Aplicație pentru predicția consumului de medicamente

Autor: Ursulean Ciprian

Coordonator științific: Lect. Dr. Cristian Frăsinaru

Cuprins

Motivatie

- Problema actuală intalnită în viata de zi cu zi
- Rezolvarea acestei probleme aduce multiple beneficii și posibil salvarea de vieți
- Problema poate fi extinsă și la alte domenii

Descrierea problemei

- Ne propunem sa realizam anumite predicții de consum a medicamentelor
- Predicțiile au la bază analiza unor seturi de date reale de la spitale
- În funcție de o anumită data și de un medicament ales vom putea estima o cantitate necesara a acestuia
- Pe întreg parcursul dezvoltării aplicației s-au folosit informații din următorul articol: click aici

Seturi de date

- Datele prezinta cea mai importanta parte a aplicatiei
- Calitatea datelor impacteaza în mod direct calitatea predicțiilor
- Aplicația folosește doua seturi de date reale de la doua spitale diferite
- Datele ne pun la dispoziție detalii legate de medicamente, pacienti, secții de spital și alte informații relevante

Modul de rezolvare al problemei

- Predicțiile se realizează pe baza algoritmului de regresie liniara
- Se construieste un model de regresie pentru fiecare medicament in parte
- Pornind de la data de utilizare se calculeaza cantitatea viitoare
- Modelele si functionalitațile lor sunt expuse cu ajutorul unor API uri

Tehnologii utilizate

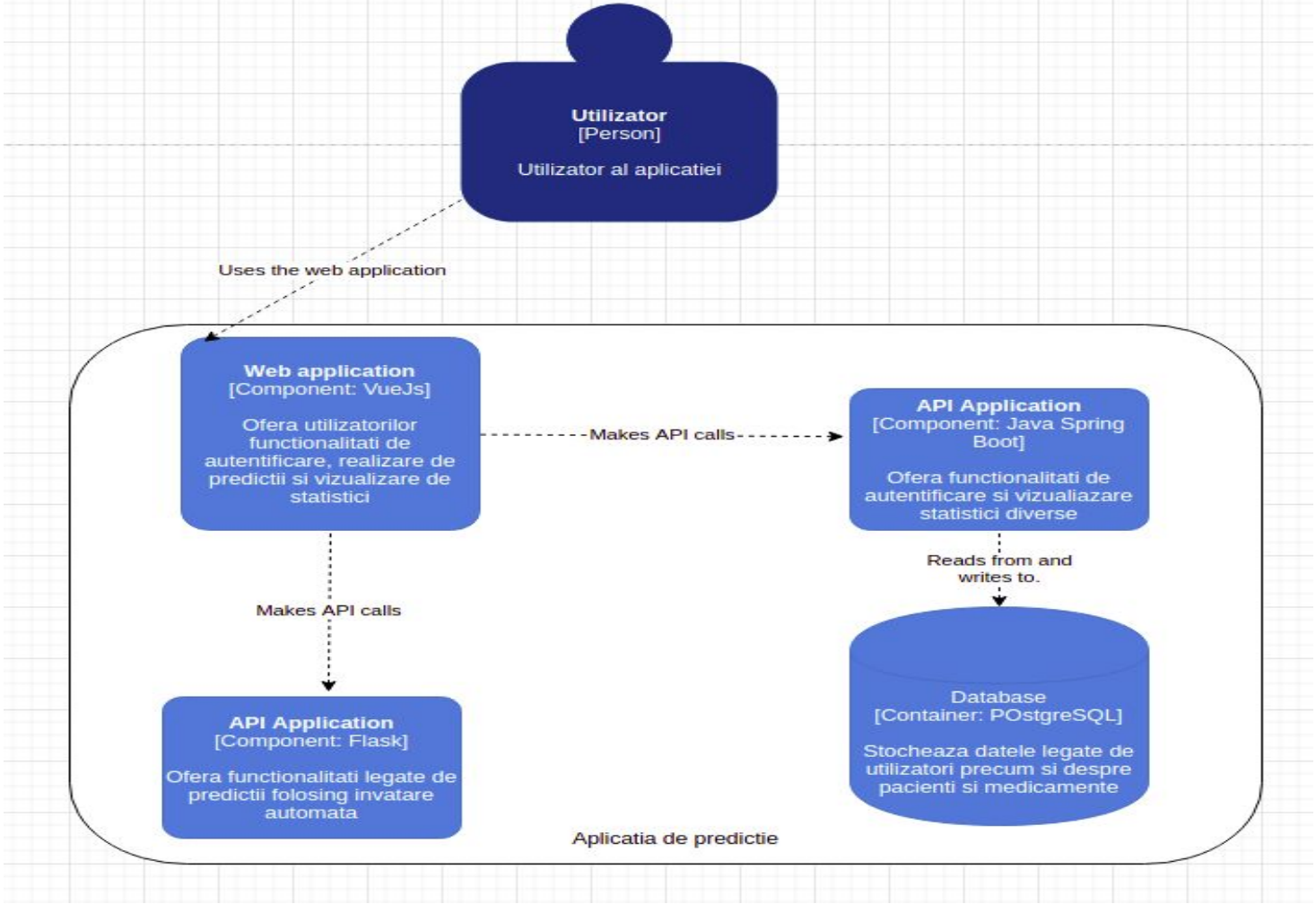
Pentru realizarea interfeței web s-a utilizat: HTML5, CSS3, VueJs, ChartJs, Axios

