Proiect la materia „Programare Web folosind Tehnologii Java”

Planificator de Conferințe

Student: Cătălin-Gabriel Sasu

Grupa: 405

Semi-grupa: 4

Cuprins

Introducere

Planificarea conferințelor este un lucru extraordinar de important deoarece vizează principii precum punctualitatea și evoluția grupurilor sau indivizilor. De aceea am ales să construiesc o aplicație care să permită organizarea și managementul conferințelor. Deși prin aplicație mă refer la *backend*-ul acesteia, în documentație vor fi menționați și aspecte de *frontend* și *Graphic User Interface* aferente.

Astfel, documentația prezintă următoarele aspecte: funcționalitățile *back*end-ului și entitățile acestuia, logica aplicației și cum a fost gândită să funcționeze, modul de stocare al datelor sau întreținere a bazei de date dar și *best practices* pentru partea de *frontend.*

I. Obiectivele aplicației

Pagina web vizează planificarea conferințelor, vizualizarea acestora dar și posibilitatea de înrolare la conferințele preferate.

Practic, utilizatorul poate intra în aplicație fără să dețină un cont, pentru autentificare fiind necesară doar adresa de email. O dată intrat în aplicație acesta poate vedea conferințele disponibile și se poate înrola la acestea. De asemenea, acesta poate adăuga o conferință fiind nevoie să introducă diferite câmpuri precum: categoria conferinței, *speaker*-ul, data de început și sfârșit, numele conferinței dar și altele. Utilizatorul poate vizualiza conferințele create de el într-un *tab* separat.

În continuare va fi prezentată baza de date dar și funcționalitățile din *backend* și vor fi menționate cazurile de folosire a acestora.

II. Stocarea informațiilor

Pentru a crearea și administrarea bazei de date, am folosit *MySQL Workbench* și serverul bazei fiind *hostat* cu ajutorul programului *XAMPP*.

Baza aplicației este formată din 10 tabele. În continuare, voi detalia pe scurt scopul fiecărei tabele și câmpurile memorate:

- *Admin*: Pentru ca anumite *features* sunt disponibile doar pentru administratorii aplicației, această tabelă este menită să stocheze *email*-urile adminilor pentru a le oferi accesul la diferite funcționalități la momentul logării: ex: ca administrator vreau să pot adăuga o localitate nouă în lista de localități,
- *DictionaryDistrict:* Această tabelă este un dicționar care stochează localitățile țării, cheia primară fiind coloana *DictionaryDistrictId,*
- *DictionaryCity*: La fel ca și *DictionaryDistrict*, tabela reține orașele din fiecare localitate, cheia principală fiind *DictionaryCityId,* iar cea străină care face legătura cu tabela localităților este *DictionaryDistrictId,*
- *DictionaryConferenceCategory:* Stochează toate categoriile pe care le poate lua o conferință. Asignarea se realizează cu ajutorul câmpului: *DictionaryConferenceCategoryId,*
- *DictionarySpeaker:* Asignează fiecărui *speaker* câte un *id* unic, câmpurile fiind: *id-ul* și numele *speaker*-ului,
- *DictionaryParticipantStatus:* În acest dicționar sunt reținute cele trei status-uri pe care le poate avea un participant: *Joined, Attended, Withdrawn*, asignând fiecăruia un *id.*
- *Location:* Memorează adresa unei locații, *id*-ul orașului acesteia și asignează un *id* unic,
- *Conference:* Probabil cea mai importantă tabelă deoarece aici sunt reținute conferințele. Câmpurile sunt:
    - *ConferenceId: id*-ul conferinței,
    - *ConferenceName:* numele conferinței,
    - *StartDate:* data de început,
    - *EndDate:* data de sfârșit,
    - *OrganiserEmail:* emailul organizatorului,
    - *LocationId: id*-ul locației
    - *DictionaryConferenceCategoryId:* id-ul categoriei.

- *ConferenceXSpeaker:* Deoarece un *speaker* poate susține mai multe conferințe iar o conferință poate avea mai mulți *speaker*s, această tabelă asociativă marchează relația de *many-to-many* dintre cele două tabele,
- *ConferenceAttendance:* În această tabelă sunt reținuți participanții unei conferințe și statusurile acestora.

III. Modularizarea backend-ului

Modularizarea claselor *backend*-ului este făcută cu ajutorul pachetelor: *config, controller, DTOs, entities, exception, repository, service.* Fiecare pachet memorând fișierele aferente. O altă parte importantă este fisierul *test* unde sunt memorate toate testele create.

