Universitatea Hyperion din București

Facultatea de Științe Exacte și Inginerești

Specializarea: Automatică și Informatică Aplicată

**Proiect de diplomă**

Coordonator științific:

Lector univ. dr. ing. Liviu Șerbănescu

Absolvent:

Rădoi Andrei

București – 2024

Universitatea Hyperion din București

Facultatea de Științe Exacte și Inginerești

Specializarea: Automatică și Informatică Aplicată

**Proiectarea unei aplicații mobile pentru localizarea a diferite puncte de interes**

Coordonator științific:

Lector univ. dr. ing. Liviu Șerbănescu

Absolvent:

Rădoi Andrei

București – 2024

Tema proiectului

Să se realizeze un sistem compus din un server și o aplicație pentru dispozitive mobile, prin intermediul căreia un utilizator să poată accesa informații legate de diferite puncte de interes (De exemplu: băi publice, fântâni de apă potabilă etc.). Proiectul trebuie să respecte următoarele cerințe:

* Arhitectura proiectului trebuie să fie orientată pe utilizator în scopul moderării conținutului;
* Server-ul trebuie să expună un API RESTful prin care se pot accesa datele în format JSON;
* Aplicația de mobil trebuie să se conecteze într-un mod sigur la server și să consume datele oferite de către API;
* Interfața aplicației de mobil trebuie să fie ușor de utilizat, să aibe un design intuitiv și să fie accesibilă pentru persoane cu probleme medicale de vedere;
* Pentru o coordonată specificată de către utilizator, să se caute și să se afișeze toate punctele de interes intr-o rază de 2km.

Cuprins

Introducere 4

Capitolul I – Aplicații mobile: Istoric, arhitecturi și tehnologii 5

1. Arhitectura client-server 5

1.1. Ce este arhitectura client-server? 5

1.2. Ce înseamnă RESTful? 6

2. Aplicații de tip client 7

2.1. Scurt istoric 7

2.2. Tehnologii populare 7

2.2.1. Cod nativ 7

2.2.2. Ionic 7

2.2.3. Flutter 8

2.2.4. React Native 9

Bibliografie 10

# **Introducere**

Această lucrare își propune studierea diferitelor concepte ce țin de dezvoltarea aplicațiilor pentru dispozitive mobile, precum: arhitectura client-server, diferitele moduri și tehnologii prin intermediul cărora se dezvoltă aplicațiile mobile, scurt studiu al popularității framework-urilor pentru cod backend și al factorilor care trebuie luați în considerare pentru alegerea unuia dintre acestea. De asemenea, se va urmării realizarea unei aplicații care să ruleze pe dispozitive mobile care să aibă capabilități de afișare și de colectare a informațiilor care descriu diferite puncte de interes de care oamenii au nevoie în viața cotidiană.

**Arhitectura client-server** constituie o componentă de bază a aplicațiilor care rulează pe dispozitivele de tip client (navigatoare web, telefoane etc.). O bună analogie pentru a explica această arhitectură este aceea a unei bucătării a unui restaurant, aplicația de tip client fiind un client fizic al restaurantului iar bucătăria fiind un server, clientul (utilizatorul aplicației) face diferite cereri către bucătarie (server-ul) care pregătește preparatele (datele solicitate de către client). Această arhitectură are diferite beneficii, precum separarea sarcinilor (în contextul analogiei prezentate, clientul nu trebuie să stie cum se fac diferitele preparate), scalabilitatea (în contextul analogiei prezentate, se pot deschide mai multe restaurante pentru deservirea a mai multor clienți) și securitatea (clienții au acces doar la rezultatul final al bucătăriei).

Pe parcursul dezvoltării aplicațiilor de mobile s-au creat diferite **metode de dezvoltare** pentru a creea aceste aplicații. Inițial acestea s-au realizat prin scrierea de cod nativ în limbajele Java pentru Android și respectiv Objective-C pentru iOS. De-a lungul timpului s-a urmărit simplificarea dezvoltării acestor aplicații prin intermediul a diferite alte limbaje si framework-uri (de exemplu: Ionic, React Native, Flutter etc.), aceste soluții la rândul lor având propriile limitări.

Pentru partea de backend se vor studia **framework-uri populare** pentru realizarea de servere web. Acestea vor fi comparare din punct de vedere al avantajelor și dezavantajelor aferente și al arhitecturii acestora în scopul alegerii unuia dintre acestea pentru realizarea unui API de tip RESTful.

