

サイバー大学IT総合学部

専門応用科目

JavaScriptフレームワークによるWeb開発

第6回 REST API実習

小園井康志

第6回 学習目標

- REST APIについて理解し説明できる
- Postmanについて理解し説明できる
- Node.jsでREST APIを作成できる

第6回 授業構成

- 第1章 REST API概要
- 第2章 Postman概要
- 第3章 REST API実習

JavaScriptフレームワークによるWeb開発

第6回 REST API実習

第1章

REST API概要

第1章 学習目標

- REST APIについて理解し説明できる

Expressルーティング前回のおさらい

作業内容

- ルーティングについて学習
（Express Generatorで作成されたファイルを確認（app.js））
- 画面を作成するテンプレートについて理解
- Expressでの仕組みで新しいページを作成

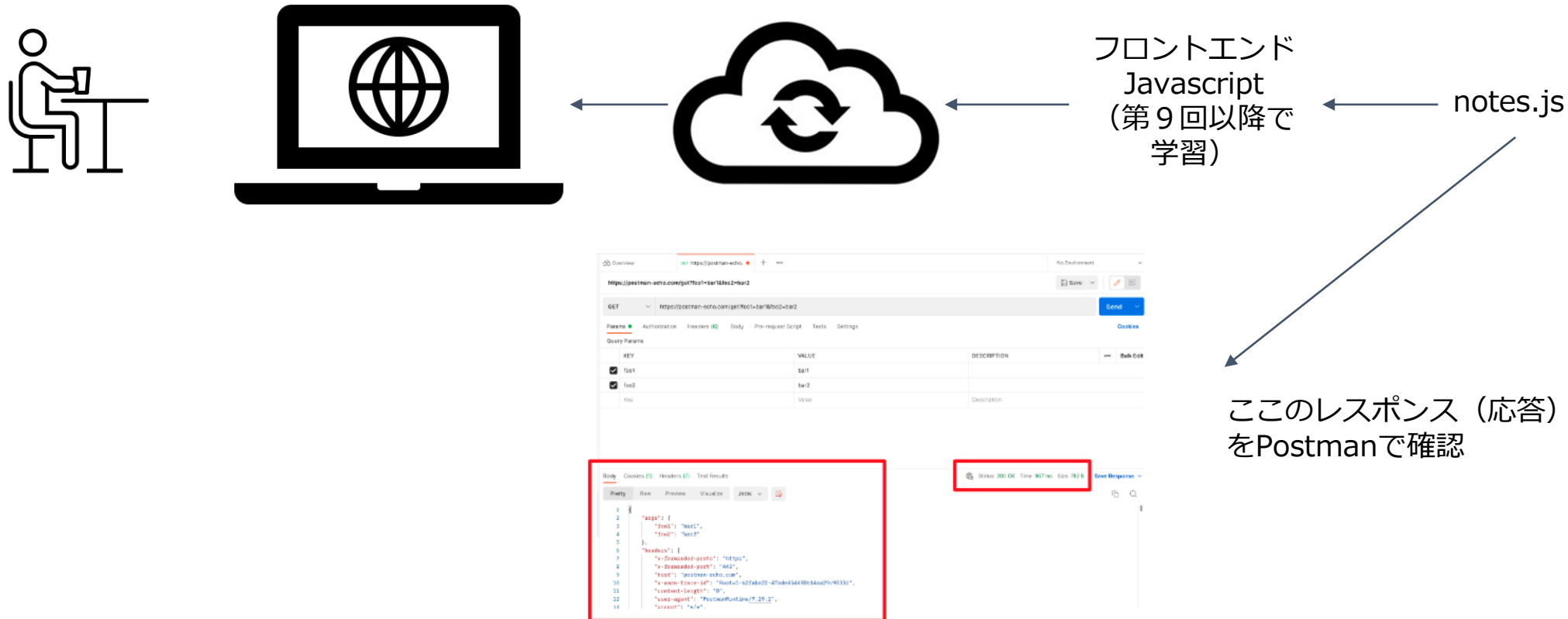
ここでおさらい

第5回では以下のように画面を作成



REST APIを利用した画面作成

第5回では以下のように画面を作成



REST APIについて

REST APIについて説明する前に、まずAPIとは何かから説明する。

この章の主な流れ

- 1. API
- 2. WebAPI
- 3. SOAP
- 4. REST API
- 5. REST APIの定義

APIとWebAPI

- API

Application Programming Interfaceの略で、異なるソフトウェア・アプリケーションが、お互いにやりとりするために使用するインタフェースのことを指す。アプリケーションからライブラリーなどを利用するときに使われることが多い

