Univerzitet u Sarajevu
Prirodno-matematički fakultet
Odsjek za matematiku

Dokumentacija za projekat iz predmeta
Razvoj mobilnih aplikacija, na temu

Chat aplikacija

Studenti:
Kreso Tarik
Ruhotina Belmin

Sadržaj

Uvod

Tema našeg projekta je, kako sam naslov kaže, Chat aplikacija. U današnje vrijeme, bilo koja vrsta Chat aplikacije je nezaobilazna, a ponekad, i ta jedna nije dovoljna. Pouceni primjerima raznih, ali donekle slicnim, implementacijama takvog rješenja, medu kojima je malo poznata neka koja nema svoju bazu korisnika, projekat Chat aplikacije nam omogucava da u jednoj ideji obuhvatimo, otprilike, vecinu najvažnih koncepata web komunikacije, android developmenta, te nadasve komercijalnog programiranja, uz to posjedujuci real-life aplikaciju spremnu za upotrebu. Ideja bilo koje implementacije Chat aplikacije je ista: korisnik kreira svoj racun sa kojim pristupa "imaginarnoj" chat sobi, unutar koje se nalaze i ostali trenutno prijavljeni korisnici sa ranije kreiranim racunima. U takvoj chat sobi, korisnici medjusobno razmjenjuju poruke, slike, i razne druge vrste fajlova. Da bi se takvo što uspjelo implementirati, potrebno je savladati, kao što je vec navedeno, razlicite koncepte. Na prvom mjestu, s obzirom da se radi o chat aplikaciji za mobitele, potrebno je implementirati aplikaciju (izgled i funkcionalnost) u pogodnom razvojnom okruženju, u našem slucaju, radi se o Javi i Android aplikaciji. S obzirom da se poruke koje korisnici razmjenjuju prenose internetom, potrebno je instancirati server koji ce biti most u komunikaciji medu korisnicima. Jedna takav server treba da spasi dospjelu poruku i njene pripadajuce informacije, te na osnovu zaglavlja te poruke, proslijedi istu naznacenom korisniku i/ili korisnicima. Na kraju, potrebno je implementirati funkcionalnost vezanu za samu komunikaciju aplikacije i web servera. Ta funkcionalnost sastoji se od dva dijela, dio implementacije na host-u, tj. aplikaciji, gdje se nalaze potrebne zavisnosti za slanje/primanje poruka od servera te njihovo renderanje (validni prikaz) na view-u aplikacije, a sa druge strane dio implementacije na web serveru, koji bi vršio usluge korisnicima na osnovu njihovih zahtjeva (HTTP zahtjeva, preciznije).

Aktivnosti

Aplikacija se sastoji od dvije aktivnosti :

- MainActivity
- Chat

MainActivity

Prvo ćemo predstaviti izgled ove aktivnosti, pa ćemo u nastavku objasniti klasu MainActivity.

Slika 1. Izgled aktivnosti MainActivity (Portrait)

Ova aktivnost predstavlja login formu, koja trenutno nema implementiranu autentifikaciju , gdje korisnik unosi svoj username, te klikom na dugme „Pridruzi se“ biva prebaćen u chat sobu u kojoj može pogledati poruke ostalih učesnika u konverzaciji, te naravno i sam učestvovati slanjem svojih poruka.
Što se tiće **activity_main.xml** fajla, on se sastoji od sljedećih komponenti :

- LinearLayout
    - ImageView (chat ikonica)
    - TextView („Unesite Vase ime“)
    - EditText (bijelo polje koje korisniku omogućava unos svog username-a)
    - Button („Pridruzi se“)
    - Space (potreban za razmake između pojedinih komponenti radi ljepšeg izgleda)

Nazivi komponenti EditText,TextView i Button su definisani u **strings.xml** fajlu.

```
<resources>
    <string name="app_name">Chatroom</string>

    <string name="login_text">Unesite Vase ime</string>
    <string name="login_username"></string>
    <string name="login_button">Pridruzi se</string>
    <string name="login_toast">Unesite Vas username</string>

    <string name="send">Posalji</string>
    <string name="message"></string>


</resources>
```

Slika 2. Nazivi komponenti MainActivity-a

Za komponente EditText i Button su definisani odgovarajući **id**-ovi jer će biti potrebno pokupiti sadržaj EditText-a , te klikom na dugme Button izvršiti odgovarajuću akciju.

