if = "Option::is_none")]

    pub name: Option<String>,
}

impl RoomDto {
    pub fn new() -> RoomDto {
        RoomDto {
            room_id: None,
            name: None,
        }
    }
}
```

sciota

Techstack: Rust



Crate ‘sea-orm’

sciota

Modelldefinition

```rust
#[derive(Clone, PartialEq, DeriveEntityModel, Serialize, Deserialize)]
#[sea_orm(table_name = "room")]
pub struct Model {
    #[sea_orm(primary_key)]
    #[serde(skip_deserializing)]
    pub id: i64,
    pub name: String,
}

#[derive(Copy, Clone, EnumIter, DeriveRelation)]
pub enum Relation {
    #[sea_orm(has_many = "super::alarm::Entity")]
    Alarm,
    #[sea_orm(has_many = "super::schedule::Entity")]
    Schedule,
}

impl Related<super::alarm::Entity> for Entity {
    fn to() -> RelationDef {
        Relation::Alarm.def()
    }
}

impl Related<super::schedule::Entity> for Entity {
    fn to() -> RelationDef {
        Relation::Schedule.def()
    }
}

impl ActiveModelBehavior for ActiveModel {}
```

- ORM für relationale DBs
- Model Definition mittels `structs` und Annotationen
- `Model`
  → Read Operations
- `ActiveModel` (Model+Metadata)
  → Write Operations
- `sea-orm-cli` kann Entities aus DB-Tables generieren

Crate ‘sea-orm’

sciota

ORM Examples

```
// reading
let all_rooms: Vec<room::Model> = room::Entity::find()
    .all(&db_conn)
    .await?;


// filtering
let scheds = schedule::Entity::find()
    .filter(schedule::Column::RoomId.eq(…))
    .all(&db_conn)
    .await?;


// writing
alarm::ActiveModel {
    id: NotSet,
    reason: Set(value.0.event_type),
    timestamp: Set(value.1),
    room_id: Set(value.0.room_id),
    acknowledged: Set(false),
}.save(&db_conn).await?;


// joins
let res = room::Entity::find()
    .select_only()
    .column_as(room::Column::Id, "room")
    .column_as(alarm::Column::Id, "alarm")
    .join(
        sea_orm::JoinType::InnerJoin,
        alarm::Relation::Room.def(),
    )
   .order_by_asc(alarm::Column::Timestamp);
```

- DB Queries über
  - ORM `Entity::find()`
  - SeaQuery `Query::select()`

SeaQuery Examples

```
// reading
let stmt = state.conn.get_database_backend().build(
    &Query::select()
        .column(Room::Id)
        .column(Room::Name)
        .from(Room::Table)
        .to_owned()
);
let res = room::Model::find_by_statement(stmt).all(&state.conn).await?;
```

Crate 'sea-orm-migration'

sciota

main.rs

```
// run migrations
Migrator::up(&db_conn, None).await.expect("Migrations failed!");
```

src/alarmservice-rs/src/persistence/migration/v1_create_tables.rs

```
#[derive(DeriveMigrationName)]
pub struct Migration;

#[async_trait::async_trait]
impl MigrationTrait for Migration {
    async fn up(&self, manager: &SchemaManager) -> Result<(), DbErr> {
        // create 'room' table
        manager
            .create_table(
                Table::create()
                    .if_not_exists()
                    .table(Room::Table)
                    .col(
                        ColumnDef::new(Room::Id)
                            .big_integer()
                            .not_null()
                            .auto_increment()
                            .primary_key(),
                    )
                    .col(ColumnDef::new(Room::Name).string().not_null())
                    .to_owned(),
            )
            .await?;

        // …
```

- Migrations Framework
- Kümmert sich um Ausführung und Persistierung der nötigen Migrations
- „händisch" im Code via `Migrator::`
  - `up()`
  - `down()`
  - `fresh()`
  - `status ()`
  - `refresh ()`
  - `reset ()`
- Oder mittels `sea-orm-cli`
  - Ähnlich `liquibase` Command

Exkurs: Mapping

sciota

Examples

```rust
// our Dto struct
pub struct RoomDto {
    pub room_name: String,
    // ...
}

// our DB Model struct
pub struct Model {
    pub name: String,
    // ...
}

impl From<RoomDto> for Model {
    fn from(value: RoomDto) -> Self {
        Model {
            name: value.room_name,
        }
    }
}

fn map_it() {
    // implicit
    let room_db = Model::from( RoomDto { room_name: "Room_001".to_string() } );

    // explicit type definition
    let room_db: Model = RoomDto { room_name: "Room_001".to_string() }.into();

