Teknisk Dokumentation
Team04 Tävlingssystem

Team04

16 februari 2026

Innehåll

1 Introduktion

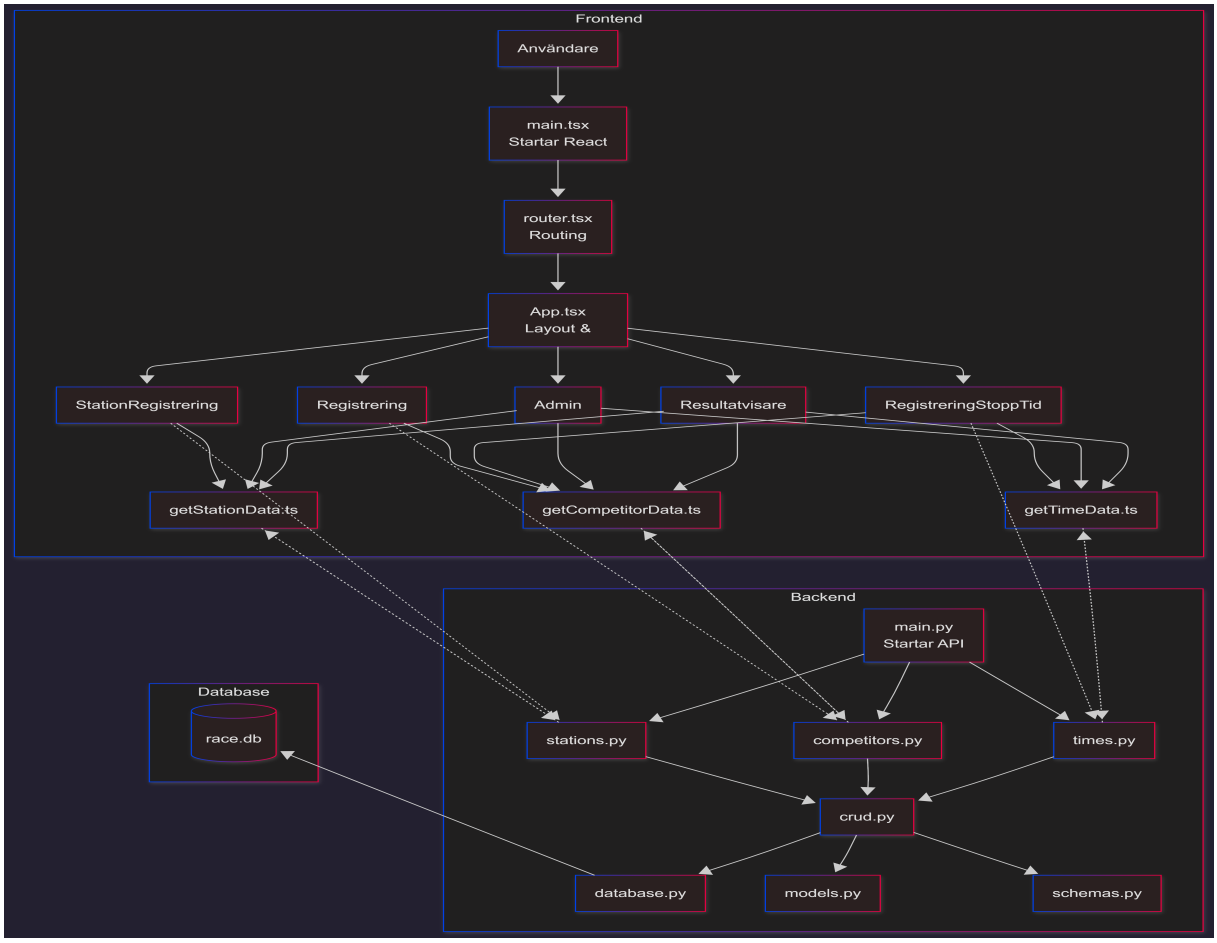
Detta dokument beskriver den överordnade arkitekturen för ett tävlingshanteringssystem. Syftet är att ge en teknisk överblick för framtida utvecklingsteam som ska vidareutveckla eller underhålla systemet.

