

# Online Voting System

Iosif Daniel-Ionel

13 aprilie 2022

## **Rezumat**

Proiectul a fost creată in cadrul disciplinei "Dezvoltarea aplicațiilor mobile" de la Universitatea de Medicină, Farmacie, Științe și Tehnologie „George Emil Palade” din Târgu Mureș și are ca scop învățarea și dezvoltarea capacității de a crea aplicații Android.

## Cuprins

<b>1</b>	<b>Introducere</b>	<b>2</b>
<b>2</b>	<b>Tehnologii Folosite</b>	<b>2</b>
2.1	Spring . . . . .	2
2.2	Spring Data JPA , H2 DataBase și Hibernate . . . . .	4
2.3	Retrofit . . . . .	6
2.4	Salvarea imaginilor în baza de date . . . . .	8
<b>3</b>	<b>Criptarea Datelor</b>	<b>8</b>
<b>4</b>	<b>BindingFragment</b>	<b>9</b>
<b>5</b>	<b>Schema de bloc a aplicației</b>	<b>10</b>
<b>6</b>	<b>Functionalitate</b>	<b>11</b>
<b>7</b>	<b>Bibleografie</b>	<b>12</b>
7.1	Android for Absolute Beginners - Getting Started with Mobile Apps Development Using the Android Java SDK by Grant Allen	12
7.2	Learn Kotlin for Android Development The Next Generation Language for Modern Android Apps Programming by Peter Späth (z-lib.org) . . . . .	12
7.3	<a href="https://docs.spring.io/spring-framework/docs/current/reference/html/">https://docs.spring.io/spring-framework/docs/current/reference/html/</a>	12
7.4	<a href="https://square.github.io/retrofit/">https://square.github.io/retrofit/</a> . . . . .	12

## 1 Introducere

Proiectul este alcătuit din 2 părți: partea de client fiind constituită din aplicația Android dezvoltată în "Android Studio" prin intermediul limbajului limbajul Kotlin , iar cealaltă parte fiind serverul dezvoltat în Java prin intermediul Framework-ului Spring.

Scopul acestuia fiind depunerea candidaturii pentru cele 3 roluri de către orice utilizator și votarea unui candidat pe care fiecare utilizator îl vede ca reprezentant, toate acestea realizându-se într-un mod sigur, datele rămânând confidențiale.

## 2 Tehnologii Folosite

### 2.1 Spring

Frameworkul **Spring** este o platformă cu sursă deschisă pentru simplificarea scrierii aplicațiilor în limbajul Java, dar există și o versiune pentru Platforma .NET.

În acest proiect am folosit **Spring Web** care oferă toate tehnologiile necesare pentru crearea unui server REST(Representational State Transfer), comunicarea

bazandu-se pe transferul datelor prin protocolul HTTP cu ajutorul metodelor GET si POST. GET fiind folosit pentru cererea de date de la o resursă specificată , iar POST fiind folosit pentru trimiterea de date la server sau pentru actualizarea datelor din baza de date. Datele fiind transmise in format JSON. Un server REST nu oferă doar posibilitatea de a transmite date într-un mod cât mai ușor posibil, dar și retransmite către cel ce folosește metodele GET, POST un cod de răspuns pentru a-l anunța ce s-a intamplat cu cererea sa. Toate aceste metode care fac legătura cu aplicația Android și baza de date se află într-o clasă Controller.

```
@Autowired
private UserService userService;

@PostMapping("/user/save")
public User saveUser(@RequestBody User newUser) {

    return userService.saveUser(newUser);
}

@GetMapping("/publickey")
public String getPublicKey() { return userService.sendPublicKey(); }

@GetMapping("/checkUser/{username}")
public boolean checkUser( @PathVariable("username") String username) { return userService.checkUser(username); }

@GetMapping("/user/password/{username}")
public String getPasswordForUsername( @PathVariable("username") String username){

    User user = userService.getPasswordForUser(username);
    return user.getPassword();
}

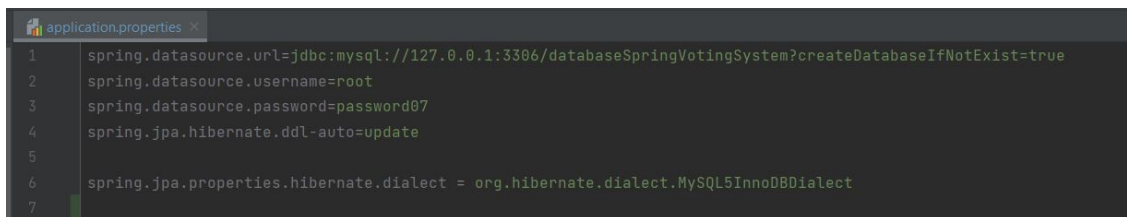
@GetMapping("/user/id/{username}")
public Long getIdForUsername( @PathVariable("username") String username){

    return userService.getIdForUser(username);
}
```

