### HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY AND EDUCATION

### FACULTY FOR HIGH QUALITY TRAINING

### 

**NEW TECHNOLOGIES IN SOFTWARE ENGINEERING**

**Bui Tuan Anh**

**Student ID**: **17110001**

**Le Phuong Nam**

**Student ID**: **17110054**

**Major**: **INFORMATION TECHNOLOGY**

**Advisor**: **LE VINH THINH, PhD.**

Ho Chi Minh City, January 2020

**Thank you**

To complete this project, we would like to express our sincere thanks to Le Vinh Thinh, PhD, who spent his valuable time and experience, personally instructed us to complete the course. Sincere thanks to the help of friends who helped us find and collect information during the project implementation. Thanks again to everyone who took the time and effort to help us complete the course. In the course of implementing the project, of course, inevitably mistakes and shortcomings, we wish teachers and friends to overlook and forgive.

Students:

**Bùi Tuấn Anh**

**Lê Phương Nam**

Contents

[HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY AND EDUCATION 1](#_Toc60653649)

[FACULTY FOR HIGH QUALITY TRAINING 1](#_Toc60653650)

[** 1](#_Toc60653651)

[1.1 An Introduction to course technology 6](#_Toc60653652)

[1.1.1 Nodejs 6](#_Toc60653653)

[- What is node.js? 6](#_Toc60653654)

[- Why node.js? 6](#_Toc60653655)

[- What Can Node.js Do? 6](#_Toc60653656)

[1.1.2 Angular 9](#_Toc60653657)

[1.2 Goal of thesis 11](#_Toc60653658)

[2.1 Usecase diagram 13](#_Toc60653659)

[2.2 Scenario 14](#_Toc60653660)

[- *Sign up* 14](#_Toc60653661)

[- *Sign in* 15](#_Toc60653662)

[- *Log out* 16](#_Toc60653663)

[- *Add task* 16](#_Toc60653664)

[- *Add list* 17](#_Toc60653665)

[- *Edit list* 17](#_Toc60653666)

[- *Delete list* 18](#_Toc60653667)

[- *Edit task* 18](#_Toc60653668)

[- *Delete task* 19](#_Toc60653669)

[3.1 Sign up 20](#_Toc60653670)

[3.2 Sign in with token,sign in with google, facebook 21](#_Toc60653671)

[3.3 Add, delete, update list 24](#_Toc60653672)

[3.4 Add, delete, update task 26](#_Toc60653673)

[3.5 Complete task 26](#_Toc60653674)

[27](#_Toc60653675)

[3.6 Filter list, task 27](#_Toc60653676)

[27](#_Toc60653677)

Chapter 1: INTRODUCTION

# An Introduction to course technology

## Nodejs

## What is node.js?

+ Node.js is an open-source and cross-platform JavaScript runtime environment. It is a popular tool for almost any kind of project.

## Why node.js?

+ Node.js eliminates the waiting, and simply continues with the next request.

+ Node.js runs single-threaded, non-blocking, asynchronously programming, which is very memory efficient.

## What Can Node.js Do?

+ Node.js can generate dynamic page content

+ Node.js can create, open, read, write, delete, and close files on the server

+ Node.js can collect form data

+ Node.js can add, delete, modify data in your database

**MONGODB**



*Figure 1.1: MongoDB introduction.*

* + MongoDB is a NoSQL database.
  + NoSQL is a generation of databases whose main characteristics are
    - Non-relational
    - Distributed
    - Open source (open source)
    - Horizontal scalable can store and process from very small amounts to extremely large data, up to petabytes of data in the system that need high load capacity, high fault tolerance with low hardware resource requirement.



