one24

Title

Implementation of messaging services with Node.js.

Introduction

This paper is aimed to demonstrate the possible message-servicing systems which are there to be easily integrated with Node.js ecosystem without much heckle. It provides a comparative study and code implementation on these packages/modules depending upon the factors like being affordable, not having too many dependencies, lightweight etc.

Architecture

External dependencies/packages involved in the project:

1. **Express**

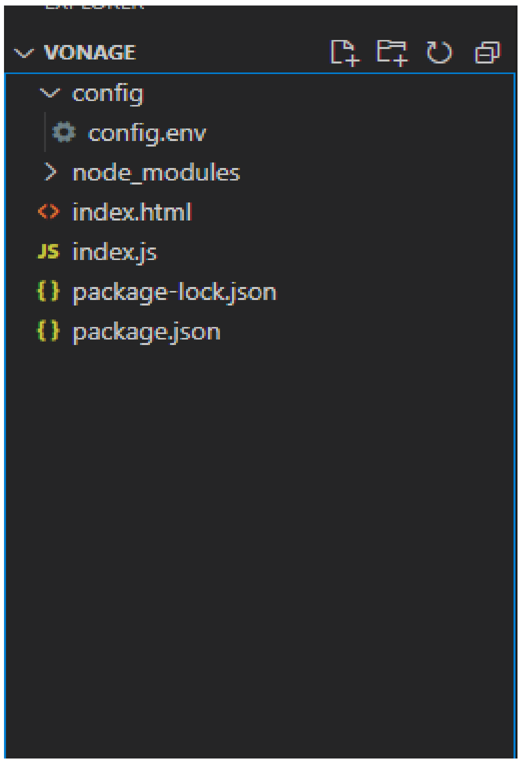
   Express is a Node.js web application framework that provides a robust set of features to develop web and mobile applications. It facilitates the rapid development of Node based Web applications. Following are some of the core features of Express framework −

- Allows to set up middlewares to respond to HTTP Requests.
- Defines a routing table which is used to perform different actions based on HTTP Method and URL.
- Allows to dynamically render HTML Pages based on passing arguments to templates.

2. **Dotenv**

   Dotenv is a zero-dependency module that loads environment variables from a .env file to make it available on the process.env variable

Directory Structure:



```
VONAGE
  config
    config.env
  node_modules
  index.html
  index.js
  package-lock.json
  package.json
```

Code implementation

For any of the packages which access the messaging service, will consider always a couple of data fields, i.e.,

1. The number on which text message to be sent and,
2. The message which is to be sent to the specified number

is being collected from the client side and sent to the respective router of the server via ajax/ any other techniques, so that it will be available on the server side as an object likewise,

- req.body.number
- req.body.message

where as, name and message are the `name` attributes of the `input fields` which accepts the number and message.

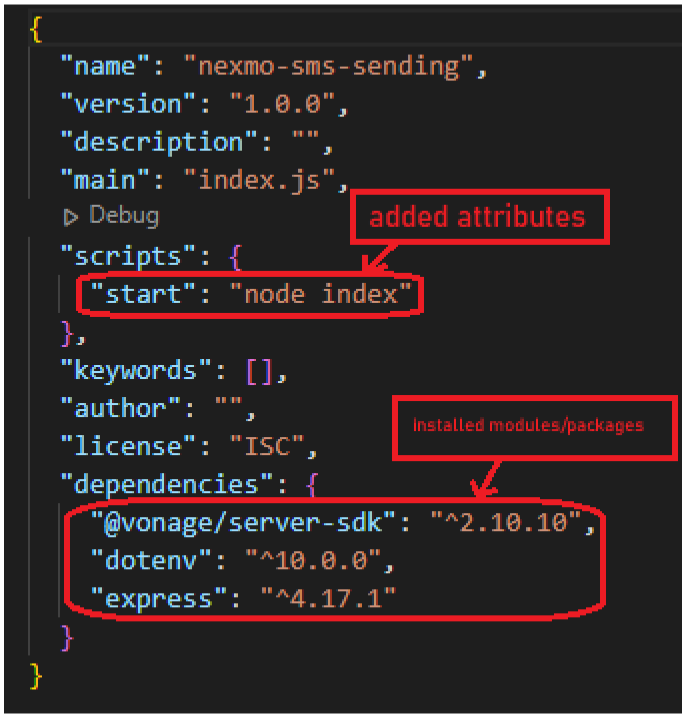
Let's start with the implementation of a service called **Nexmo.**

To start the project, in the project directory, execute the following command:
(Make sure that the latest version of Node.js and npm are installed on the system)

npm init -y

This will create a package.json file (using -y flag does not prompt for any details)
Thereafter, to install the external dependencies required for this project, execute the command:

npm install --save @vonage/server-sdk, dotenv

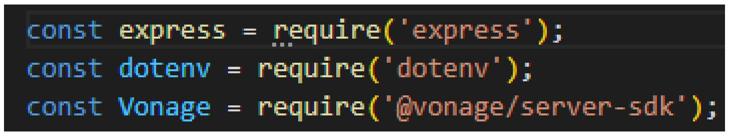


```
{
  "name": "nexmo-sms-sending",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "node index"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "@vonage/server-sdk": "^2.10.10",
    "dotenv": "^10.0.0",
    "express": "^4.17.1"
  }
}
```

Also, I have edited the ***start*** script, so that node will serve ***index.js*** on start.

Let's implement the code structure and learn how nexmo message service communicates with the number and sends a text SMS.

Now, coming onto the implementation part, first we will create the index.js server file in the same directory where these package.json is kept and start with importing the modules or packages required whichever be installed using the npm packages *i.e.,* express, dotenv, @vonage/server-sdk.



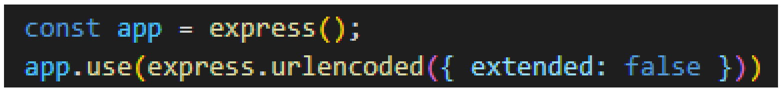
```
const express = require('express');
const dotenv = require('dotenv');
const Vonage = require('@vonage/server-sdk');
```

Next,
we call an express() function so that we use the feature of express.
As express() function creates a new express application for you and it internally calls a createApplication() function from the lib/express.js file which is the default export.

For example,

const app = express();

where the constant variable `app` returned from this function is one that we use in our application code.



```
const app = express();
app.use(express.urlencoded({ extended: false }))
```

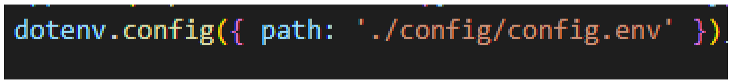
Whereas,
use() function or the **app.use()** function is used to mount the specified middleware function(s) at the path which is being specified. It is mostly used to set up middleware for our application

The **express.urlencoded()** function is a built-in middleware function in Express. It parses incoming requests with urlencoded payloads and is based on body-parser.
syntactically,

express.urlencoded([option])

Parameter:The \`option\` parameter contains various properties like extended, inflate, limit, verify, etc.
Return Value: It returns an object.