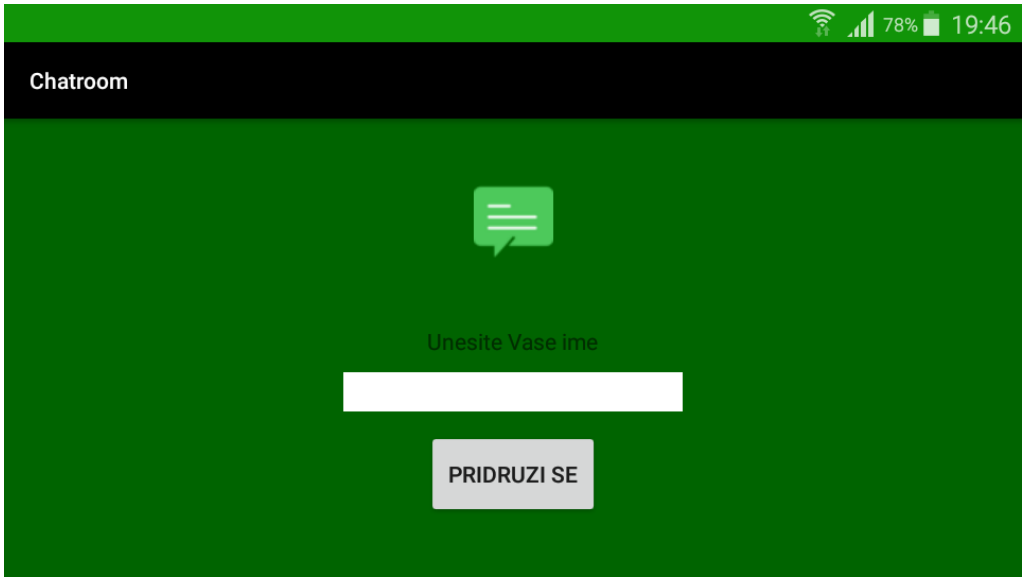
```
<EditText
    android:id="@+id/input"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:ems="10"
    android:background="@color/white"
    android:inputType="textPersonName"
    android:text="" />
```

Slika 3.1 Komponenta EditText

```
<Button
    android:id="@+id/submit"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/login_button" />
```

Slika 3.2 Komponenta Button

U slučaju rotacije ekrana, definisan je i **landscape** izgled u **activity_main.xml (land)** fajlu.



Slika 4. Izgled aktivnosti MainActivity (Landscape)

Pređimo sada na objašnjavanje značenja i svrhe pojedinih **atributa** i **metoda** vezanih za klasu **MainActivity**.

Klasa MainActivity sadrži sljedeće atribute :

```
private static final String USER = "username";

private EditText input;
private Button submit;
```

Slika 5. Atributi klase MainActivity

Atribut **input** nam je potreban kako bi mogli imati pristup sadržaju teksta kojeg je korisnk unio u komponentu EditText, dok nam je atribut **submit** potreban kako bi mogli izvršiti odgovarajuću akciju, u našem slučaju je to pokretanje aktivnosti **Chat**.
Atribut **USER** nam je potreban radi rotacije ekrana. To ćemo detaljnije objasniti u nastavku.