*Figure 1.2: MongoDB install.*

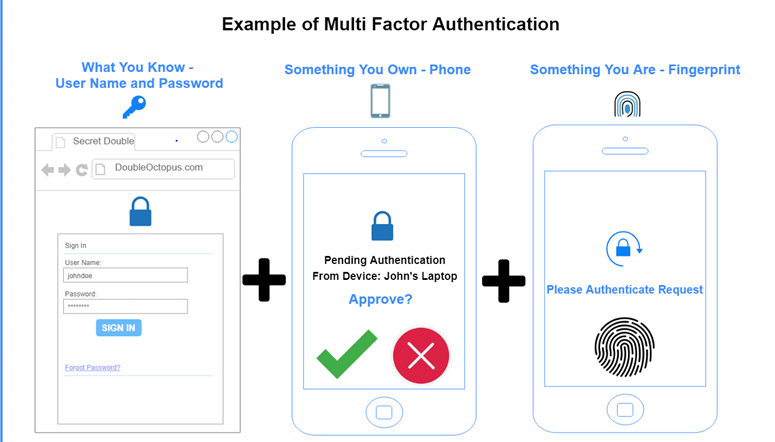
**EXPRESS**



*Figure 1.3: Express introduction.*

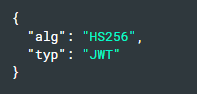
* + ExpressJS is a web application framework for NodeJS.
  + Express is the most popular Node web framework, and is the underlying library for a number of other popular Node web frameworks. It provides mechanisms to:
    - Write handlers for requests with different HTTP verbs at different URL paths (routes).
    - Integrate with "view" rendering engines in order to generate responses by inserting data into templates.
    - Set common web application settings like the port to use for connecting, and the location of templates that are used for rendering the response.
    - Add additional request processing "middleware" at any point within the request handling pipeline.

**NODE AUTHENTICATION**



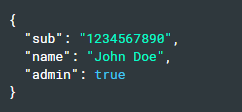
*Figure 1.4: E xample of muti factor authentication*

* + JSON Web Token
    - Header : The header typically consists of two parts: the type of the token, which is JWT, and the signing algorithm being used, such as HMAC SHA256 or RSA.

****

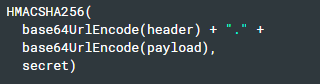
*Figure 1.5: Header example*

* + - Payload : The second part of the token is the payload, which contains the claims. Claims are statements about an entity (typically, the user) and additional data. There are three types of claims: registered, public, and private claims.



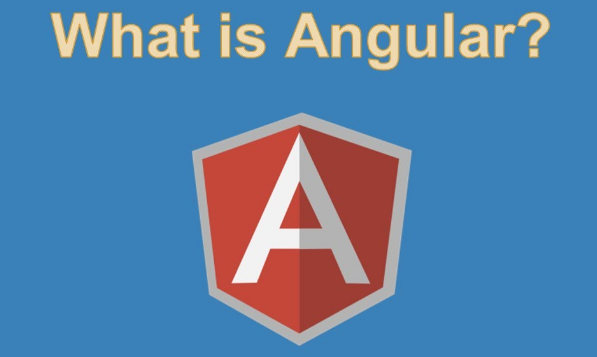
*Figure 1.6: Payload example*

* + - Signature ; To create the signature part you have to take the encoded header, the encoded payload, a secret, the algorithm specified in the header, and sign that.