În pachetul *entities* se regăsesc clasele fiecărei entități din baza de date, cu aceleași câmpuri, în timp ce în *DTOs* sunt clasele modelelor care servesc pentru afișarea modelelor care nu au aceeasi structură ca în bază: de exemplu: în entitatea *Conference* se regăsesc câmpurile: *LocationId, DictionaryConferenceCategoryId* în timp ce în *DTO ConferenceModel* se regăsesc câmpurile: *LocationAddress, CityName, DistrictName, CategoryName* etc.

În pachetul *controller* sunt cele 9 *controllers* pentru entitățile bazei. Fiecare dintre acestea sunt marcate ca fiind *@RestController* folosind adnotația din *springframework.web.* Putem observa cuvântul cheie „*Rest*" care ne anunță că acest *controller* va fi folosit pentru implementarea metodelor *REST.* Pe lângă adnotare, fiecare *controller* primește ruta de bază a acestuia care ajută la apelarea corectă a *API*-urilor: *@RequestMapping(„city")*. Folosind *Dependency Injection,* fiecare controller va avea un câmp de tipul serviciului aferent, putând astfel să folosească metodele serviciului.

În pachetul *service* se regăsesc toate serviciile aplicației, fiecare serviciu putând folosi funcțiile din *repository*-ul aferent datorită metodei *DI.*

Repository stochează fișierele marcate ca fiind *@Repository*. Aici se regăsesc implementările funcțiilor și *call*-urile în baza de date.

La modul general, dacă se va apela metoda cu ruta *x/y* și tipul *PUT,* în spate se vor întâmpla următoarele:

1. Se va folosi *controller*-ul care are *@RequestMapping /x,*

2. Se va căuta metoda *PUT* cu *@PutMapping /y,*
3. Se va apela funcția corespunzătoare metodei din *service*-ul *controllerNameService* prin intermediul instanței de *service* injectate,
4. În intermediul *service*-ului se va apela funcția aferentă prin intermediul instanței de *repository* injectate.
5. În funcția din *repo* se vor manipula datele din bază prin intermediul instanței *jdbcTemplate.*

IV. Detalierea endpoint-urilor

Fiecare *controller* are funcțiile sale iar fiecare funcție are un rol bine definit. În continuare vom descrie aceste funcții și rolurile acestora.

AdminController

Metoda *GET getAllAdmins* apelează metoda *getAllAdmins* din *adminService* care la rândul ei aplelează *getAllAdmins* din *adminRepository* care va întoarce din bază toți adminii aplicației. Aceasta poate fi folosită la afișarea tuturor adminilor în aplicație.

Metoda *POST addAdmin* cu ruta *admin/save* adaugă un admin în tabela *Admin.*

CategoryController

Metodele *GET getAllCategory* și *getCategoryById* au ruta *category/all* respectiv *category/{id}.* Afișarea categoriilor este efectuată atunci când utilizatorul trebuie să selecteze o categorie pentru conferința sa. Selectarea unei categorii după id, poate fi folosită la afișarea conferințelor: în *DTO Conference* există câmpul *ConferenceName* care va fi populat cu ajutorul acestei metode.

Metodele *POST* respectiv *PUT*, *addCategory* și *updateCategory* au ambele ruta *category/save*, diferența dintre ele fiind făcută în momentul apelării *endpoint*-ului specifincând metoda *REST* dorită.

O altă metodă este cea de ștergere a unei categorii, *feature* la care are acces un admin, *tagul* metodei fiind *@DeteleMapping(„/delete”).*

DistrictController

În momentul în care utilizatorul dorește să introducă o conferință acesta va selecta localitatea în care conferința va avea loc. Metoda *GET, getAllDistricts* va furniza datele necesare. Pentru afișarea corectă a localității unei conferințe, a fost creată metoda *getDistrictById.*

Administratorul aplicației poate adăuga, modifica și șterge localități folosind funcțiile *POST, PUT, DELETE, addDistrict, updateDistrict* și *deleteDistrict* cu rutele aferente: *district/save* pentru primele două și *district/delete.*

CityController

În momentul imediat următor selectării localității la introducerea conferinței, se dorește afișarea orașelor din acea localitate, astfel, a fost creată metoda de tip *GET getCityById*. Asemănător cu *DistrictController,* administratorul poate adăuga, modifica și șterge orașe.

