REST API v rôznych programovacích jazykoch

[C#, Python, Java, Kotlin, JavaScript, NodeJS, MicroPython]

Cieľom tejto prakticky zameranej kapitoly je ukázať prácu s REST API využitím rôznych programovacích jazykov. Ukážky budú založené na scenári automatizovanej správy konfigurácie sieťových zariadení využitím modelovo riadeného prístupu. Pracovať sa tiež bude s rôznymi vývojovými prostrediami (Visual Studio, PyCharm, IntelliJ, Android Studio, Thonny).

Prerekvizity

Táto kapitola je vhodná aj pre úplných začiatočníkov v programovaní. Čitateľom stačí mať základné vedomosti z počítačových sietí (základná konfigurácia smerovača).

Využité technológie

V rámci kapitoly sa bude pracovať s nasledujúcimi technológiami:

1. Rámec .NET, programovací jazyk C#, rámec WPF [Visual Studio]
2. Programovací jazyk Python a knižnica Tkinter [PyCharm]
3. Programovací jazyk Java a knižnica Swing [IntelliJ]
4. Programovací jazyk Kotlin [Android Studio]
5. YANG modely, RESTCONF, JSON, REST API
6. Programovací jazyk Python a knižnica Flask [PyCharm]
7. Programovací jazyk JavaScript a NodeJS [IntelliJ]
8. Programovací jazyk Python – FastAPI [PyCharm]
9. Programovací jazyk MikroPython – ESP32 [Thonny]

Zbierka riešených úloh

Zbierka úloh pozostáva z ukážky vzorového riešenia ôsmich úloh zameraných na precvičenie praktických zručností práce s REST API v rôznych programovacích jazykoch. Pre jednoduchosť bude ukázaná úloha implementácie vyčítania a zmeny názvu zariadenia. Cieľom ale bude ukázať, ako je možné vytvoriť používateľské rozhranie, ako pracovať s REST API (HTTP GET/PUT) a s certifikátmi.

Úloha 0: Inštalácia vývojových prostredí

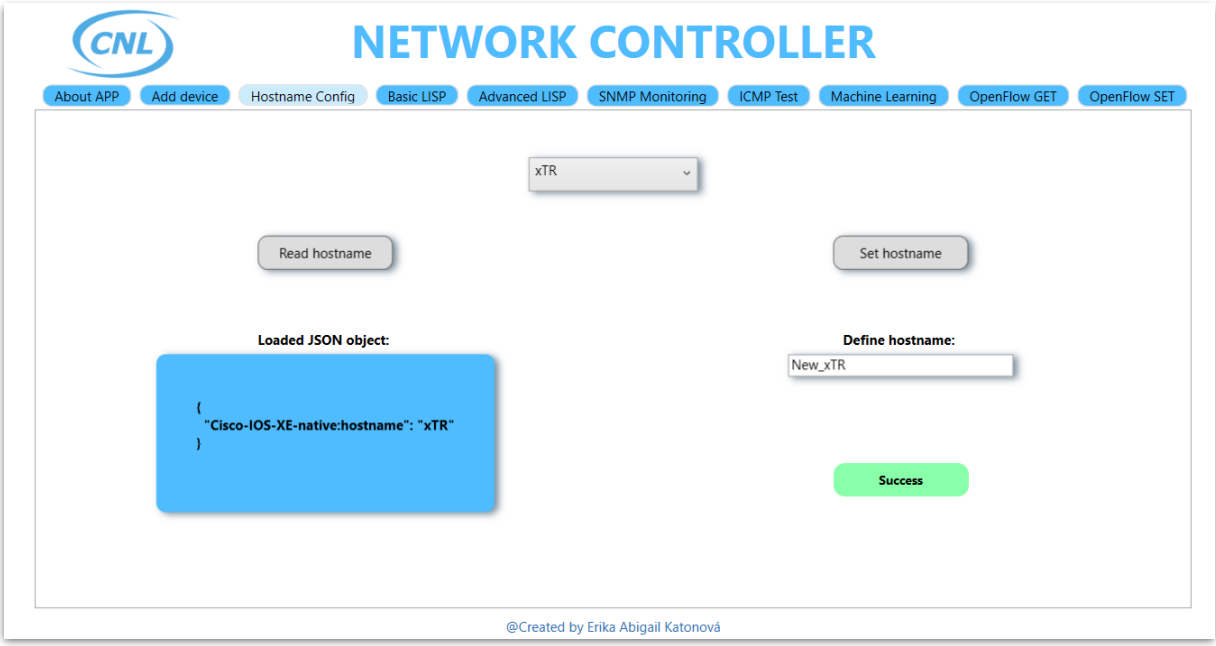
Pri inštalácii vývojových prostredí stačí postupovať štandardným spôsobom, ktorý je možné nájsť na oficiálnych stránkach poskytovaných výrobcami jednotlivých IDE. Nasledujúci zoznam obsahuje linky, odkiaľ je možné stiahnuť vývojové prostredia:

- **Visual Studio [C#]**: https://visualstudio.microsoft.com/cs/downloads/
- **PyCharm [Python]**: https://www.jetbrains.com/pycharm/download/#section=windows
- **IntelliJ [Java]**: https://www.jetbrains.com/idea/download/#section=windows
- **Android Studio [Kotlin]**: https://developer.android.com/studio
- **Thonny [MicroPython]**: https://thonny.org/

Po inštalácii vytvorte v každom vývojovom prostredí nový projekt, ktorý sa zvyčajne vytvára cez záložku **File → New Project**.

Úloha 1: C# – Načítanie a zmena názvu zariadenia

Cieľom tejto úlohy je ukázať, ako je možné využitím programovacieho jazyka C# a rámca WPF vytvoriť používateľské rozhranie a obslužné funkcie pre čítanie a zmenu názvu zariadenia modelovo riadeným prístupom. Obrázok 9.1 znázorňuje ukážku možného používateľského rozhrania:



Obrázok 9.1 Používateľské rozhranie vytvorenej aplikácie

Zdrojový kód pre vytvorenie používateľského rozhrania vyzerá nasledovne:

```
    <Grid Margin="0,20,0,0">
        <Grid.RowDefinitions>
            <RowDefinition Height="1*"/>
            <RowDefinition Height="1*"/>
            <RowDefinition Height="4*"/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="1*"/>
            <ColumnDefinition Width="1*"/>
        </Grid.ColumnDefinitions>
        <ComboBox Name="cmb_hostname" Grid.Row="0" Grid.ColumnSpan="2" Width="150" Height="30"
HorizontalAlignment="Center">
            <ComboBox.Effect>
                <DropShadowEffect Direction="-25" Color="#FFB6C2CB" ShadowDepth="4" BlurRadius="10"/>
            </ComboBox.Effect>
        </ComboBox>

        <Button Name="get_hostname" Content="Read hostname" Grid.Row="1" Grid.Column="0" Height="30"
Width="120" VerticalAlignment="Center" HorizontalAlignment="Center" Click="get_hostname_click">
            <Button.Effect>
                <DropShadowEffect Direction="-25" Color="#FFB6C2CB" ShadowDepth="4" BlurRadius="10"/>
            </Button.Effect>
            <Button.Resources>
                <Style TargetType="Border">
                    <Setter Property="CornerRadius" Value="8"/>
                </Style>
            </Button.Resources>
        </Button>
        <Button Name="set_hostname" Content="Set hostname" Grid.Row="1" Grid.Column="1" Height="30"
Width="120" VerticalAlignment="Center" HorizontalAlignment="Center" Click="set_hostname_click">
            <Button.Effect>
                <DropShadowEffect Direction="-25" Color="#FFB6C2CB" ShadowDepth="4" BlurRadius="10"/>
            </Button.Effect>
            <Button.Resources>
                <Style TargetType="Border">
                    <Setter Property="CornerRadius" Value="8"/>
```

```
            </Style>
         </Button.Resources>
      </Button>

      <Grid Grid.Row="2" Grid.Column="0">
         <Grid.RowDefinitions>
            <RowDefinition Height="1*"/>
            <RowDefinition Height="4*"/>
         </Grid.RowDefinitions>
         <Label Content="Loaded JSON object: " Grid.Row="0" HorizontalAlignment="Center"
VerticalAlignment="Bottom" FontWeight="Bold"/>
         <Label x:Name="get_hostname_content" Content="" Grid.Row="1" Height="140" Width="300"
VerticalAlignment="Top" HorizontalContentAlignment="Center" VerticalContentAlignment="Center"
FontWeight="Bold" Background="#ccebff" BorderThickness="0">
            <Label.Effect>
              <DropShadowEffect Direction="-25" Color="#FFB6C2CB" ShadowDepth="4" BlurRadius="10"/>
            </Label.Effect>
            <Label.Resources>
               <Style TargetType="Border">
                  <Setter Property="CornerRadius" Value="8"/>
               </Style>
            </Label.Resources>
         </Label>
      </Grid>

      <Grid Grid.Row="2" Grid.Column="1">
         <Grid.RowDefinitions>
            <RowDefinition Height="1*"/>
            <RowDefinition Height="4*"/>
         </Grid.RowDefinitions>
         <Label Content="Define hostname: " Grid.Row="0" HorizontalAlignment="Center"
VerticalAlignment="Bottom" FontWeight="Bold"/>
         <TextBox Name="hostname" Grid.Row="1" Height="20" Width ="200" VerticalAlignment="top"
HorizontalAlignment="Center">
            <TextBox.Effect>
              <DropShadowEffect Direction="-25" Color="#FFB6C2CB" ShadowDepth="4" BlurRadius="10"/>
            </TextBox.Effect>
         </TextBox>
         <Label Name="set_hostname_notify" Content="" Grid.Row="1" Width="120" Height="30"
VerticalAlignment="Center" HorizontalAlignment="Center" HorizontalContentAlignment="Center"
VerticalContentAlignment="Center" FontWeight="Bold" FontSize="11">
            <Label.Resources>
               <Style TargetType="Border">
                  <Setter Property="CornerRadius" Value="10"/>
               </Style>
            </Label.Resources>
         </Label>
      </Grid>
   </Grid>
```