*Figure 1.7: Signature example*

## Angular

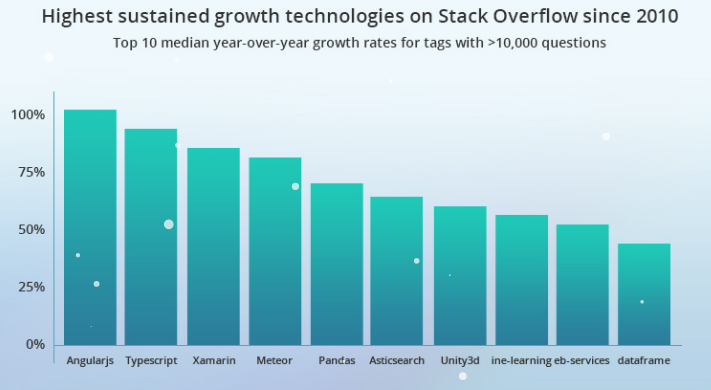


*Figure 1.8: What is angular?*

* Angular is a platform and framework for building single-page client applications using HTML and TypeScript. Angular is written in TypeScript. It implements core and optional functionality as a set of TypeScript libraries that you import into your apps.
  + **Modules**
    - An NgModule declares a compilation context for a set of components that is dedicated to an application domain, a workflow, or a closely related set of capabilities. An NgModule can associate its components with related code, such as services, to form functional units.
  + **Components**
    - Every Angular application has at least one component, the root component that connects a component hierarchy with the page document object model (DOM). Each component defines a class that contains application data and logic, and is associated with an HTML template that defines a view to be displayed in a target environment.
  + **Templates, directives, and data binding**
    - A template combines HTML with Angular markup that can modify HTML elements before they are displayed. Template directives provide program logic, and binding markup connects your application data and the DOM. There are two types of data binding:
      * Event binding lets your app respond to user input in the target environment by updating your application data.
      * Property binding lets you interpolate values that are computed from your application data into the HTML.
  + **Services and dependency injection**
    - A service class definition is immediately preceded by the @Injectable() decorator. The decorator provides the metadata that allows other providers to be injected as dependencies into your class.
    - Dependency injection (DI) lets you keep your component classes lean and efficient.
  + **Routing**
    - The Angular Router NgModule provides a service that lets you define a navigation path among the different application states and view hierarchies in your app. It is modeled on the familiar browser navigation conventions:
      * Enter a URL in the address bar and the browser navigates to a corresponding page.
      * Click links on the page and the browser navigates to a new page.
      * Click the browser's back and forward buttons and the browser navigates backward and forward through the history of pages you've seen.

# Goal of thesis

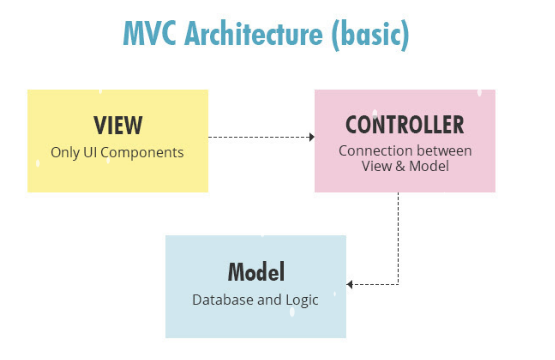
* In recently, angular rapidly develop from AngularJS in 2010 to Angular 5 in 2017 and more. To every developer understand and access the new technology



*Figure 1.9: Highest sustained growth technologies*

* Why we choose angular
  + Supported by Google : the biggest advantage of Angular is that it is supported by Google. The best part about it is Google’s Long-Term Support (LTS).

* + TypeScript: With TypeScript, you can easily take the existing ES5 or ES2015+ JS code and it will compile it down based on what you are configuring. It fully supports core ES2015 and ES2016/ES2017 features such as decorators or async/await.
  + Declarative UI; Angular uses HTML to define the UI of the application. HTML, as compared to JavaScript, is a less convoluted language. HTML is also a declarative and intuitive language.
  + Easy Testing: Angular.js modules has the application parts, which are easy to manipulate. With module separation, you can load the necessary services, while effectively performing automatic testing.
  + Simplified MVC Pattern

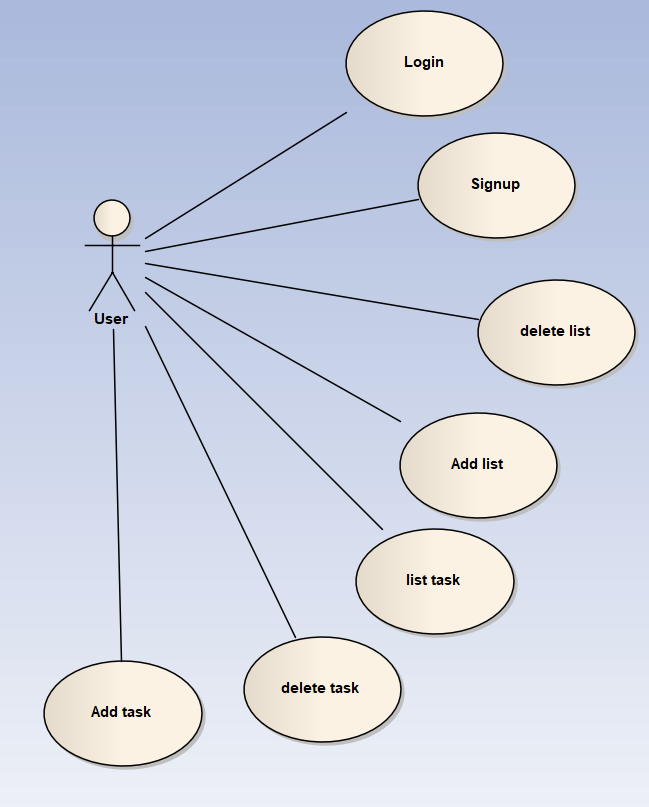


*Figure 1.10: MVC model*

* + Code Consistency
    - Reusability
    - Improved Readability
    - Ease of Maintenance