# **Capitolul I – Aplicații mobile: Istoric, arhitecturi și tehnologii**

## **1. Arhitectura client-server**

### 1.1. Ce este arhitectura client-server?

Arhitectura client-server este un model de rețea utilizat pentru a organiza aplicațiile și resursele într-un mod eficient și scalabil. În acest model, există două părți principale: clientul și serverul. Clientul este dispozitivul sau aplicația care solicită servicii sau resurse, iar serverul este dispozitivul sau aplicația care furnizează aceste servicii sau resurse.

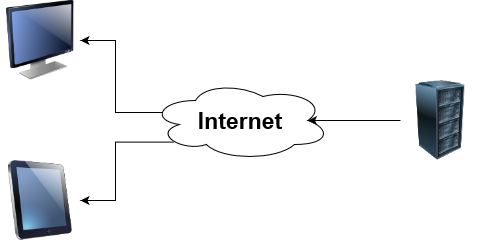
 Arhitectură client-server reflectă mai îndeaproape arhitectura de bază a web-ului în sine. Deși nu este specifică unui protocol anume, arhitectura REST a fost dezvoltată pe baza și împreună cu protocolul HTTP. Arhitectura REST definește constrângeri asupra utilizării HTTP. Aceasta încearcă să descrie o aplicație web bine proiectată: o aplicație fiabilă care funcționează bine, se extinde, are un design simplu și elegant și poate fi modificată ușor [1, pp. 6-7].

Figura 1 Reprezentare a arhitecturii client-server

Clientul trimite o cerere către server prin intermediul unei rețele, cum ar fi Internetul. Serverul primește cererea, procesează informațiile necesare și returnează un răspuns clientului. Acest model este utilizat pe scară largă în diverse aplicații, inclusiv în navigarea pe web, unde browserul web acționează ca un client, iar serverul web furnizează paginile web solicitate.

Un avantaj major al arhitecturii client-server este separarea clară a responsabilităților. Clientul se ocupă de interfața cu utilizatorul și de prezentarea datelor, în timp ce serverul gestionează stocarea, procesarea și securitatea datelor. Acest lucru permite dezvoltatorilor să îmbunătățească și să actualizeze fiecare componentă separat, fără a afecta cealaltă parte.

Scalabilitatea este un alt beneficiu al arhitecturii client-server. Serverele pot fi extinse sau adăugate în funcție de necesitățile utilizatorilor. De asemenea, acest model permite distribuirea sarcinilor între mai multe servere, asigurând astfel o performanță și o fiabilitate mai mare.

Securitatea este un alt aspect important al arhitecturii client-server. Serverele pot implementa măsuri de securitate pentru a proteja datele și resursele împotriva accesului neautorizat. Autentificarea, criptarea și firewall-urile sunt câteva dintre metodele utilizate pentru a asigura securitatea în acest model.

### 1.2. Ce înseamnă RESTful?

RESTful se referă la aplicațiile web care respectă principiile și constrângerile arhitecturii REST (Representational State Transfer). O aplicație RESTful utilizează HTTP pentru a efectua operațiuni CRUD (Create, Read, Update, Delete) pe resurse identificate de URI-uri (Uniform Resource Identifiers).

Arhitectura REST implică câteva reguli principale pentru organizarea unei aplicații web, acestea sunt: organizarea datelor în resurse, executarea a diferite acțiuni prin verbe bine definite și adresarea resurselor prin intermediul URI-urilor.

O resursă web este pur și simplu ceva disponibil pe Web. Definiția unei resurse web s-a extins de-a lungul timpului. Inițial, se referea la un document sau fișier static, adresabil. Ulterior, a fost definită într-un mod mai abstract și acum include orice entitate care poate fi identificată, denumită, adresată sau gestionată pe Web. Câteva exemple de resurse sunt o pagină web HTML tradițională, un document, un fișier audio și un fișier imagine. Resursele pot face referire și la lucruri pe care nu le-ai găsi în biblioteca ta digitală tipică, cum ar fi un dispozitiv hardware, o persoană sau o colecție de alte resurse. [1, pp. 38-40]

În REST, verbele descriu tipuri de operațiuni care se pot realiza pe resurse și corespund cu metode HTTP cărora le sunt asociate semnificații specifice. Principalele acțiuni sunt:

* GET pentru operațiuni de citire;
* POST pentru crearea de resurse;
* PUT pentru modificarea completă a unei resurse;
* PATCH pentru modificarea parțială a unei resurse;
* DELETE pentru ștergerea unei resurse.