Používateľské rozhranie obsahuje dvojicu tlačidiel. Každé z tlačidiel volá po stlačení obslužnú funkciu, ktorá vykoná požadované akcie. Nasledujúci výpis zobrazuje zdrojový kód pre funkciu na čítanie názvu zariadenia:

```
   //získanie názvu spravovaného zariadenia
   string name = cmb_hostname.Text;
   //ak používateľ zvolil spravované zariadenie, vykoná sa pokus o nadviazanie RESTCONF komunikácie
   if (name.Length != 0) {
      //získanie IP adresy cieľového spravovaného zariadenia - zo slovníka na základe názvu
      string ip_add = managed_devices[name];
      //inicializácia klienta a priradenie URI identifikátora
      var client = new RestClient("https://" + ip_add + "/restconf/data/native/hostname");
      client.RemoteCertificateValidationCallback = (sender, certificate, chain, sslPolicyErrors) =>
true;
      client.Timeout = -1;

      //definovanie typu požiadavky na vyčítanie dát = GET
      var request = new RestRequest(Method.GET);
      //tvorba hlavičiek definujúcich formát odosielania a príjmu dát = YANG dáta v JSON formáte
      request.AddHeader("Content-Type", "application/yang-data+json");
```

```
      request.AddHeader("Accept", "application/yang-data+json");
      //vytvorenie autentifikačnej hlavičky, ktorá vzniká Base64 algoritmom na základe mena: cisco
a hesla: cisco123!
      request.AddHeader("Authorization", "Basic Y2lzY286Y2lzY28xMjMh");

      //odoslanie HTTPS požiadavky môže skončiť chybou
      try {
         //odoslanie požiadavky
         IRestResponse response = client.Execute(request);
         //ak bola úspešne prijatá odpoveď = StatusCode 2xx
         if (response.IsSuccessful) {
            //zmena pozadia na modrú farbu
            get_hostname_content.Background = (SolidColorBrush)new
BrushConverter().ConvertFromString("#4EBCFF");
            //vypísanie názvu zariadenia v JSON formáte - do aplikácie
            get_hostname_content.Content = response.Content;
         }
         else {
            //zmena pozadia na červenú farbu - indikácia chyby
            get_hostname_content.Background = (SolidColorBrush)new
BrushConverter().ConvertFromString("#f99b9b");
            //výpis chybovej hlášky
            get_hostname_content.Content = "Connection error";
         }
      }
      //v prípade chyby dôjde k vypísaniu chybovej hlášky
      catch {
         //zmena pozadia na červenú farbu - indikácia chyby
         get_hostname_content.Background = (SolidColorBrush)new
BrushConverter().ConvertFromString("#f99b9b");
         //výpis chybovej hlášky
         get_hostname_content.Content = "Connection error";
      }
   }
   //ak nebolo zvolené spravované zariadenie, tak sa do aplikácie vypíše chybová hláška
   else {
      //vyplnenie obsahu značky chybovou hláškou
      get_hostname_content.Background = (SolidColorBrush)new
BrushConverter().ConvertFromString("#f99b9b");
      get_hostname_content.Content = "Choose a device from the list";
   }
```

Ako je možné vidieť, prvým krokom je vytvorenie objektu `RestClient`, ktorý dostane ako argument cieľový zdroj. Následne je potrebné vypnúť kontrolu certifikátu. Ďalším krokom je vytvorenie typu požiadavky ako **`GET`** a nastavenie príslušných hlavičiek. Nakoniec stačí uvedené objekty prepojiť a požiadavku odoslať. Na základe úspešnosti prijatia odpovede je možné používateľovi vypísať informácie o názve zariadenia. Ak došlo k chybe, dôjde k výpisu chybovej hlášky.

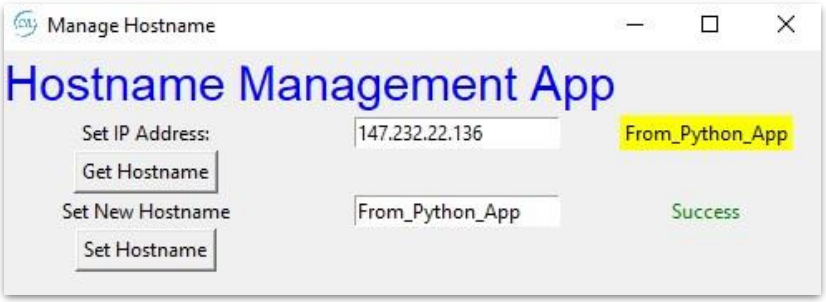
Nastavenie názvu zariadenia je založené na rovnakom princípe ako jeho čítanie. Rozdiel je, že sa použije metóda **`PUT`** a okrem štandardných hlavičiek sa vytvorí aj obsah správy, ktorý bude obsahovať JSON objekt opisujúci YANG model konfigurácie názvu zariadenia, čo znázorňuje nasledujúci výpis:

```
   //funkcia na zobrazenie "In progress.." správy - identifikuje vykonávanie programu
   in_Progress(set_hostname_notify);

   //získanie názvu spravovaného zariadenia
   string d_name = cmb_hostname.Text;

   //overenie, či bola zadaná hodnota pre nový názov zariadenia a zároveň, či bolo zvolené cieľové
spravované zariadenie
   if (!string.IsNullOrWhiteSpace(hostname.Text) && !string.IsNullOrEmpty(d_name)) {
      //získanie IP adresy cieľového zariadenia
      string ip_add = managed_devices[d_name];
      //vyčítanie používateľom zadanej hodnoty nového názvu zariadenia
      string name = hostname.Text;
      //vyčistenie formulára pre zadanie názvu zariadenia
      hostname.Clear();
```

```
      //vytvorenie objektu pre uloženie JSON objektu
      dynamic obj = new JObject();
      //priradenie načítaného názvu zariadenia a vytvorenie potrebného JSON objektu
      obj["Cisco-IOS-XE-native:hostname"] = name;
      //serializácia JSON objektu do stringu - na odoslanie
      var jsonString = Newtonsoft.Json.JsonConvert.SerializeObject(obj);

      //inicializácia klienta a priradenie URI
      var client = new RestClient("https://" + ip_add + "/restconf/data/native/hostname");
      client.RemoteCertificateValidationCallback = (sender, certificate, chain, sslPolicyErrors) =>
true;
      client.Timeout = -1;

      //vytvorenie PUT požiadavky s príslušnými hlavičkami a telom správy
      var request = new RestRequest(Method.PUT);
      request.AddHeader("Content-Type", "application/yang-data+json");
      request.AddHeader("Accept", "application/yang-data+json");
      request.AddHeader("Authorization", "Basic Y2lzY286Y2lzY28xMjMh");
      request.AddParameter("application/yang-data+json", jsonString, ParameterType.RequestBody);

      //odoslanie požiadavky
      IRestResponse response = client.Execute(request);
      //overenie úspešnosti zmeny názvu zariadenia - úspešná návratová hodnota je 2xx
      if (response.IsSuccessful) {
         //pri úspešnej zmene
            //zmena pozadia na zelenú farbu
         set_hostname_notify.Background = (SolidColorBrush)new
BrushConverter().ConvertFromString("#89ffaa");
            //vypísanie názvu zariadenia v JSON formáte - do aplikácie
         set_hostname_notify.Content = "Success";
      }
      else {
         //ak došlo k chybe, napr. používateľ zadal nesprávny reťazec - napr. obsahujúci medzeru
            //zmena pozadia na červenú farbu
         set_hostname_notify.Background = (SolidColorBrush)new
BrushConverter().ConvertFromString("#f99b9b");
            //vypísanie chybovej hlášky
         set_hostname_notify.Content = "Failed";
      }
   }
   else {
      //ak nebolo zvolené cieľové zariadenie, alebo nebol vo formulári zadaný žiaden reťazec
         //zmena pozadia na červenú farbu
      set_hostname_notify.Background = (SolidColorBrush)new
BrushConverter().ConvertFromString("#f99b9b");
         //vypísanie chybovej hlášky
      set_hostname_notify.Content = "Failed";
   }
```