Chapter 2: REQUIREMENTS SPECIFICATION

# Usecase diagram



*Figure 2.1: Usecase Diagram*

# Scenario

# Sign up

|  |  |
| --- | --- |
| Name | Signup |
| Brief Description | User signup for an account to the system |
| Actor(s) | User has not logged in |
| Basic Flow | The use case starts when the user needs to perform certain functions of the system that need to register an account  1. The system displays a window for the user to enter the username and password  2. User enters username and password.  3. System verifies Username and password.  4. The system see if the username already exists.  5. The system opens the login screen. |
| Alternative Flow | If the user enters a username that is already existed. The following tasks were performed:  1. The system will prompt the user to enter username and password again.  3. When the user re-enters the username and password, step 3 in Basic flow is performed again. |
| Pre-conditions | (none) |
| Post-condition |  |
| Success | A new account is added to the system. |
| Failure | The system is not changed. |
| Extension Point |  |
| (none) |  |

# Sign in

|  |  |
| --- | --- |
| Name | Login |
| Brief Description | User login into the system |
| Actor(s) | Users have not logged in |
| Basic Flow | The use case starts when the user needs to perform certain functions of the system that need to authenticate access  1. The system displays a window for the user to enter the username and password  2. User enters username and password  3. System verifies Username and password.  4. The system determines the user's role  5. The system opens the home screen and opens the functions that are accessible to the user. |
| Alternative Flow | If the user enters the wrong username or / and password. The following tasks were performed:  1. The system describing the reason for authentication is wrong  2. The system will prompt the user to login again  3. When the user re-enters the username and password, step 3 in Basic flow is performed again. |
| Pre-conditions | The user already has registered a account before. |
| Post-condition |  |
| Success | The user is authenticated and the system shows the homepage content |
| Failure | User can not access to system |
| Extension Point |  |
| (none) |  |

# Log out

|  |  |
| --- | --- |
| Name | Logout |
| Brief Description | User log out of the system |
| Actor(s) | User has logged in |
| Basic Flow | The use case starts when the user is in the home page  1. User will se a button with the text “logout”  2. User click logout button  3. The system opens the login  4.The usecase ends |
| Alternative Flow | (none) |
| Pre-conditions | The user already has logged into the system. |
| Post-condition |  |
| Success | The user will see the login page. |
| Failure | User can not access to system |
| Extension Point |  |
| (none) |  |

# Add task

|  |  |
| --- | --- |
| Name | Add task |
| Brief Description | Add tasks to the system |
| Actor(s) | User has logged in |
| Basic Flow | The use case starts when the user is in the home page  1. User will type the name of the task  2. User will click add task button  3. New task will be displayed to the page  4.The usecase ends |
| Alternative Flow | (none) |
| Pre-conditions | The user already has logged into the system. |
| Post-condition |  |
| Success | The user will see the login page. |
| Failure | User can not access to system |
| Extension Point |  |
| (none) |  |

# Add list

|  |  |
| --- | --- |
| Name | Add list |
| Brief Description | Add tasks to the system |
| Actor(s) | User has logged in |
| Basic Flow | The use case starts when the user is in the home page  1. User will click to the button add list  2. User will see a new page  3. User will enter the task name  4. User will click the button add list  5. User should see the new list  6. User will see the home page and the usecase ends |
| Alternative Flow | (none) |
| Pre-conditions | The user already has logged into the system. |
| Post-condition |  |
| Success | The user will see the login page. |
| Failure | User can not access to system |
| Extension Point |  |
| (none) |  |

# Edit list

|  |  |
| --- | --- |
| Name | Edit list |
| Brief Description | Add lists to the system |
| Actor(s) | User has logged in |
| Basic Flow | The use case starts when the user is in the home page  1. User will click to the button edit task next to the task name  2. User will see a new page  3. User will enter a new list name  4. User will click the button edit list  5. User should see the new name of the list  6. User will see the home page and the usecase ends |
| Alternative Flow | (none) |
| Pre-conditions | The user already has logged into the system. |
| Post-condition |  |
| Success | The user will see the login page. |
| Failure | User can not access to system |
| Extension Point |  |
| (none) |  |