Figura 1: Post si Get

## 2.2 Spring Data JPA , H2 DataBase și Hibernate

**Hibernate** este un instrument folosit pentru maparea datelor într-o bază de date prin crearea unui Model . Cu ajutorul **Spring Data JPA și H2 DataBase**, Hibernate devine un instrument destul de puternic și ușor de folosit în ceea ce privește gestionarea unei baza de date. Totul începe cu crearea modelui(o clasa care conține toate câmpurile ce le va avea o entitate din baza de date, aici fiind folosite diverse adnotări pentru specificarea tipului datelor), ca mai apoi acest model să fie folosit într-un DataSet care constituie tabela din baza de date(acesta fiind o interfață unde putem scrie metode noi pentru interogarea bazei de date). Aceste metode pot conține Adnotări cu codul SQL specifi pentru interogarea bazei de date sau pur și simplu specificarea a ceea ce dorim din baza de date sub forma denumirii metodei( bineînțeles ca trebuie respectat un tipar). Legătura cu baza de date se face dintr-un fișier unde este specificată adresa URL acesteia, precum si usernamul si parola.



```
1 spring.datasource.url=jdbc:mysql://127.0.0.1:3306/databaseSpringVotingSystem?createDatabaseIfNotExist=true
2 spring.datasource.username=root
3 spring.datasource.password=password07
4 spring.jpa.hibernate.ddl-auto=update
5
6 spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5InnoDBDialect
7
```

Figura 2: Legătură bază de date

Toate operațiile asupra bazei de date sunt realizate într-o clasă care are adnotarea **Service**, aici având o instanță a repository-ului (tabelului) asupra caruia executăm operațiile dorite.

```

@Entity
@NoArgsConstructor
public class User {

    private @Id @GeneratedValue Long id;
    private @Column(nullable = false) String fname;
    private @Column(nullable = false) String lname;
    private @Column(nullable = false) String username;
    private @Column(nullable = false) String cnp;
    private @Column(nullable = false) String phone;
    private @Column(nullable = false) String email;
    private @Column(nullable = false) String password;
    private @Column(nullable = false) String sessionKey;
    private @Column(nullable = false) int presedinte;
    private @Column(nullable = false) int parlament;
    private @Column(nullable = false) int europarlamentari;

    public int getPresedinte() { return presedinte; }

    public void setPresedinte(int presedinte) { this.presedinte = presedinte; }

    public int getParlament() { return parlament; }

    public void setParlament(int parlament) { this.parlament = parlament; }

    public int getEuroparlamentari() { return europarlamentari; }

    public void setEuroparlamentari(int europarlamentari) { this.europarlamentari = europarlamentari; }
}

```

Figura 3: Model entitate

```

public interface UserRepository extends JpaRepository<User, Long> {

    Optional<User> findByUsername(String username);
    User findByusername(String username);
    User findById(Long id);
}

```

Figura 4: DataSet-Repository

```

@Service
public class UserService {

    private static final Logger LOGGER = LoggerFactory.getLogger(UserService.class);

    @Autowired
    private UserRepository repository;

    @Async
    public String sendPublicKey() { return ServerApplication.getPublicKey(); }

    @Async
    public boolean checkUser(String username){

        return repository.findByUsername(username).isPresent();

    }

    @Async
    public User getPasswordForUser(String username) { return repository.findByusername(username); }
}

```

Figura 5: Service

## 2.3 Retrofit

**Retrofit** este o librărie folosită în aplicația android și constituie o un client REST ce ajută la creerea cererilor HTTP și procesarea acestora.

Folosirea acesteia constituie în primul rând crearea unei interfețe care conține metode GET SI POST pentru interacțiunea cu serverul, apoi a unui serviciu unde este specificată adresa serverului, cât și a conversiei de date în JSON sau pastrea acestora în formă scalară(int , boolean, string). Al treilea pas în folosirea acestei librării este apelarea metodei dorite din interfață și verificarea răspunsului de la server. Aceasta fiind realizată prin intermediul unei corutine. Corutinele fiind folosite în Kotlin pentru a asigura programarea asincronă și paralelă.

```

interface UserApi {

    @POST( value: "/user/save")
    fun save(@Body user: User): Call<User>

    @GET( value: "/publickey")
    fun publicKey(): Call<String>

    @GET( value: "/checkUser/{username}")
    fun checkUser(@Path( value: "username") username: String): Call<Boolean>

    @GET( value: "/user/password/{username}")
    fun getPasswordForUsername(@Path(value="username") username: String) : Call<String>

    @GET( value: "/user/id/{username}")
    fun getIdForUsername(@Path(value="username") username: String) : Call<Long>

    @GET( value: "/user/numeprenume/{id}")
    fun getName(@Path(value="id") id: Long) : Call<String>

    @POST( value: "/candidate")
    fun saveCandidate(@Body candidate: Candidate): Call<Candidate>

    @GET( value: "/candidate/president")
    fun getCandidatesPresident() : Call<List<CandidateGet>>
}

