

TECHNICAL UNIVERSITY OF KOSICE

Department of Computer and Informatics

DOCUMENTATION

Django with React

Contents

1. Project Overview.....	3
2. Introduction.....	4
3. Django Installation.....	5
i) Install Django	
ii) Starting a project in Django	
4. Files of Django.....	7
i) Urls.py	
ii) Views.py	
iii) Settings.py	
iv) Manage.py	
5. React Installation.....	15
i) Install npm and npx packages	
ii) Installing react	
iii) Other essential packages	
6. Connecting React with Django.....	18
7. Files of React.....	19
i) App.js	
ii) Components of React	
8. Output.....	21
9. Conclusion.....	22
10. References and Contact.....	25

I. Project Overview

This full-stack web application, crafted by Sarukesh Boominathan, seamlessly blends frontend and backend components to deliver a holistic user experience. Here's a detailed examination of each element:

Frontend Components:

1. Header Component:

- Presents the project title and an accompanying image.
- Emphasizes Sarukesh Boominathan's name as a focal point.

2.Navbar Component:

- Furnishes navigation links for effortless exploration within the application.
- Incorporates a direct link to an "About Me" section.
- Offers buttons for data retrieval and documentation download.

3. Input Form Component:

- Facilitates user submission of personal details: name, mobile number, and email address.
- User-centric design ensures ease of data input.

Backend Components:

1. URL Configuration:

- Specifies routing for various endpoints within the application.

2.Views:

- Hosts functions dedicated to request handling and response generation.
- Encompasses operations like rendering HTML templates, processing form data, and serving API endpoints.

3. Settings:

- Houses configuration parameters pertinent to the Django project.
- Manages settings for static files, internationalization, and other project-specific configurations.

This project overview encapsulates the core functionalities provided by Sarukesh Boominathan, accentuating both frontend and backend components, along with their respective roles and functionalities.

2. Introduction

Welcome to the **full-stack web application** crafted by Sarukesh Boominathan, which seamlessly integrates **React.js and Django frameworks**. This project serves as a comprehensive platform for users to interact with various features, leveraging frontend technologies for **user input collection** and backend technologies for **data storage and retrieval**.

In this application, React.js is employed on the frontend to provide an intuitive user interface for seamless interaction. Users can conveniently inputs, including **id, title, and due date**, via a user-friendly form. React.js ensures a smooth and responsive experience, enhancing user engagement and satisfaction.

On the backend, Django plays a pivotal role in handling **data storage** and management. Instead of employing a traditional database system, Django is configured to **store user data in a text file**. This approach offers simplicity and efficiency, eliminating the need for complex database configurations while ensuring data persistence and accessibility.

Additionally, the application features a convenient **documentation download** functionality. Users can effortlessly access project documentation by clicking on the designated button, simplifying the process of acquiring essential information about the application's features and functionalities.

With the combination of React.js and Django, this web application delivers a robust and user-centric solution, catering to the diverse needs of users while maintaining simplicity and efficiency in data management and documentation access.

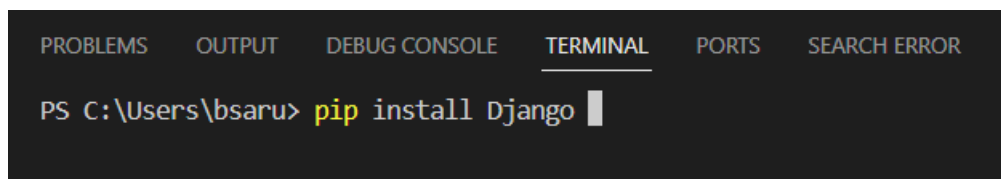
Starting a Project:

- Create a Folder for your new project
- Click the file path and type "cmd" to open Command prompt inside your folder
- I have used a code editor called "Visual Studio Code" for my project. So I am using a command called "code ." to open Visual Studio Code inside my folder.
- Now click on "Terminal" in navigation bar and click "New Terminal"
- Use the terminal to navigate to the directory where you want to create your Django project.
- Now follow the below mentioned steps to start you Full Stack project .

