

1. 目標：ユーザー登録及びログイン認証アプリケーション

2.使用した技術：

表 1 課題で使用した技術

区別	技術
フロントエンド	HTML5.0
	CSS
	VUE.js
	VUE-cli
	VUE-router
バックエンド	Node.js
	Express.js
	MySQL

開発ツール：IntelliJ IDEA 2021.1.2 x64 版

バージョン管理用ツール：Git version 2.24.1.windows.2 版

3.実行例

3.1 Windows Terminal を 2 つ開き、それぞれがフロントエンドとバックエンドのアプリケーションを実行する。図 1 のように、ここでは、IntelliJ IDEA の Terminal 部を使っている。

Local タグには cd コマンドでアプリケーションの service フォルダーに開く。Local(2)タグにはアプリケーションの Root フォルダーlogin-registerに残る

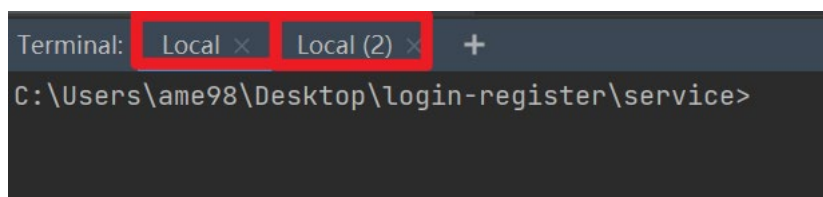


図 1 IntelliJ IDEA の Terminal 部

3.2 Local タグは「node app.js」コマンドでバックエンド部分を実行する。



図 2 バックエンド部分実行図

図2のように「success,アプリケーションが実行しました」という提示があったら、バックエンド部分の実行が成功した。

3.3 Local タグ開いたままで、Local(2)タグに「npm run serve」コマンドでフロントエンド部分を実行する。

```
C:\Users\ame98\Desktop\login-register>npm run serve

> login-register-vue-node@0.1.0 serve C:\Users\ame98\Desktop\login-register
> vue-cli-service serve

INFO Starting development server...
40% building 21/23 modules 2 active C:\Users\ame98\Desktop\login-register\no
. Please run:
npx browserslist@latest --update-db
98% after emitting CopyPlugin

DONE Compiled successfully in 2042ms

App running at:
- Local: http://localhost:8080/
- Network: http://192.168.0.151:8080/

Note that the development build is not optimized.
To create a production build, run npm run build.
```

図3 フロントエンド部分実行図

図3のように「DONE Compiled successfully in xxx ms」という提示があったら、フロントエンド部分の実行が成功した。

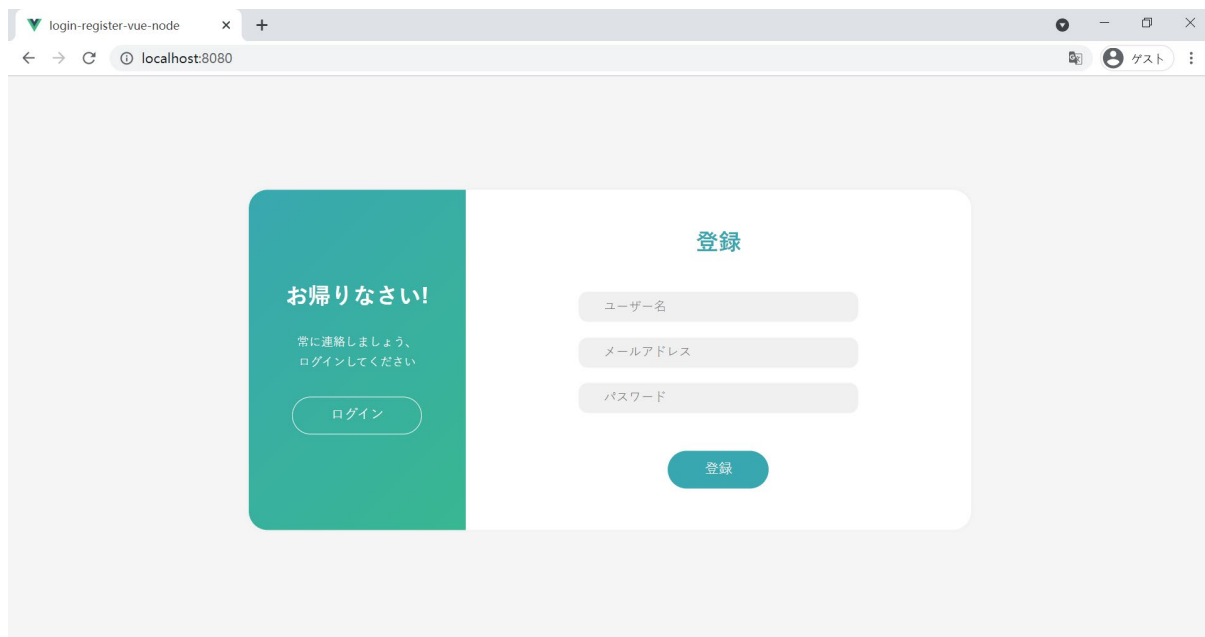


図4 ユーザー登録画面

3.4 図4のように、ブラウザで <http://localhost:8080/> でローカルサーバーにアクセスする。ローカルエリアネットワーク（LAN）のアクセスアドレスは <http://192.168.0.151:8080/> である。