Úloha 2: Python – Načítanie a zmena názvu zariadenia

Cieľom tejto úlohy je ukázať, ako je možné využitím programovacieho jazyka Python a knižnice *Tkinter* vytvoriť používateľské rozhranie a obslužné funkcie pre načítanie a zmenu názvu zariadenia modelovo riadeným prístupom. Obrázok 9.2 znázorňuje ukážku používateľského rozhrania:



Obrázok 9.2 Používateľské rozhranie vytvorenej aplikácie

Zdrojový kód pre vytvorenie používateľského rozhrania je uvedený v nasledujúcom výpise:

```
def T8_Telnet_Capture():
    packets = sniff(filter="port 80", count=int(T8_number_of_http_msg_value.get()))
    auth_data_array = []
    for packet in range(0, int(T8_number_of_http_msg_value.get())):
        print(packets[packet].summary())
        index = 3
        if packets[packet].haslayer(HTTPRequest):
            method = packets[packet][HTTPRequest].Method.decode()
            if (packets[packet].haslayer(Raw)) and (method == "POST") and
(packets[packet][Raw].load.decode().find("username")!= -1):
                auth_data = packets[packet][Raw].load.decode().split("&")
                auth_json = {
                    "username": auth_data[0],
                    "password": auth_data[1]
                }
                auth_data_array.append(auth_json)
from tkinter import *
from PIL import ImageTk, Image

import json
import requests
        requests.packages.urllib3.disable_warnings()

if __name__ == '__main__':
    root = Tk()
    root.title("Manage Hostname")
    root.iconphoto(True, ImageTk.PhotoImage(Image.open("cnllogo.png")))
    root.geometry("700x300")

    ip_address_value = StringVar(root, "")
    hostname_value = StringVar(root, "")

    Label(root, text="Hostname Management App", font=("Helvetica", 22), fg="blue",
).grid(columnspan=2, row=0)
    Label(root, text="Set IP Address:").grid(column=0, row=1)
    Entry(root, textvariable=ip_address_value).grid(column=1, row=1)
    Button(root, text="Get Hostname", command=get_hostname).grid(column=0, row=2)

    Label(root, text="Set New Hostname").grid(column=0, row=3)
    Entry(root, textvariable=hostname_value).grid(column=1, row=3)
    Button(root, text="Set Hostname", command=set_hostname).grid(column=0, row=4)

    root.mainloop()
```

Používateľské rozhranie obsahuje dvojicu tlačidiel – jedno pre čítanie a druhé pre zmenu konfigurácie názvu zariadenia. Nasledujúci výpis obsahuje zdrojový kód obslužných funkcií:

```
def get_hostname():
    url = "https://"+ip_address_value.get()+"/restconf/data/native/hostname"
    payload = {}
    headers = {
        'Accept': 'application/yang-data+json',
        'Content-Type': 'application/yang-data+json',
        'Authorization': 'Basic Y2lzY286Y2lzY28xMjMh'
    }
    #sekcia, ktorá pošle GET žiadosť a prevedie prijaté dáta do dict(JSON object) štruktúry
    new_get = requests.request("GET", url, headers=headers, data=payload, verify=False)
    data = new_get.json()
    if new_get.status_code == 200:
        Label(root, text=data["Cisco-IOS-XE-native:hostname"], bg="yellow").grid(column=2, row=1)

def set_hostname():
    url = "https://"+ip_address_value.get()+"/restconf/data/native/hostname"
    payload = {"Cisco-IOS-XE-native:hostname": hostname_value.get()}
    headers = {
        'Accept': 'application/yang-data+json',
        'Content-Type': 'application/yang-data+json',
        'Authorization': 'Basic Y2lzY286Y2lzY28xMjMh'
    }
```

```
#sekcia, ktorá pošle GET žiadosť a prevedie prijaté dáta do dict(JSON object) štruktúry
new_put = requests.request("PUT", url, headers=headers, data=json.dumps(payload), verify=False)
if new_put.status_code == 204:
   Label(root, text="Success", fg="green").grid(column=2, row=3)
else:
   Label(root, text="Failed", fg="red").grid(column=2, row=3)
```

Práca s REST API využitím jazyka Python je veľmi jednoduchá. Stačí vytvoriť `url` adresu, hlavičky, zvoliť metódu (**GET**/**PUT**) prípadne `payload` a odoslať požiadavku využitím knižnice *Requests*. Na základe úspešnosti odpovede je potom možné notifikovať používateľa o úspechu/neúspechu vykonania akcie.

Úloha 3: Java – Načítanie a zmena názvu zariadenia

Cieľom tejto úlohy je ukázať, ako je možné využitím programovacieho jazyka Java a knižnice *Swing* vytvoriť používateľské rozhranie a obslužné funkcie pre čítanie a zmenu názvu zariadenia modelovo riadeným prístupom. Obrázok 9.3 znázorňuje ukážku možného používateľského rozhrania:

Obrázok 9.3 Používateľské rozhranie vytvorenej aplikácie

Nasledujúci výpis znázorňuje tvorbu používateľského rozhrania:

```
JFrame window = new JFrame("Network Controller");
window.setSize(400,300);
Image icon =
Toolkit.getDefaultToolkit().getImage("D:\\Programming\\Network_Java_App\\src\\cnllogo.png");
window.setIconImage(icon);

JLabel title = new JLabel("Network Controller APP");
title.setBounds(130,10,200,30);
window.add(title);

JLabel set_ip_add = new JLabel("Set IP address:");
set_ip_add.setBounds(10,40,100,30);
window.add(set_ip_add);

JTextField ip_add_input = new JTextField();
ip_add_input.setBounds(110,40,100,25);
window.add(ip_add_input);

JLabel result_label = new JLabel();
result_label.setBounds(160,80,100,30);
window.add(result_label);

JLabel result = new JLabel();
result.setBounds(210,80,100,30);
window.add(result);

JButton button = new JButton("Get Hostname");
button.setBounds(10,80,130,30);
button.setBackground(Color.blue);
button.setForeground(Color.WHITE);
```

Po vložení jednotlivých grafických elementov do používateľského rozhrania je možné vytvoriť `listener`, ktorý po detekcii udalosti (stlačenie tlačidla), začne vykonávanie obslužnej funkcionality. Nasledujúci výpis zobrazuje kód pre uskutočnenie `GET` požiadavky na vyčítanie názvu zariadenia:

```
button.addActionListener(new ActionListener() {
   @Override
   public void actionPerformed(ActionEvent e) {
      String response_string = "";
      try {
         // vytvorenie trust manažéra neoverujúceho certifikáty
         TrustManager[] trustAllCerts = new TrustManager[] {new X509TrustManager() {
            public java.security.cert.X509Certificate[] getAcceptedIssuers() {return null;}
            public void checkClientTrusted(X509Certificate[] certs, String authType) {}
            public void checkServerTrusted(X509Certificate[] certs, String authType) {}
         }};
         // inštalácia dôverujúceho trust manažéra
         SSLContext sc = SSLContext.getInstance("SSL");
         sc.init(null, trustAllCerts, new java.security.SecureRandom());
         HttpsURLConnection.setDefaultSSLSocketFactory(sc.getSocketFactory());
         // vytvorenie dôverujúcich overovateľov
         HostnameVerifier allHostsValid = new HostnameVerifier() {public boolean verify(String
hostname, SSLSession session) {return true;}};
         // inštalácia dôverujúcich overovateľov
         HttpsURLConnection.setDefaultHostnameVerifier(allHostsValid);

         URL url = new URL("https://"+ip_add_input.getText()+"/restconf/data/native/hostname");
         HttpsURLConnection con = (HttpsURLConnection) url.openConnection();
         con.setRequestProperty("Content-Type", "application/yang-data+json");
         con.setRequestProperty("Accept", "application/yang-data+json");
         con.setRequestProperty("Authorization", "Basic Y2lzY286Y2lzY28xMjMh");

         if (con.getResponseCode() == HttpURLConnection.HTTP_OK) {
            BufferedReader in = new BufferedReader(new InputStreamReader(con.getInputStream()));
            String inputLine;
            StringBuffer response = new StringBuffer();

            while ((inputLine = in.readLine()) != null) {
               response.append(inputLine);
            }
            in.close();
            response_string = response.toString();
         }
      } catch (IOException ioException) {
         ioException.printStackTrace();
      } catch (NoSuchAlgorithmException noSuchAlgorithmException) {
         noSuchAlgorithmException.printStackTrace();
      } catch (KeyManagementException keyManagementException) {
         keyManagementException.printStackTrace();
      }
      Gson gson = new Gson();
      HostnameClass hostname = gson.fromJson(response_string, HostnameClass.class);

      result_label.setText("Result:");
      if (response_string != "") {
         result_label.setText("Result:");
         result.setText(hostname.hostname);
         result.setForeground(Color.BLUE);
      } else {
         result.setText("ERROR");
         result.setForeground(Color.RED);
      }
   }
});
window.add(button);
```