3. Django Installation

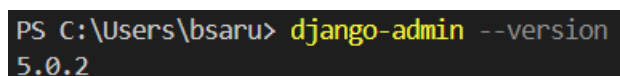
I. Install Django

Once you ensure that you are in the correct directory, you can install Django pip:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SEARCH ERROR
PS C:\Users\bsaru> pip install Django
```

You can verify that Django has been installed correctly by checking its version:



```
PS C:\Users\bsaru> django-admin --version
5.0.2
```

II. Starting a Project in Django

To start a project in Django, Use the “Django-admin” command. This step must be continued on the same directory.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR

PS C:\Users\bsaru> django-admin startproject project1
PS C:\Users\bsaru> 
```

Once it is installed you can navigate to your project folder using:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR

PS C:\Users\bsaru> cd project1
PS C:\Users\bsaru\project1> 
```

now I am inside my project folder. Now to start Django development server to see your project in action:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR

PS C:\Users\bsaru\project1> py manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
March 15, 2024 - 21:30:31
Django version 5.0.2, using settings 'project1.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.


```

4. Files of Django

Manage.py

manage.py is a command-line utility provided by Django to perform various administrative tasks within a Django project. Here's a concise explanation:

Management Commands:

- Executes management commands such as creating applications, running development servers, and applying database migrations.

Project Configuration:

- Sets up the Django project environment, including settings, database connections, and installed applications.

Development Server:

- Starts the development server to run the Django project locally during development.

Database Migrations:

- Manages database schema changes and data migrations using Django's built-in migration system.

Utility Functions:

- Provides utility functions for tasks like creating superusers, running tests, and managing static files.

In essence, `manage.py` serves as the entry point for interacting with a Django project's management commands and administrative tasks.

Urls.py

In Django, `urls.py` serves as a configuration file where you define URL patterns for your web application. It acts as a bridge between incoming URLs and the views that handle them. Each URL pattern is associated with a specific view function or class-based view, allowing you to map URLs to the appropriate functionality within your application. `urls.py` enables you to organize and manage your application's URL structure, providing a clear and structured approach to handling incoming requests.

```
#urls.py

from django.contrib import admin
from django.urls import path
from .views import index
from .views import store_name
from .views import get_data
from .views import download_data

urlpatterns = [
    path("", index),
    path('store-name/', store_name, name='store_name'),
    path('documentation/downloads', download_data, name='download_documentation'),
    path('get-data/', get_data, name='get_data'),
    path('admin/', admin.site.urls),
]
```


This `urls.py` file configures the URL patterns for a Django project:

1. It imports necessary modules (`admin` and `path`) from Django.
2. It maps URLs to corresponding view functions (`index`, `store_name`, `get_data`, `download_data`).
3. Each URL pattern is defined using the `path()` function.
4. The `name` parameter assigns unique names to URL patterns for easy referencing.
5. The `admin/` URL is reserved for Django's admin interface.

In essence, this file defines how incoming URLs are handled and which views should respond to them within the Django project.

Views.py

The `views.py` file contains Python functions that handle HTTP requests and generate HTTP responses in a Django project:

1. It imports necessary modules for handling requests and generating responses.
2. Each function represents a specific view or endpoint in the application.
3. The functions process incoming requests, perform necessary operations (such as data retrieval or manipulation), and return appropriate responses.
4. Views can render HTML templates, serve API endpoints, or perform other tasks based on the application's requirements.
5. View functions are linked to URL patterns defined in `urls.py`, determining which function should be invoked for a particular URL.

