UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” IAȘI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DOMENIUL: Calculatoare și tehnologia informației

Platformă de proiectare a rețelelor feroviare în vederea simulării acestora

Temă la disciplina

Baze de date

Student: Obrocea Traian - 1306A

Cadru didactic coordonator: Cătălin Mironeanu

Cuprins

1. Introducere

Această platformă a fost dezvoltată cu scopul de a facilita proiectarea unei rețele feroviare și a părților componente a acesteia, anume:

- Operatori feroviari
- Gări
- Linii
- Tipuri de serviciu (Local, InterCity, InterRegio etc.)
- Trenuri/Rame electrice

În acest scop, platforma dispune de o interfață tabelară și de pagini pentru efectuarea operațiilor CRUD (Create/Read/Update/Delete) asupra entităților menționate mai sus.

2. Tehnologii folosite și instalarea aplicației

1. Back-end

Pentru partea de back-end, s-a folosit Flask, un web framework pentru Python. Dezvoltarea unei aplicații web cu Flask este simplă și intuitivă deoarece nu sunt necesare configurări complexe iar rutele se construiesc prin asociere cu funcții definite de programator.

Pentru conexiunea la baza de date, s-a folosit driver-ul python-oracledb, ce permite interacționarea cu o bază de date de tip Oracle printr-o interfață ce respectă Python Database API.

