Organizator distribuit de orare

Roibu Radu-Gheorghe

1 Context si motivatie

Proiectul *Organizator distribuit de orare* isi propune sa modeleze si sa gestioneze, intr-un mod scalabil si robust, orarele scolare / universitare (discipline, grupe, sali, profesori si intervale orare), folosind o arhitectura bazata pe microservicii si tehnologii specifice sistemelor concurente si distribuite.

Pe masura ce numarul disciplinelor, grupelor si salilor creste, iar constrangerile de orar devin tot mai complexe (evitarea suprapunerilor, respectarea disponibilitatilor profesorilor, folosirea eficienta a salilor), o solutie monolitica devine greu de extins si de mentinut. O platforma distribuita permite:

- procesarea simultana a mai multor cereri de generare sau actualizare a orarului;
- scalarea orizontala a componentelor critice (de exemplu, motorul de generare a orarelor);
- separarea clara a responsabilitatilor intre servicii si o mai buna rezilienta la erori.

Structura proiectului respecta cerintele din enuntul oficial al proiectului SCD 2025, privind utilizarea autentificarii SSO, a microserviciilor containerizate, a unei baze de date relationale si a cel putin unui modul scalabil orizontal cu functionalitati avansate specifice sistemelor concurente si distribuite.

2 Obiectivul proiectului

Scopul proiectului este de a realiza o platforma web distribuita care sa ofere tuturor actorilor implicati (secretariat, profesori, studenti) functionalitatile esentiale pentru lucrul cu orarele:

- administrarea disciplinelor, grupelor, profesorilor si salilor;
- definirea si actualizarea orarelor, cu verificarea automata a suprapunerilor;
- generarea automata de orare pe baza unor constrangeri si preferinte;
- notificarea utilizatorilor cu privire la modificarile aparute in orar;
- expunerea unei interfete REST API care poate fi folosita atat de un frontend web, cat si de instrumente de automatizare sau integrare.

3 Arhitectura generala a solutiei

3.1 Vedere de ansamblu

Solutia va fi implementata ca un set de microservicii, fiecare rulat in propriul container Docker si orchestrate folosind Docker Swarm. Fiecare componenta va fi definita ca serviciu in fisierul

de tip stack (`.yml`), iar comunicatia dintre servicii se va realiza prin numele DNS generate de Docker si prin variabile de mediu.

Arhitectura include urmatoarele categorii de componente:

- **Servicii open-source**:
  - **Keycloak** – serviciu de autentificare SSO si management de identitati;
  - **PostgreSQL** – baza de date relationala principala;
  - **RabbitMQ** – broker de mesaje pentru procesare asincrona;
  - **Nginx / API Gateway** – punct unic de intrare pentru apelurile HTTP catre microservicii;
  - (optional) **Redis** – cache pentru orare frecvent accesate;
  - (optional) **Prometheus** si **Grafana** – colectare si vizualizare metrice pentru monitorizare.
- **Microservicii dezvoltate** (componente proprii):
  - **Timetable Management Service**;
  - **Scheduling Engine Service**;
  - **Notifications Service** (optional, dar planificat).

Toate serviciile vor fi conectate in una sau mai multe retele Docker de tip overlay, configurate astfel incat fiecare serviciu sa poata comunica doar cu serviciile necesare (de exemplu, serviciile de business vor comunica cu baza de date si cu RabbitMQ).

3.2 Containerele ce vor fi folosite

La nivel de stack Docker Swarm, vor fi definite cel putin urmatoarele servicii:

- **keycloak-service** – instanta Keycloak pentru SSO;
- **db-service** – baza de date PostgreSQL;
- **rabbitmq-service** – broker de mesaje RabbitMQ;
- **api-gateway-service** – Nginx sau alt gateway HTTP;
- **timetable-service** – microserviciu propriu pentru gestionarea orarelor;
- **scheduling-engine-service** – microserviciu propriu pentru generarea automata a orarelor (modul scalabil orizontal);
- **notifications-service** – microserviciu propriu pentru procesarea evenimentelor si trimiterea de notificari;
- (optional) **redis-service** – cache pentru orare si rezultate frecvent accesate;
- (optional) **monitoring-services** – Prometheus, Grafana.