This `views.py` file contains Python functions used to handle HTTP requests and generate responses in a Django project:

```
from django.shortcuts import render
from django.http import JsonResponse
from django.http import HttpResponseRedirect
from django.http import FileResponse
from django.views.decorators.csrf import csrf_exempt
from rest_framework.decorators import api_view, authentication_classes, permission_classes
from rest_framework.authentication import TokenAuthentication
from rest_framework.permissions import IsAuthenticated
import json
import os
```

- "render": Renders HTML templates.
- "JsonResponse": Returns JSON-formatted responses.
- "HttpResponse": Returns basic HTTP responses.
- "FileResponse": Sends files as HTTP responses.
- "csrf_exempt": Exempts views from CSRF protection.
- "api_view", "authentication_classes", "permission_classes": Decorators for API views.
- "TokenAuthentication", "IsAuthenticated": Components for authentication and permissions.
- "json": Module for JSON serialization.
- "os": Module for interacting with the operating system.

```
def index(request):
    return render(request, 'index.html')
```

The `index` function renders the `index.html` template when a user visits the root URL of the application.

```
@api_view(['POST'])
@csrf_exempt
@authentication_classes([TokenAuthentication])
@permission_classes([IsAuthenticated])
def store_name(request):
    if request.method == 'POST':
        try:
            data = json.loads(request.body)
            name = data.get('name')
            mobile = data.get('mobile')
            email = data.get('email')
            if name:
                with open('name.txt', 'a') as f:
                    f.write( '\nName: ' + name + '\nMobile Number : ' + mobile + '\nEmail ID: ' + email)
                return JsonResponse({'success': True, 'message': 'Name stored successfully'})
            else:
                return JsonResponse({'success': False, 'message': 'No name provided'}, status=400)
        except Exception as e:
            print(e)
            return JsonResponse({'success': False, 'message': 'Failed to store name'}, status=500)
    else:
        return JsonResponse({'success': False, 'message': 'Method not allowed'}, status=405)
```

This “**store_name**” function is a view in the Django project, designed to handle POST requests to store user information. Here's a concise explanation of its functionality.

Decorator:

- `@api_view(['POST'])`: Marks the function as an API view that only accepts POST requests.

CSRF Exemption:

- `@csrf_exempt`: Exempts the view from CSRF (Cross-Site Request Forgery) protection.

Authentication and Permission Classes:

- `@authentication_classes([TokenAuthentication])`: Specifies token-based authentication for the view.

- `@permission_classes([IsAuthenticated])`: Requires users to be authenticated to access the view.

Request Handling:

- Checks if the request method is POST.
- Parses the JSON data from the request body to extract name, mobile, and email.
- Stores the user information in a text file named `name.txt`.
- Returns a JSON response indicating whether the operation was successful or not.

Error Handling:

- Handles exceptions that may occur during the process, such as JSON parsing errors or file writing errors.
- Returns appropriate error responses with status codes.

In summary, the “**store_name**” view function processes POST requests, extracts user data, stores it in a text file, and returns JSON responses to indicate the success or failure of the operation. It also includes measures for authentication, permission, and error handling.

```
def get_data(request):
    try:
        with open('name.txt', 'r') as file:
            data = file.read()
        return HttpResponse(data, content_type='text/plain')
    except FileNotFoundError:
        return HttpResponse("File not found", status=404)
    except Exception as e:
        return HttpResponse(str(e), status=500)
```

The `get_data` function in the Django project serves as a view to retrieve stored data from a text file named `name.txt`. Below is a simplified explanation of its functionality:

- File Reading:

- Attempts to open the `name.txt` file in read mode ("r") using a **with** statement.
- Reads the contents of the file into the **data** variable.

HTTP Response:

- Returns an HTTP response containing the data read from the file.
- The `content_type` parameter specifies the MIME type of the response, set to 'text/plain'.

Error Handling:

- If the `name.txt` file is not found, returns an HTTP response with status code 404 (File Not Found).
- Handles any other exceptions that may occur during file reading and returns an HTTP response with status code 500 (Internal Server Error), along with the error message.

In summary, the `get_data` function attempts to read the contents of the `name.txt` file and returns them as a plain text HTTP response. It includes error handling to manage situations where the file is not found or if any other error occurs during the file reading process.

```
def download_data(request):
    try:
        file_path = "doc.pdf"
        if os.path.exists(file_path):
            return FileResponse(open(file_path, 'rb'), as_attachment=True)
        else:
            return HttpResponse("File not found", status=404)
    except Exception as e:
        return HttpResponse(str(e), status=500)
```

The `download_data` function serves as a view in the Django project to allow users to download a PDF file named `doc.pdf`. Below is a simplified explanation of its functionality.