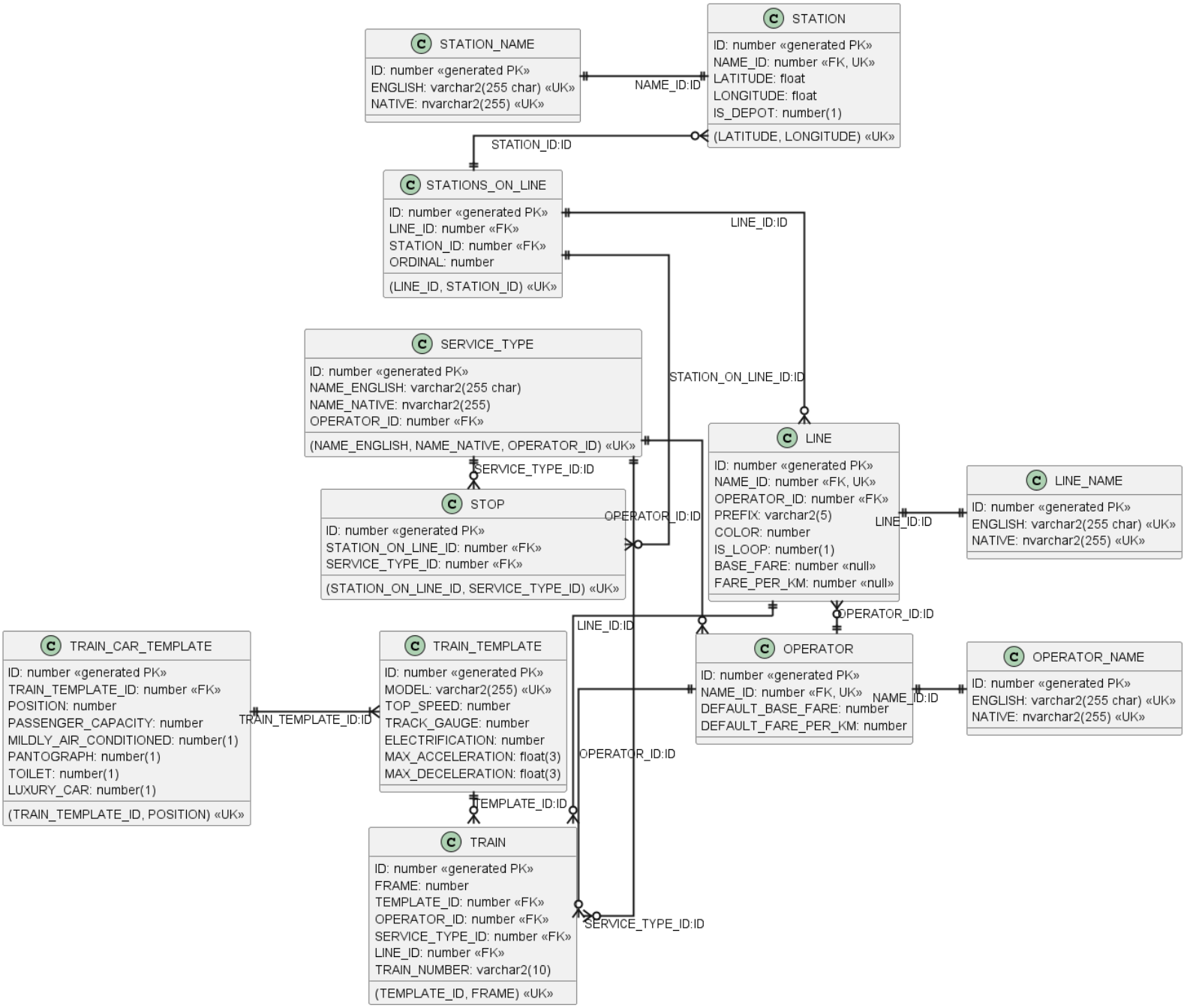
2. Front-end

Pentru partea de front-end, s-a folosit HTML + CSS + JavaScript. Paginile sunt generate de server cu ajutorul procesorului de șabloane Jinja, inclus în Flask, ce permite crearea unei pagini sub formă de șablon HTML, datele fiind introduse în pagină în momentul în care utilizatorul pagina respectivă. Secvențe de cod JavaScript sunt folosite pe unele pagini pentru a îmbunătăți experiența utilizatorului acolo unde nu se putea realiza acest lucru prin HTML (cum ar fi cereri web asincrone)

3. Instalare

- Se instalează Python 3.12+ de pe https://python.org sau de pe repo (pentru Linux, în funcție de disponibilitate)
- Se creează un virtual environment (venv) cu comanda: `$python3.12 -m venv ./venv` (Linux) sau `>py -m venv ./venv` (Windows)
- Se rulează comanda `$source ./venv/bin/activate` (Linux) sau `>./venv/Scripts/Activate` (Windows) pentru a activa venv-ul
- Se rulează comanda `pip install -r requirements.txt` pentru a instala toate dependințele
- Se lansează aplicația cu comanda `python main.py`

3. Structura tabelelor



În cadrul proiectării tabelelor, pentru fiecare entitate s-au ales informații relevante inițializării unei simulări, nu date care ar apărea spre exemplu într-o aplicație de tip ERP.

S-a ales folosirea unei coloane de tip autoincrement drept cheie primară pentru toate tabelele din mai multe motive:

- Simplificarea tabelelor și a lucrului cu acestea prin eliminarea cheilor primare compuse acolo unde ar fi apărut
- Separarea „business logic” de cheia primară
- Simplificarea aplicației prin existența unei chei primare de un singur tip

Pentru toate tabelele _name s-au folosit constrângeri UNIQUE pentru a preveni introducerea a două entități cu același nume. Pentru câmpurile de tip boolean (care sunt de fapt de tip NUMBER(1), s-au folosit constrângeri de tip CHECK pentru a verifica că valorile introduse sunt 0 sau 1. Toate atributele de tip _id au constrângeri de tip FOREIGN KEY către tabela numită.

1. Operator

În cadrul acestei tabele sunt definiți operatori feroviari și tarifele lor implicite

Tabela operator_name este în 3NF deoarece fiecare valoare este atomică și fiecare coloană admite valori unice (nu există atribute non-cheie)

Tabela operator este în 3NF deoarece fiecare valoare este atomică, foreign key-ul către nume este unic (este cheie candidat), atributele non-cheie (tarifele) depind de cheia primară și nu depind de atribute non-cheie.

Constrângeri:

- UNIQUE_OPERATOR_NAME pentru a preveni existența a doi operatori care împar

2. Station

În cadrul acestei tabele sunt definite gările/stațiile, locațiile acestora și dacă beneficiază de depou.

Tabela station_name este în 3NF analog operator_name.

Tabela station este în 3NF analog operator.