Dokumentet beskriver:

- Systemets övergripande arkitektur
- Ingående program och filer
- Bygg- och körinstruktioner
- Intern struktur och ansvarsfördelning
- Använda designprinciper och mönster

2 Övergripande Arkitektur och filer

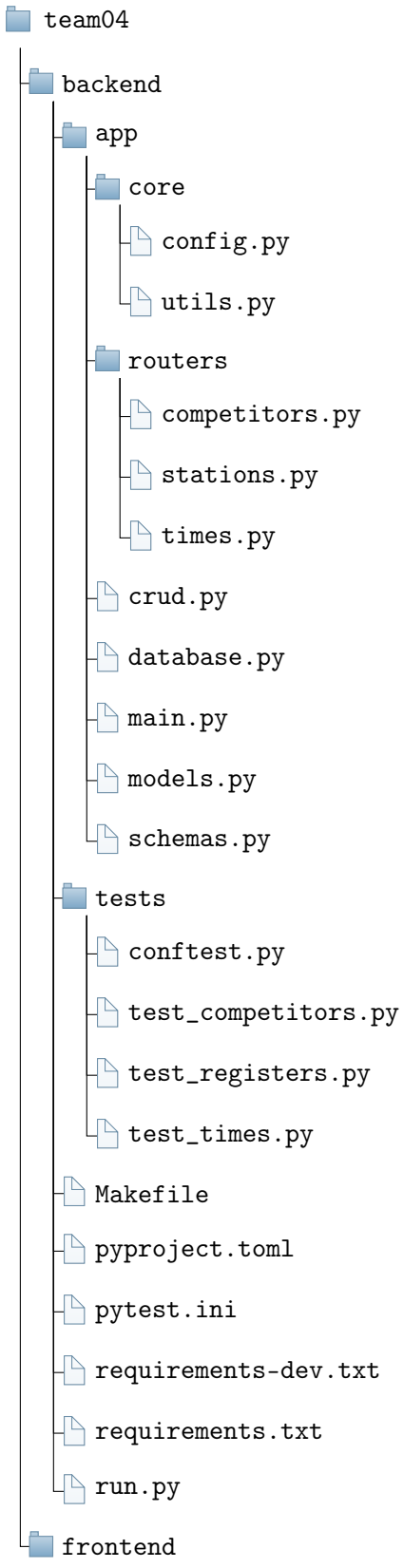
Systemet är uppbyggt enligt en klient–server-arkitektur och består av två huvuddelar: Backend och frontend.

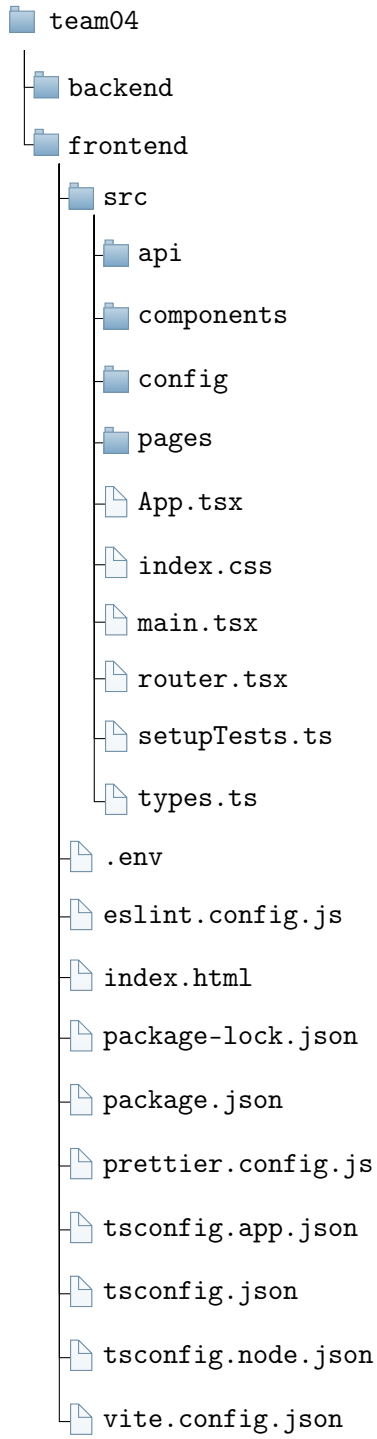


Figur 1: UML av systemets arkitektur. Frontend ansvarar för presentation och användarinteraktion. Backend ansvarar för logik, datavalidering och datalagring.

2.1 Systemets Program och Filer

Nedan visas huvudrepots struktur:




team04
backend
frontend
src
api
components
config
pages
App.tsx
index.css
main.tsx
router.tsx
setupTests.ts
types.ts
.env
eslint.config.js
index.html
package-lock.json
package.json
prettier.config.js
tsconfig.app.json
tsconfig.json
tsconfig.node.json
vite.config.json

2.1.1 Backend

Backend är implementerad i Python med FastAPI och SQLAlchemy.