SpeakerController

În acest controller sunt definite *endpoint*-urile care permit urmatoarele:

- selectarea tuturor *speaker*-ilor,
- selectarea *speaker*-ilor în funcție de id,
- selectarea în funcție de nume, pentru o potențială filtrare,
- selectarea *speaker*-ului in funcție de conferință,
- inserare, editare și ștergere cu rutele: *POST/PUT /save* și *DELETE /delete*

LocationController

Pentru a oferi o funcționalitate corectă aplicației, acest controller implementează metodele de selectare a tuturor locațiilor, a locațiilor în funcție de id și funcționalitățile de inserare și editare folosind metodele *REST*.

ParticipantStatusController

ParticipantStatusController permite selectarea statusurilor și adăugarea de status nou.

ConferenceAttendanceController

Acest controller se ocupă de adaugarea în bază a participanților la o conferință. De exemplu: mitică@gmail.com apasă pe butonul de *Attend,* atunci se va apela metoda *POST*

cu ruta */save addAttendance* și va introduce în tabela *ConferenceAttendance* datele aferente. Dacă Mitică se răzgândește și apasă pe *Withdrawn* atunci se va apela metoda *PUT updateAttendance* cu aceeași rută și va modifica statusul.

Dacă utilizatorul dorește să vadă toți participanții la o conferință, acest lucru este posibil apelând metoda *GET getAllAttendances.*

ConferenceController

Acest *controller*¸ permite:

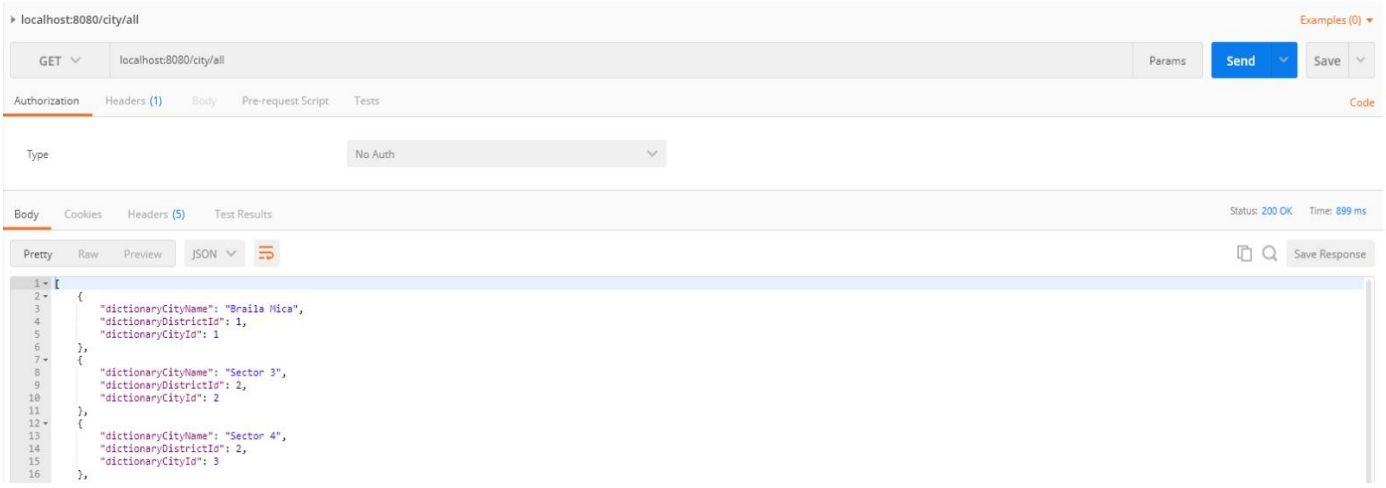
- vizualizarea tuturor conferințelor, prin metoda *GET getAllConferences,*
- vizualizarea conferințelor create de utilizator, prin metoda *getConferencesByOrganiserEmail*
- Adăugarea de conferințe, prin metoda *POST /save, addConference*
- Editarea unei conferințe, prin metoda *PUT /save, updateConference*
- Ștergerea unei conferințe. prin metoda *DELETE /save, deleteConference*

V. Testare

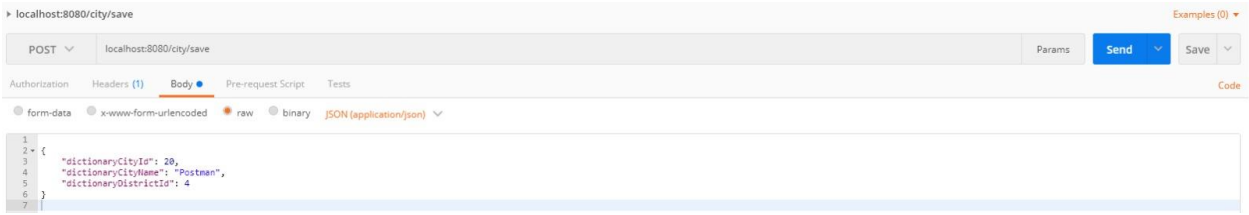
V.I Postman

Pentru a demonstra funcționalitatea fiecărui *endpoint,* acesta a fost testat folosind aplicația *Postman.* Mai jos se regăsesc câteva imagini care evidențiază acest lucru.