    // mapping in vectors
    let rooms_db = vec![RoomDto{…}, RoomDto{…}, …]
        .iter()
        .map(Model::from)
        .collect();
}
```

- Mapping mittels `From<T>` und `Into<T>` traits
- `From<T>` und `Into<T>` sind tauschbar
- Ermöglicht saubere Projektstrukturen

Techstack: Rust

sciota

- Cargo

Config (Env & Files)
- clap (args)
- config-rs (env & files)
- (serde)

Logging & Monitoring
- tracing
- metrics-prometheus

Routing & HTTP-Handler
- axum

DTO (De-)Serialisierung
- serde

ORM
- sea-orm

DB Versioning
- sea-orm-migration

Dependency Management & Build
- Cargo

(Integration-) Tests

alarmservice-rs

Demo …

Benchmarks!*

sciota

	Build Package Size	Startup Time	Memory	1.000 Events (2 par. Rqst.)
Rust	~19 MB	~320ms (clean DB) ~160ms (initialized)	~3.8 MB**	~3.4s ~48ms avg.
spring	~75 MB	~9.5s (clean DB) ~8.9s (initialized)	~400..500 MB	~5.8s ~67ms avg.

*)Test auf Windows 11 Entwicklermaschine; kein belastbarer Benchmark!

Benchmarks!*

sciota

	Build Package Size	Startup Time	Memory	1.000 Events (2 par. Rqst.)
	~19 MB	~320ms (cl ~160ms (initialized)	~3.8 MB**	~3.4s ~48ms avg.
spring	~75 MB	~9.5s (clean DB) ~8.9s (initialized)	~400..500 MB	~5.8s ~67ms avg.

*)Test auf Windows 11 Entwicklermaschine; kein belastbarer Benchmark!

Memory*

sciota

alarmservice-rs
(load)

\\BRNO	
Process	alarmservice-rs
% Privileged Time	0,000
% Processor Time	0,000
% User Time	0,000
Creating Process ID	8.216,000
Elapsed Time	629,046
Handle Count	195,000
ID Process	42.620,000
IO Data Bytes/sec	0,000
IO Data Operations/sec	0,000
IO Other Bytes/sec	0,000
IO Other Operations/sec	0,000
IO Read Bytes/sec	0,000
IO Read Operations/sec	0,000
IO Write Bytes/sec	0,000
IO Write Operations/sec	0,000
Page Faults/sec	0,000
Page File Bytes	5.382.144,000
Page File Bytes Peak	5.611.520,000
Pool Nonpaged Bytes	21.864,000
Pool Paged Bytes	181.608,000
Priority Base	8,000
Private Bytes	5.382.144,000
Thread Count	19,000
Virtual Bytes	4.457.078.784
Virtual Bytes Peak	4.460.224.512
Working Set	27.230.208,000
Working Set - Private	4.001.792,000
Working Set Peak	27.516.928,000

alarmservice-spring
(idle)

\\BRNO	
Process	java#2
% Privileged Time	0,000
% Processor Time	0,000
% User Time	0,000
Creating Process ID	7.736,000
Elapsed Time	72,369
Handle Count	735,000
ID Process	24.460,000
IO Data Bytes/sec	0,000
IO Data Operations/sec	0,000
IO Other Bytes/sec	64,474
IO Other Operations/sec	2,015
IO Read Bytes/sec	0,000
IO Read Operations/sec	0,000
IO Write Bytes/sec	0,000
IO Write Operations/sec	0,000
Page Faults/sec	0,000
Page File Bytes	438.730.752,000
Page File Bytes Peak	721.563.648,000
Pool Nonpaged Bytes	48.728,000
Pool Paged Bytes	499.896,000
Priority Base	8,000
Private Bytes	438.730.752,000
Thread Count	63,000
Virtual Bytes	14.891.225.088,0000
Virtual Bytes Peak	14.912.962.560,0000
Working Set	387.571.712,000
Working Set - Private	366.940.160,000
Working Set Peak	592.494.592,000

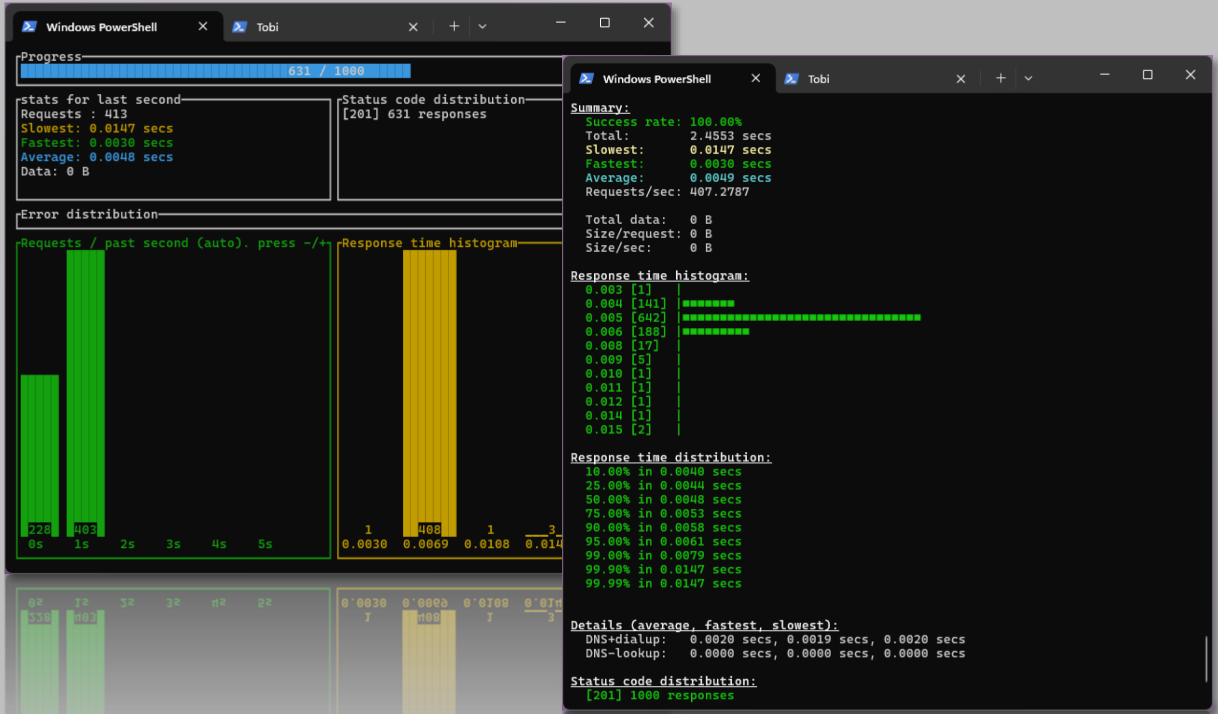
alarmservice-spring
(load)

\\BRNO	
Process	java#2
% Privileged Time	1,572
% Processor Time	86,459
% User Time	84,887
Creating Process ID	7.736,000
Elapsed Time	143,371
Handle Count	730,000
ID Process	24.460,000
IO Data Bytes/sec	47.267,834
IO Data Operations/sec	247,476
IO Other Bytes/sec	55.289,665
IO Other Operations/sec	1.739,371
IO Read Bytes/sec	0,000
IO Read Operations/sec	0,000
IO Write Bytes/sec	47.267,834
IO Write Operations/sec	247,476
Page Faults/sec	24.941,715
Page File Bytes	526.651.392,000
Page File Bytes Peak	721.563.648,000
Pool Nonpaged Bytes	50.936,000
Pool Paged Bytes	500.344,000
Priority Base	8,000
Private Bytes	526.651.392,000
Thread Count	61,000
Virtual Bytes	14.893.182.976,0000
Virtual Bytes Peak	14.912.962.560,0000
Working Set	474.918.912,000
Working Set - Private	454.287.360,000
Working Set Peak	592.494.592,000

*)Test auf Windows 11 Entwicklermaschine; kein belastbarer Benchmark!

oha (おはよう)

- HTTP load generator
- Concurrent requests
- TUI during execution
- Prettified Summary or JSON resultset