Fil/Mapp	Ansvar
app/	Applikationskod
models.py	Databasmodeller
schemas.py	Pydantic-scheman (API-kontrakt)
crud.py	Databasoperationer
database.py	Databasanslutning
routers/	API-endpoints
run.py	Startar backend-servern
tests/	Backend-tester

2.1.2 Frontend

Frontend är implementerad i React med TypeScript och Vite.

Fil/Mapp	Ansvar
src/pages/	Applikationens sidor
src/components/	Återanvändbara UI-komponenter
src/api/	API-anrop till backend
router.tsx	Routing-konfiguration
package.json	Projektkonfiguration

3 Bygg och Körning

3.1 Backend

```
cd backend
make install
make run
```

Alternativt:

```
pip install -r requirements.txt
python run.py
```

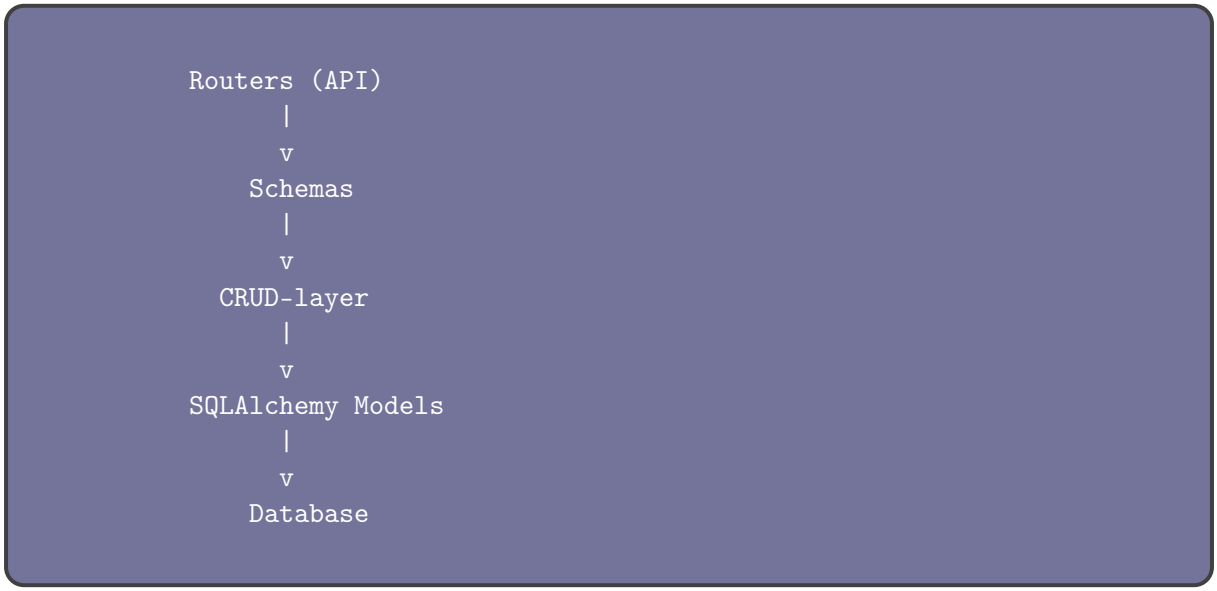
Vänligen notera att det finns en ytterligare fil `requirements-dev.txt` som innehåller utvecklingsberoenden, inklusive testverktyg. Dessa är inte nödvändiga, men används av teamet och är ett smidigt verktyg. För att installera dessa kan följande kommando användas innan du kör python-filen:

```
pip install -r requirements-dev.txt
```

3.2 Frontend

```
cd frontend
npm install
npm run dev
```

4 Backendens Interna Struktur



4.1 Databasmodeller och Relationer

Systemets kärna består av tre huvudsakliga modeller:

4.1.1 Competitor

Representerar en tävlande i systemet.

- id (primärnyckel)
- start_number (unik)
- name

Varje tävlande kan ha flera tidsregistreringar.

4.1.2 Station

Representerar en kontrollstation i tävlingen.

- id (primärnyckel)
- station_name
- order (ordning i tävlingen)

4.1.3 TimeEntry

Representerar en registrerad tid.

- id (primärnyckel)
- competitor_id (foreign key)

- station_id (foreign key)
- timestamp

Relationer:

- En Competitor kan ha flera TimeEntry (1:N).
- En Station kan ha flera TimeEntry (1:N).
- TimeEntry binder ihop tävlande och station.