- *localhost:8080/city/all* – furnizarea tuturor orașelor



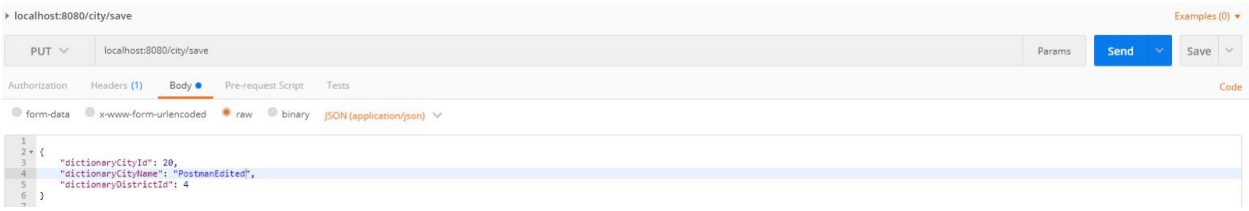
- *localhost:8080/city/save – POST* inserarea unui nou oraș



verificarea în bază:

DictionaryCityId	DictionaryCityName	DictionaryDistrictId
1	Braila Mica	1
2	Sector 3	2
3	Sector 4	2
4	Sector 5	2
5	Murfatlar	5
6	Galati Mic	7
7	Harghita Mica	8
8	Giurgiu Mic	6
9	Bihor Mic	4
10	Caras Zeppelin	3
20	Postman	4
NULL	NULL	NULL

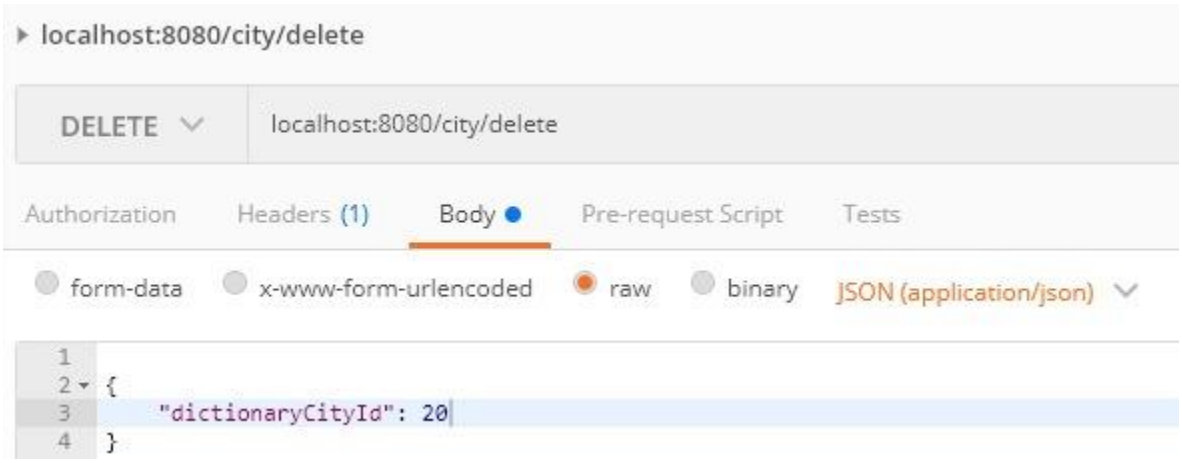
- *localhost:8080/city/save – PUT* editarea unui oraș



verificarea în bază

	DictionaryCityId	DictionaryCityName	DictionaryDistrictId
	1	Braila Mica	1
	2	Sector 3	2
	3	Sector 4	2
	4	Sector 5	2
	5	Murfatlar	5
	6	Galati Mic	7
	7	Harghita Mica	8
	8	Giurgiu Mic	6
	9	Bihor Mic	4
	10	Caras Zeppelin	3
▶	20	PostmanEdited	4
*	NULL	NULL	NULL