Constrângeri:

- UNIQUE_NAME analog
- UNIQUE_POSITION pentru a preveni existența a două stații în aceeași poziție
- POSITION_IN_RANGE pentru a verifica încadrarea coordonatelor în limitele [-90, 90] de grade respectiv [-180, 180] de grade

3. Line

În cadrul acestei tabele sunt definite căile feroviare, operatorul deținător, tarifele specifice (dacă nu se folosesc cele implicite operatorului) și date cu privire la afișarea pe hartă (prefixul adăugat stațiilor și culoarea folosită).

Tabela line_name este în 3NF analog operator_name

Tabela line este în 3NF analog analog operator.

Constrângeri:

- COLOR_IN_RANGE: culoarea este stocată sub forma unui întreg de tip 0xRRGGBB, se verifică dacă valoarea introdusă este între 0x000000 și 0xFFFFFF

4. Stations_on_line

Aceasta este o tabelă de mapare care realizează legătura many-to-many între stații și linii (o linie trece prin mai multe stații iar o stație poate fi străbătută de mai multe linii) cu un număr de ordine („Stația X este a Y-a statie pe linia Z”)

Tabela stations_on_line este în 1NF deoarece toate valorile sunt atomice.

În plus, este în 2NF deoarece este în 1NF iar atributul non-cheie (numărul de ordine) depinde în întregime de cheia candidat (line_id, station_id).

În final, este în 3NF deoarece atributul non-cheie depinde în mod direct de cheia primară.

Constrângeri:

- UNIQUE_LINE_STATION_COMBINATION: interzice asocierea unei stații cu o linie de mai multe ori. Această constrângere previne crearea unei linii care se autointersectează (caz întâlnit în viața reală), dar s-a decis ca aplicația să nu implementeze această funcționalitate
- ORDINAL_POSITIVE: se impune ca numărul de ordine să fie pozitiv. Teoretic, nu este necesară această limitare dar sistemul este mai ușor de înțeles daca se poate spune că numărul de ordine reprezintă spre exemplu poziția stației într-o listă.

5. Service_type

În această tabelă sunt definite regimurile sub care circulă trenurile fiecărui operator (Un tren care circulă sub un anumit regim s-ar putea să nu oprească la o anumită stație de-a lungul unei linii.

Tabela service_type este în 3NF deoarece toate valorile sunt atomice și nu există atribute non-cheie.

Constrângeri:

- UNIQUE_NAME_PER_OPERATOR: Un operator nu poate avea două regimuri cu nume identice, dar doi operatori pot avea câte un regim cu același nume.

6. Stop

În această tabelă sunt definite opririle fiecărui regim de circulație a trenurilor la o stație de-a lungul unei linii („Trenul de tip X care circulă pe linia Y oprește la stația Z”).

Tabela stop este în 3NF deoarece toate valorile sunt atomice și nu există atribute non-cheie.
Constrângeri:

- UNIQUE_STOPS (service_type_id, station_on_line_id): Previne oprirea unui tip de tren la o anumită stație de două ori pe aceeași linie. Conform explicației de la UNIQUE_LINE_STATION_COMBINATION, se consideră că o linie nu se autointersectează, deci tabela Stop nu conține un număr de ordine pentru fiecare oprire, ordinea fiind dată de ordinea stațiilor pe lini. Astfel, opririle repetate ar fi o eroare.

7. Train_template

În această tabelă sunt definite familii de trenuri (specificații tehnice, numărul de vagoane)

Tabela train_template este în 3NF deoarece toate valorile sunt atomice deoarece nu există chei candidat compuse iar toate atributele non-cheie depind în mod direct de cheia primară.

Constrângeri:

- UNIQUE_MODEL: nu se permite definirea a două familii de tren cu același nume. Acest caz nu este neapărat neplauzibil, dar s-a ales a nu include producătorul familiei în baza de date deoarece nu este relevant pentru o simulare, așa că două familii cu nume identice dar diferite prin restul atributelor ar crea confuzie.

8. Train_car_template

În această tabelă sunt definite vagoanele specifice unei familii definite în tabela precedentă și atributele fiecărui vagon, cum ar fi dacă dispune de pantograf sau toaletă. În plus, este definită familia de care aparține (FK către train_template) și poziția de-a lungul trenului unde se află.