4.2 Flöde: Från Endpoint till Databas

När en HTTP-request når backend sker följande:

1. Routern tar emot anropet och validerar inkommande data via Pydantic-schema.
2. En databassession skapas via `get_db()`.
3. Routern anropar motsvarande funktion i CRUD-lagret.
4. CRUD-lagret skapar eller hämtar SQLAlchemy-objekt.
5. `db.add()`, `db.commit()` och `db.refresh()` används vid skrivning.
6. Objektet returneras till routern.
7. FastAPI serialiserar objektet till JSON enligt response_model.

4.3 Ansvarsfördelning

- **Routers**: Hanterar HTTP-anrop och returnerar JSON-respons.
- **Schemas**: Definierar datamodeller för API-kommunikation.
- **CRUD**: Utför databasoperationer.
- **Models**: Definierar databasens struktur.

5 Frontendens Interna Struktur

Frontend är uppbyggd enligt en komponentbaserad arkitektur.

```
Pages
  |
  v
Components
  |
  v
API-layer
```

5.1 Ansvarsfördelning

- **Pages**: Representerar applikationens olika vyer. Dessa är kopplade till router och ansvarar för att hämta data och rendera UI
  - Admin sida
  - Registreringssida
  - Tid regristreringssida
  - Stationsregistreringssida
  - Resultatsida
- **Components**: Återanvändbara UI-komponenter.
- **API**: Abstraktion för kommunikation med backend.
  - Både FETCH och PUT anrop används för att hämta och skicka data till backend.

---

6 Kommunikation mellan Delarna

Kommunikationen sker via REST API över HTTP.
Flöde:

1. Frontend skickar HTTP-request.
2. Router tar emot anropet.
3. CRUD-lagret utför databasoperation.
4. Resultat returneras som JSON enligt definierade Pydantic-scheman.

6.1 Exempel på JSON-struktur

6.1.1 Registrera tävlande

Request:

```
POST /api/competitors/register

{
  "start_number": 101,
  "name": "Anna Andersson"
}
```

Response:

```
{
  "start_number": 101,
  "name": "Anna Andersson"
}
```

6.1.2 Registrera tid

Request:

```
POST /api/times/record

{
  "start_number": 101,
  "timestamp": "2025-01-01T12:00:00",
  "station_id": 1
}
```

Response:

```
{
  "id": 15,
  "competitor_id": 3,
  "station_id": 1,
  "timestamp": "2025-01-01T12:00:00"
}
```

6.1.3 Hämta tider för tävlande

Request:

```
GET /api/times/101
```

Response:

```
[
  {
    "id": 15,
    "competitor_id": 3,
    "station_id": 1,
    "timestamp": "2025-01-01T12:00:00"
  }
]
```

---

7 Distribution under Tävling

Under en tävling distribueras systemet enligt följande:

- Backend körs på en central server.
- Frontend körs i webbläsare på klientdatorer.
- Kommunikation sker via lokalt nätverk.

---

8 Vidareutveckling

För att lägga till ny funktionalitet eller uppdatera denna dokumentationen kan följande sektioner hjälpleda utvecklingen.

Vänligen notera att det praktiseras **Test Driven Development** och förutsätts att man följer det för de delar listad nedan. Även refaktorering av kod och uppdatering av dokumentation är en del av vidareutvecklingen, se då till att testerna fortfarande är gröna och att dokumentationen är uppdaterad.

8.1 Backend

1. Skapa ny modell i `models.py` vid behov.
2. Skapa motsvarande schema i `schemas.py`.
3. Implementera CRUD-funktion.
4. Lägg till endpoint i `routers/`.

8.2 Frontend

1. Skapa ny sida i `pages/`.
2. Implementera nödvändiga komponenter.
3. Lägg till API-anrop i `api/`.
4. Registrera route i `router.tsx`.

8.3 Teknisk Dokumentation

Vid större förrändringar av systemet, se till att uppdatera denna tekniska dokumentation. Detta så att den alltid är aktuell och användbar för framtida utvecklingsteam. Detta inkluderar:

- Uppdatera fildiagram och UML vid större förändringar.
  - Vi rekommenderar att använda verktyget Mermaid för att skapa och uppdatera UML-diagramet, men välj fritt själv. Det finns inga krav för specifika verktyg.
- Lägg till beskrivningar av nya moduler och filer.
- Dokumentera API-kontrakt i `schemas.py`.

---

9 Sammanfattning

Systemet är modulärt uppbyggt med tydlig separation mellan presentation, logik och datalager.

Detta dokument syftar till att ge framtida utvecklingsteam en snabb och strukturerad introduktion till systemets uppbyggnad.