# Delete list

|  |  |
| --- | --- |
| Name | Delete list |
| Brief Description | Add tasks to the system |
| Actor(s) | User has logged in |
| Basic Flow | The use case starts when the user is in the home page  1. User will click to the button delete list next to the button edit task  2. User should see that the list is no longer there  3. the usecase ends |
| Alternative Flow | (none) |
| Pre-conditions | The user already has logged into the system. |
| Post-condition |  |
| Success | The user will see the login page. |
| Failure | User can not access to system |
| Extension Point |  |
| (none) |  |

# Edit task

|  |  |
| --- | --- |
| Name | Edit Task |
| Brief Description | Edit task already have in task list |
| Actor(s) | User |
| Basic Flow | 1. User login and application shows the main UI of the software  2. User choose list want to add task  3. User clicks on the edit button  4. User edit task name.  5. User clicks Save button.  6. The application show task that user edit. |
| Alternative Flow |  |
| (none) |  |
| Pre-conditions |  |
| Logined as admin | The user has been logged in. |
| Post-condition |  |
| Success | The task is successfully edited to the system |
| Failure | The exception is prompted to the user. |
| Extension Point |  |
| (none) |  |

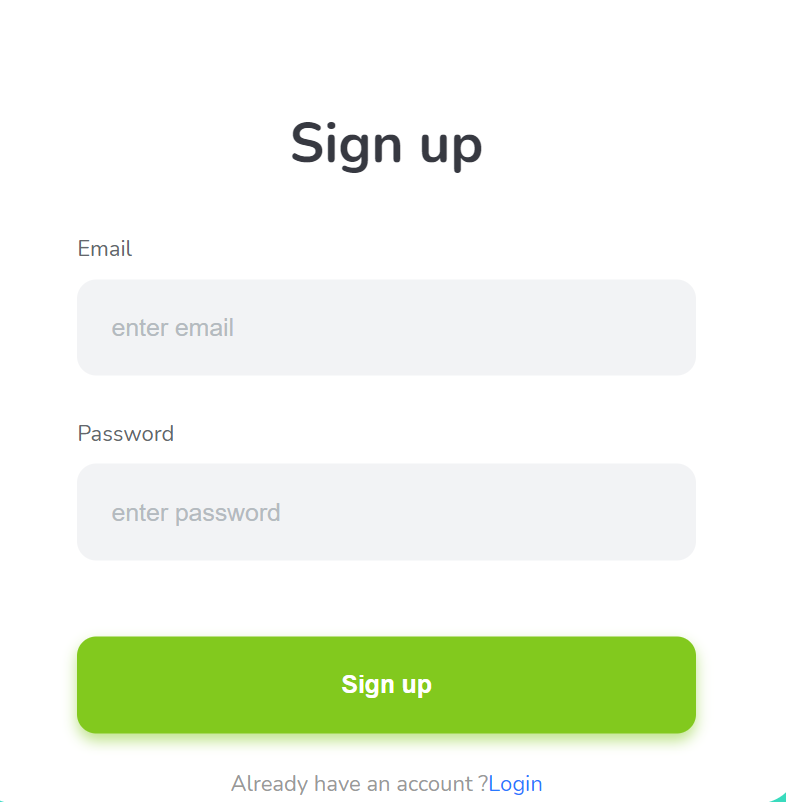
# Delete task

|  |  |
| --- | --- |
| Name | Delete Task |
| Brief Description | Delete task already have in task list |
| Actor(s) | User |
| Basic Flow | 1. User login and application shows the main UI of the software  2. User choose list want to add task  3. User clicks on the delete button  4. Task has been removed from the list |
| Alternative Flow |  |
| (none) |  |
| Pre-conditions |  |
| Logined as admin | The user has been logged in. |
| Post-condition |  |
| Success | The task is successfully deleted to the system |
| Failure | The exception is prompted to the user. |
| Extension Point |  |
| (none) |  |