Once done with express setup, Let's move to the environment file,
To deal with the environment file we have used a package of npm i.e., ***dotenv*** which is a zero-dependency module that loads environment variables from a .env file into ***process.env.***



```
dotenv.config({ path: './config/config.env' })
```

the above code snippet says to configure the environment file for the project.
the ***config()*** function is a built-in function in the dotenv package.
we also use it as,

require("dotenv").config();

if the environment file is not kept in side any of the folder, if it is, then we have to provide the \`path\` to the ***config()*** function as an object.

On the above code, an object is passed followed by a key attribute \`path\` which shows the path of the environment file, that is given by (in our case) a value attribute i.e., './config/config.env'.

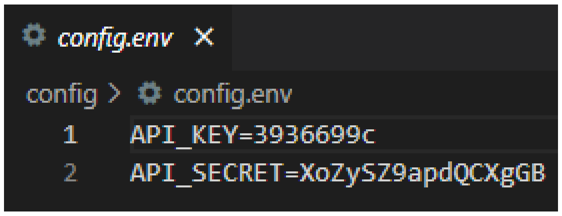
To execute the variable assigned in the environment file, we use

process.env.*Variable_name*

First Implementation,

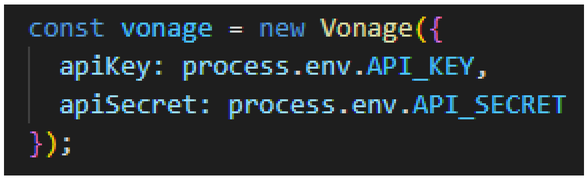
Using Vonage API :

Let it be understood by configuring the Vonage API.



```
API_KEY=3936699c
API_SECRET=XoZySZ9apdQCXgGB
```

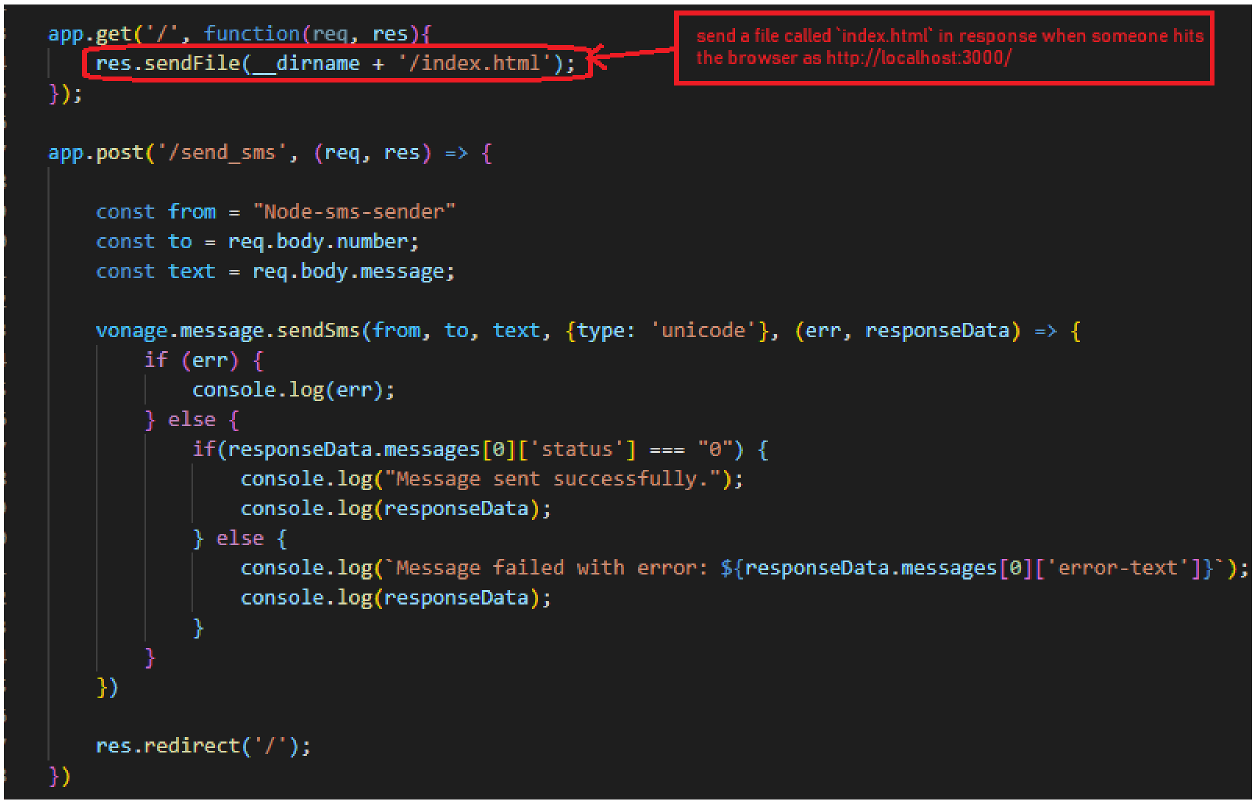
The above picture belongs to the ***.env file(environment file)*** which is having two variables ***API_KEY*** and ***API_SECRET*** with their respective values as well. Also, vonage requires you to register on the official site, and get an Api key and Api secret for your test project. To process these values and to configure the Vonage Api we, do is,



```
const vonage = new Vonage({
  apiKey: process.env.API_KEY,
  apiSecret: process.env.API_SECRET
});
```

We have created a Vonage object and provide a JSON object followed by apiKey and apiSecret for the authentication purpose which is now provided by their environment variable respectively by using process.env.API_KEY and process.env.API_SECRET.
By creating a Vonage object, we store it to a constant variable `vonage`. Now we use a `vonage` variable wherever it needs to be used, instead of writing the whole Vonage Object Code.