Prvá časť zdrojového kódu rieši problém s certifikátmi. Následne sa vytvárajú objekty pre tvorbu HTTPS požiadavky, ku ktorým patrí tvorba hlavičiek a požadovanej metódy. Ďalej je potrebné vytvoriť objekt pre príjem dátového toku, ktorý je potrebné uložiť pre neskoršie použitie. Uvedené dáta sú uložené v podobe textového reťazca. Pre ich lepšie spracovanie je reťazec prevedený do podoby JSON objektu využitím knižnice *GSON*. Po úspešnom spracovaní je používateľom zobrazený názov zariadenia. V opačnom prípade je vypísaná chybová hláška. Pre prácu s JSON dátami je potrebné vytvoriť triedu, ktorá bude obsahovať požadované atribúty. Nasledujúci výpis obsahuje kód triedy pre tvorbu požadovanej štruktúry:

```
import com.google.gson.annotations.SerializedName;

public class HostnameClass {
   @SerializedName("Cisco-IOS-XE-native:hostname")
   public String hostname;
}
```

Obslužná funkcia pre `PUT` požiadavku na zmenu názvu zariadenia vyzerá podobne a preto v nasledujúcom výpise je uvedený len fragment kódu, ktorý sa líši (je len v `PUT` požiadavkách):

```
URL url = new URL("https://"+ip_add_input.getText()+"/restconf/data/native/hostname");
HttpsURLConnection con = (HttpsURLConnection) url.openConnection(Proxy.NO_PROXY);
con.setRequestMethod("PUT");
con.setDoOutput(true);
con.setRequestProperty("Content-Type", "application/yang-data+json");
con.setRequestProperty("Accept", "application/yang-data+json");
con.setRequestProperty("Authorization", "Basic Y2lzY286Y2lzY28xMjMh");

String payload = "{\"Cisco-IOS-XE-native:hostname\": \""+new_hostname_input.getText()+"\"}";
byte[] out = payload.getBytes(StandardCharsets.UTF_8);

OutputStream stream = con.getOutputStream();
stream.write(out);
statusCode=con.getResponseCode();

con.disconnect();
```

Jediným rozdielom je zmena metódy a vytvorenie payload-u, ktorý je následne odoslaný cez tok zápisu. Na základe kódu návratovej hodnoty je následne notifikovaný používateľ o úspešnosti/neúspešnosti zmeny názvu zariadenia.

Posledný krok pre spustenie aplikácie využíva nasledujúce tri príkazy a umožní nastaviť parametre pre zobrazenie a uzavretie okna aplikácie, čo je možné vidieť v nasledujúcom výpise:

```
Window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
window.setLayout(null);
window.setVisible(true);
```

Úloha 4: Kotlin – Načítanie a zmena názvu zariadenia

Cieľom tejto úlohy je ukázať, ako je možné využitím programovacieho jazyka Kotlin vytvoriť používateľské rozhranie a obslužné funkcie pre čítanie a zmenu názvu zariadenia modelovo riadeným prístupom. Obrázok 9.4 znázorňuje ukážku možného používateľského rozhrania:

Obrázok 9.4 Používateľské rozhranie vytvorenej aplikácie

Používateľské rozhranie je možné v rámci vývoja mobilných aplikácií vytvárať využitím palety objektov a pracovného plátna jednoduchým ťahaním myši. Po vytvorení rozhrania vývojové prostredie vygeneruje XML súbor, ktorý obsahuje opis a prepojenie jednotlivých elementov. V našom prípade sa bude pracovať s troma elementmi a to TextBox (vkladanie textu), TextBox (s podčiarknutým názvom = umožňuje používateľský vstup) a posledným požadovaným elementom je tlačidlo, ku ktorému bude priradená obslužná funkcia vykonávajúca požadovanú akciu.

K jednotlivým elementom používateľského rozhrania je možné pristupovať cez ich **id**. Po vytvorení rozhrania a obslužných funkcií je možné vytvorenú aplikáciu po jej spustení okamžite nainštalovať do mobilného Android zariadenia a testovať jej funkcionalitu.

Vzhľadom k tomu, že je potrebné komunikovať využitím protokolu HTTPS, tak je potrebné pracovať s certifikátmi. Keďže smerovač, s ktorým sa pracuje v tejto kapitole používa self-signed certifikát, tak prvou funkciou, ktorú je potrebné implementovať bude funkcia na umožnenie dôverovať takýmto self-signed certifikátom. Pred implementáciou tejto funkcie ale najskôr ukážeme, štruktúru funkcie `Main`, ktorá ma za úlohu vykresliť používateľské rozhranie a aplikovať na elementy tlačidiel listnenery, ktoré po stlačení tlačidla zavolajú príslušnú obslužnú funkciu.

Nasledujúci výpis zobrazuje zdrojový kód funkcie `Main`:

```
class MainActivity : AppCompatActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContentView(R.layout.activity_main)
      findViewById<Button>(R.id.read_hostname_button).setOnClickListener {
         getHostname()
      }
      findViewById<Button>(R.id.set_hostname_button).setOnClickListener {
         setHostname()
      }
   }
}
```

Ďalšou implementovanou funkciou je funkcia pre riešenie problémov s certifikátmi, ktorej zdrojový kód je znázornený na nasledujúcom výpise:

```
private fun solveCertificationProblem() {
    val policy = StrictMode.ThreadPolicy.Builder().permitAll().build()
    StrictMode.setThreadPolicy(policy)
    // vytvorenie trust manažéra neoverujúceho certifikáty
    val trustAllCerts = arrayOf<TrustManager>(object : X509TrustManager {
        override fun getAcceptedIssuers(): Array<X509Certificate>? = null
        override fun checkClientTrusted(chain: Array<X509Certificate>, authType: String) = Unit
        override fun checkServerTrusted(chain: Array<X509Certificate>, authType: String) = Unit })
    val sc = SSLContext.getInstance("SSL")
    sc.init(null, trustAllCerts, java.security.SecureRandom())
    HttpsURLConnection.setDefaultSSLSocketFactory(sc.socketFactory)
    // vytvorenie dôverujúcich overovateľov
    val allHostsValid = HostnameVerifier { _, _ -> true }
    // inštalácia dôverujúcich overovateľov
    HttpsURLConnection.setDefaultHostnameVerifier(allHostsValid)
}
```

Hlavným cieľom uvedenej funkcie je nastavenie dôvery pre všetky možné certifikáty. Pre prax to nie je vhodným riešením, no pre testovacie účely v rámci tejto kapitoly je tento spôsob vhodný. V praxi by stačilo, ak by spravované zariadenia mali certifikáty podpísané známou certifikačnou autoritou alebo by boli dôveryhodné certifikáty importované do aplikácie.