Objasnimo sada šta se dešava ukoliko korisnik klikne dugme „Pridruži se“.

```
submit = (Button) findViewById(R.id.submit);
submit.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        if(MainActivity.this.input.getText().toString().equals("")){
            Toast.makeText( context: MainActivity.this,R.string.login_toast,Toast.LENGTH_SHORT).show();
        }else{
            Intent intent = new Intent( packageContext: MainActivity.this,Chat.class);
            intent.putExtra( name: "user", MainActivity.this.input.getText().toString());
            startActivity(intent);
        }

    }
});
```

Slika 6. OnClickListener za Button submit

Klikom na dugme „Pridruži se“, izvršava se funkcija **onClick** gdje se prvo provjerava da li je korisnik unio svoj username u komponentu EditText. Ukoliko nije, prikazuje se **toast** u dnu ekrana koji govori korisniku da unese svoj username.

Slika 7. Prikazivanje toast-a

Ukoliko je korisnik unio svoj username, onda pravimo objekat **intent** klase **Intent** u koji smještamo sadržaj koji je korisnik unio u EditText (na ovaj način vršimo prenos podataka sa jedne aktivnosti na drugu), te pokrećemo aktivnost **Chat**.

Ukoliko je korisnik krenuo nakucavat svoj username u EditText komponentu, pa iz nekog razloga izvršio **rotaciju** ekrana, potrebno je sačuvati to što je nakucao te postaviti u EditText nakon što se prikazao **landscape** izgled aplikacije.

Prije same rotacije, moramo sačuvati to što je nakucao na sljedeći način:

```
@Override
protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    outState.putString(USER,input.getText().toString());
}
```

Slika 8. Spašavanje sadržaja EditText-a prije rotacije

Nakon rotacije ekrana, aplikacija se uništila pa pokrenula iz početka, što znači da je potrebno nakon njenog ponovnog pokretanja postaviti sadržaj EditText-a na prethodni što je urađeno na sljedeći način:

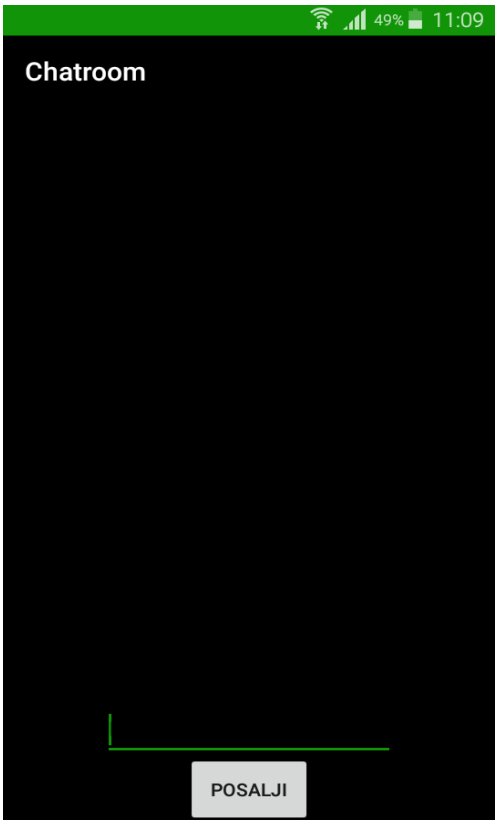
```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    input = (EditText) findViewById(R.id.input);

    if(savedInstanceState != null) {
        input.setText(savedInstanceState.getString(USER));
    }
```

Slika 9. Postavljanje sadržaja EditText-a

Iz prethodnog smo ujedno i vidjeli svrhu atributa **USER**, jer se podaci spašavaju u obliku **key-value** vrijednosti , pri čemu su key vrijednosti striktno stringovi.

Chat activity

Prvo ćemo predstaviti izgled ove aktivnosti, pa ćemo u nastavku objasniti klasu Chat.



Slika 1. Izgled Chat activity-a (Portrait)

Ova aktivnost predstavlja chat sobu gdje korisnik može da vidi poruke ostalih učesnika u konverzaciji, te da i sam razmjenjuje poruke sa njima.