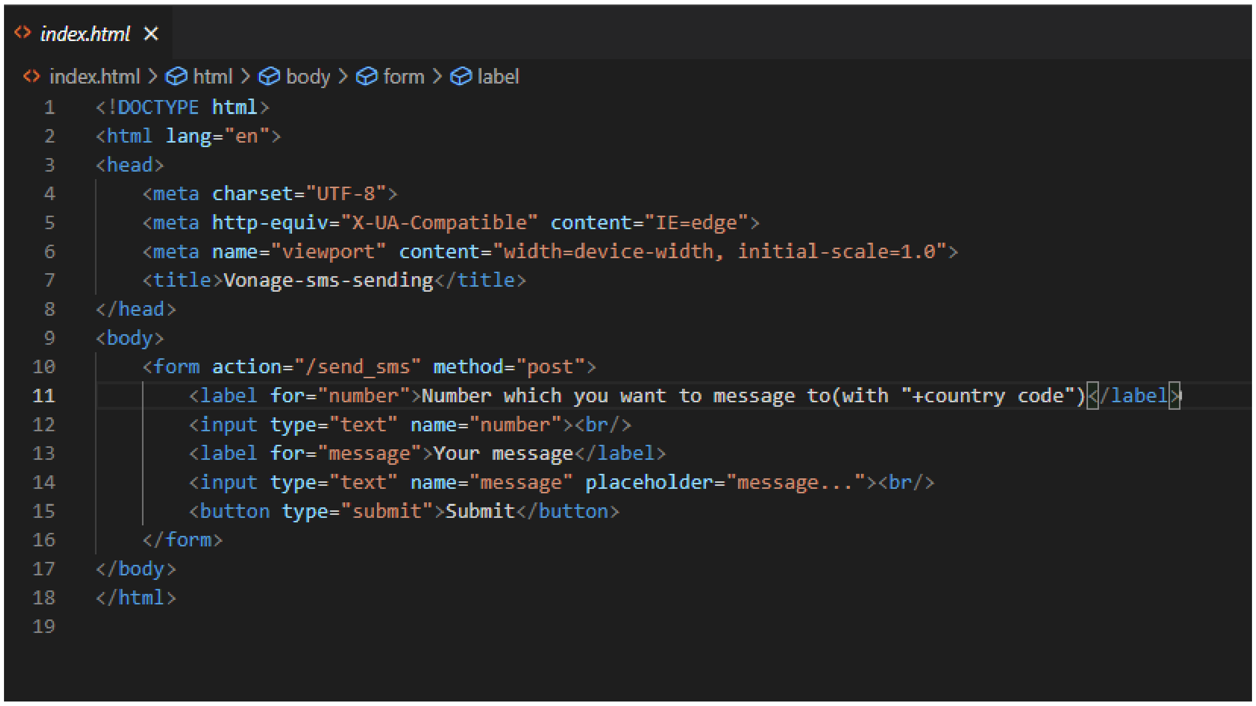
Next,
Further move to the Basic Routing feature of Express.



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

app.post('/send_sms', (req, res) => {

    const from = "Node-sms-sender"
    const to = req.body.number;
    const text = req.body.message;

    vonage.message.sendSms(from, to, text, {type: 'unicode'}, (err, responseData) => {
        if (err) {
            console.log(err);
        } else {
            if(responseData.messages[0]['status'] === "0") {
                console.log("Message sent successfully.");
                console.log(responseData);
            } else {
                console.log(`Message failed with error: ${responseData.messages[0]['error-text']}`);
                console.log(responseData);
            }
        }
    })

    res.redirect('/');
})
```

send a file called `index.html` in response when someone hits the browser as http://localhost:3000/

Picture-1: sends a file in response when user hits to the browser as (http://localhost:3000/)



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Vonage-sms-sending</title>
</head>
<body>
    <form action="/send_sms" method="post">
        <label for="number">Number which you want to message to(with "+country code")</label>
        <input type="text" name="number"><br/>
        <label for="message">Your message</label>
        <input type="text" name="message" placeholder="message..."><br/>
        <button type="submit">Submit</button>
    </form>
</body>
</html>
```

Picture-2: code structure of index.html file

In **Picture-1**, we see that ***app.get('/')*** serves file in response to the browser using ***sendFile()*** function.

res.sendFile():

The **res.sendFile()** function basically transfers the file at the given path and it sets the Content-Type response HTTP header field based on the filename extension.
And,
the index.html file looks like in **Picture-2**, in which a "form" takes input as "number" and "message" using the method "POST" and sends the data to ***"/send_sms"*** when the user clicks to "Submit" button that present in the form.
The ***"/send_sms"*** route will handle the data and send the SMS to the destination phone number

the number and the message from the body(browser) takes by the name attributes of input tag which is further takes by the server(index.js) file as ***req.body.***
So, suppose,

req.body.number
req.body.message

Come to ***Picture-3,*** at the ***post('/send_sms')*** function.

this function/method takes input from the form:
Here,
We have used the free tier version of vonage API, and we can use any title such as 'Node-sms-sender' as the sender;s name (may not be visible in free testing)

const from = "Node-sms-sender"

and respectively in `` `to` `` and `` `text` `` followed by the form in the browser.

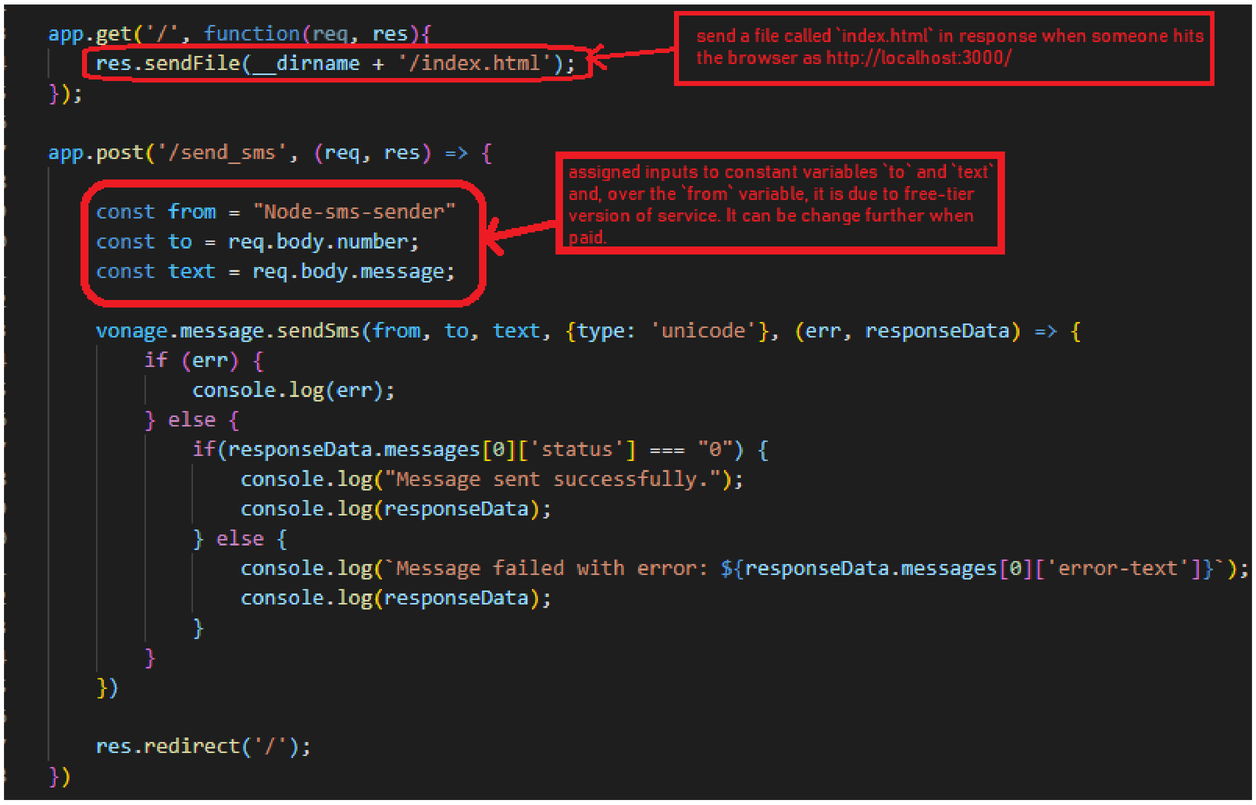
Now Next,
Vonage API has a built-in function called the ***sendSms()*** which is found under the ***message*** object function of ***vonage***.
that's why, It would written as,

vonage.message.sendSms()

This ***sendSms()*** function takes 3 arguments ,i.e, ***`from`, `to`, `text`*** and a call back function which ensures that the message is sent or not.