Pentru construirea serviciilor s-a utilizat: Java (Spring Boot), Python (Flask)

Baza de date folosită a fost PostgreSQL

Arhitectura aplicației

- arhitectura modulară bazată pe servicii web
- componentele sunt independente și comunica prin apeluri HTTP
- serviciul construit cu Spring Boot este responsabil de utilizatori și de statistici medicamente/pacienți
- serviciul construit cu Flask este responsabil de predicțiile cu învățare automată


Utilizator
[Person]
Utilizator al aplicatiei
Uses the web application
Web application
[Component: VueJs]
Ofera utilizatorilor functionalitati de autentificare, realizare de predictii si vizualizare de statistici
Makes API calls
API Application
[Component: Java Spring Boot]
Ofera functionalitati de autentificare si vizualiazare statistici diverse
Reads from and writes to.
Makes API calls
API Application
[Component: Flask]
Ofera functionalitati legate de predictii folosing invatare automata
Database
[Container: POstgreSQL]
Stocheaza datele legate de utilizatori precum si despre pacienti si medicamente
Aplicatia de predictie

Detalii de implementare

```
<ChartDrug :key="componentKey"
           :type="type"
           :label="label"
           :labels="this.labels"
           :data="this.data"
           :backgroundColor="this.backgroundColor"
           :background="this.background" />
```

You, seconds ago • Uncommitted changes

Figura 1 - Componenta Chart

```python
def train_models(model_config):
    for file_name, model_name in model_config.items():
        try:
            model_config[file_name].prepare_model()
        except Exception as ex:
            print(ex)
            print(file_name)


@app.before_first_request
def prepare_models():
    global models_config_

    print("Preparing models...")
    models_config_ = generate_models_from_csv()
    print("DONE preparing models...")

    print("Training models...")
    train_models(models_config_)
    print("DONE training models... ")


@app.route('/predict')
def get_prediction():
    print(request.args)
    if "drug" in request.args and "timestamp" in request.args:
        drug_name = request.args.get("drug")
        timestamp = request.args.get("timestamp")
        if drug_name + ".csv" in models_config_:
            return str(models_config_.get(drug_name + ".csv").get_prediction(np.array(timestamp).reshape(-1, 1)))
        return "No data"
    else:
        abort(400, 'Invalid url format, provide valid drug name and timestamp')
```

Figura 2 - Expunerea functionalitatii de predicție prin apeluri HTTP

```java
@RestController
@RequestMapping("/api/v1/users")
@CrossOrigin(origins = "*", allowedHeaders = "*")
public class UserController {
    @Autowired
    UserService userService;

    @PostMapping("/register")
    public ResponseEntity<Map<String, String>> registerUser(@RequestBody Map<String, Object> userMap) {
        String firstName = (String) userMap.get("first_name");
        String lastName  = (String) userMap.get("last_name");
        String email     = (String) userMap.get("email");
        String password  = (String) userMap.get("password");
        User newUser = userService.registerUser(firstName, lastName, email, password);
        return new ResponseEntity<>(JwtConfig.generateJwtToken(newUser), HttpStatus.OK);
    }

    @PostMapping("/login")
    public ResponseEntity<Map<String, String>> loginUser(@RequestBody Map<String, Object> userMap) {
        String email = (String) userMap.get("email");
        String password = (String) userMap.get("password");
        User user = userService.validateUser(email, password);
        return new ResponseEntity<>(JwtConfig.generateJwtToken(user), HttpStatus.OK);
    }
}
```

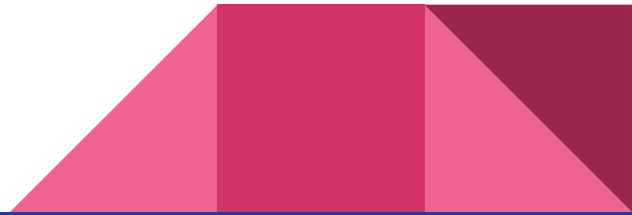
Figura 3 - Implementarea autentificarii

DEMO

YOUTUBE LINK

Concluzii

- Procesul dezvoltării aplicației a fost unul plin de provocări dar satisfăcător
- Utilizarea unei palete largi de tehnologii m-a facut sa înțeleg mai bine cum se integrează părțile aplicației
- Dificultăți tehnice intalnite: la politicile CORS, mici erori la procesarea datelor, mici erori la lucrul cu servicul de învățare automată



Mulțumesc