```

Figura 6: Interfata

```

class RetrofitService {

    private var retrofit: Retrofit

    init{
        retrofit = Retrofit.Builder()
            .baseUrl( baseUrl: "http://192.168.56.1:8080")
            .addConverterFactory(GsonConverterFactory.create(Gson()))
            .build()
    }

    public fun getRetrofit(): Retrofit{
        return retrofit
    }
}

```

Figura 7: ServieRetrofit

```

val retrofitService = RetrofitService()
val userApi = retrofitService.getRetrofit().create(UserApi::class.java)

CoroutineScope(Dispatchers.Default).launch { //this CoroutineScope

    userApi.getCandidatesEuroParliamentary().enqueue(object : Callback<List<CandidateGet>> {
        @RequiresApi(Build.VERSION_CODES.O)
        override fun onResponse(call: Call<List<CandidateGet>>, response: Response<List<CandidateGet>>) {
            if(response.code() == 200){

                candidatelist = response.body()!!
                itemAdapter = ItemAdapter(candidatelist)
                itemAdapter.setOnItemClickListener(object : ItemAdapter.OnItemClickListener {
                    override fun onItemClick(position: Int) {

                        val intent = Intent(getActivity, VoteActivity::class.java)
                        startActivity(intent)

                    }
                })
                mRecyclerView.adapter = itemAdapter
            }
        }

        override fun onFailure(call: Call<List<CandidateGet>>, t: Throwable) {
            Log.e(tag: "rau", msg: "grea la incarcare lista")
        }
    })
}

```

Figura 8: Corutina și retrofit

## 2.4 Salvarea imaginilor în baza de date

Imaginile sunt salvate în baza de date în forma de vector binar. Conversia acestora fiind realizată cu ajutorul unei clase utilitare.

```

class ImageConvertor {

    companion object{

        fun getBitmapAsByteArray(bitmap: Bitmap): ByteArray {
            val outputStream = ByteArrayOutputStream()
            bitmap.compress(CompressFormat.PNG, quality: 0, outputStream)
            return outputStream.toByteArray()
        }

        fun getBitmapImage(byteArray: ByteArray): Bitmap{

            return BitmapFactory.decodeByteArray(byteArray, offset: 0, byteArray.size)
        }
    }
}

```

Figura 9: Conversie imagini pentru stocarea în baza de date

## 3 Criptarea Datelor

Serverul generează la fiecare pornire o pereche de chei: cheia privată și cheia publică cu ajutorul algoritmului RSA. La pornirea aplicației, aceasta primește cheia publică de la server și generează cheia de sesiune( generată cu algoritmul



DES) care mai apoi este criptată cu cheia publică și este transmisă serverului, care o decriptează cu cheia privată. Folosindu-se de această cheie realizându-se transferul de date dintre client și server.

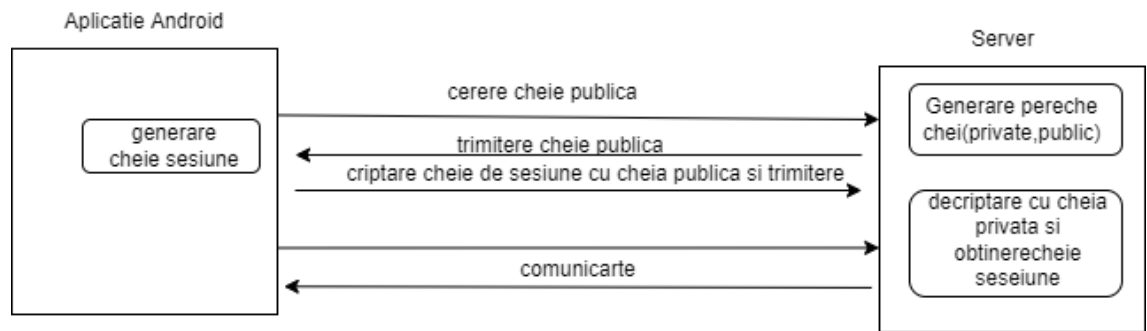


Figura 10: modalitate criptare

## 4 BindingFragment

Legarea fragmentelor cu binding permite accesarea widgeturilor din fragmente pentru extragerea informațiilor care mai apoi urmează a fi transmise către baza de date.

```

class LoginFragment : Fragment() {

    private var _binding: FragmentLoginBinding? = null
    private val binding get() = _binding!!
    private lateinit var text: TextView
    private lateinit var loginBtn: Button
    private lateinit var forgotPasswordBtn: Button
    private lateinit var username: EditText
    private lateinit var password: EditText

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        _binding = FragmentLoginBinding.inflate(inflater, container, attachToParent: false)

        forgotPasswordBtn = binding.root.findViewById(R.id.forgotPassword)
        loginBtn = binding.root.findViewById(R.id.btnLoggin)
        text = binding.root.findViewById(R.id.textViewError)
    }
}
    
```