Tabela train_car_template este în 3NF deoarece toate valorile sunt atomice, atributele non-cheie depind în întregime de cheia candidat (train_template_id, position) și depind în mod direct de cheia primară.

Constrângeri:

- NO_2_CARS_IN_SAME_POSITION pentru a preveni existența a două vagoane în aceeași poziție de-a lungul trenului
- POSITIVE_(PASSENGER_CAPACITY/POSITION) pentru a verifica că aceste valori sunt pozitive (deci au sens)

9. Train

În această tabelă sunt definite unitățile existente ale familiilor definite anterior și detalii specifice fiecărui tren în parte:

- „frame” – un număr specificat de producător pentru a identifica în mod unic acea unitate
- Familia de care aparține
- Operatorul deținător
- Regimul sub care circulă
- Linia pe care circulă
- „train_number” – un identificator specific operatorului pentru a identifica o unitate care se poate schimba (spre exemplu, analog mașinilor de poliție, „frame” este VIN-ul iar „train_number” este „autospeciala 56”)

Tabela train este în 3NF deoarece toate valorile sunt atomice, atributele non-cheie depind în întregime de cheia candidat (frame, train_template_id) și depind în mod direct de cheia primară.

Constrângeri:

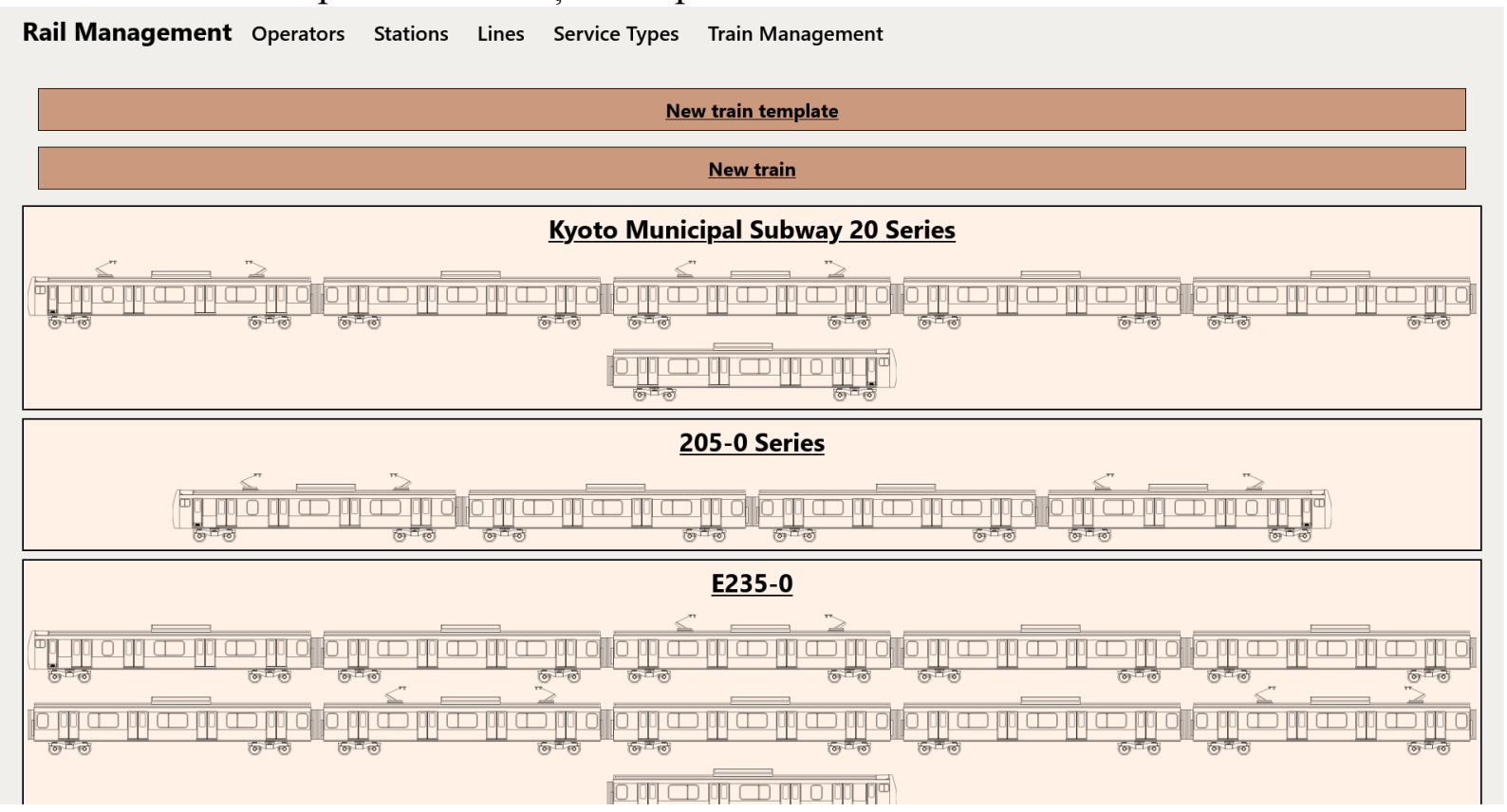
- UNIQUE_FRAME_PER_TEMPLATE: nu se permit două trenuri cu același număr unic de identificare