{type: 'unicode'} -> This is an optional parameter to the function which tells the encoding format. ***{ type: 'unicode' }*** of the message. It will make sure that you can send emojis too. If the message is sent/ fails to be sent , It will display the appropriate message in the console.

öne24



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

app.post('/send_sms', (req, res) => {

    const from = "Node-sms-sender"
    const to = req.body.number;
    const text = req.body.message;

    vonage.message.sendSms(from, to, text, {type: 'unicode'}, (err, responseData) => {
        if (err) {
            console.log(err);
        } else {
            if(responseData.messages[0]['status'] === "0") {
                console.log("Message sent successfully.");
                console.log(responseData);
            } else {
                console.log(`Message failed with error: ${responseData.messages[0]['error-text']}`);
                console.log(responseData);
            }
        }
    })

    res.redirect('/');
})
```

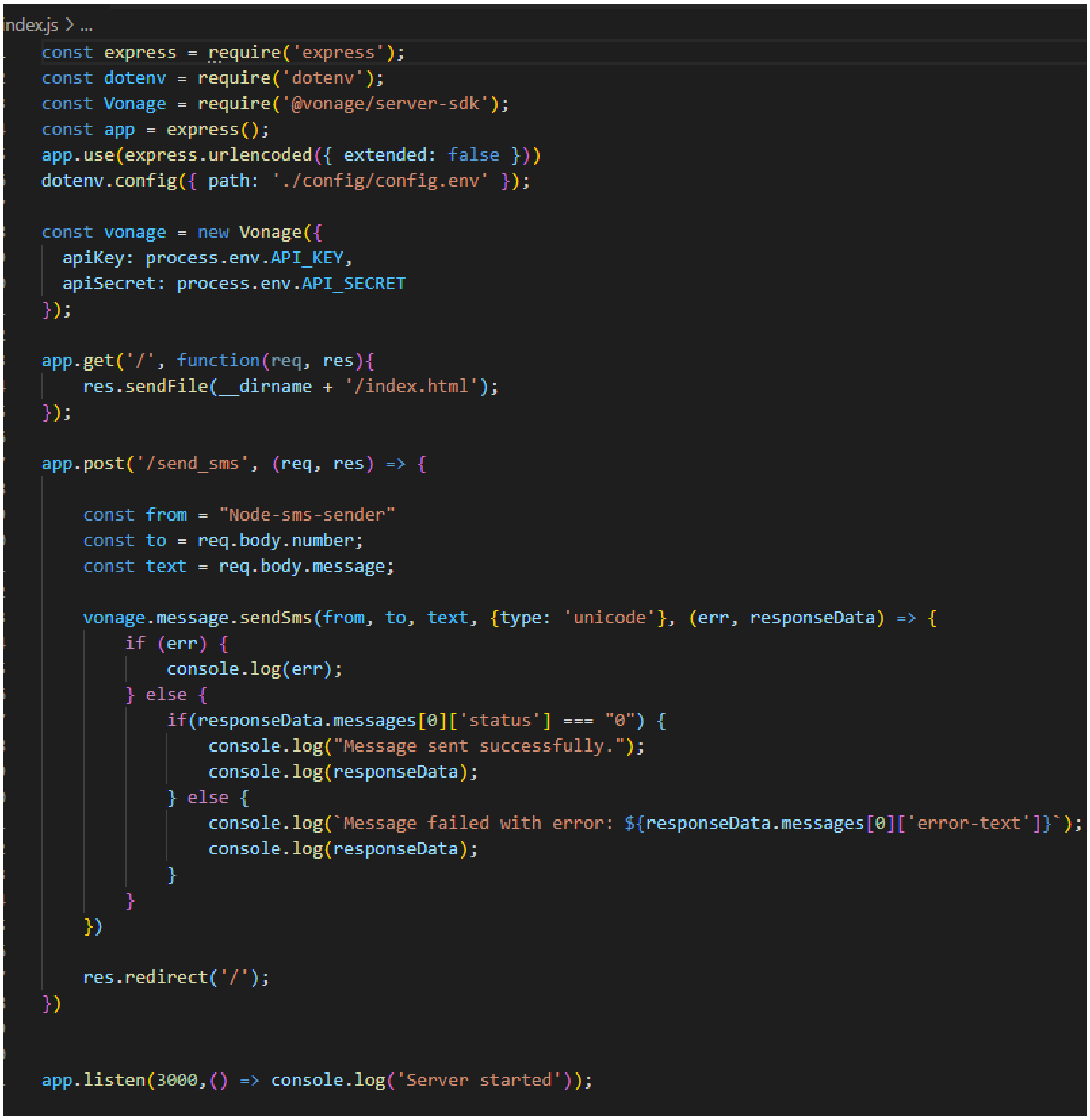
send a file called `index.html` in response when someone hits the browser as http://localhost:3000/

assigned inputs to constant variables `to` and `text` and, over the `from` variable, it is due to free-tier version of service. It can be change further when paid.

Picture-3: Code snippet for the implementation of vonage sendSms() function

Complete Code:

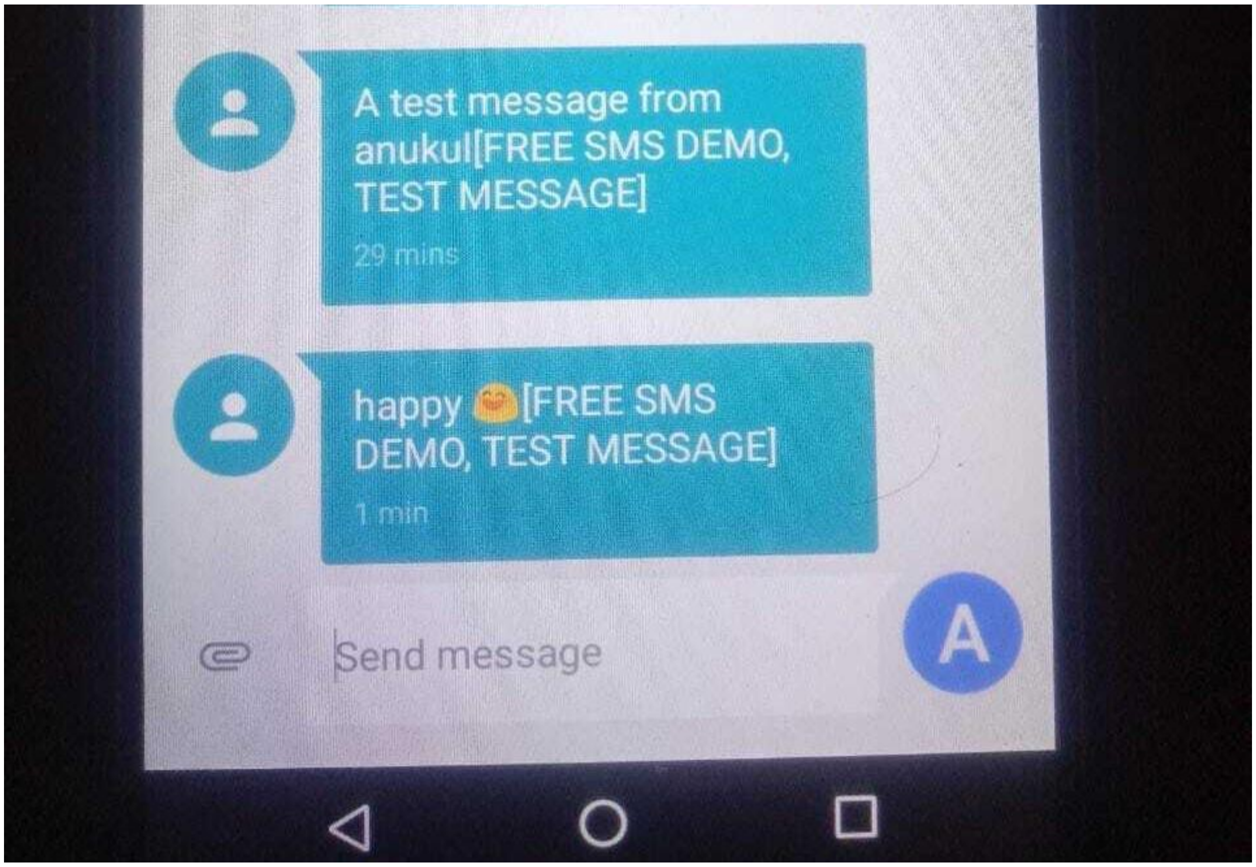
index.js