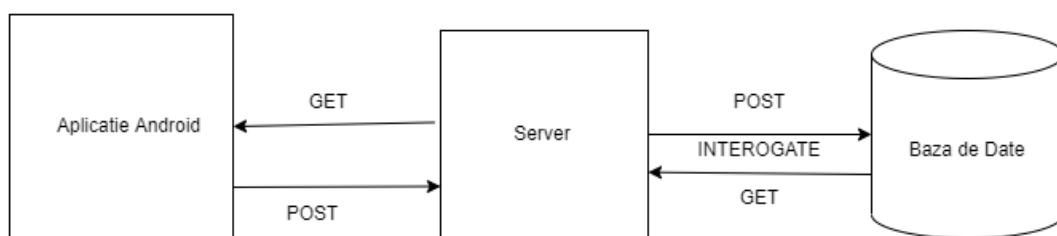
Figura 11: Binding

## 5 Schema de bloc a aplicației

Aplicatie android:

POST: creare cont, trimitere cerere candidatura, votare, trimitere cheie de sesiune,

GET: primire cheie publica, obtinere date pt autentificare, obtinere date candidati



Server:

POST: creare cont in baza de date, modificare scor vot, modificare cheie de sesiune

GET: obtinere date din baza de date

Figura 12: scehma bloc

## 6 Functionalitate

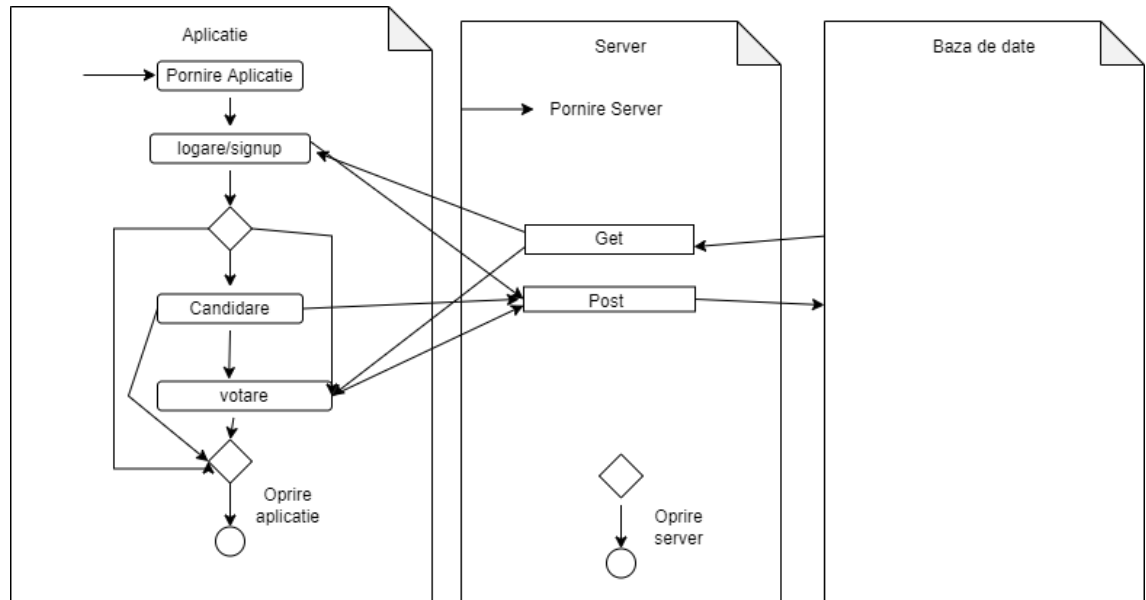


Figura 13: functionalitate

## 7 Bibleografie

- 7.1 Android for Absolute Beginners - Getting Started with Mobile Apps Development Using the Android Java SDK by Grant Allen
- 7.2 Learn Kotlin for Android Development The Next Generation Language for Modern Android Apps Programming by Peter Späth (z-lib.org)
- 7.3 <https://docs.spring.io/spring-framework/docs/current/reference/html/>
- 7.4 <https://square.github.io/retrofit/>