Aceste resurse sunt structurate și adresate prin intermediul URI-urilor, structura indicând și  
relația sau ierarhia dintre resurse.

## 2. Aplicații de tip client

### 2.1. Scurt istoric

În 2007 Apple lansează primul iPhone și SDK-ul iOS. Dezvoltatorii folosesc Objective-C pentru a crea aplicații native pentru iOS. Aplicatiile native sunt optimizate pentru performanță și interacțiune directă cu hardware-ul telefonului.

După succesul Apple, în 2008 Google lansează Android și Android SDK. Limbajul de programare principal pentru dezvoltarea aplicațiilor native Android este Java. Dezvoltarea nativă pe Android oferă acces complet la API-urile sistemului și la resursele hardware.

Prin anii 2010 apar diferite soluții care se folosesc de tehnologii web pentru dezvoltarea aplicațiilor mobile. În 2011 Adobe lansează PhoneGap (acum Apache Cordova), care permite dezvoltatorilor să folosească tehnologii web pentru a crea aplicații mobile ce pot fi distribuite prin App Store și Google Play.

În 2013 Facebook lansează React Native, un framework care permite dezvoltarea aplicațiilor mobile folosind JavaScript și React. React Native transpune componentele React în componente native, oferind performanță aproape nativă și o experiență de utilizare fluidă. În 2015 React Native devine open source, iar popularitatea sa crește rapid datorită comunității de dezvoltatori activi.

Google lansează Flutter în 2018, un nou framework pentru dezvoltarea aplicațiilor mobile cross-platform. Flutter folosește limbajul de programare Dart și oferă un motor de randare rapid și flexibil. Flutter permite dezvoltarea aplicațiilor cu o performanță aproape nativă și o interfață de utilizator consistentă pe iOS și Android.

### 2.2. Tehnologii populare

#### 2.2.1. Cod nativ

În ciuda apariției noilor abordări de dezvoltare a aplicațiilor mobile, dezvoltarea prin cod nativ înca este o varianta populara deoarece oferă cel mai mare grad de control asupra comportamentului aplicației și datorită performanței aplicațiilor dezvoltate. Cu toate că inițial se foloseau limbajele Java pentru Android și respectiv Objective-C pentru iOS, în prezent acestea au scăzut în popularitate odata cu apariția limbajului Kotlin pentru Android și respectiv a limbajului Swift pentru iOS.

#### 2.2.2. Ionic

Ionic + Capacitor este succesorul Apache Cordova, inițial Ionic a fost un framework construit peste Apache Cordova și suporta doar AngularJS, în prezent Ionic + Capacitor oferă o soluție completă pentru a dezvolta aplicații mobile prin intermediul tehnologiilor web.