```
index.js > ...
const express = require('express');
const dotenv = require('dotenv');
const Vonage = require('@vonage/server-sdk');
const app = express();
app.use(express.urlencoded({ extended: false }))
dotenv.config({ path: './config/config.env' });

const vonage = new Vonage({
  apiKey: process.env.API_KEY,
  apiSecret: process.env.API_SECRET
});

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

app.post('/send_sms', (req, res) => {

    const from = "Node-sms-sender"
    const to = req.body.number;
    const text = req.body.message;

    vonage.message.sendSms(from, to, text, {type: 'unicode'}, (err, responseData) => {
        if (err) {
            console.log(err);
        } else {
            if(responseData.messages[0]['status'] === "0") {
                console.log("Message sent successfully.");
                console.log(responseData);
            } else {
                console.log(`Message failed with error: ${responseData.messages[0]['error-text']}`);
                console.log(responseData);
            }
        }
    })

    res.redirect('/');
})


app.listen(3000,() => console.log('Server started'));
```

index.html *[Refer Picture-2]*

Output:


A test message from anukul[FREE SMS DEMO, TEST MESSAGE]
29 mins

happy 😊[FREE SMS DEMO, TEST MESSAGE]
1 min

Picture-4: The final output screen.

Second Implementation,

Using Fast2SMS service :

Architecture

External dependencies/packages involved in the project:

1. **Express**

   Express is a Node.js web application framework that provides a robust set of features for developing web and mobile applications. It facilitates the rapid development of Node based Web applications. Following are some of the core features of Express framework −

   - Allows to set up middlewares to respond to HTTP Requests.
   - Defines a routing table which is used to perform different actions based on HTTP Method and URL.
   - Allows to dynamically render HTML Pages based on passing arguments to templates.

2. **Dotenv**

   Dotenv is a zero-dependency module that loads environment variables from a .env file into process.env.

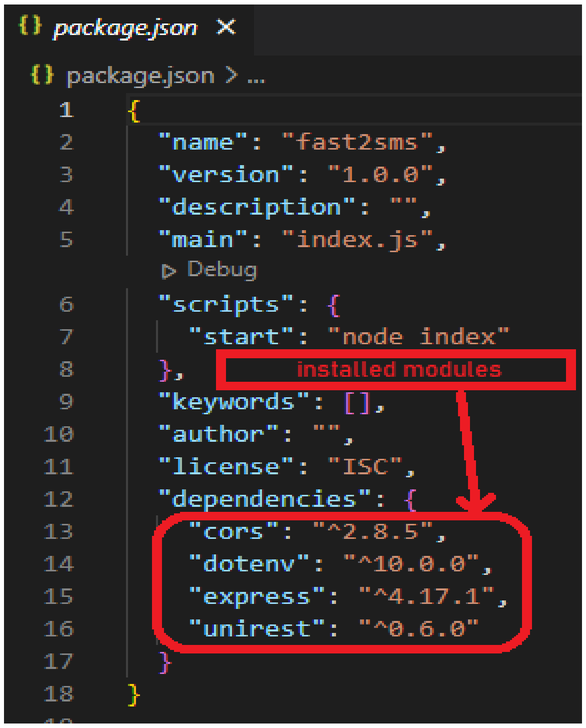
3. **unirest**

   Unirest is **a library to simplify making HTTP REST requests**. It's open source and available in multiple languages like Node.js, JAVA, .Net, Python, Ruby, etc

4. **Cors**

   Cross-Origin Resource Sharing (CORS) is an HTTP-header based mechanism that allows a server to indicate any origins (domain, scheme, or port) other than its own from which a browser should permit loading resources.

Installed dependencies:



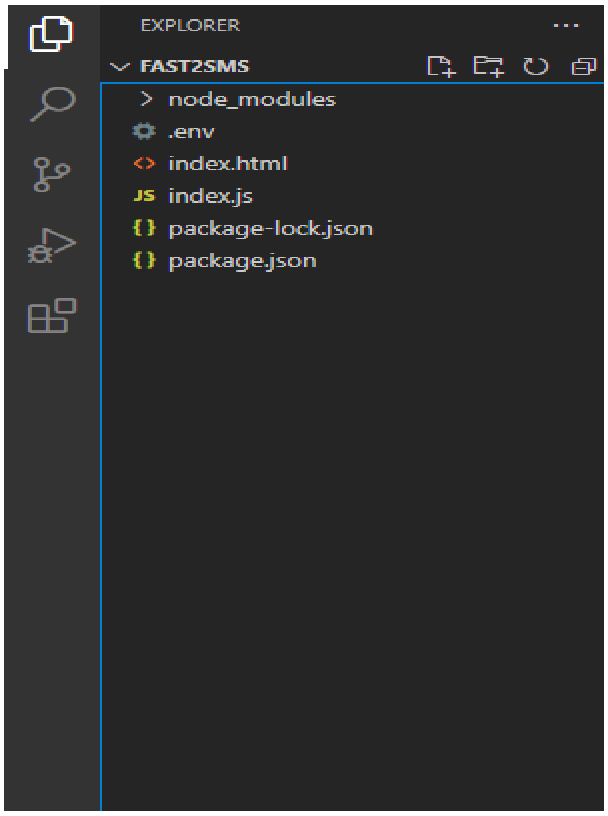
package.json ×

package.json > ...