Nasledujúci výpis obsahuje zdrojový kód pre obslužnú funkciu načítania názvu zariadenia:

```
private fun getHostname() {
    solveCertificationProblem()
    val url =
URL("https://"+findViewById<TextView>(R.id.input_ip_address).text+"/restconf/data/native/hostname")
    val conn = url.openConnection()
    conn.setRequestProperty("Content-Type", "application/yang-data+json")
    conn.setRequestProperty("Accept", "application/yang-data+json")
    conn.setRequestProperty("Authorization", "Basic Y2lzY286Y2lzY28xMjMh")
    var sb = StringBuilder()
    BufferedReader(InputStreamReader(conn.getInputStream())).use { inp ->
        var line: String?
        while (inp.readLine().also { line = it } != null) {
            println(line)
            sb.append(line)
        }
    }
    val netConfig = JSONObject(sb.toString())
    findViewById<TextView>(R.id.result_hostname).text = netConfig.getString("Cisco-IOS-XE-
native:hostname").toString()
    findViewById<TextView>(R.id.result_hostname).setBackgroundResource(R.color.hostname_output)
}
```

Proces práce s HTTPS protokolom je rovnaký ako v iných programovacích jazykoch. Stačí vytvoriť `url` (identifikuje cieľový zdroj) a vytvoriť objekt pre nadviazanie spojenia, v ktorom sa definujú požadované parametre v podobe hlavičiek. Následne sa vytvorí objekt pre `StringBuilder()`, ktorý bude postupne vytvárať reťazec na základe dátového toku prijatého ako odpoveď na HTTPS GET požiadavku. Posledným krokom je vytvorenie JSON objektu (potrebné len pre lepšiu prácu s prijatými dátami), získanie pre nás zaujímavých dát a ich výpis do používateľského rozhrania.

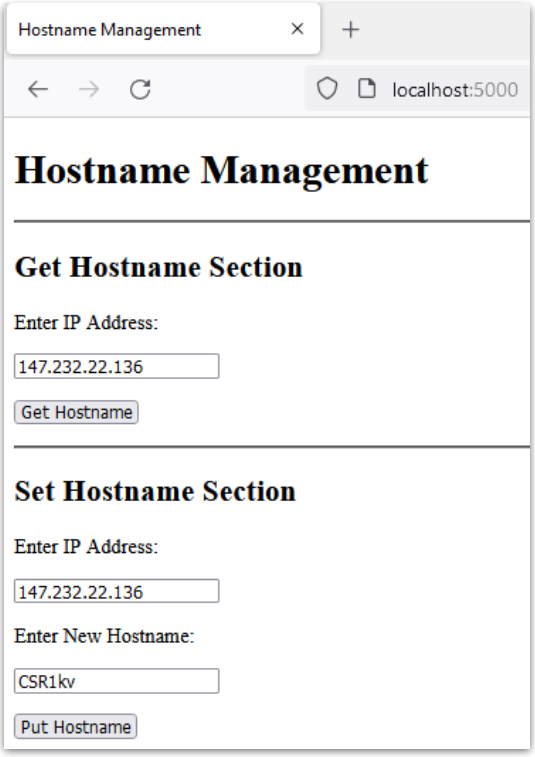
Poslednou funkciou implementovanou vo vytváranej mobilnej aplikácii je funkcia na zmenu názvu zariadenia. Princíp tejto funkcie je veľmi podobný, ako v predchádzajúcom prípade, jediným rozdielom bude to, že sa musí definovať HTTPS metóda – v tomto prípade metóda `PUT` (štandardne je `GET`). Ďalším krokom je vytvorenie obsahu správy, ktorá bude odosielaná. Obsah sa vytvorí ako JSON objekt, ktorý sa prevedie do podoby reťazca (serializácia) a odošle sa na cieľové zariadenie využitím dátového toku pre zápis. Aj PUT požiadavka má návratovú hodnotu (StatusCode), na základe ktorého je používateľ notifikovaný o úspechu/neúspechu zmeny nastavenia názvu zariadenia.

Nasledujúci výpis zobrazuje kód pre zmenu názvu zariadenia:

```
private fun setHostname() {
   solveCertificationProblem()
   // vytvorenie JSON objektu
   val jsonObject = JSONObject()
   jsonObject.put("Cisco-IOS-XE-native:hostname",
findViewById<TextView>(R.id.user_input_hostname).text)
   val jsonObjectString = jsonObject.toString()
   val url =
URL("https://"+findViewById<TextView>(R.id.input_ip_address).text+"/restconf/data/native/hostname")
   val conn = url.openConnection() as HttpsURLConnection
   conn.setRequestProperty("Content-Type", "application/yang-data+json")
   conn.setRequestProperty("Accept", "application/yang-data+json")
   conn.setRequestProperty("Authorization", "Basic Y2lzY286Y2lzY28xMjMh")
   conn.requestMethod = "PUT"
   val outputStreamWriter = OutputStreamWriter(conn.outputStream)
   outputStreamWriter.write(jsonObjectString)
   outputStreamWriter.flush()
   // Check if the connection is successful
   val responseCode = conn.responseCode
   if (responseCode.toInt() == 204) {
      findViewById<TextView>(R.id.change_status).text = "Success"
      findViewById<TextView>(R.id.change_status).setBackgroundResource(R.color.success)
   } else {
      findViewById<TextView>(R.id.change_status).text = "Failed"
      findViewById<TextView>(R.id.change_status).setBackgroundResource(R.color.failed)
   }
}
```

Úloha 5: Python-Flask – Načítanie a zmena názvu zariadenia

Cieľom tejto úlohy je ukázať ako je možné, využitím programovacieho jazyka Python, vytvoriť webovú aplikáciu, používateľské rozhranie a obslužné funkcie pre čítanie a zmenu názvu zariadenia modelovo riadeným prístupom. Obrázok 9.5 znázorňuje ukážku používateľského rozhrania:



Obrázok 9.5 Používateľské rozhranie vytvorenej aplikácie

Webová aplikácia bola vytvorená využitím rámca `Flask`. Základný kód pre zobrazenie formulárov a spustenie servera je znázornený v nasledujúcom výpise:

```
from flask import Flask, redirect, url_for, request, render_template

app = Flask(__name__)

@app.route('/')
def get_hostname():
   return render_template("get_form.html")

if __name__ == '__main__':
   app.run(debug = True)
```

Vzhľad používateľského rozhrania je vytvorený v html súbore, ktorého obsah je uvedený v nasledujúcom výpise:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Hostname Management</title>
</head>
<body>
   <h1>Hostname Management</h1>
   <hr>
   <h2>Get Hostname Section</h2>
   <form action="http://localhost:5000/show_hostname" method = "post">
      <p>Enter IP Address:</p>
      <p><input type = "text" name = "ip_add"/></p>
      <p><input type = "submit" value="Get Hostname"/></p>
   </form>
   <hr>
   <h2>Set Hostname Section</h2>
   <form action="http://localhost:5000/show_hostname_status" method = "post">
      <p>Enter IP Address:</p>
      <p><input type = "text" name = "p_ip_add"/></p>
      <p>Enter New Hostname:</p>
      <p><input type = "text" name = "p_new_hostname"/></p>
      <p><input type = "submit" value="Put Hostname"/></p>
   </form>
</body>
</html>
```

Každé tlačidlo, uvedené vo formulároch, spôsobí odoslanie zapísaných údajov na príslušnú URL adresu, ku ktorej je v hlavnom kóde priradená obslužná funkcia. Nasledujúce výpisy zobrazujú html zápisy pre zobrazenie výsledkov načítania názvu zariadenia a status o úspešnosti zmeny:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Show hostname</title>
</head>
<body>
   <h1>Hostname: {{name}}</h1>
   <a href="/">Back</a>
</body>
</html>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Status</title>
</head>
<body>
   <h1>Hostname Management</h1>
   {% if status %}
      <h2>Change successful</h2>
   {% else %}
      <h2>Change Failed</h2>
   {% endif %}
   <a href="/">Back</a>
</body>
</html>
```

V html kóde je cez {{}} možné odovzdať parametre z Python kódu a prekresliť obsah zobrazených stránok. Nasledujúci zdrojový kód zobrazuje funkciu pre načítanie názvu zariadenia:

```
@app.route('/show_hostname', methods=["POST"])
def show_hostname():
    ip_address = request.form["ip_add"]
    url = "https://" + ip_address + "/restconf/data/native/hostname"
    payload = {}
    headers = {
        'Accept': 'application/yang-data+json',
        'Content-Type': 'application/yang-data+json',
        'Authorization': 'Basic Y2lzY286Y2lzY28xMjMh'
    }
    #sekcia, ktorá pošle GET žiadosť a prevedie prijaté dáta do dict(JSON object) štruktúry
    new_get = requests.request("GET", url, headers=headers, data=payload, verify=False)
    data = new_get.json()
    if new_get.status_code == 200:
        return render_template("show_hostname.html", name=data["Cisco-IOS-XE-native:hostname"])
    else:
        return render_template("show_hostname.html", name="Get Failed!!!")
```