3.3 Componente proprii (microservicii de business)

Timetable Management Service. Acest serviciu expune API-urile principale pentru gestionarea entitatilor din domeniu (discipline, grupe, sali, profesori, sloturi orare) si pentru administrarea orarelor efective. El va asigura logica de validare a cererilor de modificare a orarului, verificari simple de suprapuneri si interactiunea cu baza de date prin intermediul unui ORM (de exemplu, SQLAlchemy).

Scheduling Engine Service. Acest serviciu reprezinta motorul de generare si ajustare automata a orarelor. El primeste cereri de generare (sau recalculare) a orarului pentru o anumita grupa, un anumit set de discipline sau pentru un intreg semestru. Cererile vor fi plasate intr-o coada RabbitMQ, iar una dintre replicile acestui serviciu va procesa job-ul, va aplica algoritmi de alocare si va salva rezultatele in baza de date. Serviciul va fi configurat cu mai multe replici in Swarm, reprezentand modulul scalabil orizontal al aplicatiei.

Notifications Service. Acest serviciu consuma evenimente din brokerul de mesaje (de exemplu, "orar generat", "curs mutat", "profesor indisponibil") si declanseaza notificari catre utilizatori (email, mesaje interne in aplicatie sau alte mecanisme). De asemenea, el poate genera rapoarte periodice sau agregate privind modificarile de orar.

4 Module si functionalitati

4.1 Module comune

1. **Modul de autentificare** (Keycloak):
   - gestionarea autentificarii utilizatorilor (SSO, OAuth2 / OpenID Connect);
   - emiterea de token-uri de acces utilizate de microserviciile proprii.
2. **Modul de profil utilizator si roluri** (User Profile integrat in Timetable Management Service):
   - extrage informatii din token-ul Keycloak;
   - creeaza si actualizeaza profilul intern al utilizatorului;
   - gestioneaza rolurile: *admin*, *secretariat*, *profesor*, *student*.
3. **Modul de baza de date** (PostgreSQL + ORM):
   - stocheaza entitatile de business (utilizatori, roluri, discipline, grupe, sali, orare);
   - asigura persistenta si consistenta datelor prin tranzactii.

4.2 Module proprii (avansate)

Timetable Management Module. Functionalitati principale:

- administrarea disciplinelor, grupelor, profesorilor si salilor;
- definirea intervalelor orare (zi, ora inceput, ora sfarsit);
- gestionarea orarelor (creare, modificare, vizualizare);
- verificari de baza pentru evitarea suprapunerilor evidente.

Scheduling Engine Module. Functionalitati principale:

- preluarea cererilor de generare sau ajustare a orarelor dintr-o coada RabbitMQ;
- aplicarea unor algoritmi de alocare a disciplinelor pe sloturi orare, respectand constrangeri precum:
  - disponibilitatea profesorilor;
  - disponibilitatea salilor;
  - evitarea suprapunerilor pentru aceeasi grupa;

  - preferinte de orar (de exemplu, evitarea cursurilor foarte tarzii);
- salvarea solutiei gasite in baza de date;
- generarea de rapoarte de conflict (de exemplu, cand nu exista solutie completa).

Notifications and Updates Module. Functionalitati principale:

- ascultarea evenimentelor emise de celelalte microservicii (orar nou, modificare orar, anulare curs);
- trimiterea de notificari catre utilizatorii afectati;
- logarea actiunilor relevante in entitatea *AuditLog*;
- generarea de rapoarte simple (de exemplu, cate modificari au avut loc intr-un anumit interval).