＊ライブラリー：ある特定の機能を持つプログラムを定型化して、他のプログラムが引用できる状態にしたもの

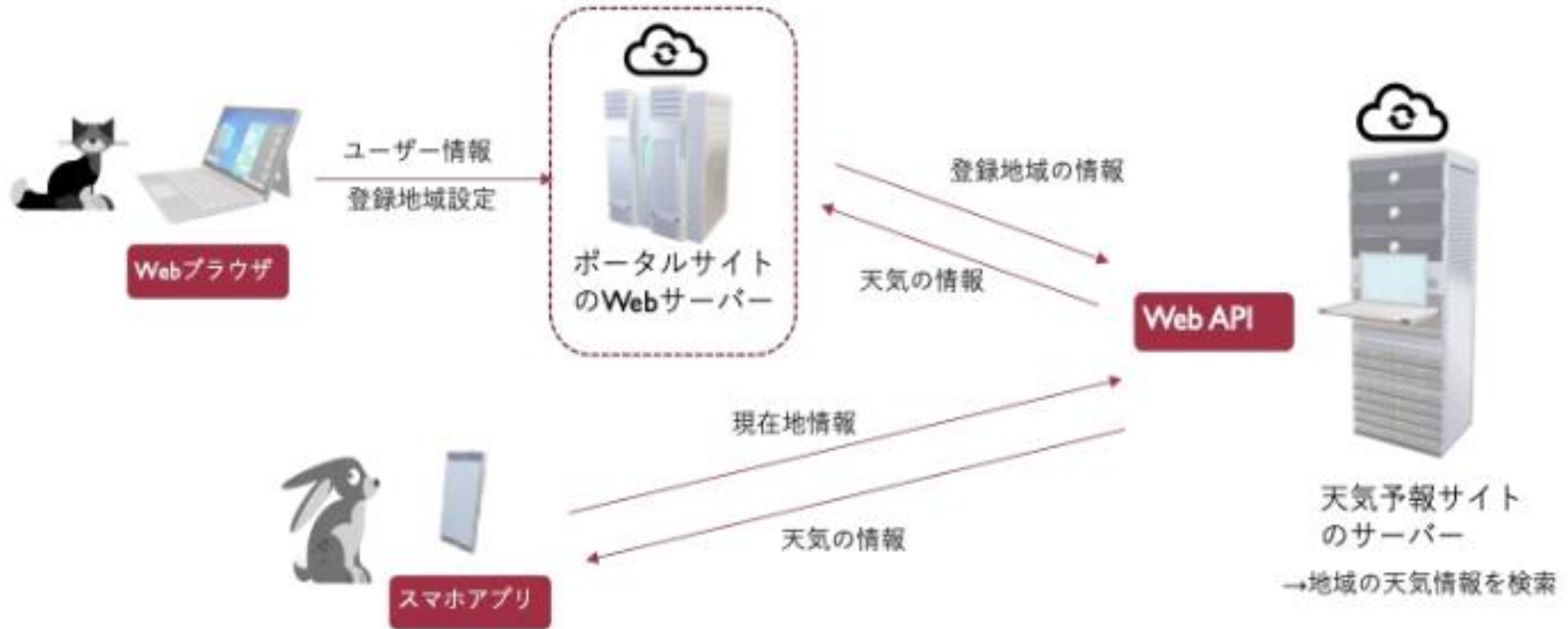
- WebAPI

APIの利用を、HTTP／HTTPSプロトコルを使ったインターネットなどで実現するAPIのことを指す。

RESTful APIとも呼ばれる。HTTPのメソッドを使って利用する。

インターネット上の様々なWebサービスがこのWebAPIを利用している。

REST API使用例



SOAPとREST API

Web APIの中にも、実装方式としてREST APIやSOAP API などが存在している。

- SOAP (Simple Object Access Protocol)

XMLデータのやり取りを行うRPCプロトコル。

RPC (Remote Procedure Call) は、別のプロセス (別コンピュータ) に対して、メソッド呼び出しを行うものを指している。

あくまで別環境のメソッド呼び出しを行うものに過ぎない。

- REST API (REpresentational State Transfer)

URIでリソースを一意に識別し、リソースの操作はHTTPメソッドで指定し操作する実装方式。セッション管理や状態管理は行わず (ステートレス)、同じURIの呼び出し結果は、常に同じ結果が返されることが必要。

結果は、JSONやXML、HTMLで返されるが、JSONが主流。

処理結果はHTTPステータスコードで通知する。

リソース (データ) へアクセスしたい場合の実装方針。

他に最近では GraphQL、gRPCなどがある

REST APIの定義

REST APIの定義は、メソッドとURIを決めること。

HTTPメソッド

REST APIで使われるメソッド

メソッド名	役割	幂等性	安全性
GET	リソースの取得	○	○
POST	リソースの作成	×	×
PUT	リソースの全体更新（置き換え）	○	×
PATCH	リソースの部分更新（一部を更新）	×	×
DELETE	リソースの削除	○	×

URIとは

URIは、URL（Web上のどこにあるか）と
URN（Web上にあるファイル名）の総称。

実際には、URLやURIは厳密に区別されていない。

URI とは URL, URNのどちらか または 両方を指したりする。

Webページ（<https://ibm.com>）は、URIでありURL。

<https://>はURIの一部。

REST APIの定義

REST APIの定義の例。

URI	メソッド	説明
/notes	GET	ノートデータを全件取得する。
/notes/[ID]	GET	IDのノートデータを取得する。
/notes	POST	ノートデータを新規登録する。
/notes/[ID]	PUT	IDのノートデータを更新する。
/notes/[ID]	DELETE	IDのノートデータを削除する。

REST APIの定義 補足

一般に、以下のようなルールで作成される

- ◆ URLに[ID]を設定する場合には、特定のIDに対応したデータが処理対象。
IDを設定しない場合には全量が対象。
- ◆ GETはデータを読み取り・取得する用途。
- ◆ POSTはデータを書き込み・登録する用途。
- ◆ PUTはデータを更新する用途。
- ◆ DELETEはデータを削除する用途。

第1章 まとめ

- APIとは、異なるソフトウェア・アプリケーションが互いにやりとりするために使用するインタフェースを指す。
- WebAPIとは、HTTP/HTTPSプロトコルを使ったインターネットなどで実現するAPIを指す。
- WebAPIには種類があり、以下の2種類を紹介した。
 - SOAP
 - REST API : URIでリソースを一意に識別し、リソース操作はHTTPメソッドで指定して操作する実装方式。
- URIとは、URLとURNの総称。
- REST APIに使われるメソッドにはGet、Put、Post、Deleteなどがある。

JavaScriptフレームワークによるWeb開発

第6回 REST API実習

第1章

REST API概要

終わり

JavaScriptフレームワークによるWeb開発
第6回 REST API実習

第2章 Postman概要

第2章 学習目標

- Postmanについて理解し説明できる

Postmanについて概要

- Web APIのテストクライアントサービスの一つで、開発した REST APIなどを簡単にテストすることができる。
 - APIを開発・テストするためのコラボレーションプラットフォーム。Postmanでは、Token（トークン）など認証情報を設定し、APIのエンドポイントにリクエストを投げ込めば Response（レスポンス）を確認することができる。
- SOAP、REST、GraphQL、gRPCなどの方式に対応している

それ以外の主な機能

機能一覧

- API client: REST、SOAP、GraphQLのリクエストを直接実行
- Automated Tessitn: 自動テスト機能を提供
- Design & Mock: サーバ機能を提供しエンドポイントとレスポンスをシミュレートし、予想される動作を行う
- Documentation: API定義からAPI仕様書を生成
- Monitor: モニタリング機能（パフォーマンス、レスポンスなど）
- Workspace: チーム間でのAPI共同開発機能