Šti se tiče **activity_chat.xml** fajla, on se sastoji od sljedećih komponenti :

- RelativeLayout
  - ScrollView
    - LinearLayout
  - LinearLayout
    - EditText
    - Button

ScrollView nam je potreban jer kada se čitav ekran popuni porukama, ostale nećemo moći vidjeti ukoliko nemamo ScrollView.
EditText je potreban jer on omogućuje unos poruke korisniku, dok Button omogućava slanje te iste poruke na server (server će biti detaljnije objašnjen u nastavku).

Nazivi komponenti EditText i Button su definisani u **strings.xml** fajlu.

```
<resources>
    <string name="app_name">Chatroom</string>

    <string name="login_text">Unesite Vase ime</string>
    <string name="login_username"></string>
    <string name="login_button">Pridruzi se</string>
    <string name="login_toast">Unesite Vas username</string>

    <string name="send">Posalji</string>
    <string name="message"></string>

</resources>
```

Slika 2. Nazivi komponenti EditText-a i Button-a

Također, za komponente EditText i Button su postavljeni odgovarajući **id**-ovi jer je potrebno dohvatiti sadržaj EditText-a nakon unosa korisnika te klikom na Button poslati taj sadržaj na server.

```
<EditText
    android:id="@+id/mess"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:ems="10"
    android:inputType="textPersonName"
    android:hint="Unesite poruku .."
    android:text="@string/message" />

<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/send" />
```

Slika 3. ID-ovi komponenti

U slučaju rotacije ekrana, definisan je i **landscape** izgled u **activity_chat.xml (land)** fajlu.

Slika 4. Izgled aktivnosti Chat (Landscape)

Pređimo sada na objašnjavanje značenja i svrhe pojedinih **atributa** i **metoda** vezanih za klasu **Chat**.
Klasa Chat posjeduje sljedeće atribute :

```
private final String URL = "https://chatroom-android.herokuapp.com/";

private String currentUser;
private Button send;
private EditText et1;
private LinearLayout linearLayout;

private RequestQueue requestQueue;

private Handler handler = new Handler();
private Runnable runnable;
private int delay = 300;
```

Slika 5. Atributi klase Chat

Atribut **currentUser** čuva username korisnika koji je proslijeđen od strane **MainActivity**-a.
Atributi **et1** i **send** su nam potrebni za pristup EditText-u i Button-u, dok ćemo pomoću atributa **linearLayout** postavljati poruke na ekran. Ostali atributi će biti objašnjeni u nastavku kada dođe pogodno vrijeme za njihovo objašnjenje.

Objasnimo sada šta se dešava ukoliko korisnik klikne dugme „Pošalji“.

```
send = (Button) findViewById(R.id.button1);
send.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        Calendar cal = Calendar.getInstance();
        Date date=cal.getTime();
        DateFormat dateFormat = new SimpleDateFormat( pattern: "HH:mm:ss");
        String formattedDate=dateFormat.format(date);

        String data = "{"+
                "\"username\"" + ":\"" + Chat.this.currentUser + "\","+
                "\"message\"" + ":\"" + Chat.this.et1.getText().toString() + "\","+
                "\"time\"" + ":\"" + formattedDate + "\""+
                "}";

        Chat.this.et1.setText("");
        Chat.this.sendData(data);
    }
});
```

Slika 6. onClickListener za Button Pošalji

Klikom na Button **Pošalji**, izvršava se funkcija **onClick**. Prvo, uzmemo vrijeme kada je korisnik kliknuo dugme, zatim napravimo **JSON** koji sadržava username korisnika, poruku koju je unio u EditText, te vrijeme kada je kliknuo button. Poslije toga, ispraznimo sadržaj EditText-a, te pozovemo funkciju **sendData** kojoj proslijedimo JSON, a ona taj JSON šalje na server.