4.3 Fluxuri principale de utilizare

Cateva scenarii reprezentative pe care platforma le va acoperi sunt:

- Secretaria creeaza un nou semestru, defineste disciplinele, grupele si salile disponibile;
- Se lanseaza o cerere de generare automata a orarului pentru un set de grupe;
- Scheduling Engine Service primeste job-ul, calculeaza un orar valid si salveaza rezultatul;
- Studentii si profesorii se autentifica prin Keycloak si vizualizeaza orarul personalizat;
- Atunci cand o sala devine indisponibila sau un profesor anunta o constrangere noua, se genereaza o cerere de recalculare partiala a orarului;
- Notifications Service trimite notificari catre utilizatorii afectati de modificarile de orar.

5 Functionalitati avansate

Pentru a demonstra concepte specifice sistemelor concurente si distribuite, proiectul va include urmatoarele functionalitati avansate:

5.1 1. Engine distribuit de generare automata a orarelor

Scheduling Engine Service reprezinta un modul de procesare intensiva care primeste cereri dintr-o coada RabbitMQ si ruleaza algoritmi de generare a orarelor pe baza unui set de constrangeri. Acest serviciu va fi configurat cu mai multe replici in Docker Swarm, astfel incat mai multe job-uri sa poata fi procesate in paralel. Functionalitatea ilustreaza:

- scalare orizontala (replicarea unui microserviciu);
- distribuirea sarcinii de lucru intre workeri;
- izolarea unui modul complex intr-un serviciu dedicat.

5.2 2. Procesare asincrona cu cozi de mesaje

Platforma va utiliza RabbitMQ ca broker de mesaje pentru a decupla microserviciile. Cererile de generare sau recalculare a orarului, precum si evenimentele de modificare a orarului vor fi modelate ca mesaje in cozi, procesate asincron de workerii din Scheduling Engine Service si Notifications Service. Aceasta abordare permite:

- reducerea timpului de raspuns pentru operatiile initiate de utilizator;
- cresterea robustetii in fata erorilor temporare;
- scalarea independenta a modulelor care proceseaza volume mari de mesaje.

5.3 3. Mecanisme de consistenta si control concurent pentru resurse

Rezervarea salilor, a profesorilor si a intervalelor orare reprezinta o problema intrinseca de consistenta a datelor. Proiectul va include mecanisme explicite pentru:

- evitarea suprapunerilor (un profesor sau o grupa nu poate avea doua cursuri in acelasi timp);
- evitarea conflictelor de sala (aceeasi sala nu poate gazdui doua cursuri in acelasi interval);
- verificari concurente la nivelul bazei de date, folosind tranzactii si constrangeri adecvate;
- validari suplimentare in logica de business pentru a preveni stari inconsistente.

6 Tehnologii utilizate

- **Limbaj backend**: Python;
- **Framework REST API**: FastAPI (sau alternativ Flask / Django Rest Framework);
- **ORM**: SQLAlchemy;
- **Baza de date**: PostgreSQL;
- **Autentificare si autorizare**: Keycloak (SSO, OAuth2 / OpenID Connect);
- **Broker de mesaje**: RabbitMQ;
- **Cache** (optional): Redis;
- **Monitorizare** (optional): Prometheus + Grafana;
- **Containerizare**: Docker;
- **Orchestrare**: Docker Swarm (stack definit intr-un fisier `.yml`);
- **Testare API**: Postman si scripturi automate acolo unde este cazul;
- **Control versiuni**: Git.

7 Concluzii pentru Milestone 1

Acest document defineste tema proiectului *Organizator distribuit de orare*, contextul si motivatia alegerii sale, arhitectura generala a solutiei, modulele principale, functionalitatile avansate, precum si tehnologiile care vor fi utilizate.

Structura propusa respecta cerintele generale ale proiectului SCD 2025: utilizarea autentificarii SSO, integrarea unei baze de date relationale printr-un ORM, separarea aplicatiei in microservicii containerizate, folosirea unui stack Docker Swarm, existenta unui modul scalabil orizontal si integrarea unor functionalitati avansate relevante pentru sisteme concurente si distribuite.

In etapele urmatoare (Milestone 2 si Milestone 3) se va trece la implementarea efectiva a microserviciilor, integrarea lor in stack-ul Docker Swarm, scrierea testelor si realizarea unei demonstratii functionale a aplicatiei complete.