Postmanのこれらの機能を利用することによってAPI開発が簡略化され作業効率の向上を実現することができる

Postmanについて

Postmanは、Web版とアプリ版がある。
(どちらも基本的に使い方は同じ。)

Web版は下記からSign Upして利用可能。

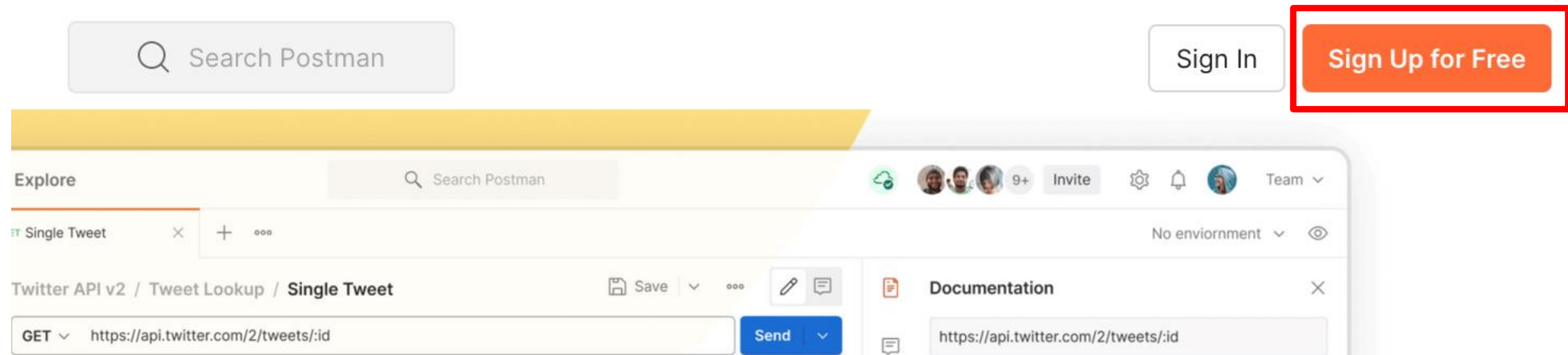
<https://www.postman.com>

アプリ版は下記からインストールが可能。

<https://www.postman.com/downloads/>

Postmanの使い方（初期登録）

- ここではPostmanを初めて使う方向けにWeb版の手順を実施する。
- 次のサイトにアクセス <https://www.postman.com>
- 「Sign Up for Free」をクリック。



Postmanの使い方（初期登録）

名前とロールを入力して「Continue」をクリック
名前とロールは任意の文字を入れる



Welcome to Postman! Tell us a bit about yourself.

Your name

eg: John Doe

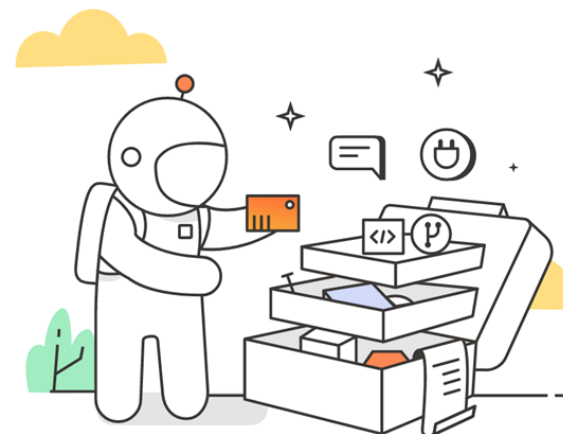
❗ Please tell us your name

What is your role?

Knowing your role will help us offer you a better experience

Select your role

Continue



Postmanの使い方（初期登録）

そのまま「Continue without team」をクリック



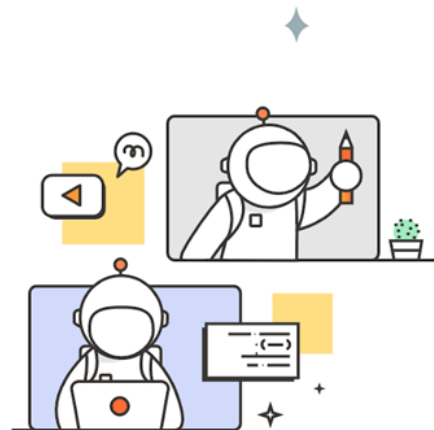
Bring people you want to collaborate with on Postman.

Invite people via ☒ Email ☐ Invite Link

Enter an email address

Finish

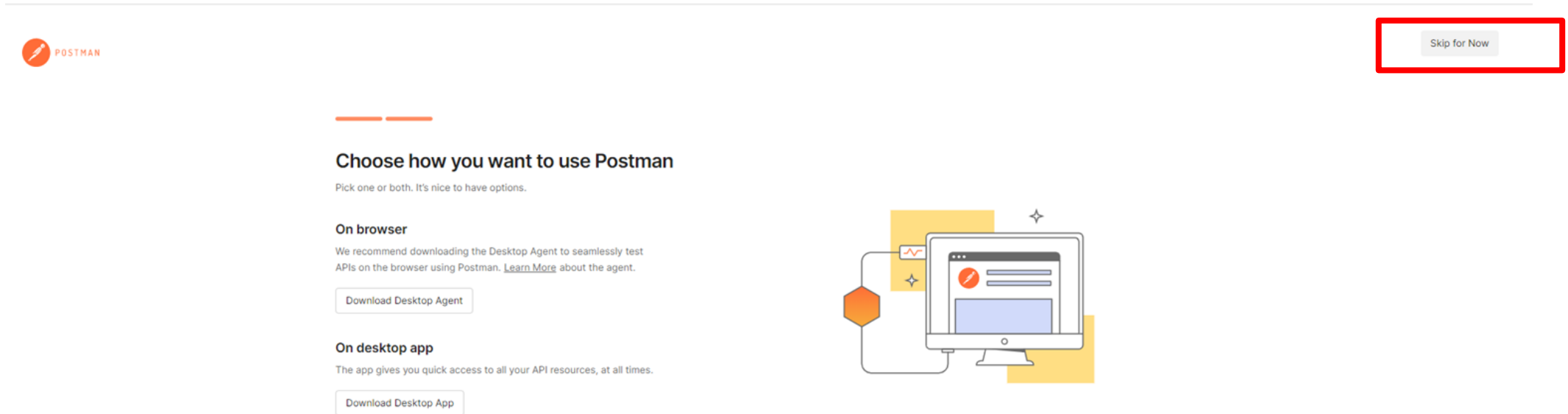
[Continue without team](#)



- ✓ Collaborate on requests, collections, API's and more.
- ✓ Review and manage changes to an API using version-control and API versioning.
- ✓ Monitor your API's health with monitors.