```
user@machine:/$ cargo install oha
user@machine:/$ oha -c 2 -n 1000 \
--method POST -H 'Content-Type:application/json' \
-d '{…}' \
http://localhost:9001/event
```

Wrap-up

- Das Rust **Ökosystem** für Web Apps **ist etabliert** und gibt alles her, was man als Entwickler gewöhnt ist
  - Crates sind **exzellent dokumentiert** und liefern **gute Beispiele** mit
- Wir haben mit überschaubarem Aufwand eine **Production-ready** Web App gebaut
- Spring is all about convention - Rust ist expliziter; aber weniger „Magie“ erfordert entsprechende Handarbeit für Verdrahtung (**Gluecode**)
- VSCode + rust-analyzer = ❤️
- Typische **Compilererrors** (Borrow Checking, Lifetimes, usw.) spielen **weniger selten** eine Rolle, als zu befürchten → die Bibliotheken sind gut designed und i.d.R. hervorragend dokumentiert
- **Startupzeit** und **Memory** sind ein Traum; dazu kommt ganz nebenbei die Speichersicherheit

#wasfehltheute

- Unit- und Integrationtests
- Rust bench
- Build / CI
- Loco-rs: „It’s Like Ruby on Rails, but for Rust”

Vielen Dank !

Karol Manterys
sciota GmbH
karol.manetrys@sciota.io

Tobias Nebel
sciota GmbH
tobias.nebel@sciota.io

sciota

Demo Repos

alarmservice-spring
https://github.com/sciotaio/alarmservice-spring

alarmservice-rs
https://github.com/sciotaio/alarmservice-rs

sciota