```json
{
  "name": "fast2sms",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "node index"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "cors": "^2.8.5",
    "dotenv": "^10.0.0",
    "express": "^4.17.1",
    "unirest": "^0.6.0"
  }
}
```

öne24

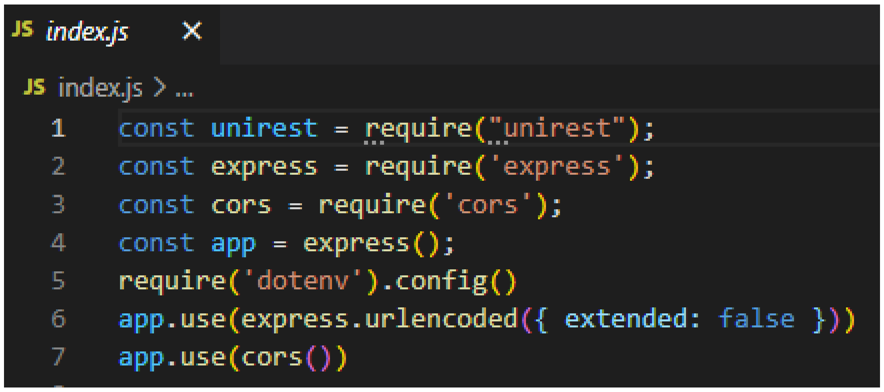
Directory Structure:



```
EXPLORER
FAST2SMS
  > node_modules
  .env
  index.html
  index.js
  package-lock.json
  package.json
```

Code Implementation:

First import all the required modules/packages which are installed by using ***node-package-manager(npm)***.



index.js > ...

```javascript
const unirest = require("unirest");
const express = require('express');
const cors = require('cors');
const app = express();
require('dotenv').config()
app.use(express.urlencoded({ extended: false }))
app.use(cors())
```

Why unirest ?

A request can be initiated by invoking the appropriate method on the unirest object, then calling .end() to send the request. Alternatively you can send the request directly by providing a callback along with the url.

Note: It is light-weight library, just to make API calls, that is why we have used it for making the small project, otherwise, we have more options also available such as axios, request, etc. for production level API calls.

Syntax:

unirest(method [, uri, headers, body, callback]) or
unirest[method]([uri, headers, body, callback])

- method: Request type (GET, PUT, POST, etc…)
- uri: *optional;* When passed will return a Request object. Otherwise returns generated function with method pre-defined (e.g. unirest.get)
- headers(Object): *optional;* HTTP Request headers
- body(Mixed): *optional;* HTTP Request body
- callback(Function): *optional;* Invoked when Request has finalized with the argument Response

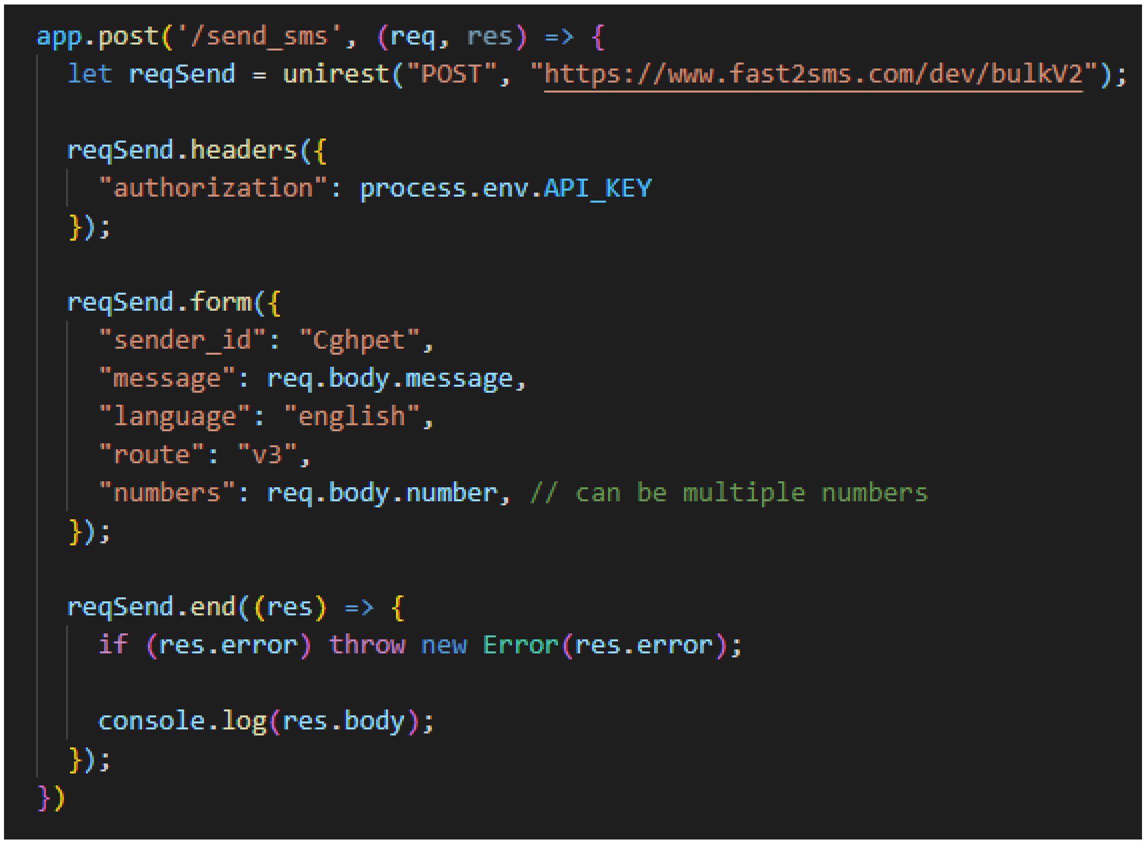
Why cors ?
CORS is a way to whitelist requests to your web server from certain locations, by specifying response headers like 'Access-Control-Allow-Origin'. It's **an important protocol for making cross-domain requests possible**, in cases where there's a legitimate need to do so.

To use cors in the program, do,

app.use(cors())

we use ***require("dotenv").config()***, as we configure the environment file exter**nally *[Refer Directory Structure]*** segment.

Next,