Postmanの使い方（初期登録）

この画面が出てきたら”skip for now”をクリック



Postmanの使い方（初期登録）

作業中にトレーニングのポップアップが出てきたら
"End Lesson"をクリック

	bar1	
	bar2	
	Value	Des

ts

JSON ▾

33% completed

End lesson

Response body

This is the response returned by the server.

Next

67% completed

End lesson

Save request

Save API requests in collection to set common authorisation, variables and easily share with colleagues.

Save ▾

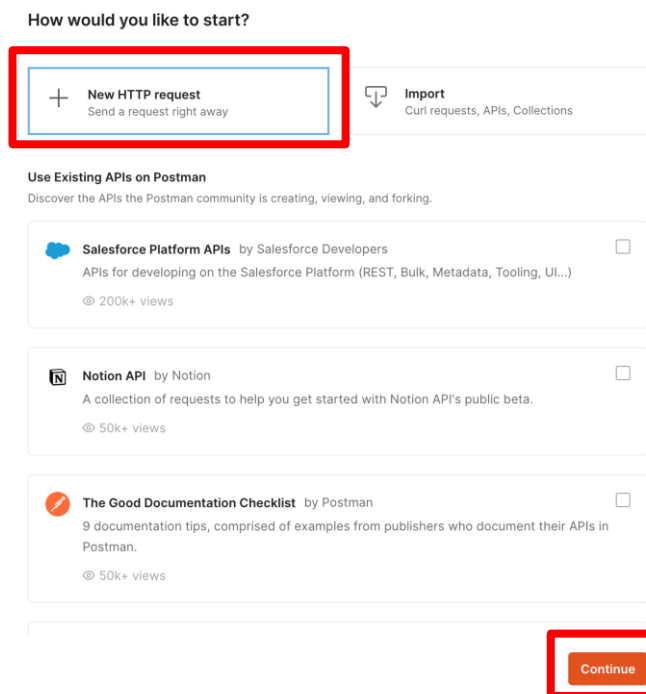
Prev

DESCRIPTION

Postmanの使い方

「How would you like to start?」と表示されたら、
「New HTTP request」をクリックし、「Continue」をクリック。

表示されない場合、Start with something newのCreate Newをクリック



または

Good morning!

Pick up where you left off.

Recently visited workspaces

My Workspace

Get started with Postman

Start with something new

Create a new request, collection, or API in a workspace

Create New →

Import an existing file

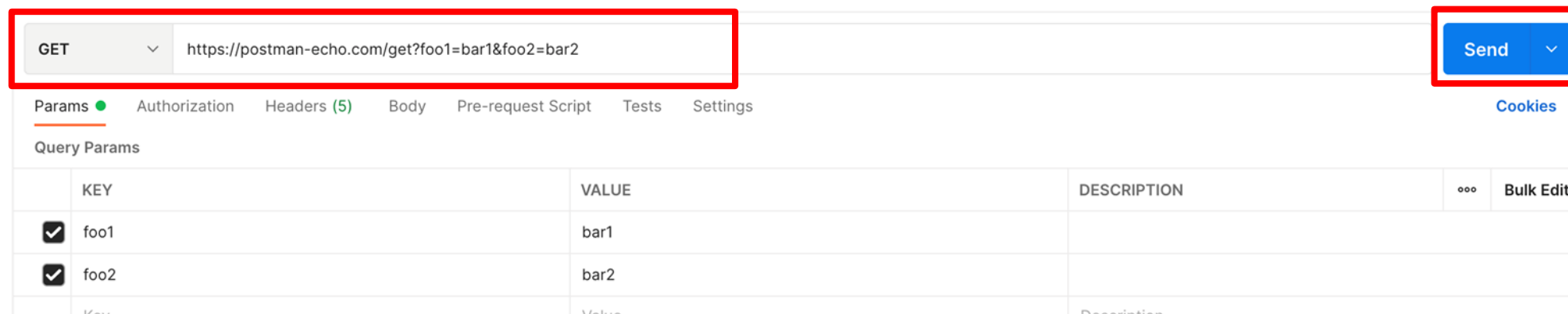
Import any API schema file from your local drive or Github