3.5 ユーザー登録機能具体に実行する。図5のように、データを入力して、「登録」ボタンを押す。

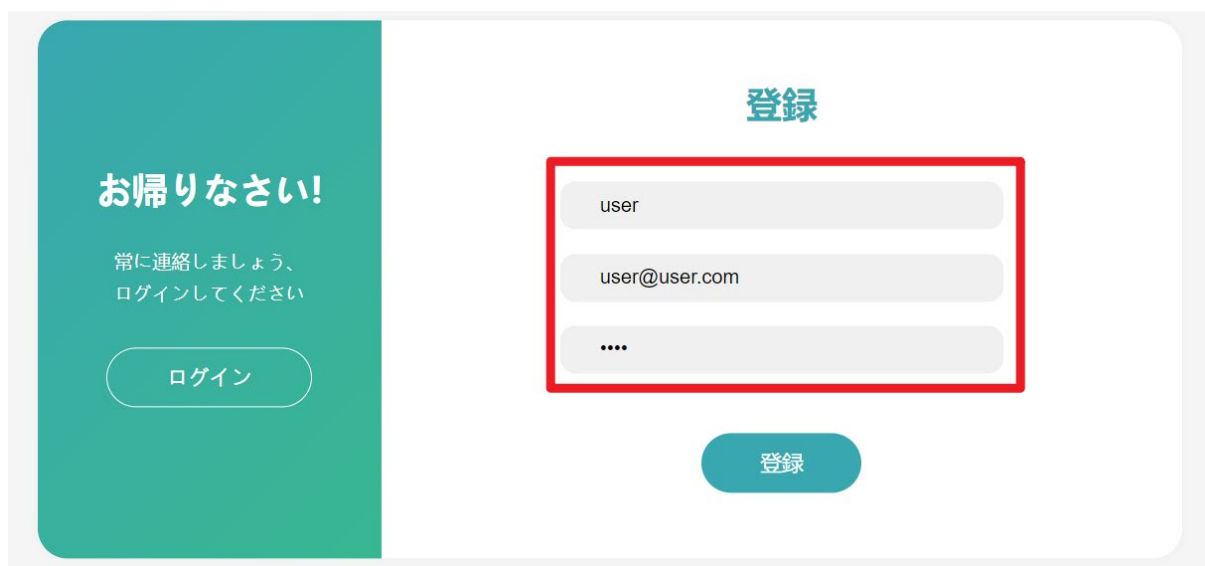


図5 ユーザー登録例



図 6 登録とログインが成功

```

select * from user where username = 'user'
OkPacket {
  fieldCount: 0,
  affectedRows: 1,
  insertId: 0,
  serverStatus: 2,
  warningCount: 0,
  message: '',
  protocol41: true,
  changedRows: 0
}
select * from user where email = 'user@user.com'
[
  RowDataPacket {
    username: 'user',
    password: 'user',
    email: 'user@user.com'
  }
]

```

図6のように登録成功とログインが成功の内容が二回表示する。ここではユーザーが登録した後自動的にログインすることに配慮している。データがデータベースに記入したことも分かる。具体的な内容は後でコード部分に説明する。

3.6 ユーザー登録ログイン具体に実行する。図7のように左側の「ログイン」ボタンを押すと、ログインの画面が表示する。同じように右側の「登録」ボタンを押すと、登録に画面の切り替えることができる。