4. Conectarea la baza de date din aplicație

Pentru conectarea la baza de date Oracle, s-a folosit driver-ul python-oracledb. Funcția `connect()` din modulul util apelează funcția corespunzătoare driver-ului. Numele de utilizator și parola sunt stocate în fișierul .env, de unde sunt încărcate de aplicație. Pentru controlul tranzacțiilor dispunem de metodele commit() și rollback().

```
def connect():
    global connection
    if connection is None:
        connection = oracledb.connect(
            user=os.getenv("ORACLE_USER"),
            password=os.getenv("ORACLE_PW"),
            host="81.180.214.85",
            port=1539,
            service_name="orcl",
            conn_class=ConnectionWithLogging,
        )
```

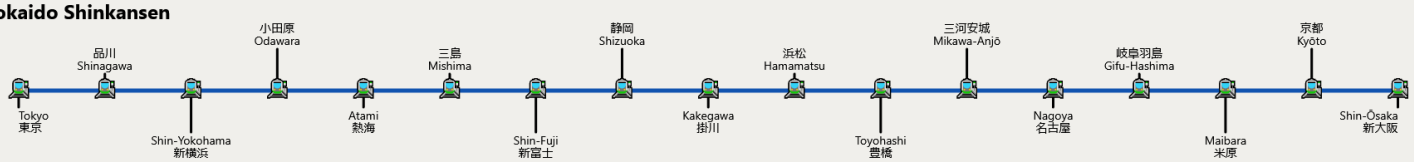
5. Capturi de ecran și exemple de cod



Lista familiilor de trenuri

Rail Management Operators Stations Lines Service Types Train Management

Edit Tokaido Shinkansen



Name (English)	Tokaido Shinkansen
Name (Native)	東海道新幹線
Prefix	T
Operator	JR Central/JR東海
Base fare (default: 110)	
Fare per km (default: 11)	
Color	
Is loop?	☐
Stations	× Tokyo/東京 × Shinagawa/品川 × Shin-Yokohama/新横浜 × Odawara/小田原 × Atami/熱海 × Mishima/三島 × Shin-Fuji/新富士 × Shizuoka/静岡 × Kakegawa/掛川 × Hamamatsu/浜松 × Toyohashi/豊橋 × Mikawa-Anjō/三河安城 × Nagoya/名古屋 × Gifu-Hashima/岐阜羽島 × Maibara/米原 × Kyōto/京都 × Shin-Ōsaka/新大阪