Import file →

Postmanの使い方

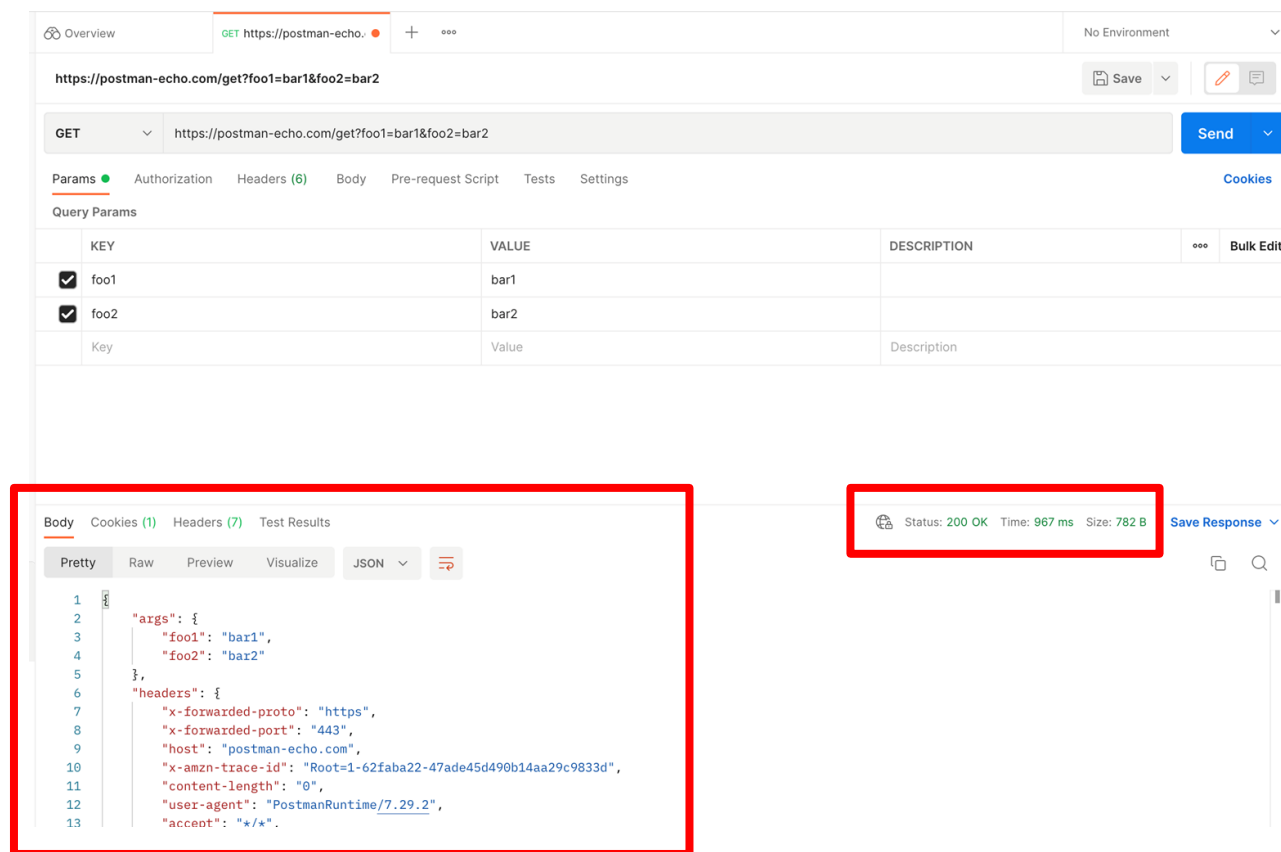
- ここではPostmanの公式ドキュメントに記載されているAPIを実行し、その結果を確認する。
- GETの右の入力フォームに以下を入力し、Sendボタンをクリック

`https://postman-echo.com/get?foo1=bar1&foo2=bar2`



Postmanの使い方

APIの応答結果が下の赤枠の部分に表示される。



Postmanの使い方

Testsタブにテストコードを記載するとAPIの応答に対してOK・NGを判定できる。

The screenshot displays the Postman interface for a GET request to `https://postman-echo.com/get?foo1=bar1&foo2=bar2`. The **Tests** tab is selected, showing the following JavaScript code:

```
1 pm.test("Status code is 200", function () {  
2   pm.response.to.have.status(200);  
3 });
```

The **Test Results (1/1)** section shows the result of the test:

Test Results (1/1)
PASS Status code is 200

Additional details visible in the interface include the **Overview** tab, the **Send** button, the **Params** tab, the **Authorization** tab, the **Headers (6)** tab, the **Body** tab, the **Pre-request Script** tab, the **Settings** tab, the **Cookies** tab, the **Save** button, the **Learn more about tests scripts** link, the **SNIPPETS** section, the **Status: 200 OK** message, the **Time: 208 ms** message, the **Size: 762 B** message, and the **Save Response** button.

Postmanの使い方

Testsタブにテストコードを記載するとAPIの応答に対してOK・NGを判定できる。

GET <https://postman-echo.com/get?foo1=bar1&foo2=bar2> [Send](#)

Params • Authorization Headers (6) Body Pre-request Script **Tests •** Settings Cookies

```
1 pm.test("Status code is 200", function () {  
2   pm.response.to.have.status(200);  
3 });
```

Test scripts are written in JavaScript, and are run after the response is received. Learn more about [tests scripts](#)

Snippets
[Send a request](#)
Status code: Code is 200
[Response body: Contains string](#)

Body Cookies (1) Headers (5) **Test Results (1/1)**

All Passed Skipped Failed

PASS Status code is 200 **テスト結果**

Status: 200 OK Time: 217 ms Size: 832 B [Save Response](#)

テストコードは右側から選ぶと
スニペット（コードが表示されます）

Postmanの使い方

動画を全画面表示にして視聴してください。

第2章 まとめ

- Postmanとは、Web APIのテストクライアントサービスの一つ。開発したREST APIなどをテストすることができる。
- Postmanの使い方を学習した。
 - ローカルPC環境、オンラインWeb環境があることを学習
 - オンラインWeb環境での操作方法を学習
 - サンプルのRest APIサービスを使い応答を確認する

JavaScriptフレームワークによるWeb開発

第6回 REST API実習

第2章

Postman概要

終わり

JavaScriptフレームワークによるWeb開発
第6回 REST API実習

第3章

REST API実習

第3章 学習目標

- REST APIサービスを作成することができるようにする
 - 1、 2章で学習してことをベースにREST APIサービス機能を作成
サーバ上で動作させPostmanで確認

REST API実習

- ここでは 1 章で定義したREST APIの一部を実際に実装する。

URI	メソッド	説明
→ /notes	GET	ノートデータを全件取得する。
/notes/[ID]	GET	IDのノートデータを取得する。
/notes	POST	ノートデータを新規登録する。
/notes/[ID]	PUT	IDのノートデータを更新する。
/notes/[ID]	DELETE	IDのノートデータを削除する。