Pogledajmo sada implementaciju funkcije **sendData**.

```
private void sendData(String data) {

    final String savedata= data;

    requestQueue = Volley.newRequestQueue(getApplicationContext());

    StringRequest stringRequest = new StringRequest(Request.Method.POST, URL, new Response.Listener<String>() {
        @Override
        public void onResponse(String response) {
            try {
                JSONObject objres = new JSONObject(response);
                //Toast.makeText(getApplicationContext(),objres.toString(),Toast.LENGTH_LONG).show();

            } catch (JSONException e) {
                Toast.makeText(getApplicationContext(), text: "Server Error",Toast.LENGTH_LONG).show();
            }

        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {

            Toast.makeText(getApplicationContext(), error.getMessage(), Toast.LENGTH_SHORT).show();

        }
    }) {
```

```
        @Override
        public String getBodyContentType() { return "application/json; charset=utf-8"; }

        @Override
        public byte[] getBody() throws AuthFailureError {
            try {
                return savedata == null ? null : savedata.getBytes( charsetName: "utf-8");
            } catch (UnsupportedEncodingException uee) {

                return null;
            }
        }

    };
    requestQueue.add(stringRequest);

}
```

Slika 7. Implementacija funkcije sendData

Atribut **requestQueue** nam je potreban jer u njega dodajemo HTTP zahtjeve kao što su GET i POST u našem slučaju i on vodi svu brigu oko njih i mi nemamo potrebe da brinemo oko nekih pozadinskih procesa jer on to sve radi za nas.

Naravno, pored slanja poruka na server, potrebno je i dohvatiti te poruke sa servera i prikazati ih na ekranu. U svrhu dohvatanja poruka sa servera, implementirana je funkcija **getData.** Pogledajmo njenu implementaciju.

```
private void getData() {

    requestQueue = Volley.newRequestQueue(getApplicationContext());

    JsonArrayRequest request = new JsonArrayRequest(URL, new Response.Listener<JSONArray>() {
                @Override
                public void onResponse(JSONArray response) {
                    try{

                        Chat.this.linearLayout.removeAllViews();

                        for(int i=0;i<response.length();i++){

                            JSONObject obj = response.getJSONObject(i);

                            String user = obj.getString( name: "user");
                            String message = obj.getString( name: "mess");
                            String time = obj.getString( name: "time");

                            String text = time + " " + user + " : " + message;

                            if(user.equals(Chat.this.currentUser)) {
                                Chat.this.newTextView(text, where: "right");
                            }else{
                                Chat.this.newTextView(text, where: "left");
                            }
```

```
                    }
                }catch (JSONException e){
                    e.printStackTrace();
                    //Toast.makeText(getApplicationContext(),"Error: " + e.getMessage(),Toast.LENGTH_LONG).show();
                }
            }
        },
        new Response.ErrorListener(){
            @Override
            public void onErrorResponse(VolleyError error){
                //Toast.makeText(getApplicationContext(),error.getMessage(),Toast.LENGTH_LONG).show();
            }
        });

    requestQueue.add(request);
}
```

Slika 8. Implementacija funkcije getData

Sa servera smo dobili niz **JSON** objekata koji sadrže **username** korisnika, njegovu **poruku** i **vrijeme** slanja. Opet nam je potreban atribut **requestQueue** koji upravlja HTTP zahtjevima. Kod obje funkcije sendData i getData, koristili smo **Volley** biblioteku. Za njeno korištenje, bilo ju je potrebno uključiti u projekat. To smo uradili tako što smo **implementation 'com.android.volley:volley:1.1.0'** u **dependenciens** u **app gradle** fajl. Da bi mogli koristiti ovu bliblioteku, morali smo još i dodati **android.permission.INTERNET** u **AndroidManifest.xml**.