Chapter 3: RESULT

# 3.1 Sign up

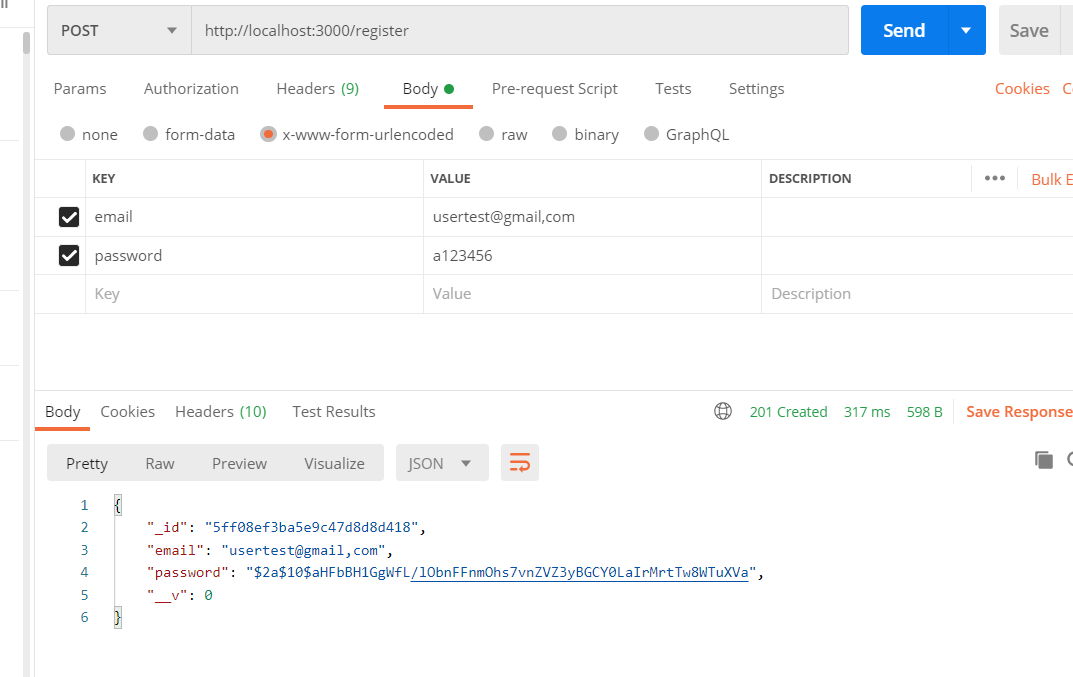
* Sign up page



*Figure 3.1: sign up page*

* Test sign up on serve using Postman

+ sign up with email : [usertest@gmail.com](mailto:usertest@gmail.com) , password : a123456