Nasledujúci zdrojový kód zobrazuje funkciu pre zmenu názvu zariadenia:

```
@app.route('/show_hostname_status', methods=["POST"])
def change_hostname_status():
    ip_address = request.form["p_ip_add"]
    new_hostname = request.form["p_new_hostname"]

    url = "https://" + ip_address + "/restconf/data/native/hostname"
    payload = {"Cisco-IOS-XE-native:hostname": new_hostname}
    headers = {
        'Accept': 'application/yang-data+json',
        'Content-Type': 'application/yang-data+json',
        'Authorization': 'Basic Y2lzY286Y2lzY28xMjMh'
    }
    # sekcia, ktorá pošle GET žiadosť a prevedie prijaté dáta do dict(JSON object) štruktúry
    new_put = requests.request("PUT", url, headers=headers, data=json.dumps(payload), verify=False)
    if new_put.status_code == 204:
        return render_template("change_hostname_status.html", status=True)
    else:
        return render_template("change_hostname_status.html", status=False)
```

Úloha 6: JavaScript (NodeJS) – Načítanie a zmena názvu zariadenia

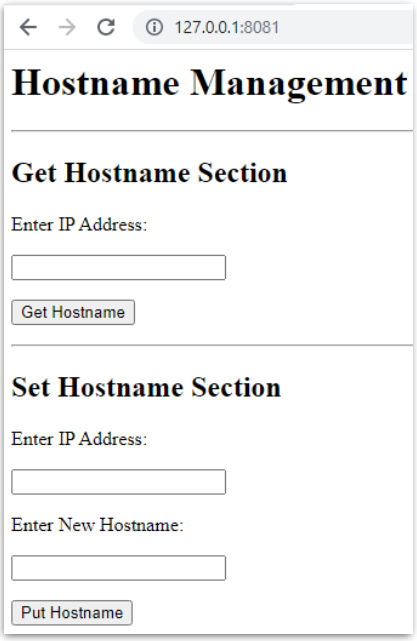
Webové stránky často používajú pre vykonávanie funkcionalít JavaScript. Nasledujúca ukážka znázorní vytvorenie Webového servera a riadenie konfigurácie sieťových zariadení protokolom RESTCONF. Vhodnou knižnicou pre zjednodušenie vývoja je práca s *ExpressJS*. Nasledujúci výpis znázorňuje všetky potrebné závislosti, základné smerovanie a spustenie servera na porte 8081:

```
var express = require('express');
var request = require('request').defaults({ rejectUnauthorized: false });
var app = express();

app.use(express.static('src'));
app.set('view engine', 'pug')
app.set('views', "./views");


app.get('/', function (req, res) {
   res.sendFile(__dirname + "/" + "index.html" );
})

var server = app.listen(8081, function () {
   console.log("Server Started at: ", server.address().port)
})
```

Pre zobrazenie webových stránok sa používa smerovač, ktorý na základe zvolenej URL spustí príslušnú funkcionalitu, prípadne zobrazí webovú stránku, ktorá môže byť napísaná v HTML jazyku, alebo v podobe „pug“ šablóny. Spustenie servera je možné využitím CMD príkazom **`node file.js`**. Nasledujúci zdrojový kód zobrazuje vytvorenie obsahu webovej stránky *index.html*:

```
<html>
<body>
   <h1>Hostname Management</h1>
   <hr>
   <h2>Get Hostname Section</h2>
   <form action="http://localhost:8081/show_hostname" method = "get">
      <p>Enter IP Address:</p>
      <p><input type = "text" name = "ip_add"/></p>
      <p><input type = "submit" value="Get Hostname"/></p>
   </form>
   <hr>
   <h2>Set Hostname Section</h2>
   <form action="http://localhost:8081/show_hostname_status" method = "get">
      <p>Enter IP Address:</p>
      <p><input type = "text" name = "p_ip_add"/></p>
      <p>Enter New Hostname:</p>
      <p><input type = "text" name = "p_new_hostname"/></p>
      <p><input type = "submit" value="Put Hostname"/></p>
   </form>
</body>
</html>
```

Obrázok 9.6 znázorňuje vzhľad používateľského rozhrania:



Obrázok 9.6 Používateľské rozhranie vytvorenej aplikácie

Po stlačení tlačidiel, sa získané dáta z formulára odošlú na vo formulári definovanú cieľovú adresu, pre ktorú stačí vytvoriť obslužnú funkcionalitu. Nasledujúce výpisy znázorňujú vytvorenie šablón pre zobrazenie výstupu spracovania požiadaviek:

```
html
   head
      title Show Hostname
   body
      h1 Hostname Get Result
      hr
      h2 Hostname: #{name}
      a(href='/') Back
```

```
html
   head
      title Show Hostname Status
   body
      h1 Hostname Put Result
      hr
      h2 Status: #{state}
      a(href='/') Back
```

Šablóny generujú HTML kód stránky a umožňujú dynamicky vložiť hodnoty na miesta, kam to je potrebné. V uvedených prípadoch sa názov premennej, s ktorou sa bude pracovať, nachádza v zložených zátvorkách. Obe šablóny obsahujú na konci odkaz na domovskú stránku.

Nasledujúci výpis zobrazuje zdrojový kód pre získanie dát (HTTPS `GET`) z formulára, po stlačení tlačidla, za účelom získania aktuálneho názvu spravovaného zariadenia a jeho zobrazenie:

```
app.get('/show_hostname', function (req, res) {
   var ip = req.query.ip_add;
   var options = {
      'method': 'GET',
      'url': 'https://'+ip+'/restconf/data/native/hostname',
      'headers': {
         'Content-Type': 'application/yang-data+json',
         'Accept': 'application/yang-data+json',
         'Authorization': 'Basic Y2lzY286Y2lzY28xMjMh'
      }
   };

   request(options, function (error, response) {
      if (error) throw new Error(error);
      var jobj = JSON.parse(response.body);
      res.render('get_result', { name: jobj["Cisco-IOS-XE-native:hostname"]});
   });
})
```

Nasledujúci výpis zobrazuje zdrojový kód pre zmenu (HTTPS `PUT`) názvu zariadenia na základe údajov zadaných do formulára a zobrazenie úspešnosti zmeny:

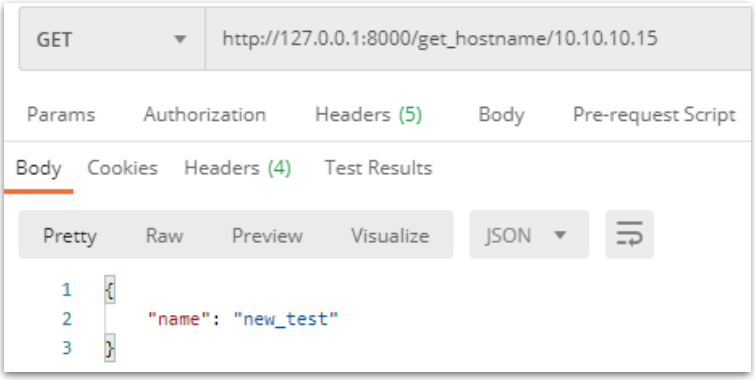
```
app.get('/show_hostname_status', function (req, res) {
   var ip_add = req.query.p_ip_add;
   var new_hostname = req.query.p_new_hostname;
   var options = {
      'method': 'PUT',
      'url': 'https://'+ ip_add + '/restconf/data/native/hostname',
      'headers': {
         'Content-Type': 'application/yang-data+json',
         'Accept': 'application/yang-data+json',
         'Authorization': 'Basic Y2lzY286Y2lzY28xMjMh'
      },
      body: JSON.stringify({"Cisco-IOS-XE-native:hostname": new_hostname})

   };
   request(options, function (error, response) {
      if (error) throw new Error(error);
         if (response.statusCode == 204) {
            res.render('put_result', { state: "Success"});
      } else {
         res.render('put_result', { state: "Failed"});
      }
   });
})
```

Úloha 7: Python (FastAPI) – Vytvorenie vlastných API

Pre sprístupnenie funkcionalít nášho riešenia externým klientom, v podobe, ktorú si definujeme, je možné využiť vytvorenie vlastných API. Implementácia je možná využitím rôznych programovacích jazykov. V rámci tejto úlohy bude ukázaná ich tvorba využitím jazyka Python a knižnice *FastAPI*. Skript spúšťajúci túto funkcionalitu je potrebné spustiť z terminálu príkazom: `uvicorn main:app –reload`. **Main** je názov skriptu, ktorý sa spúšťa a **app** predstavuje názov objektu reprezentujúci `fastAPI` inštanciu.