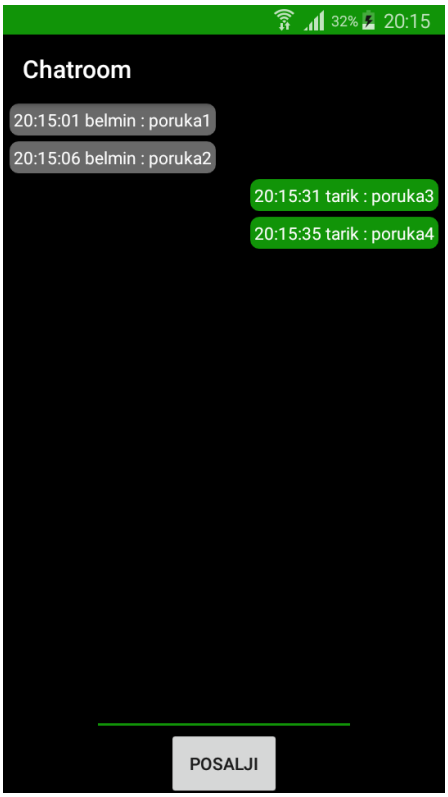
Za prikazivanje poruka na ekranu, implementirali smo funkciju **newTextView**. Pogledajmo njenu implementaciju.

```
private void newTextView(String text, String where) {


    LinearLayout ll = new LinearLayout( context: Chat.this);
    ll.setLayoutParams(new LinearLayout.LayoutParams(
            LinearLayout.LayoutParams.MATCH_PARENT,
            LinearLayout.LayoutParams.WRAP_CONTENT
    ));

    ll.setOrientation(LinearLayout.HORIZONTAL);
    if(where.equals("right")) {
        ll.setGravity(Gravity.RIGHT);
    }else{
        ll.setGravity(Gravity.LEFT);
    }



    TextView tv = new TextView( context: Chat.this);
    LinearLayout.LayoutParams params = new LinearLayout.LayoutParams(LinearLayout.LayoutParams.WRAP_CONTENT,
                                                                     LinearLayout.LayoutParams.WRAP_CONTENT);
    params.setMargins( left: 0, top: 0, right: 0, bottom: 7);
    tv.setLayoutParams(params);
    tv.setPadding( left: 5, top: 5, right: 5, bottom: 5);
```

```
int[] colors_right = {getResources().getColor(R.color.green),getResources().getColor(R.color.green)};
int[] colors_left = {getResources().getColor(R.color.grey),getResources().getColor(R.color.grey)};

GradientDrawable gd;

if(where.equals("right")){
    gd = new GradientDrawable(GradientDrawable.Orientation.TOP_BOTTOM, colors_right);
    gd.setStroke( width: 2,getResources().getColor(R.color.green));
}else{
    gd = new GradientDrawable(GradientDrawable.Orientation.TOP_BOTTOM, colors_left);
    gd.setStroke( width: 2,getResources().getColor(R.color.grey));
}

gd.setShape(GradientDrawable.RECTANGLE);
gd.setCornerRadius(10.0f);

tv.setBackground(gd);
tv.setText(text);
tv.setTextColor(Color.WHITE);

ll.addView(tv);
this.linearLayout.addView(ll);
```

Slika 9. Implementacija funkcije newTextView

Dakle, prvo napravimo **LinearLayout.** U njega smještamo **TextView** koji sadrži username korisnika, poruku i vrijeme slanja. TextView trenutnog korisnika smještamo uz desnu marginu i oni su obojeni zelenom bojom, dok TextView-e ostalih korisnika smještamo uz lijevu marginu i oni su obojeni sivom bojom.Taj **LinearLayout** smještamo na glavni **LinearLayout**.



