# Prenos informacija putem HTTP-a

## HTTP – Hypertext Transfer Protocol

HTTP je protokol aplikativnog nivoa koji se koristi za prenos informacija na Web-u. Zasnovan je na klijent-server arhitekturi, gde klijent predstavlja endpoint koji se javlja serveru radi dobavljanja i slanja podataka. Komunikacija je dvosmerna – klijent šalje zahteve, a server mu odgovara.

### Poruke zahteva

Poruke zahteva (*request messages*) se sastoje od sledećeg:

- linija zahteva – *GET /images/logo.png  HTTP/1.1*
- header-i zahteva
- prazna linija
- opciono telo zahteva

### Poruke odgovora

Poruke odgovora (*response messages*) se sastoje od sledećeg:

- status linija – *HTTP/1.1 200 OK*
- zaglavlje odgovora
- prazna linija
- opciono telo odgovora

### Metode zahteva:

- GET – dobavljanje podataka; parametri se šalju unutar linije zahteva - */images?type=png&name=logo*; treba izbegavati slanje osetljivih podataka ovom metodom
- HEAD – slično kao GET, samo što se ne vraća ceo odgovor nego samo zaglavlje
- POST – slanje podataka serveru; podaci se šalju u telu zahteva
- DELETE – brisanje određenog resursa sa servera
- PUT
- TRACE
- OPTIONS
- PATCH
- CONNECT

### Statusni kodovi

Statusni kodovi su podeljeni u sledeće grupe:

- Informacije – 1xx
- Uspešno – 2xx

- Redirekcija – 3xx
- Client error – 4xx
- Server error – 5xx

Lista statusnih kodova

### Zaglavlja

Polja zaglavlja poruke predstavljaju parametre u HTTP komunikaciji. Neki od primera polja zaglavlja poruka zahteva:

- Accept – tipovi koji su prihvatljivi kao odgovor (*Accept: text/html*)
- Authorization – kredencijali HTTP autentikacije (*Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==*)
- Content-Type – tip tela zahteva (*Content-Type: application-x-www-form-urlencoded*)

Neki od primera polja zaglavlja poruka odgovora:

- Content-Encoding – tip kodovanja podataka (*Content-Encoding: gzip*)
- Set-Cookie – HTTP cookie (*Set-Cookie: UserID=JohnDoe; Max-Age=3600; Version=1*)

## JSON struktura podataka

JSON – JavaScript Object Notation je format koji se sastoji od objekata tipa *key-value*, gde su ključevi jedinstveni unutar objekta. Primer jednog JSON objekta:

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [],
  "spouse": null
}
```

Klase u Javi koje reprezentuju JSON objekte su *JSONObject* i *JSONArray*.

Dobavljanje svih ključeva i odgovarajućih vrednosti JSON objekta:

```
Iterator<String> iterator = json.keys();
while (iterator.hasNext()) {
    String key = iterator.next();
    json.getString(key);
}
```

## HttpURLConnection

Klasa koja se koristi za uspostavljanje HTTP konekcije ka serveru. Jedna instanca ove klase odgovara jednom zahtevu poslatom ka serveru.

Kako bi Android aplikacija mogla da šalje HTTP zahteve u Manifest fajlu je potrebno dati dozvolu aplikaciji za korišćenje Interneta:

```
<manifest xlmns:android...>
 ...
 <uses-permission android:name="android.permission.INTERNET" />
 <application ...
</manifest>
```

Primer slanja HTTP GET zahteva za dobavljanje odgovora tipa JSONObject:

```
/*HTTP get json object*/
public JSONObject getJSONObjectFromURL(String urlString) throws IOException, JSONException {
    HttpURLConnection urlConnection = null;
    java.net.URL url = new URL(urlString);
    urlConnection = (HttpURLConnection) url.openConnection();
    /*header fields*/
    urlConnection.setRequestMethod("GET");
    urlConnection.setRequestProperty("Accept", "application/json");
    urlConnection.setReadTimeout(10000 /* milliseconds */ );
    urlConnection.setConnectTimeout(15000 /* milliseconds */ );
    try {
        urlConnection.connect();
    } catch (IOException e) {
        return null;
    }
    BufferedReader br = new BufferedReader(new InputStreamReader(url.openStream()));
    StringBuilder sb = new StringBuilder();
    String line;
    while ((line = br.readLine()) != null) {
        sb.append(line + "\n");
    }
    br.close();

    String jsonString = sb.toString();
    Log.d("HTTP GET", "JSON obj- " + jsonString);
    int responseCode =  urlConnection.getResponseCode();
    urlConnection.disconnect();
    return responseCode == SUCCESS ? new JSONObject(jsonString) : null;
}
```

Primer slanja HTTP POST zahteva:

```
/*HTTP post*/
public boolean postJSONObjectFromURL(String urlString, JSONObject jsonObject)
    throws IOException, JSONException {
        HttpURLConnection urlConnection = null;
        java.net.URL url = new URL(urlString);
        urlConnection = (HttpURLConnection) url.openConnection();
        urlConnection.setRequestMethod("POST");
        urlConnection.setRequestProperty("Content-Type", "application/json;charset=UTF-8");
        urlConnection.setRequestProperty("Accept","application/json");
        /*needed when used POST or PUT methods*/
        urlConnection.setDoOutput(true);
        urlConnection.setDoInput(true);
        try {
            urlConnection.connect();
        } catch (IOException e) {
            return false;
        }
        DataOutputStream os = new DataOutputStream(urlConnection.getOutputStream());
        /*write json object*/
        os.writeBytes(jsonObject.toString());
        os.flush();
        os.close();
        int responseCode =  urlConnection.getResponseCode();
        Log.i("STATUS", String.valueOf(urlConnection.getResponseCode()));
        Log.i("MSG" , urlConnection.getResponseMessage());
        urlConnection.disconnect();
        return (responseCode==SUCCESS);
}
```

*openStream* metoda dobavlja *stream* iz koga se čitaju podaci. Ova metoda vraća objekat tipa *java.io.InputStreamReader*, tako da se čitanje svodi na čitanje iz input stream-a. Slično, slanje podataka se vrši pomoću *output stream-a*.

*BufferedReader* – klasa koja služi za čitanje iz input stream-a; u konstruktoru očekuje objekat koji predstavlja stream.

*DataOutputStream* – upis u output stream.

## Konkurencija u Javi

Konkurentnost u Javi je obezbeđena pomoću *Thread* klase i *Runnable* interfejsa. Razlike između ova dva pristupa:

1. Nasleđivanjem *Thread* klase, svaki od *thread*-ova ima jedinstveni objekat, a implementiranjem *Runnable* interfejsa više *thread*-ova mogu da dele istu instancu objekta
2. Kada se nasledi *Thread* klasa, ni jedna druga klasa ne može da se nasledi; sa druge strane kada se implementira *Runnable* interfejs, moguće je implementirati još drugih klasa i eventualno naslediti neku klasu po potrebi.

## Thread klasa

```
class Multi extends Thread{
public void run(){
System.out.println("thread is running...");
}
public static void main(String args[]){
Multi t1=new Multi();
t1.start();
 }
}
```

## Runnable interfejs

```
class Multi3 implements Runnable{
public void run(){
System.out.println("thread is running...");
}

public static void main(String args[]){
Multi3 m1=new Multi3();
Thread t1 =new Thread(m1);
t1.start();
 }
}
```

Napomena: Voditi računa o tome da se prikaz podataka uvek obavlja iz Main thread-a, u kome su dobavljeni View-i!