File Path:

- Defines the file path variable **file_path** pointing to the location of the PDF file.

File Existence Check:

- Checks if the file exists at the specified path using **os.path.exists()**.
- If the file exists:
 - Returns a **FileResponse** containing the PDF file in binary mode (**rb**), with the **as_attachment** parameter set to `True`` to prompt the user to download the file.
- If the file does not exist:
 - Returns an HTTP response with status code 404 (File Not Found) and a message indicating that the file was not found.
- Error Handling:
 - Handles any exceptions that may occur during the process, such as file access errors.
 - Returns an HTTP response with status code 500 (Internal Server Error) along with the error message if any exception occurs.

In summary, the **download_data** function checks for the existence of the PDF file **doc.pdf** and allows users to download it. It includes error handling to manage situations where the file is not found or if any other error occurs during the process.

5.React Installation

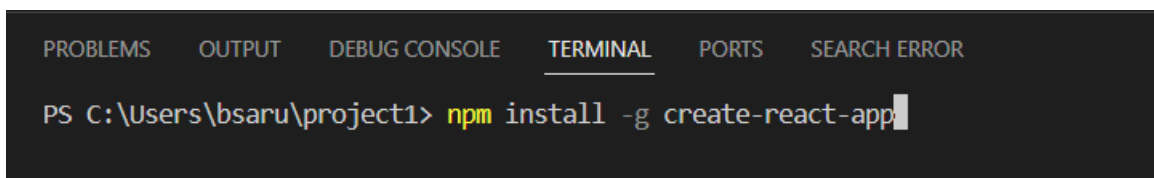
To install React, follow these steps inside the folder which we created for our Django project. React must be inside Django's root directory.

1. Node.js Installation:

- Ensure you have Node.js installed on your system. You can download and install Node.js from the official website: [Node.js Download](https://nodejs.org/).

2. Create React App:

- Open your terminal
- Run the following command to install the Create React App tool globally.

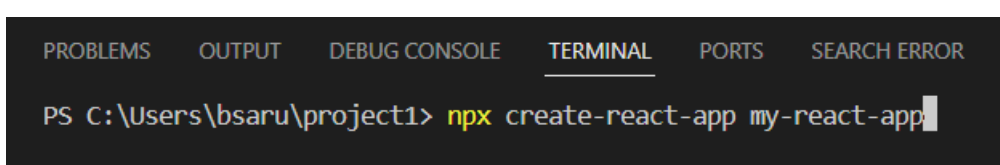


```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR  
PS C:\Users\bsaru\project1> npm install -g create-react-app
```

- This tool simplifies the process of creating and managing React applications.

3. Create a New React Project:

- Navigate to the directory where you want to create your React project.
- Run the following command to create a new React project:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR  
PS C:\Users\bsaru\project1> npx create-react-app my-react-app
```

- Replace my-react-app with the name of your project.

4. Navigate to Project Directory:

- Change into the project directory.

```
We suggest that you begin by typing:  
  
cd my-react-app  
npm start  
  
Happy hacking!  
PS C:\Users\bsaru\project1>  
PS C:\Users\bsaru\project1> cd my-react-app
```

5. Start Development Server:

- Once inside the project directory, start the development server by running:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR  
  
PS C:\Users\bsaru\project1\my-react-app> npm start
```

- This command will start the development server and open your default web browser to view the React application.

6. Verify Installation:

- Open your web browser and navigate to “<http://localhost:3000>” to see the React application running.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR  
  
Compiled successfully!  
  
You can now view my-react-app in the browser.  
  
Local:      http://localhost:3000  
On Your Network: http://192.168.56.1:3000  
  
Note that the development build is not optimized.  
To create a production build, use npm run build.  
  
webpack compiled successfully
```




Edit `src/App.js` and save to reload.

[Learn React](#)

That's it! You've successfully installed React and created a new React project. You can now begin developing your React application.

III. Additional Package

To make HTTP request we need to download “axios”

```
PS C:\Users\bsaru\project1\my-react-app> npm install axios
```

6.Connecting React with Django

-Open settings.py add the following code below.