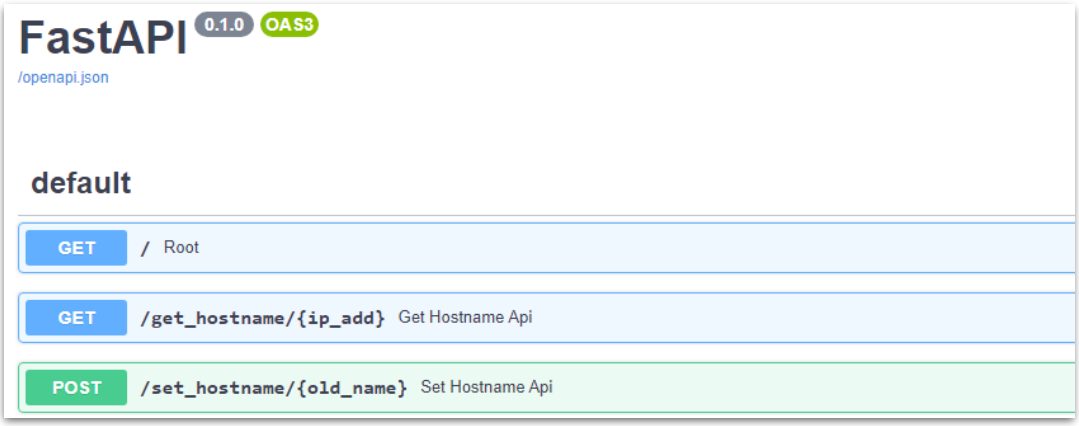
```python
from fastapi import FastAPI

#vytvorenie objektu pre prácu s API
app = FastAPI()

#základná stránka
@app.get("/")
def root():
   return {"About App": "API for managing network device"}
```

Testovanie funkčnosti je možné spustením webového prehliadača, prípadne nástroja Postman, do ktorého stačí zadať do URL adresu **127.0.0.1:8000**. Predchádzajúci výpis zobrazuje vzorový zdrojový kód, ktorý k uvedeným URL a metódam priradzuje obslužnú funkciu, ktorá vykonáva istú činnosť. Prvou činnosťou, ktorej sa budeme venovať je tvorba API, ktorá umožní používateľovi vyčítať názov sieťového zariadenia. Vzorové správanie by mohlo vyzerať nasledovne (obrázok 9.7):



Obrázok 9.7 Odoslanie GET požiadavky na vytvorené API

Ako je možné vidieť, používateľ zadá URL v tvare **127.0.0.1:8000/get_hostname/ip_address** a zvolí metódu GET. Ako výstup dostane názov zariadenia v podobe JSON štruktúry. JSON štruktúra odpovede, ako aj URL adresa a potrebné parametre, sú definované našimi API. Dokumentácia API (obrázok 9.8) sa generuje automaticky a je dostupná na adrese http://127.0.0.1:8000/docs:



Obrázok 9.8 Zobrazenie dokumentácie pre vytvorené API

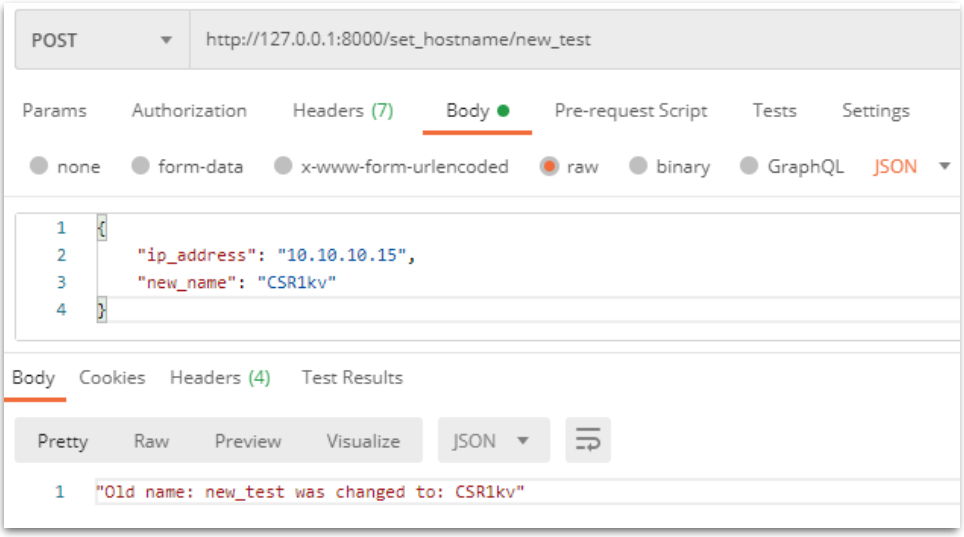
Zdrojový kód pre vytvorenie API na čítanie názvu zariadenia je zobrazený na nasledujúcom výpise:

```python
#API pre načítanie názvu zariadenia
@app.get("/get_hostname/{ip_add}", status_code=200)
def get_hostname_api(ip_add:str):
   hostname = get_hostname(ip_add)
   if hostname != 400:
      ResponseData = {
         "name": hostname
      }
      return ResponseData
   else: return "Error"
```

Ako je možné pozorovať, tak využitím samotnej URL je možné odlíšiť volania rôznych obslužných funkcií. Súčasťou URL je tiež časť zadania voliteľných parametrov, ktoré sa v kóde uvádzajú do zložených zátvoriek. Reťazec je následne možné použiť pri volaní funkcie ako názov premennej parametra. Uvedená funkcia zavolá pripravenú funkciu pre čítanie názvu zariadenia, ktorá vráti získaný názov alebo hodnotu 400 v prípade chyby. Takto, využitím API, nemusíme klientom zverejniť náš zdrojový kód ale vieme poskytnúť len výstup nášho produktu. Nasledujúci výpis znázorňuje zdrojový kód obslužnej funkcie get_hostname:

```
def get_hostname(ip_address):
    url = "https://" + ip_address + "/restconf/data/native/hostname"
    payload = {}
    headers = {
        'Accept': 'application/yang-data+json',
        'Content-Type': 'application/yang-data+json',
        'Authorization': 'Basic Y2lzY286Y2lzY28xMjMh'
    }
    try:
        #sekcia, ktorá pošle GET žiadosť a prevedie prijaté dáta do dict(JSON object) štruktúry
        new_get = requests.request("GET", url, headers=headers, data=payload, verify=False)
        data = new_get.json()
        return data["Cisco-IOS-XE-native:hostname"]
    except:
        return 400
```

Podobným spôsobom je možné pracovať s POST požiadavkami používateľa. V tomto prípade treba myslieť na to, že používateľ okrem samotnej požiadavky odošle na naše API aj telo. POST požiadavka sa často používa, ak chceme vykonať v systéme nejaké zmeny. V našom prípade bude telo zapísané v JSON podobe. Význam vytváraného API bude zmeniť názov zariadenia. Obrázok 9.9 zobrazuje možnú interakciu s vytvoreným API:



Obrázok 9.9 Odoslanie POST požiadavky na vytvorené API

Ako je možné vidieť, klient zadá požadovanú URL, zvolí metódu POST a vloží do tela správy JSON štruktúru podľa definovaného API. Po odoslaní požiadavky získa odpoveď, ktorá ho notifikuje o spracovaní jeho požiadaviek. Z pohľadu zdrojového kódu by implementácia API pre POST požiadavku mohla vyzerať nasledovne:

```
from pydantic import BaseModel

#model pre prijatie dát
class ConfigData(BaseModel):
    ip_address: str
    new_name: str
    optional: str | None = None
```

```python
#API pre zmenu názvu zariadenia
@app.post("/set_hostname/{old_name}", status_code=201)
def set_hostname_api(old_name:str, cf: ConfigData):
    cf_dict = cf.dict()
    put_hostname = set_hostname(cf_dict["ip_address"], cf_dict["new_name"])
    if put_hostname != 400:
        return "Old name: " + old_name + " was changed to: " + cf_dict["new_name"]
    else:
        return "Error"
```