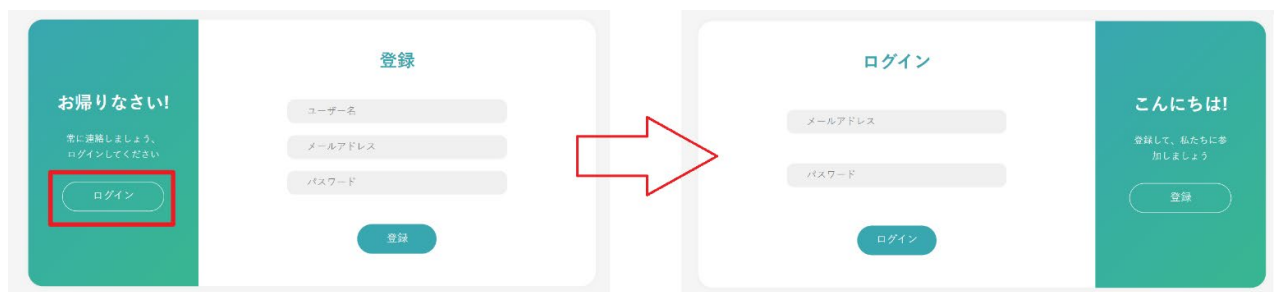


図7 ログイン画面と登録画面の切り替え

そして、図7のように先程登録したデータを正しく入力して、「ログイン」ボタンを押すと、ロ

ログインが成功の内容を提示されている。Local タグにデータがバックエンドにデータベース検索したことが分かる。

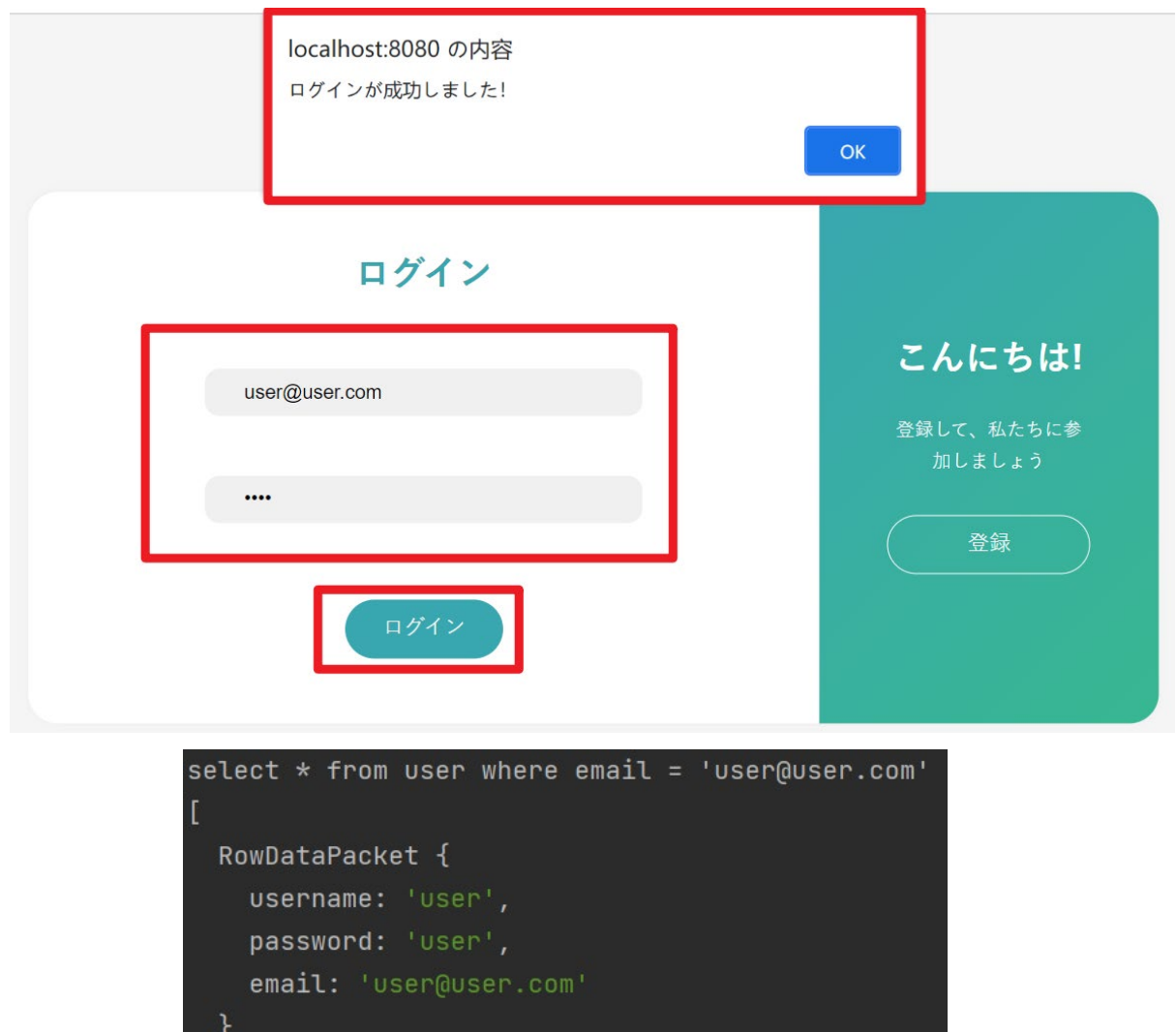


図7 ログインが成功した図

3.7 不正確な内容を入力した場合



図8 「空欄に記入」の要求した図

まず、空欄した場合は「空欄に記入してください」を提示する。データは発送しない。

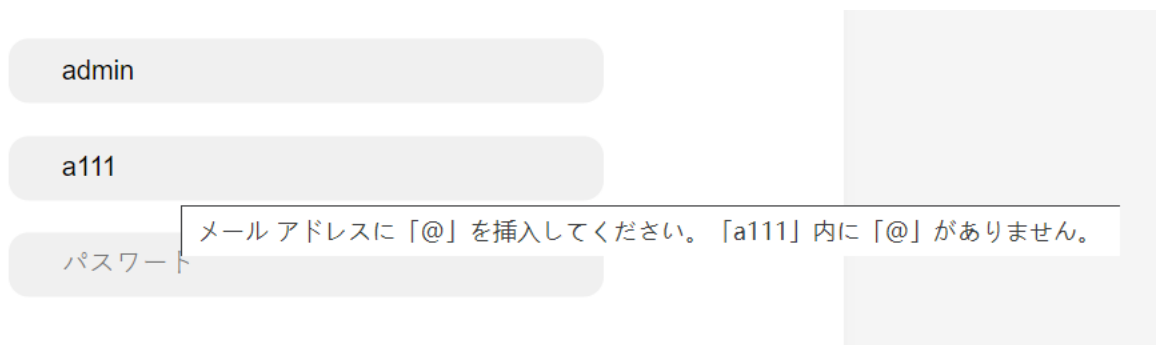


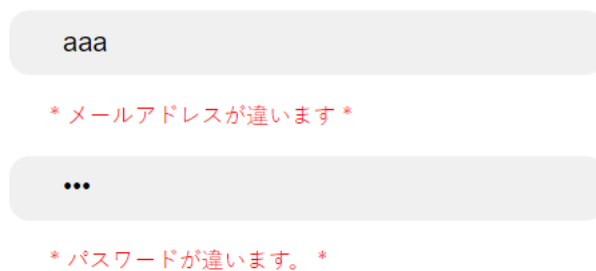
図9 「@」の要求した図

そして、アドレス欄にメールアドレスの仕様ではない場合は提示がある。しかし、「@」を要求しただけでメールアドレスの仕様に強制的に限られていない。

次は、にログインする時にメールアドレスとパスワードが一致しない場合と登録していない場合。つまりデータベースにデータがない場合。図10のよう、「メールアドレスとパスワード違い」提示している。データは発送する。図11のよう、Local タグにデータがバックエンドにデータベ

ース検索したことが分かる。

ここでは簡単に処理しているが、実際に企業級アプリケーション開発にはより複雑に処理すべき。



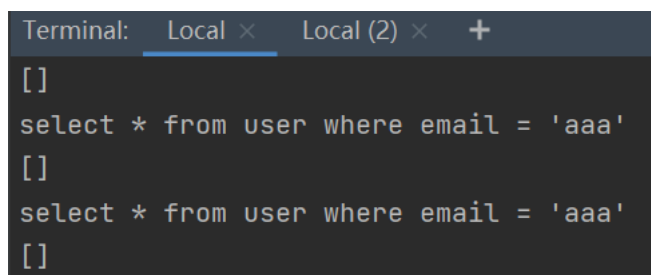
aaa

* メールアドレスが違います *

...

* パスワードが違います。*

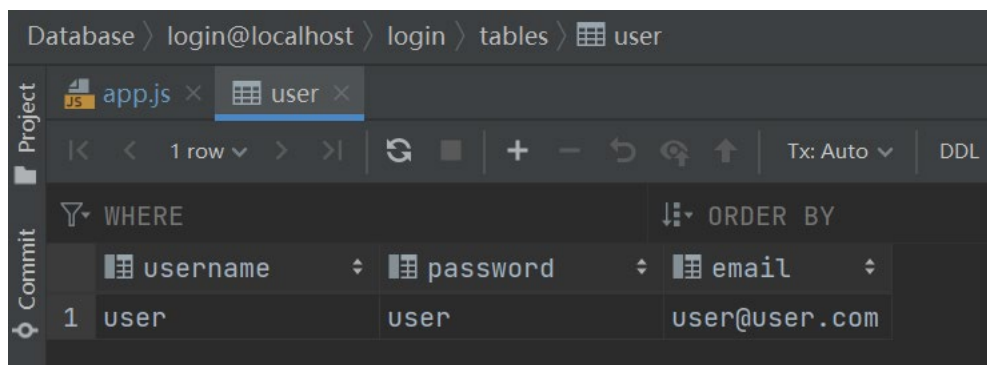
図 10 「メールアドレスとパスワード違い」提示



```
Terminal: Local × Local (2) × +
[]
select * from user where email = 'aaa'