```
from pathlib import Path
import os
```

-Add import os

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [
            os.path.join(BASE_DIR, 'frontend/build')
        ],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    ],
]
```

```
STATICFILES_DIRS = [
    os.path.join(BASE_DIR, 'frontend/build/static')
]
```

-By adding these code we will successfully connect React to Django. We are rendering the static files from react.

7.Files of React

1. index.html:

- Entry point HTML file where the React application is rendered.
- Contains a root `<div>` element with an `id` where the React app is mounted, usually `id="root"`.

2. index.js:

- JavaScript file that serves as the entry point for the React application.
- Renders the root component (`App`) into the DOM using ReactDOM.

3. App.js:

- Main component file where the root component ("App") is defined.
- Contains the structure of the entire application and renders other components.

4. App.css:

- CSS file for styling the App component.
- Contains styles specific to the App component and its children.

5. src/ directory:

- Directory where most of the application's source code resides.
- Contains JavaScript files for React components, CSS files for styling, and other assets.

6. Components:

- Individual component files (e.g., Navbar.js, Header.js, Footer.js).
- Each component typically consists of a JavaScript file defining the component's logic and a CSS file for styling.

```
import Header from "../Header";
import Content from "../Content";
import Footer from "../Footer";
import "../App.css";
import Navbar from "../Navbar";
import NameForm from "../Input";
function App() {

  return (
    <div className="App-header">
      <Navbar/>
      <Header />
      <Content />
      <NameForm />
      <Footer />
    </div>
  );
}

export default App;
```

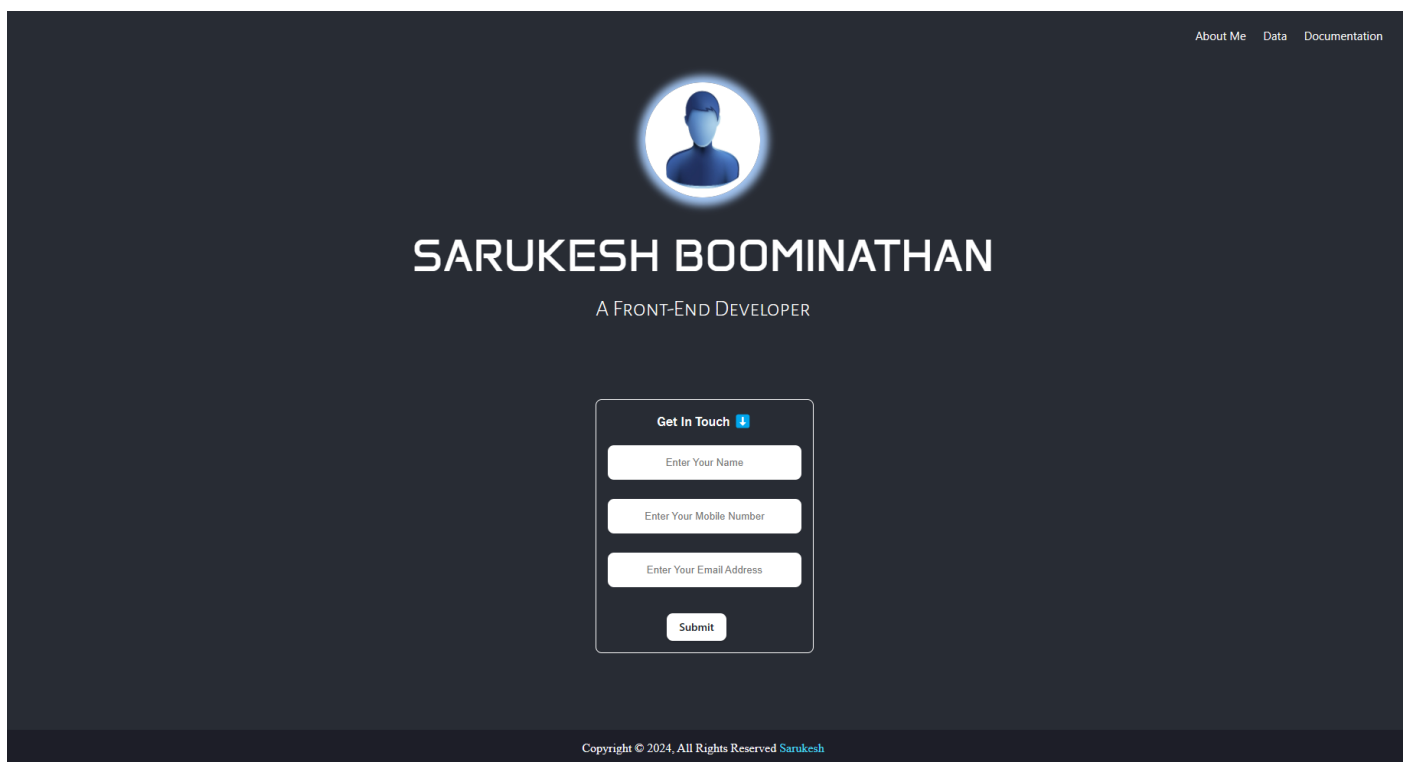
-This is **App.js** file. This file connects all the UI designs. Each element is made in separate files and connected to the **main.js**

```
import Header from "../Header";
import Content from "../Content";
import Footer from "../Footer";
import "../App.css";
import Navbar from "../Navbar";
import NameForm from "../Input";
```