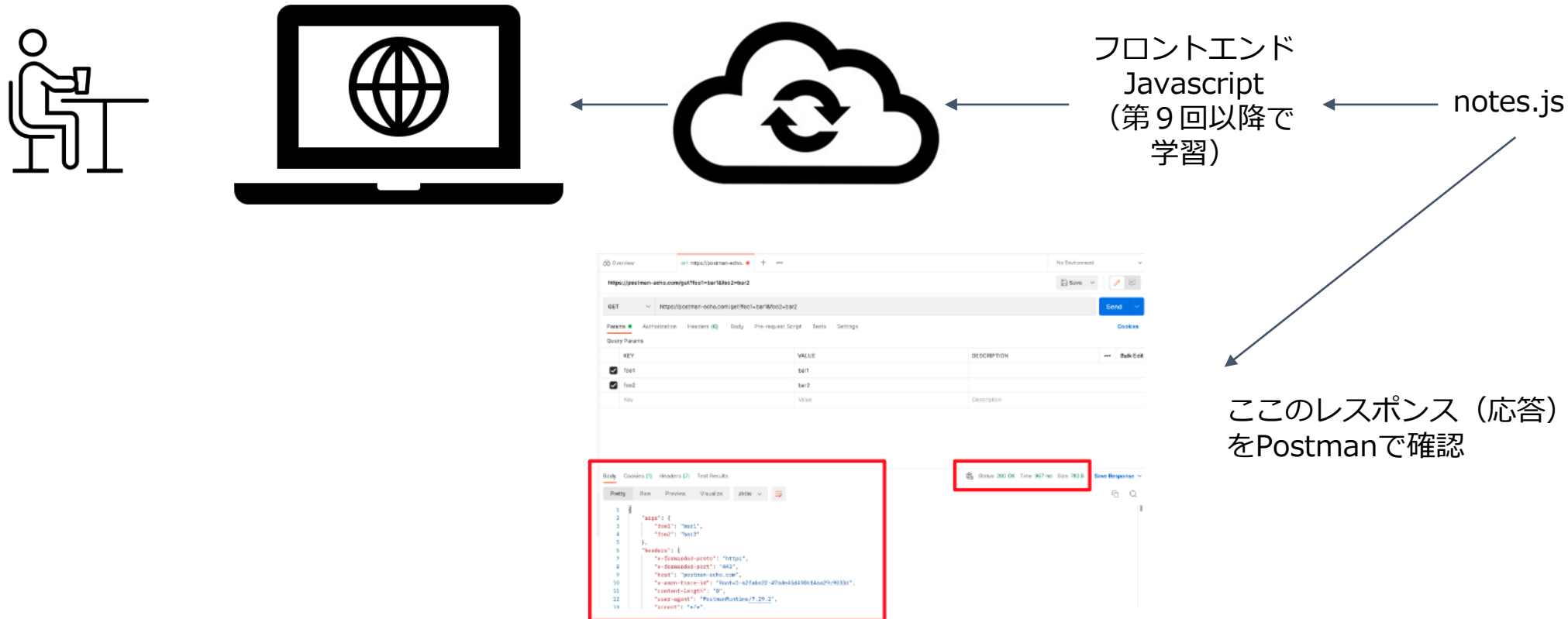
ここでおさらい

第5回では以下のように画面を作成



REST APIを利用した画面作成

第5回では以下のように画面を作成



notes.jsの追加

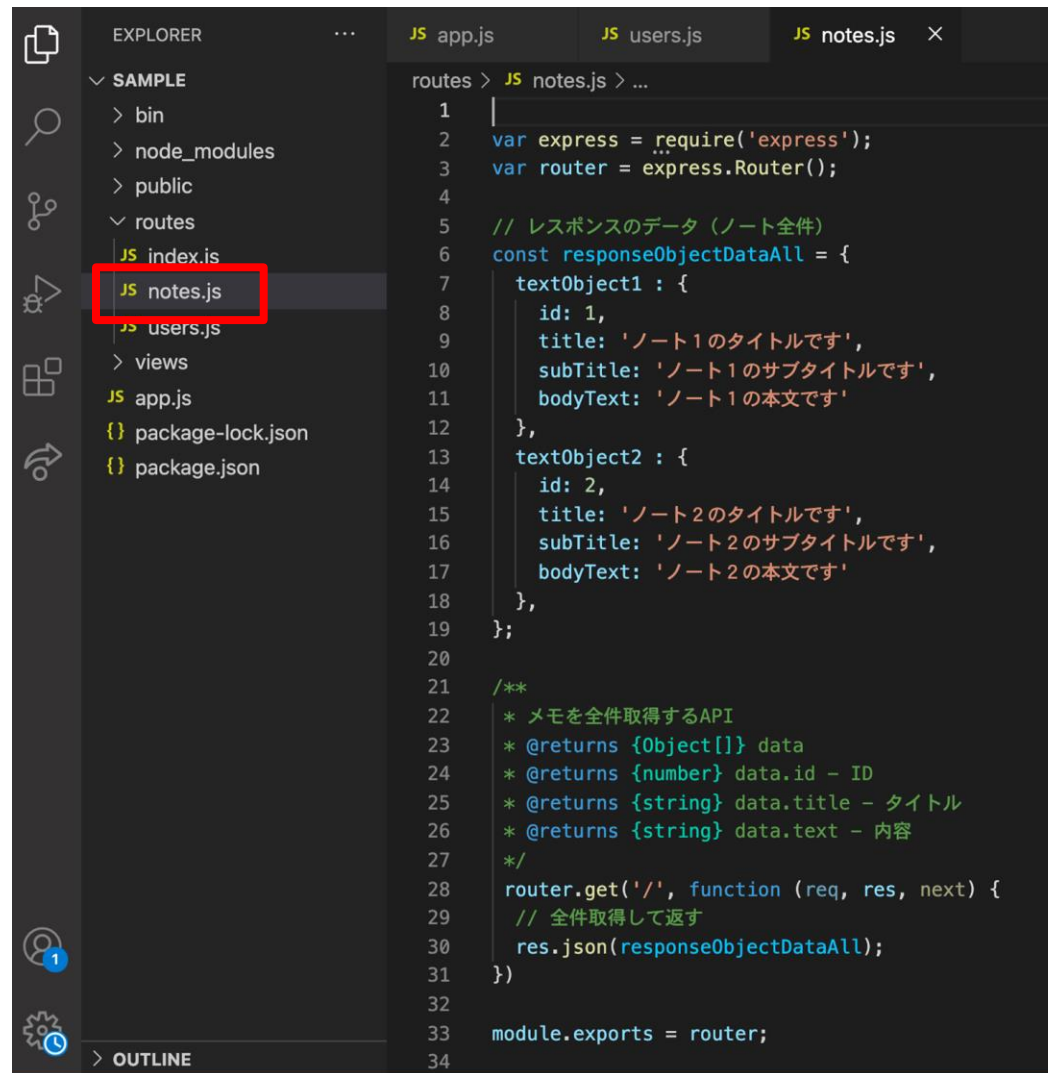
- routes配下にnotes.jsを追加。
- notes.jsに以下を記述。

```
var express = require('express');
var router = express.Router();

// レスポンスのデータ (ノート全件)
const responseObjectDataAll = {
  textObject1 : {
    id: 1,
    title: 'ノート1のタイトルです',
    subTitle: 'ノート1のサブタイトルです',
    bodyText: 'ノート1の本文です'
  },
  textObject2 : {
    id: 2,
    title: 'ノート2のタイトルです',
    subTitle: 'ノート2のサブタイトルです',
    bodyText: 'ノート2の本文です'
  },
};

/**
 * メモを全件取得するAPI
 * @returns {Object[]} data
 * @returns {number} data.id - ID
 * @returns {string} data.title - タイトル
 * @returns {string} data.text - 内容
 */
router.get('/', function (req, res, next) {
  // 全件取得して返す
  res.json(responseObjectDataAll);
})

module.exports = router;
```