Submit Changes

Cancel

Delete

Editarea unei linii

```python
@app.route( rule: "/stations/delete/<int:obj_id>", methods=["DELETE"])
async def delete_station(obj_id):
    # Implications of removing a station:
    #    We delete all rows from stations_on_line involving this station
    #    That requires wiping all stops using this station
    #    Finally we delete the station itself

    station: Station = im[Station][obj_id]
    with util.connection.cursor() as cs:
        # Collect all station_on_line ID's involving this station
        station_on_line_rows = cs.execute(
             statement: "SELECT id FROM stations_on_line WHERE station_id = :id",  parameters: {"id": station.id}
        ).fetchall()
        station_on_line_ids: list[ID] = [row[0] for row in station_on_line_rows]

        # Delete all stops involving this station
        if station_on_line_ids:
            cs.executemany( statement: "DELETE FROM stop WHERE station_on_line_id = :id", [{"id": i} for i in station_on_line_ids])

        # For every line we affect by deleting the station, we have to update the ordinal of all following stations
        # First, find affected lines
        line_rows = cs.execute(
             statement: "SELECT line_id, ordinal FROM stations_on_line WHERE station_id = :id",  parameters: {"id": station.id}
        ).fetchall()
        line_ids: list[ID] = [row[0] for row in line_rows]
        ordinals: list[int] = [row[1] for row in line_rows]

        for line_id, ordinal in zip(line_ids, ordinals):
            # Drop the station from each line
            cs.execute(
                 statement: "DELETE FROM stations_on_line WHERE station_id = :station_id AND line_id = :line_id",
                 parameters: {"station_id": station.id, "line_id": line_id},
            )

            # Decrement the ordinal from each station after the one we deleted
            cs.execute(
                 statement: "UPDATE stations_on_line SET ordinal = ordinal - 1 WHERE line_id = :line_id AND ordinal > :ordinal",
                 parameters: {"line_id": line_id, "ordinal": ordinal},
            )

        cs.execute( statement: "DELETE FROM station WHERE id = :id",  parameters: {"id": station.id})
        cs.execute( statement: "DELETE FROM station_name WHERE id = :id",  parameters: {"id": station.name.id})
    util.connection.commit()

    # no exception means success, we can update our local copy

    # Remove all stops using this station
    for stop in list(im[Stop].values()):
        if stop.station.id == station.id:
            del im[Stop][stop.id]

    # Remove station from all lines
    for line in list(im[Line].values()):
        if station in line.stations:
            line.stations.remove(station)

    # Remove station
    del im[StationName][station.name.id]
    del im[Station][obj_id]
    return "", 204, ""
```

Logica de ștergere a unei stații

```python
def train_template_get_messages_for_invalid_params(form: dict[str, str], car_count: Optional[int] = None):
    model = form["model"]
    top_speed = float(form["top_speed"])
    track_gauge = int(form["track_gauge"])
    electrification = int(form["electrification"])
    max_acceleration = float(form["max_acceleration"])
    max_deceleration = float(form["max_deceleration"])

    conditions = [
        (0 < len(model) <= 255, "Model name must be shorter than 255 characters can't be empty"),
        (0 < top_speed, "Top speed must be positive"),
        (0 < track_gauge, "Track gauge must be positive"),
        (0 < electrification, "Electrification must be positive"),
        (0 < max_acceleration, "Max acceleration must be positive"),
        (0 < max_deceleration, "Max deceleration must be positive"),
        (car_count is None or 0 < car_count, "Car count must be positive"),
    ]

    return [c[1] for c in conditions if not c[0]]
```

Logica de verificare a parametrilor pe back-end pentru actualizarea unei familii de trenuri

```python
@app.route( rule: "/service-types/delete/<int:obj_id>", methods=["DELETE"])
async def delete_service_type(obj_id: int):
    # Deleting a service types involves deleting all stops associated with it
    # This should error if any trains are still assigned to this service type
    # to allow the user to reassign them
    with util.connection.cursor() as cs:
        try:
            # Delete associated stops
            cs.execute(
                 statement: """
                DELETE FROM stop
                WHERE service_type_id = :id
            """,
                 parameters: {"id": obj_id},
            )

            # Delete the service type, this will fail if any trains are still associated
            cs.execute(
                 statement: """
                DELETE FROM service_type
                WHERE id = :id
            """,
                 parameters: {"id": obj_id},
            )

        except DatabaseError as e:
            if "02292" in str(e):  # child record found
                return "This service type cannot be deleted as it is in use by at least one train", 409, ""
            raise
    util.connection.commit()
    for stop in list(im[Stop].values()):
        if stop.service_type.id == obj_id:
            del im[Stop][stop.id]
    # if we're here, the trains have been modified in the relevant places
    del im[ServiceType][obj_id]
    return "", 204, ""
```

Logica de ștergere a unui regim de circulație