Slika 10. Izgled Chatroom-a

Naravno, potrebno je stalno osvježavati ekran kako bi trenutni korisnici imali pregled stanja dosad poslatih poruka. U tu svrhu smo implementirali **handler** koji će izvršavati funkciju **getData** svako 3 ms i na taj način smo obezbjedili ažuriranost poruka.

```
@Override
protected void onResume() {
    super.onResume();

    handler.postDelayed(runnable = (Runnable) () → {
            Chat.this.getData();
            handler.postDelayed(runnable,delay);
    },delay);

}
```

Slika 11. Implementacija handler-a

Ovdje vidimo svrhu atributa **handler, runnable i delay** koji su bili neophodni za implementaciju.
Kada aplikacija pređe u stanje **onPause,** potrebno je ovaj handler zaustaviti jer nema potrebe da se izvršava kad aplikacija nije aktivna.

```
@Override
protected void onPause() {
    super.onPause();

    handler.removeCallbacks(runnable);
}
```

Slika 12. Prekidanje handler-a kada aplikacija nije aktivna

Što se tiče preuzimanja podatka ko je trenutni korisnik, to radimo prilikom pokretanja aktivnosti Chat na sljedeći način:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_chat);

    currentUser = getIntent().getStringExtra( name: "user");
```

Slika 13. Preuzimanje podatka od aktivnosti MainActivity

Server

Za potrebe aplikacije bilo je potrebno implementirati server na koji ćemo spremati **poruke** korisnika te upravljati **GET** i **POST** zahtjevima. Implementiran je **Node.js** server i deploy-an je na **Heroku**. Zahvaljući Heroku, naš server je online.

```
var express = require('express');
var bodyParser = require("body-parser");

var app = express();

const port=process.env.PORT || 3000
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());


var conversations = [];

app.get('/', function (req, res,next) {
  res.send(conversations);
});



app.post('/', function(req, res, next) {
  var newMessage = {
    user : req.body.username,
    mess : req.body.message,
    time : req.body.time
  };
  conversations.push(newMessage);

  res.json({
    success: true
  });

});



app.listen(port, function () {
  console.log('Example app listening on port 3000!');
});
```

Slika 1. Node.js server

U listu **conversations** spremamo **JSON** objekte koji sadrže poruke korisnika. Na **GET** zahtjev, vraćamo listu JSON objekata. Kada dođe **POST** zahtjev, iz body-a izvučemo podatke o korisniku, poruci te vremenu slanja. Na osnovu tih podataka napravimo JSON objekat i spremimo ga u listu convesations. Na kraju, vraćamo odgovor da je POST zahtjev uspješno primljen i obrađen. Način na koji smo deploy-ali server na Heroku, može se pogledati na sljedećem linku : https://medium.freecodecamp.org/how-to-deploy-a-nodejs-app-to-heroku-from-github-without-installing-heroku-on-your-machine-433bec770efe

Zaključak

Kao što je objašnjeno u ovoj dokumentaciji, naš projekat za prvi dio sadrži osnovu (CORE) funkcionalnost koju mora imati bilo koja vrsta Chat aplikacije. Naša osnovna ideja za prvio dio je bila uvezivanje različitih koncepata u jedinstvenu cjelinu koja bi omogučila korisnicima pristup osnovnoj namjeni aplikacije. S obzirom da nastavljamo sa razvojem naše aplikacije, za drugi dio projekta ćemo dodati i ostale funkcionalnosti koje bi omogučile još više namjena za korisnike. Među prvim takvim, implementirati ćemo bazu podataka na našem web serveru, koja bi čuvala historiju konverzacija, vršila user managament, autentifikaciju i autorizaciju korisnika, itd. Dalje, implementacijom takve baze podataka, korisnicima će biti omogučeno da vide ko je trenutno aktivan, ko je pročitao njihovu poruku, uređuju svoj korisnički profil i slične mogučnosti. Na samom kraju, okrenut ćemo se djelu interakcije same aplikacije
sa operativnim sistemom, pomoću koje bi korisnicima omogučili da primaju obavijesti o dospjelim, nepročitanim, porukama, čak i dok aplikacija nije aktivna. Sa navedenim funkcionalnostima, koje će biti implementirane, uz već postojeće, našu će se Chat aplikaciji, moći koristiti baš kao i svaka druga na današnjem marketu.