The screenshot shows the VS Code interface. On the left, the Explorer sidebar displays a project structure with folders 'bin', 'node_modules', 'public', and 'routes'. Inside 'routes', the files 'index.js', 'notes.js', and 'users.js' are listed. 'notes.js' is highlighted with a red rectangle. The main editor area shows the content of 'notes.js', which includes the same code as the one provided in the text block, including the API endpoint definition and JSDoc comments.

notes.jsの中身

json形式のデータを定義

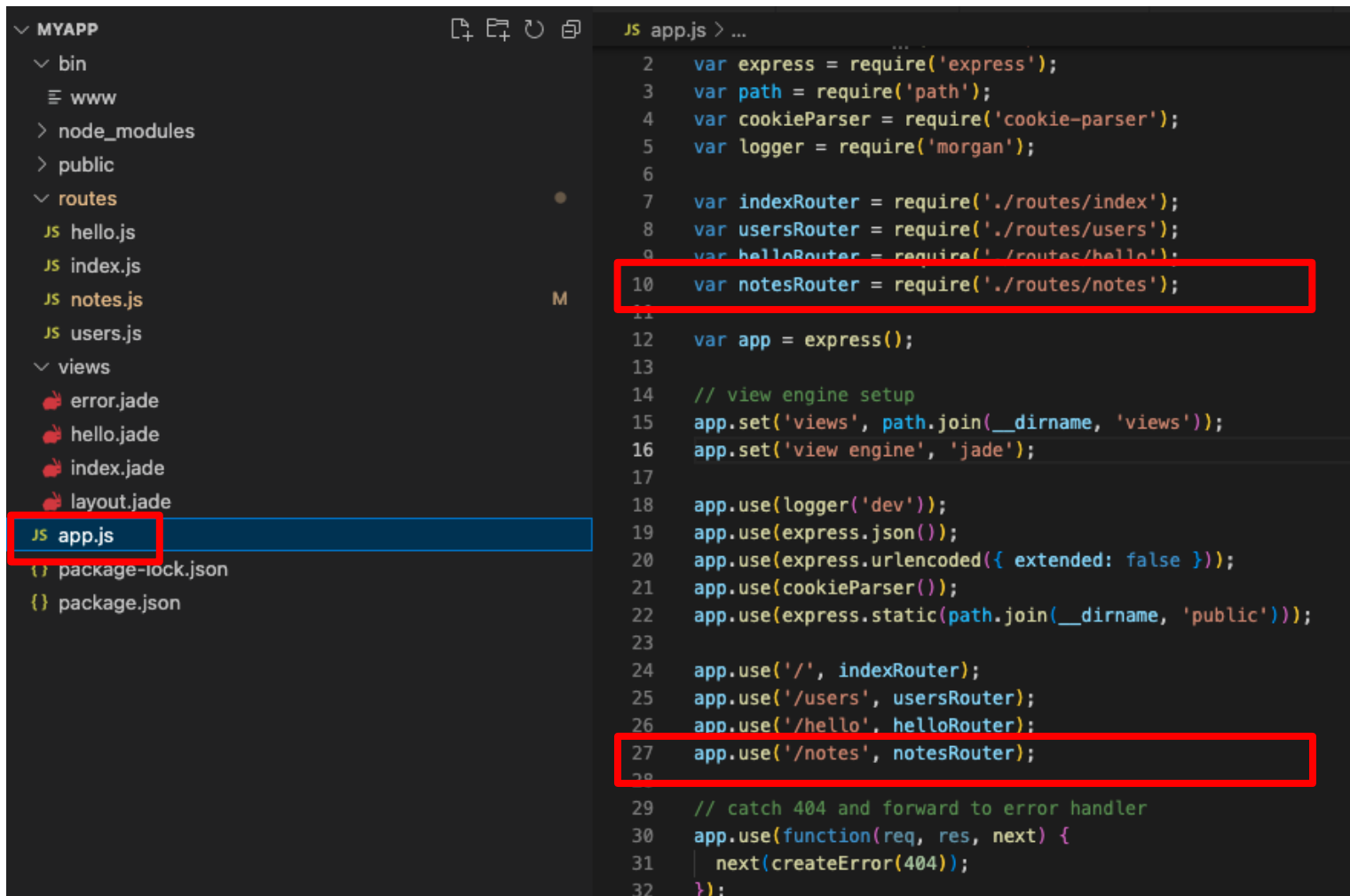
Getリクエストがあった時に上記のデータをjson形式で出力します。

```
1  var express = require('express');
2  var router = express.Router();
3
4  // レスポンスのデータ (ノート全件)
5  const responseObjectDataAll = {
6    textObject1 : {
7      id: 1,
8      title: 'ノート1のタイトルです',
9      subTitle: 'ノート1のサブタイトルです',
10     bodyText: 'ノート1の本文です'
11   },
12   textObject2 : {
13     id: 2,
14     title: 'ノート2のタイトルです',
15     subTitle: 'ノート2のサブタイトルです',
16     bodyText: 'ノート2の本文です'
17   },
18 };
19
20 /**
21  * メモを全件取得するAPI
22  * @returns {Object[]} data
23  * @returns {number} data.id - ID
24  * @returns {string} data.title - タイトル
25  * @returns {string} data.text - 内容
26  */
27 router.get('/', function (req, res, next) {
28   // 全件取得して返す
29   res.json(responseObjectDataAll);
30 })
31
32 module.exports = router;
```