-This code connects all the components. It imports the code from external file to the **App.js**.

8.Output:

The project is a full-stack web application developed using Django for the backend and React for the frontend. It serves as a platform where users can interact with the application by providing their id, title, and due date. The backend stores this information in a text file and provides endpoints for retrieving the stored data and downloading documentation.



This how the project looks after connecting with React framework.

9. Conclusion:

Key Components:

1. Backend (Django):

- Utilizes Django framework to handle HTTP requests, manage data, and serve API endpoints.
- Defines views for handling requests such as storing user information and providing data retrieval functionality.
- Uses Django REST Framework for building RESTful APIs and integrating authentication classes for securing endpoints.
- Stores user information in a text file named `name.txt`.
- Provides endpoints for storing user data, retrieving stored data, and downloading documentation.

2. Frontend (React):

- Developed using React library for building user interfaces.
- Consists of components such as Navbar, Header, Content, NameForm (input form), and Footer.
- Uses Axios library to make HTTP requests from the frontend to the backend.
- Allows users to input their name, mobile number, and email address through the NameForm component.
- Integrates with the backend to store user data and retrieve stored information.

Communication Between Frontend and Backend:

- Frontend communicates with the backend by making HTTP requests using Axios library.
- Backend processes incoming requests, performs necessary operations, and sends appropriate responses back to the frontend.
- This communication enables data exchange between the frontend and backend, allowing users to interact with the application seamlessly.

In conclusion, the project demonstrates the integration of Django and React to build a full-stack web application. It showcases the implementation of CRUD (Create, Read, Update, Delete) operations using Django's backend functionalities and React's dynamic user interface. By connecting the frontend and backend, the application enables users to input their information, store it securely, and retrieve it when needed. With its modular architecture and responsive design, the project serves as a robust foundation for further development and expansion of features.

References:

- **Programming** : [Programming in Python — 1st and 2nd semester of Programming \(tuke.sk\)](#)
- **React** : [Quick Start – React](#)
- **Django** : [The web framework for perfectionists with deadlines | Django \(djangoproject.com\)](#)
- **Chat GPT** : [ChatGPT \(openai.com\)](#)
- **Google** : [www.google.com](#)
- **Stack overflow** : <https://stackoverflow.com>
- **W3 schools** : <https://w3schools.com>

Contact

Name: SARUKESH BOOMINATHAN

Email: sarukesh.boominathan@student.tuke.sk

Website: www.sarukesh.com

Mobile: +421910240413

Address: Jedlikova 13, Kosice