[]
select * from user where email = 'aaa'
[]
```

図 11 Local タグでデータベース検索した

データベースの部分を見ると、データベースにログインで入力したデータがない場合は全てがログイン失敗として扱う。図 12 は IntelliJ IDEA のデータベース機能でデータベース「user」の一覧である。



Database > login@localhost > login > tables > user			
Project	app.js	user	
Commit			
	username	password	email
1	user	user	user@user.com

図 12 データベース表の一覧

ここまでは、アプリケーションのユーザー登録及びログイン認証機能が簡単に説明した。

4. コード説明

4.1 プロジェクトの一覧から説明する。全てのファイルから、重要な3つのフォルダー「public」「service」「src」がある。「tests」はVUEプロジェクトフォルダー化のまま、今度のレポートでは使わないので、重要なフォルダー「public」「service」「src」が説明する。

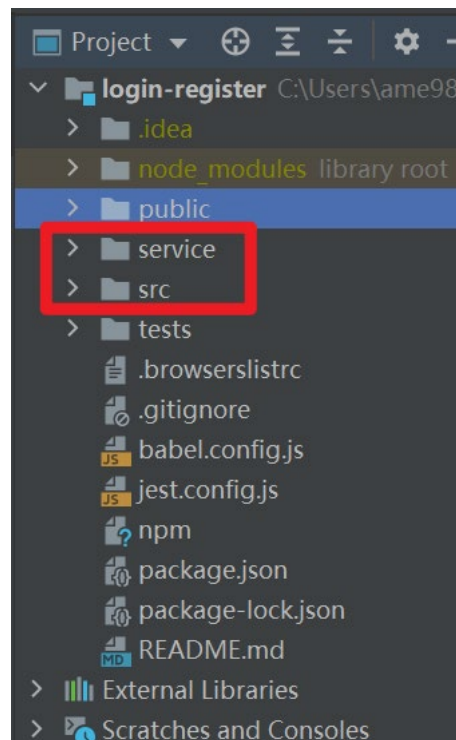


図 13 プロジェクトの一覧

4.2 「public」フォルダーで、ICON は VUE のデフォルト ICON である。「index.html」には「div id = “app”」が重要である。VUE がビルドファイルは自動的に注入されることになる。CSS ファイルは「<http://meyerweb.com/eric/tools/css/reset/>」からダウンロードした HTML5.0 の様式リセット CSS である。

```
<body>
  <div id="app"></div>
  <!-- built files will be auto injected -->
</body>
```