app.jsの変更

app.jsに以下を追加。

```
var notesRouter = require('./routes/notes');  
  
app.use('/notes', notesRouter);
```



```
JS app.js > ...  
2  var express = require('express');  
3  var path = require('path');  
4  var cookieParser = require('cookie-parser');  
5  var logger = require('morgan');  
6  
7  var indexRouter = require('./routes/index');  
8  var usersRouter = require('./routes/users');  
9  var helloRouter = require('./routes/hello');  
10 var notesRouter = require('./routes/notes');  
11  
12 var app = express();  
13  
14 // view engine setup  
15 app.set('views', path.join(__dirname, 'views'));  
16 app.set('view engine', 'jade');  
17  
18 app.use(logger('dev'));  
19 app.use(express.json());  
20 app.use(express.urlencoded({ extended: false }));  
21 app.use(cookieParser());  
22 app.use(express.static(path.join(__dirname, 'public')));  
23  
24 app.use('/', indexRouter);  
25 app.use('/users', usersRouter);  
26 app.use('/hello', helloRouter);  
27 app.use('/notes', notesRouter);  
28  
29 // catch 404 and forward to error handler  
30 app.use(function(req, res, next) {  
31   next(createError(404));  
32 });
```

ブラウザでの動作確認

- ・ npm startでアプリを起動し、ブラウザに、以下を入力。
<http://localhost:3000/notes>
- ・ 以下のように表示されたらOK。

← → ↻ ⓘ localhost:30000/notes

```
{"textObject1":{"id":1,"title":"ノート1のタイトルです","subTitle":"ノート1のサブタイトルです","bodyText":"ノート1の本文です"},"textObject2":{"id":2,"title":"ノート2のタイトルです","subTitle":"ノート2のサブ
```

サーバでの動作確認

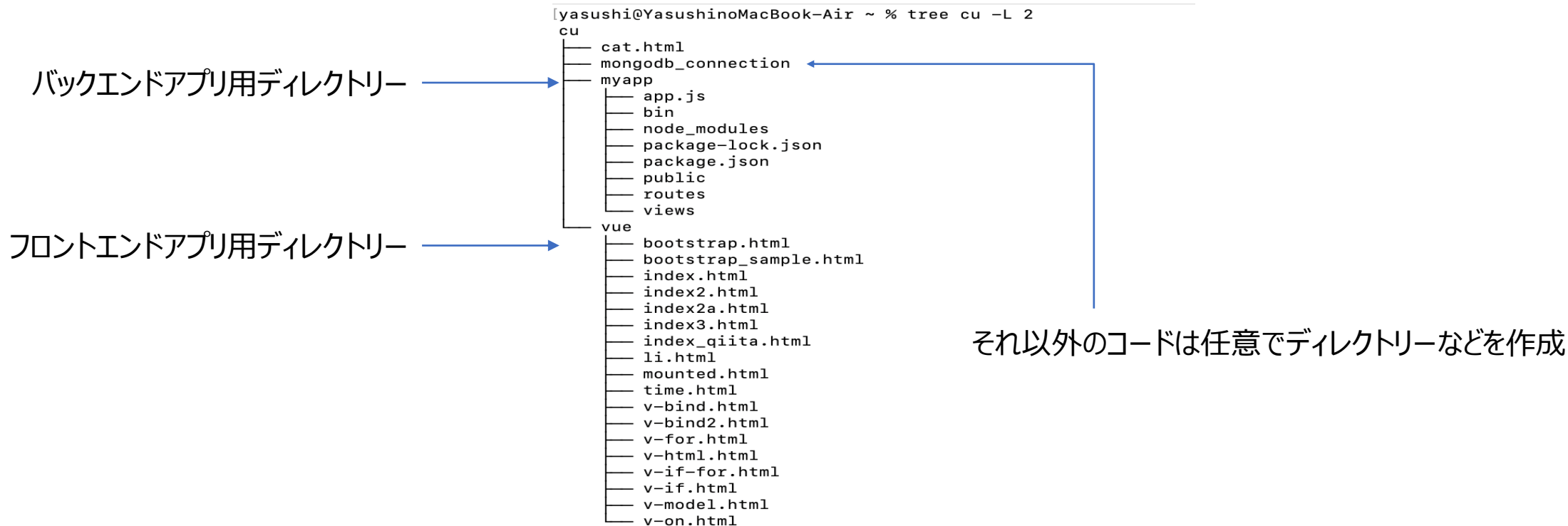
- GitHubへのコードのプッシュ
- サーバへのログイン
- Git Pullでサーバのコードを最新に
- サーバーでアプリ起動
- Postmanで動作確認

アプリの確認のためのURLは
(自分のサーバのIPアドレス:ポート番号/notes)
例: 153.120.121.157:30000/notes

実習時間

- ・ 10分程度を目安に動画を止めて前ページまでの実習をしてください。
- ・ 作業が終わったらビデオを再開して学習を進めてください。

※第1回4章で説明した通り、実習のファイル作成時には以下のディレクトリ構造を推奨しています。必要に応じて参照し、演習フォルダを整理してください。



第3章 まとめ

- REST APIについて実習した。
- Expressで作成したプログラムを編集してREST APIサービスを作成した。
- ローカル環境でREST APIの動作確認を行った。
- Postmanを使い上記の確認を行った。

第6回 まとめ

- REST APIについて学習した。
 - API・Web APIとは何か
 - SOAPとREST API
 - REST APIの定義
- Postmanについて学習した。
 - Web APIのテストクライアントサービス「Postman」の使用方法
- Npde.jsを使用したREST APIの作成方法を学習した。
 - note.jsの実装演習

JavaScriptフレームワークによるWeb開発
第6回 REST API実習

第3章 REST API実習

終わり