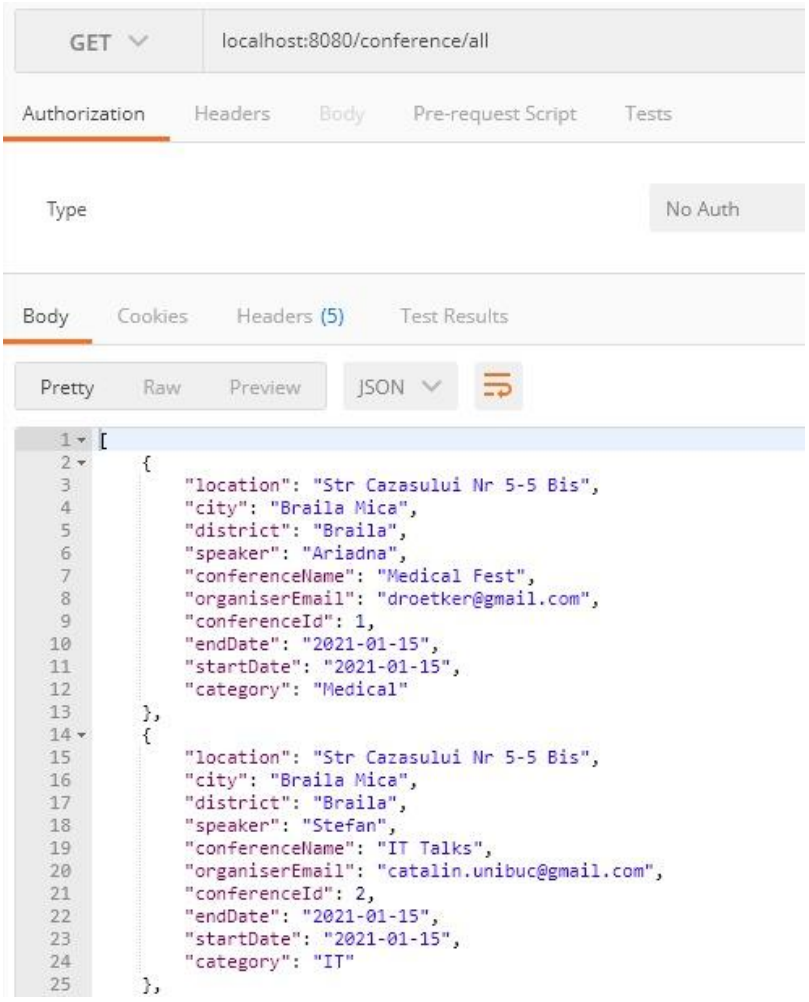
- *localhost:8080/city/delete – Delete* ștergerea unui oraș



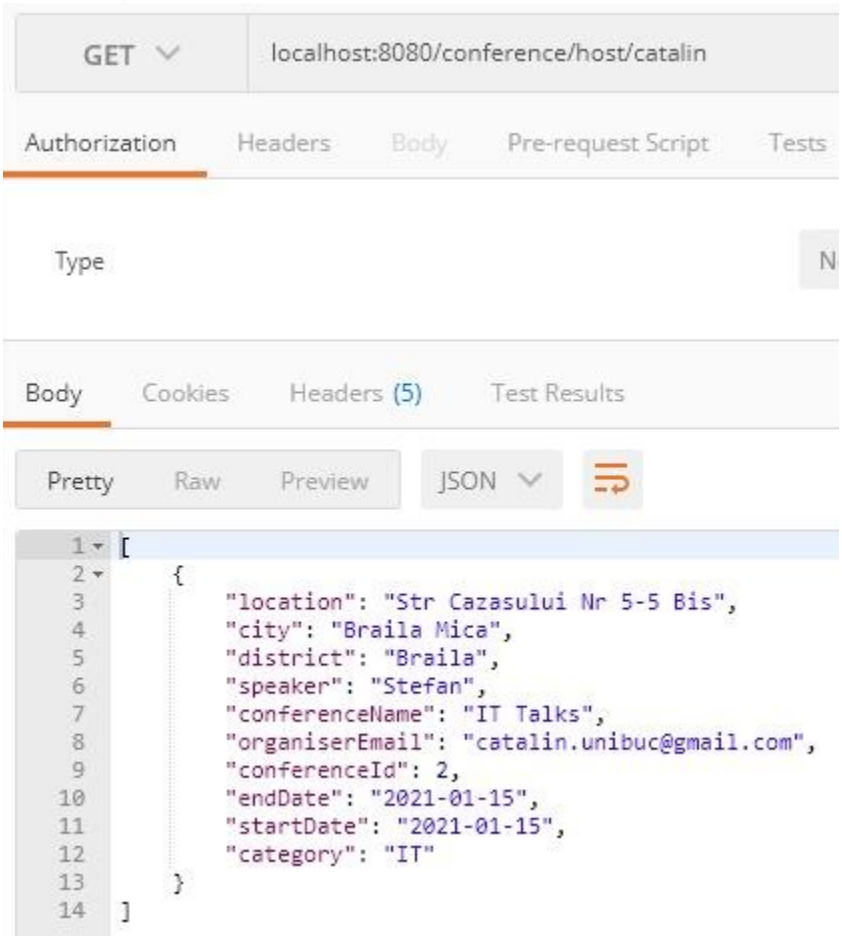
vericarea în bază:

	DictionaryCityId	DictionaryCityName	DictionaryDistrictId
	1	Braila Mica	1
	2	Sector 3	2
	3	Sector 4	2
	4	Sector 5	2
	5	Murfatlar	5
	6	Galati Mic	7
	7	Harghita Mica	8
	8	Giurgiu Mic	6
	9	Bihor Mic	4
	10	Caras Zeppelin	3
▶*	NULL	NULL	NULL

- *localhost:8080/conference/all – Get* selectarea conferințelor



- *localhost:8080/conference/host/catalin – GET* selectarea conferințelor organizate de deținătorul adresei care începe cu *catalin*.



- *localhost:8080/conference/host/catalin123 – GET Not Found*

Pe lângă verificarea *Api*-urilor cu aplicația *Postman,* am implementat în *folder*-ul destinat testării, patru mici teste: trei pentru *CityController* și unul pentru *ParticipantStatusController*. Acestea asigură corectitudinea selectării orașelor, orașelor în funcție de *distrcitId,* adăugarea de orașe și adăugarea unui nou status.

- *localhost:8080/conference/delete – DELETE,* această metodă va șterge: conferința din tabela *Conference* și *speaker*-ii acesteia din *ConferenceXSpeaker*

V.II Teste JUnit

În cadrul fișierului *test* se regăsesc testele pentru următoarele *controllers: CityControllerTest, ConferenceControllerTest, ParticipantStatusControllerTest.*

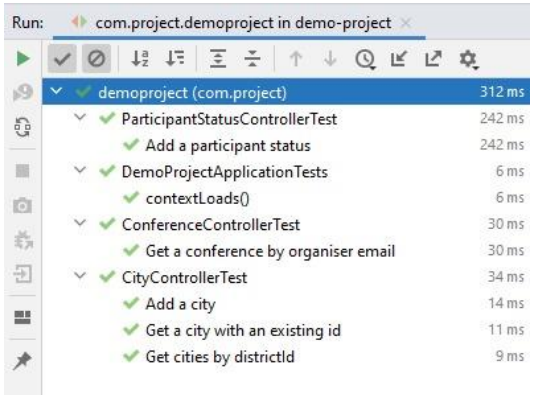
Primul *controller, CityControllerTest,* este compus din testele:

- *Get a city with an existing id* – testează selectarea corectă a orașelor după *id.*
- *Add a city* – verifică funcționalitatea de adăugare a orașelor.
- *Get a city by districtId* - testează selectarea corectă a orașelor după district *id.*

ParticipantStatusControllerTest verifică adăugarea corectă a statusurilor de participare.

ConferenceControllerTest este compus din două teste: *Get a conference by organiser email* care verifică selectarea conferințelor în funcție de email și *Get a conference by non-existent organiser email* care verifică *bad-flow*-ul metodei de selectare după *email.* Codul celui de-al doilea test este comentat din motive tehnice.

Mai jos se află o imagine cu totalitatea testelor și rezultatul acestora:



Concluzii

Documentația prezintă *backend*-ul unei aplicații *Spring Boot* folosită pentru planificarea conferințelor. Documentația prezintă în mod explicit traseul pe care îl urmează *endpoint*-urile și modul în care acestea ajung să manipuleze informațiile persistente în bază. De asemenea, este evidențiat și verificarea corectitudinii fiecărui obiectiv al *endpoint*-urilor.

Am ales această temă pentru că am considerat că îmi oferă posibilitatea de a atinge fiecare cerință arătând astfel capacitatea de creare, testare și înțelegere a conceptelor cursului.