図 1 4 index.html の内容

4.3 ここからバックエンドの部分を説明する。「service」フォルダーの中には2つフォルダーがある。それぞれがフロントエンドに提供する「API」部分とデータベースに繋ぐ「Database」部分である。

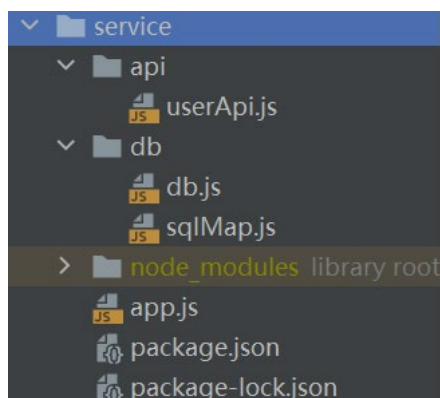


図 15 「service」フォルダーの一覧

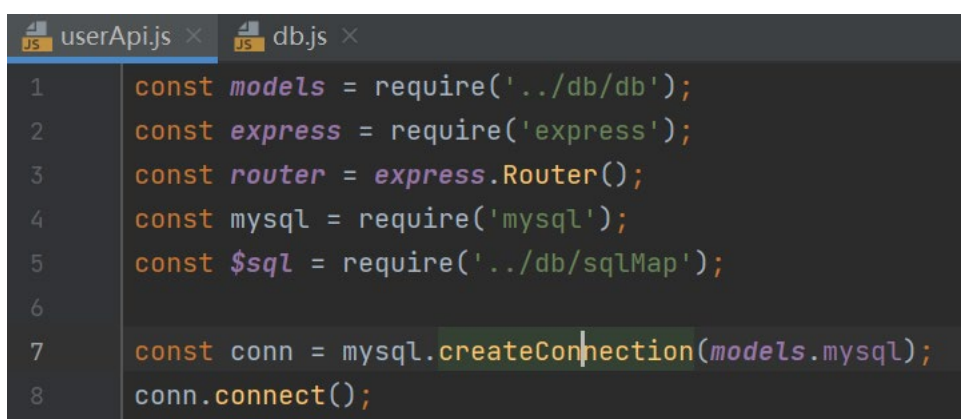


図 15 userApi.js 1 - 8 行

まずは API の部分にデータベースのモデルを導入する。図 1 5 のように、1 行目は db.js を導入した、同じように、5 行目は sqlMap.js を導入した。

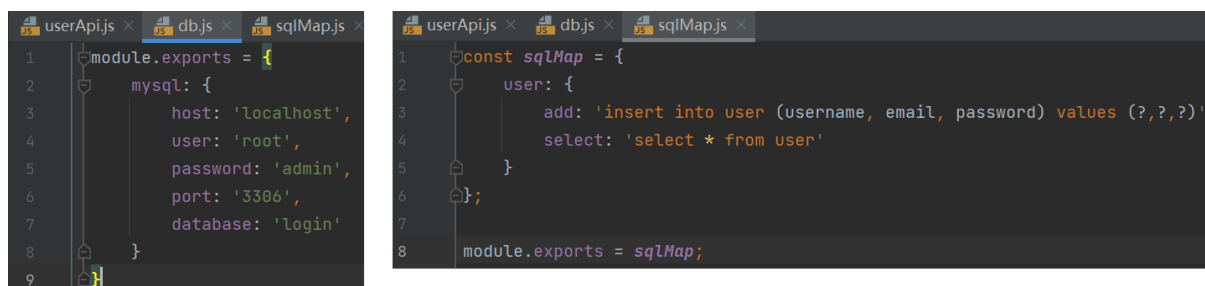


図 16 db.js と sqlMap.js

db.js ではデータベース管理システム Mysql に関するデータを保存されている、それ

ぞれが 1. Host アドレス

2. データベース管理システム Mysql のユーザー

3. データベース管理システム Mysql のユーザーのパスワード

4. 開放するポート

5. つながるデータベース名

sqlMap.js では、一つの User オブジェクトを提供している。図 1 6 右図の 2 - 5 行はその内容である。オブジェクトは 2 つのサブ名前空間を使って、それぞれが String 型の MySQL の add コマンドと Select コマンドである。データベースにデータの追加とデータ全表示の機能に定義するになっている。

それでは、userApi.js に戻る。図 1 5 では 1 行目と 5 行目は自分書いた js ファイルを導入したが、2 行目と Node モデルを導入している。2 行目は Node の express モデル、4 行目はと Node の MySQL モデルである。

7 行目と 8 行目はデータベース管理システム MySQL に接続を意味している。db.js からのデータを create Connection 関数に入れ、そして、connect 関数を使って、データベース管理システム MySQL との接続ができた。