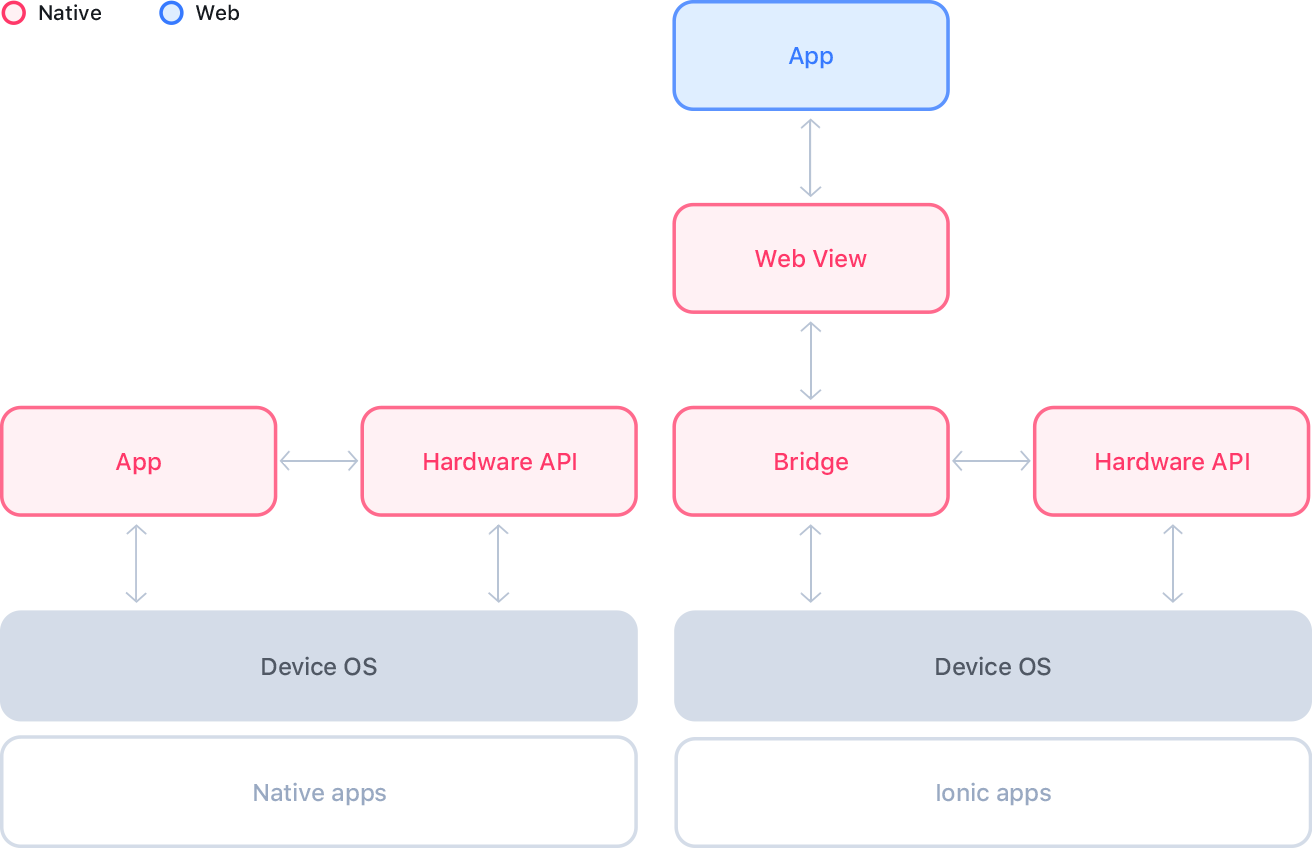


Figura 2 Arhitectura folosită de Ionic + Capacitor [4]

#### 2.2.3. Flutter

Flutter este un kit de dezvoltare software creat de Google, destinat creării de aplicații mobile, web și desktop fiind nevoie ca codul să fie scris o singură dată. Utilizând limbajul de programare Dart, Flutter permite dezvoltarea pe diferite platforme, oferind un framework UI flexibil și reactiv. Printre funcționalitățile sale notabile se numără și "hot reload", care permite vizualizarea instantanee a modificărilor de cod fără repornirea aplicației, și compilarea directă în cod nativ care asigură performanța aplicațiilor. Ecosistemul Flutter include o comunitate mare și o varietate de pluginuri și pachete disponibile.

Architectural
diagramFlutter utilizează motorul grafic open-source Skia pentru randare grafică, oferind performanță ridicată, suport multi-platformă, grafică vectorială, accelerare hardware și compatibilitate cu OpenGL și Vulkan. Acesta asigură interfețe rapide și atractive pe diverse dispozitive și sisteme de operare.

Figura 3 Arhitectura Flutter [3]

De la apariția sa, Flutter a fost într-o continuă creștere în popularitate, dupa 2020 Flutter a depășit React Native. În prezent, comparativ cu React Native, Flutter este mai căutat pe motoare de căutare, are mai multe stele pe github și se pun mai multe întrebari cu tag-ul lui pe platforme precum StackOverflow. [2]

#### 2.2.4. React Native

# Bibliografie

|  |  |
| --- | --- |
| [1] | C. C. Saternos, Client-server Web Apps with JavaScript and Java, O'Reilly Media, 2014. |
| [2] | P. Sulimau, „Flutter vs Competitors: Popularity,” 18 03 2024. [Interactiv]. Available: https://pasul.medium.com/flutter-vs-competitors-popularity-f79536688ec3. [Accesat 11 06 2024]. |
| [3] | Google, „Flutter architectural overview,” [Interactiv]. Available: https://docs.flutter.dev/resources/architectural-overview. [Accesat 12 06 2024]. |
| [4] | „Ionic Documentation, Web View,” Drifty Co., [Interactiv]. Available: https://ionicframework.com/docs/core-concepts/webview. [Accesat 11 06 2024]. |