```
app.post('/send_sms', (req, res) => {
  let reqSend = unirest("POST", "https://www.fast2sms.com/dev/bulkV2");

  reqSend.headers({
    "authorization": process.env.API_KEY
  });

  reqSend.form({
    "sender_id": "Cghpet",
    "message": req.body.message,
    "language": "english",
    "route": "v3",
    "numbers": req.body.number, // can be multiple numbers
  });

  reqSend.end((res) => {
    if (res.error) throw new Error(res.error);

    console.log(res.body);
  });
})
```

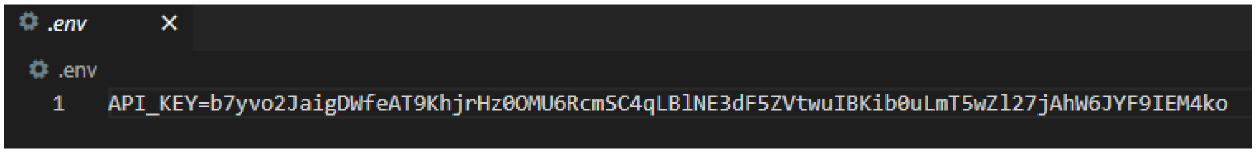
Picture-5: Implementation of Fast2Sms service.

On the above code snippet***[Refer picture-5]***, the whole procedure is the same as of Vonage API but the only change found on the Fast2Sms is the process of sending the SMS.

Let's understand step-by-step,

First, we make a call to the API of fast2sms and it is secured so we have to pass the `***headers***` using the key ***"authorization"*** and pass the value to that key using the API_KEY that we stored earlier in the environmental file(***.env file***). You can get the API key, once you create an account and login with that. ***(Refer: https://docs.fast2sms.com/)***

In our case:



.env

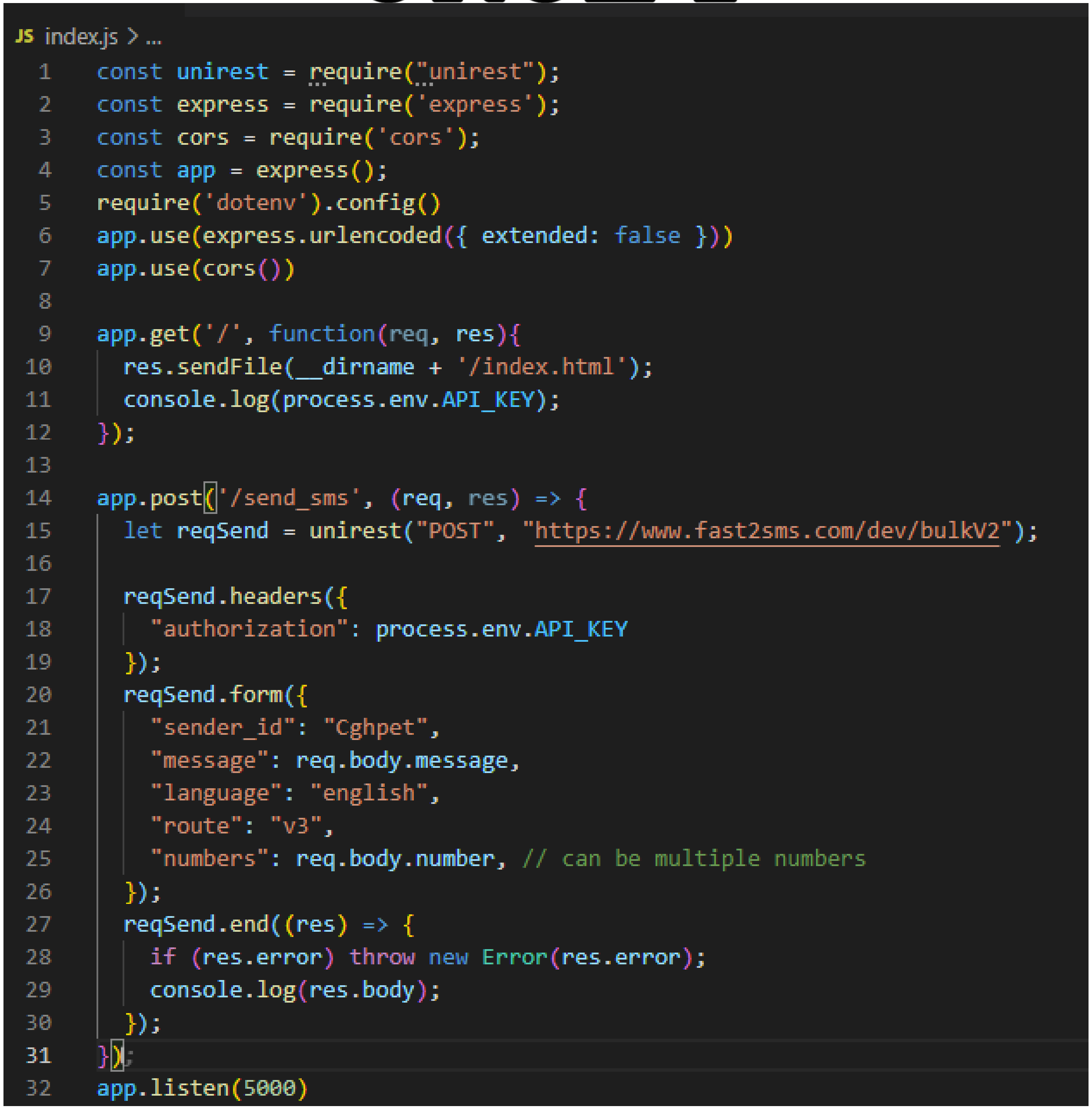
```
API_KEY=b7yvo2JaigDWfeAT9KhjrHz0OMU6RcmSC4qLBlNE3dF5ZVtwuIBKib0uLmT5wZl27jAhW6JYF9IEM4ko
```

After the auths process,
We have to provide the `number` and `message` to the ***req.body*** object, with some pair of keys and values to ***[Refer: https://docs.fast2sms.com/#post-method13]***

Note: “numbers” field is an array, so we can send messages to as many of the users we want to.

Lastly the ***end()*** function of the unirest package is used to ensure that the message which we are trying to send is sent or it gives an error while execution of the code/sending the message.

Complete Code:
index.js

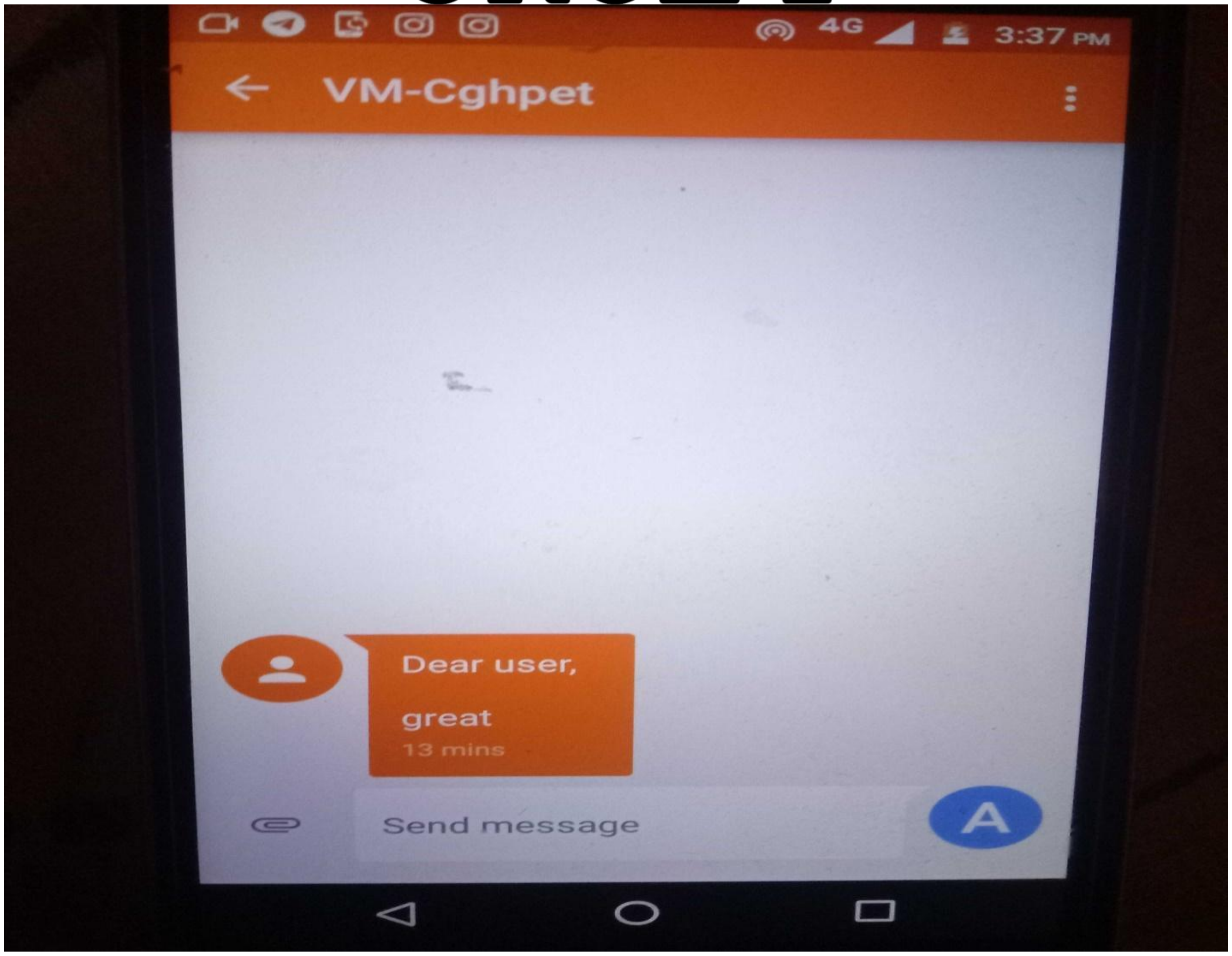


index.js > ...