3 行目は express. Router()関数を使っている。

```

// Login API
router.post( path: '/login', handlers: (req : Request<P, ResBody, ReqBody, ReqQuery, Locals> , res : Response<ResBody, Locals> )=>{
  const user = req.body;
  const sel_email = $sql.user.select + " where email = '" + user.email + "'";
  console.log(sel_email);
  conn.query(sel_email, user.email, (error, results)=>{
    if (error) {
      throw error;
    }
    console.log(results)
    if (results[0] === undefined) {
      res.send( body: "-1"); // -1 クエリが利用できない、ユーザーが存在しない、つまりメールボックスが正しく入力されていない
    } else{
      if (results[0].email == user.email && results[0].password == user.password) {
        res.send( body: "0"); // 0 ユーザーが存在し、メールパスワードが正しい
      } else{
        res.send( body: "1"); // 1 ユーザーは存在するが、パスワードが間違っている
      }
    }
  })
});

```

図 17 userApi.js の Login API 部分

userApi.js に残ったコードが 2 つの部分に分けている。まずは Login API 部分から説明する。

router.post の中が Path と大きなアロー関数式 Handler になっている。関数式の中では、リクエストとレスポンスの引数が必要となる。1 3 行目が sqlMap.js で導入した Select コマンドと 1 2 行目からのリクエスト体の email というパラメータと構成されている。

8 行目と似ているように、1 5 行目はクエリ関数を使って、Node の MYSQL モデルに入れると、結果またはエラーの内容が分かる。2 0 行から 2 8 行まではその内容とエラーを分析している。そして常に console に出力して、ブラウザにレスポンスを送送する。

表 2 LoginAPI リスponsコードと意味

リスponsコード	意味
-1	クエリが利用できない、ユーザーが存在しない、つまりメールボックスが正しく入力されていない
0	ユーザーが存在し、メールパスワードが正しい
1	ユーザーは存在するが、パスワードが間違っている

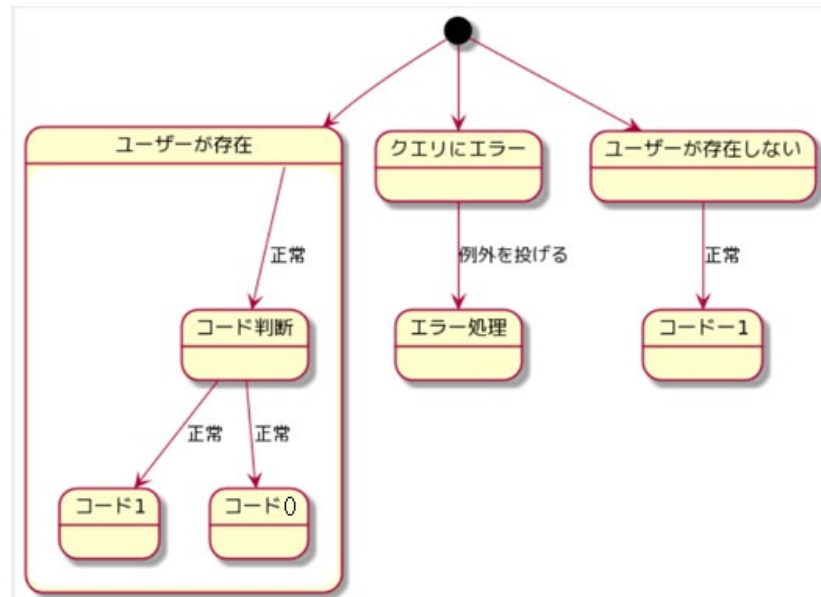


図 18 Login API 部分の状態遷移図

図 18 が Login API 部分の状態遷移図になる。表 2 からレスポンスコードと意味が分かる。

```

// register API
router.post( path: '/add', handlers: (req : Request<P, ResBody, ReqBody, ReqQuery, Locals> , res : Response<ResBody, Locals> ) => {
  const params = req.body;
  const sel_sql = $sql.user.select + " where username = '" + params.username + "'";
  const add_sql = $sql.user.add;
  console.log(sel_sql);

  conn.query(sel_sql, params.username, (error, results) => {
    if(error) {
      console.log(err);
    }
    if (results.length != 0 && params.username == results[0].username) {
      res.send( body: "-1"); // -1 ユーザー名がすでに存在していることを示す
    } else {
      conn.query(add_sql, [params.username, params.email, params.password], (err, rst) => {
        if (err) {
          console.log(err);
        } else {
          console.log(rst);
          res.send( body: "0"); // 0 ユーザー登録が成功
        }
      });
    }
  });
});

```

図 18 userApi.js の register API 部分

次に register API 部分を説明する。Login API 部分と同じように、router.post の
 中が Path と大きなアロー関数式 Handler になっている。関数式の中では、リクエスト
 とレスポンスの引数が必要となる。リクエスト体から username パラメータを取り、

sqlMap からの String 型の文字と構成している。

39 行から、54 行までは登録する時の判断である。そして、レスポンスで発送する。表

3 では register API でレスポンスコードと意味が示している。

登録

user

* 登録済みのユーザー名は登録できません。*

user@user.com

....