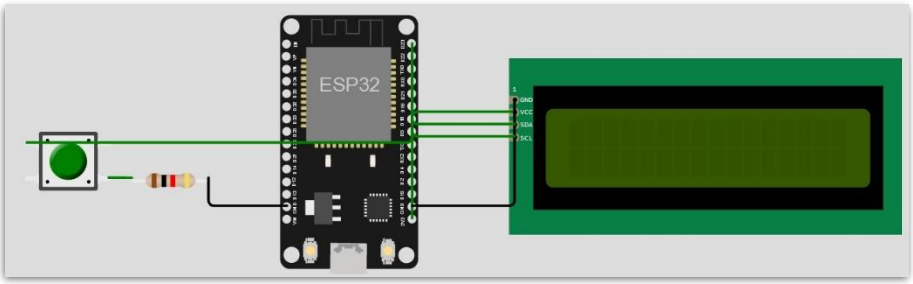
V tomto prípade je dôležité najskôr vytvoriť triedu, ktorá bude reprezentovať očakávané dáta od používateľa. Následne je tieto dáta možné vo volanej obslužnej funkcii získať, spracovať ich a pokračovať volaním funkcie pre zmenu názvu zariadenia. Na základe výstupu tejto funkcie je ďalej možné používateľovi vypísať správu o stave vykonania jeho požiadavky. Naša obslužná funkcia teda získa IP adresu a nový názov zariadenia z prijatého tela požiadavky a vykoná zmenu názvu zariadenia. Nasledujúci výpis zobrazuje kód obslužnej funkcie:

```python
def set_hostname(ip_address, new_hostname):
    url = "https://" + ip_address + "/restconf/data/native/hostname"
    payload = {"Cisco-IOS-XE-native:hostname": new_hostname}
    headers = {
        'Accept': 'application/yang-data+json',
        'Content-Type': 'application/yang-data+json',
        'Authorization': 'Basic Y2lzY286Y2lzY28xMjMh'
    }
    try:
        #sekcia, ktorá pošle GET žiadosť a prevedie prijaté dáta do dict(JSON object) štruktúry
        new_put = requests.request("PUT", url, headers=headers, data=json.dumps(payload),
verify=False)
        if new_put.status_code == 204:
            return 204
        else:
            return 400
    except:
        return 400
```

Rovnakým spôsobom by sa dala implementovať akákoľvek funkcionalita pre riadenie a monitorovanie sieťových zariadení.

Úloha 8: MikroPython (ESP32) – Načítanie a zmena názvu zariadenia

V rámci tejto úlohy bude ukázané ako pracovať so zariadením ESP32, ako ho pripojiť k sieti, odosielať z neho HTTP (GET, PUT) požiadavky, ako ich vizualizovať a ako pracovať s tlačidlom, po stlačení ktorého dôjde k nejakej akcii. Najskôr je potrebné zapojiť danú schému, čo je možné vidieť na obrázku 9.10:

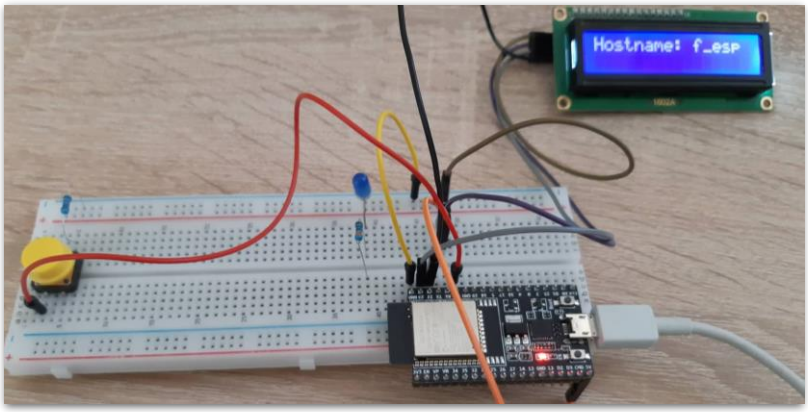


Obrázok 9.10 Schéma pre zapojenie IoT prvkov

Na zariadení ESP32 budú použité nasledujúce piny:

[GND = GND, 21 = Button, 22 = scl, 23 = sda, 5v = VCC]

Reálne zapojenie je znázornené na obrázku 9.11:



Obrázok 9.11 Reálne zapojenie IoT prvkov

Nasledujúca časť dokumentu uvádza ukážku zdrojových kódov pre interakciu s jednotlivými HW komponentmi. Začnime najjednoduchšou a to vytvorením objektu pre prepínač, čo znázorňuje nasledujúci výpis:

```
# Vytvorenie tlačidla - pripojené na pine 21
button = Pin(21, Pin.IN, Pin.PULL_UP)
```

Ďalšou dôležitou časťou je práca s LCD display-om, čo je znázornené v nasledujúcom výpise:

```
from machine import SoftI2C
from lcd_api import LcdApi
from i2c_lcd import I2cLcd
# definovanie adresy, veľkosti LCD a vytvorenie objektov pre zobrazenie pripojené na pinoch 22 a 23
I2C_ADDR = 0x27
totalRows = 2
totalColumns = 16
i2c = SoftI2C(scl=Pin(22), sda=Pin(23), freq=10000)
lcd = I2cLcd(i2c, I2C_ADDR, totalRows, totalColumns)
# funkcia pre vypísanie požadovaného textu na 5 sekúnd
def print_text_LCD(msg):
   lcd.putstr(msg)
   sleep(5)
   lcd.clear()
```

V prípade potreby komunikácie s externými entitami je potrebné vykonať pripojenie sa do WiFi siete, čo znázorňuje nasledujúci výpis:

```
def connect_to_wifi():
   import network
   # Definovanie WiFi parametrov - SSID a heslo
   WIFI_SSID = "IoT_ESP"
   WIFI_PASS = "ESP_pass"
   # vytvorenie objektu pre prácu s WiFi rozhraním pre pripojenie do siete (station interface)
   sta_if = network.WLAN(network.STA_IF)
   # ak nie je zariadenie pripojené
   if not sta_if.isconnected():
      print("connectiong to network...")
      # aktivácia rozhrania
      sta_if.active(True)
      # pripojenie do siete so zvoleným SSID a heslom
      sta_if.connect(WIFI_SSID, WIFI_PASS)
      # proces pripojenia môže trvať - počkanie na pripojenie
      while not sta_if.isconnected():
         pass
   # po pripojení sa vypíšu informácie ako priradená adresa, maska, GW
   print("Network Configuration: ", sta_if.ifconfig())
```

Štandardnou interakciou medzi zariadeniami je využitie protokolu HTTP. Nasledujúci výpis znázorňuje prácu s `GET` a `PUT` požiadavkami (na príklade čítania a zmeny názvu zariadenia):

```
import urequests
import json
# funkcia pre načítanie názvu zariadenia
def http_get(address):
   # vytvorenie URL a hlavičiek
   url = "https://" + address + "/restconf/data/native/hostname"
   headers = {
      'Accept': 'application/yang-data+json',
      'Content-Type': 'application/yang-data+json',
      'Authorization': 'Basic Y2lzY286Y2lzY28xMjMh'
   }
   payload = {}
   # vytvorenie a odoslanie GET požiadavky
   response = urequests.get(url, headers=headers, data=payload)
   # prerobenie prijatých dát do JSON a vrátenie názvu zariadenia
   data = response.json()
   response.close()
   return data["Cisco-IOS-XE-native:hostname"]

# funkcia pre zmenu názvu zariadenia
def http_put(address, new_hostname):
   # vytvorenie URL, hlavičiek a tela správy
   url = "https://" + address + "/restconf/data/native/hostname"
   headers = {
      'Accept': 'application/yang-data+json',
      'Content-Type': 'application/yang-data+json',
      'Authorization': 'Basic Y2lzY286Y2lzY28xMjMh'
   }
   payload = {"Cisco-IOS-XE-native:hostname": new_hostname}
   # vytvorenie a odoslanie PUT požiadavky
   response = urequests.put(url, headers=headers, data=json.dumps(payload).encode("utf8"))
   response.close()
```

Po príprave všetkých potrebných komponentov je možné napísať kód pre hlavnú slučku, ktorá sa bude vykonávať po spustení zariadenia. Nasledujúci výpis znázorňuje túto slučku:

```
while True:
   # pripojenie sa do siete
   connect_to_wifi()
   # načítanie názvu zariadenia
   hostname = http_get("192.168.2.5")
   # vypísanie názvu zariadenia na LCD display
   print_text_LCD("Hostname: " + hostname)
   # po stlačení tlačidla
   if not button.value():
      # notifikácia na display
      print_text_LCD("SEND_PUT")
      # zmena názvu zariadenia
      http_put("192.168.2.5", "f_esp")
```

Záver

Ako je možné vidieť, práca s REST API je principiálne rovnaká pre všetky programovacie jazyky. Pre čitateľov je postačujúce poznať princíp a následne vedia jednoducho, po krátkom štúdiu základov zápisu daného programovacieho jazyka, implementovať svoje riešenie. Podobne aj tvorba používateľských rozhraní aplikácií je založená na podobnom princípe.