```
const unirest = require("unirest");
const express = require('express');
const cors = require('cors');
const app = express();
require('dotenv').config()
app.use(express.urlencoded({ extended: false }))
app.use(cors())

app.get('/', function(req, res){
  res.sendFile(__dirname + '/index.html');
  console.log(process.env.API_KEY);
});

app.post('/send_sms', (req, res) => {
  let reqSend = unirest("POST", "https://www.fast2sms.com/dev/bulkV2");

  reqSend.headers({
    "authorization": process.env.API_KEY
  });
  reqSend.form({
    "sender_id": "Cghpet",
    "message": req.body.message,
    "language": "english",
    "route": "v3",
    "numbers": req.body.number, // can be multiple numbers
  });
  reqSend.end((res) => {
    if (res.error) throw new Error(res.error);
    console.log(res.body);
  });
});
app.listen(5000)
```

Index.html ***[Refer Picture-2]***

Output:



Dear user,

great

13 mins

Picture-5: The final output screen.

The following will be the response body, confirming that the message has been sent successfully:

```
{
        “return”: true,
        “request_id”: “lwdtp7cjyqxvfe9”,
        "message": [
                "Message sent successfully"
         ]
}
```

one24

Short Description :

1. Vonage API (aka Nexmo)

- Probably the best in the business, considering the accuracy and speed of it's services.
- Unpacked size is 494 kb, with 3 other dependencies, as listed on the official npm page of the package.
- The pricing is a bit of concern here, as the basic currency for it's credits are in Euro. It will cost around 85 INR for 34-35 messages, without any need for renewal ,i.e. until the credit ends.
- *npm install --save @vonage/server-sdk* will install this package

2. Fast2SMS

- Best suited for the job and startups in India have been using it extensively for a healthy amount of time.
- Simple,secure, affordable and fast, what else do we need..?, The pricing is quite good as it touches down the lowest possible barrier for SMS messaging in the country.
- With the highest plans, which charge around 0.20 INR per message, one can send 34-35 messages, spending only 6.8 in Indian currency,which is quite affordable
- Depending upon the time you need the service for, and the amount you want to spend, here is a clear pricing plans:
  Bulk SMS Price List in Fast2SMS
- The implementation is quite simpler, just get an API_KEY from the official site after signing up for a test message/ bulk messaging, and follow the further instructions:
- We don't have to install amy npm package, rather we can just hit the API endpoints and it will do the messy work for us
- Also, here is a perfect guide, if one wants to test this on his/her own, in any language
  API documentation
- We have considered the exact same setup from the Vonage API part, so here I will just skip to the implementation part of this API

3. PureText

- PureText offers a Texting/SMS gateway to which we can make an HTTP GET/POST request with details such as recipient number and body of the text and it will deliver the text to the recipient for us.
- In order to identify that this request belongs to us, we'll need to include our API token in the HTTP GET/POST request. We can get our API token from here
- With an unpacked size of 4.55 kb and 2 dependencies, it is a better lightweight solution, but the community is not that big as of now, which will surely get better in upcoming days.
- The price is very much affordable as compared to the others, here is a brilliant short guide on how to integrate it with our Node.js project: https://crm.puretext.us/articles/puretext-node-js-sms-api-or-node-js-sms-gateway-or-node-js-send-receive-texts/
- Often , it can prove to be great for startups and small organizations, but can slow down on the large scale.

4. Twilio

- A great community, having the highest number of downloads, and the largest community in this list
- Great for large and complex applications, which require complex exchange of data through messages, invoices etc, and a go to choice for enterprise-grade applications
- But the dependency tree is quite heavy, having an unpacked size around 13.5 mb, with 11 more dependencies, which is not suitable for a simple SMS sending tasks
- Other than that, the pricing is second highest after the Vonage API , this is around 44 INR for 34-35 messages.
- If you want to integrate it with your Node application, here is a brilliant guide to do so, https://www.twilio.com/docs/libraries/reference/twilio-node/

5. Amazon SNS

- A fully managed messaging service for both application-to-application (A2A) and application-to-person (A2P) communication.
- One can get started with the AWS free tie, and send upto first 1 million requests for free.
- Following page can help one to get started pretty quickly with it. https://aws.amazon.com/sns/getting-started/

one24

References:

- Vonage API
- Twilio
- Fast2SMS
- PureText
- Unitest
- Node.js, Node.js Tutorial
- Amazon SNS

Thank you.

Paper written by:
Anukul Kumar
National Institute of Technology (N.I.T Agartala)
Company: One24

Paper reviewed by:
Rahul Ranjan
(Signature)
SDE-3 (One 24)

One24.