登録

図 19 登録失敗の例

表3 register API リスponseコードと意味

レスポンスコード	意味
-1	ユーザー名がすでに存在していることを示す
0	ユーザー登録が成功

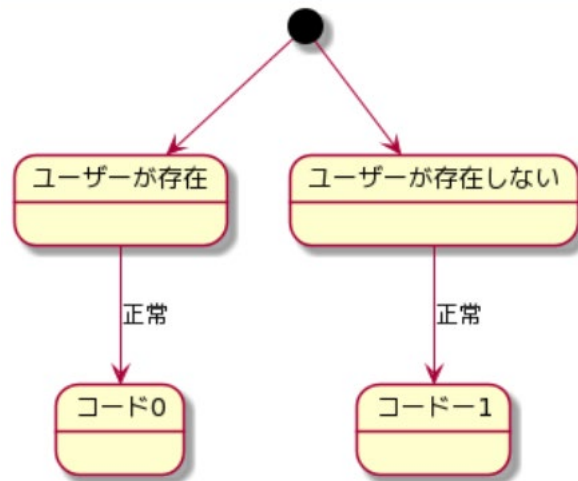


図20 register API 部分の状態遷移図

図18がregister API 部分の状態遷移図になる。表3からレスポンスコードと意味が分かる。

```

58 module.exports = router;
  
```

図21 モデル導出

最後には書いたrouter部分をモデルとして導出する。その後、app.jsでは使う。

```

app.js
1  const bodyParser = require('body-parser');
2  const express = require('express');
3  const cors = require('cors')
4  const app = express();
5
6  const userApi = require('./api/userApi.js');
7
25 app.use("/api/user",userApi);
26
27 app.listen( port: 10520);
28 console.log("success,アプリケーションが実行しました" );
  
```

図 2.2 モデル導入と使用

App.js では主に bodyParser というモデルを使って、POST 方式 x-www-form-urlencoded の解析、json の解析が完成している。クロスドメインリクエストの設定もしている。

ここまでは、アプリケーションのバックエンドの部分が完成している。

4.4 フロントエンド

フロントエンドの部分は SRC フィルターにある。

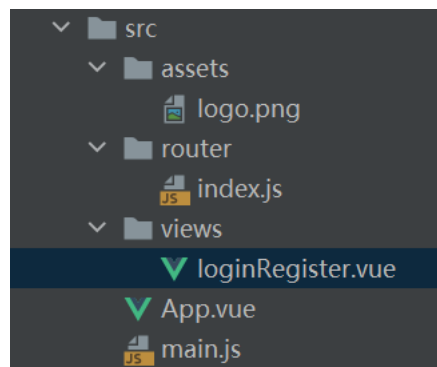
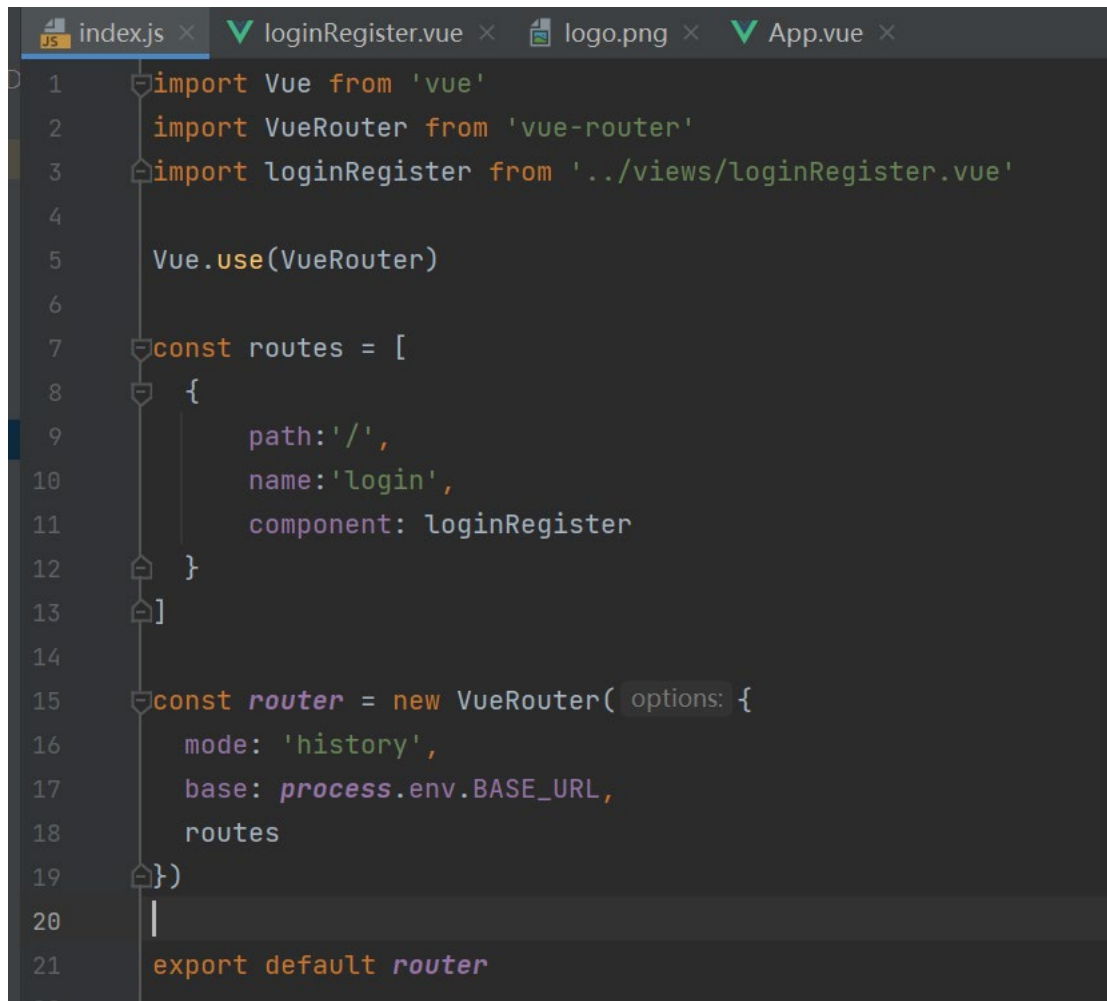


図 2.3 src フィルターの一覧

Logo.png は VUE プロジェクトのデフォルトアイコンである。これから、index.js を説明する。



```

1  import Vue from 'vue'
2  import VueRouter from 'vue-router'
3  import LoginRegister from '../views/LoginRegister.vue'
4
5  Vue.use(VueRouter)
6
7  const routes = [
8    {
9      path: '/',
10     name: 'login',
11     component: LoginRegister
12   }
13 ]
14
15 const router = new VueRouter({ options: {
16   mode: 'history',
17   base: process.env.BASE_URL,
18   routes
19 }})
20
21 export default router

```

図 24 index.js コード

まず index.js 1 行と 2 行は、Vue と Vue-router を Node モデルから導入している。この js ファイルは Vue から認識されている。そして 5 行目に Vue を Vue-router が入力パラメータと使用している。その後に routes を定義して、VueRouter に使用している。

最後は、新しい定義した Router を導出する。

見やすいように、Google Chrome の開発ツールを使って、説明する。

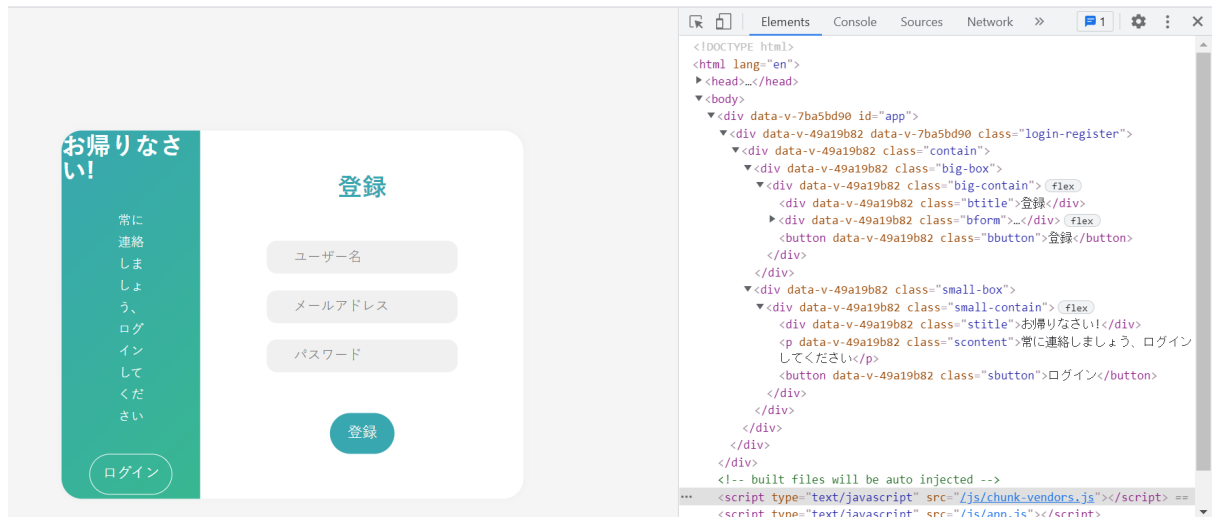


図 24 Elements コード（登録時）

図 2 4 の左側は端末が画面変更した時の様子である。右側のコードから見ると、VUE が HTML につかう全ての元素が自動的生成したことが分かる。ログインを押すと、図 2 5 のように、元素の内容が変わった。

```
<!DOCTYPE html>
<html lang="en">
<head>...</head>
<body>
  <div data-v-7ba5bd90 id="app">
    <div data-v-49a19b82 data-v-7ba5bd90 class="login-register">
      <div data-v-49a19b82 class="contain">
        <div data-v-49a19b82 class="big-box active">
          <div data-v-49a19b82 class="big-contain">
            <div data-v-49a19b82 class="btitle">ログイン</div>
            <div data-v-49a19b82 class="bform">...</div>
            <button data-v-49a19b82 class="bbutton">ログイン</button>
          </div>
        </div>
        <div data-v-49a19b82 class="small-box active">
          <div data-v-49a19b82 class="small-contain">
            <div data-v-49a19b82 class="stitle">こんにちは!</div>
            <p data-v-49a19b82 class="scontent">登録して、私たちに参加しまし
            てください</p>
            <button data-v-49a19b82 class="sbutton">登録</button>
          </div>
        </div>
      </div>
    </div>
  </div>
</body>
</html>
```

図 25 Elements コード（ログイン時）

ここまでは VUE プロジェクトでフロントエンド部分とバックエンド部分全ての説明をした。