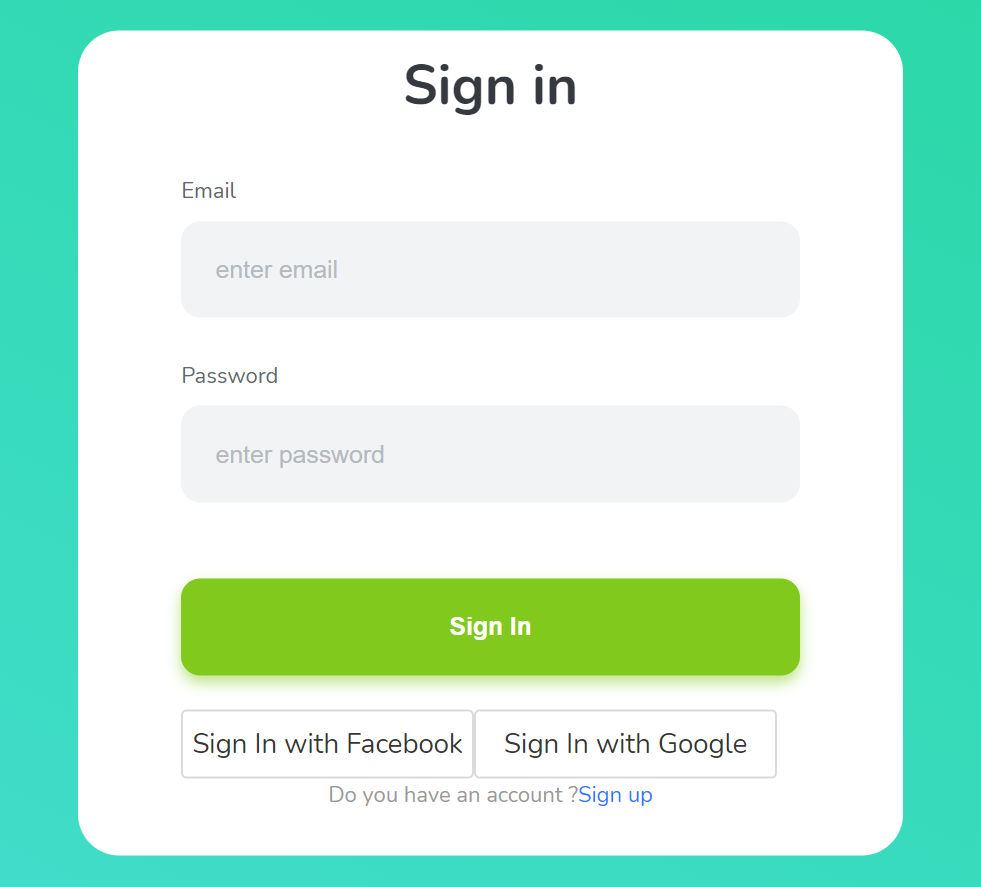


*Figure 3.2: Register using postman*

**+** After singup, website redirect to login page

# 3.2 Sign in with token,sign in with google, facebook

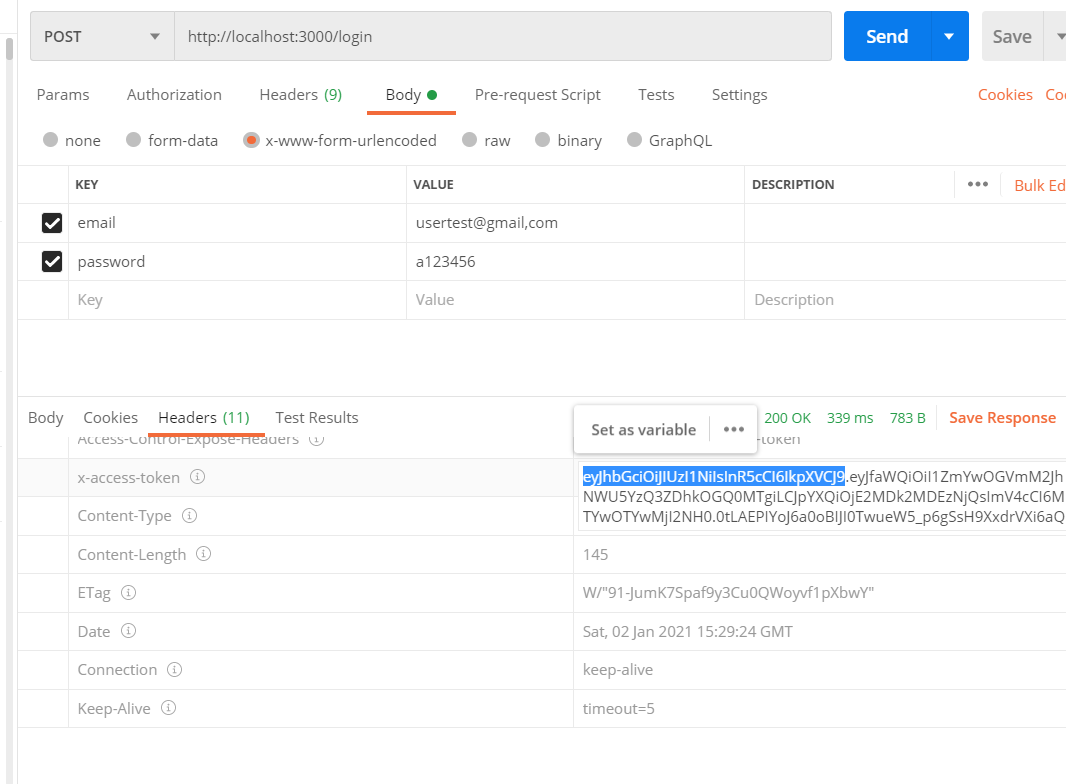
* Login page



*Figure 3.3: Login page*

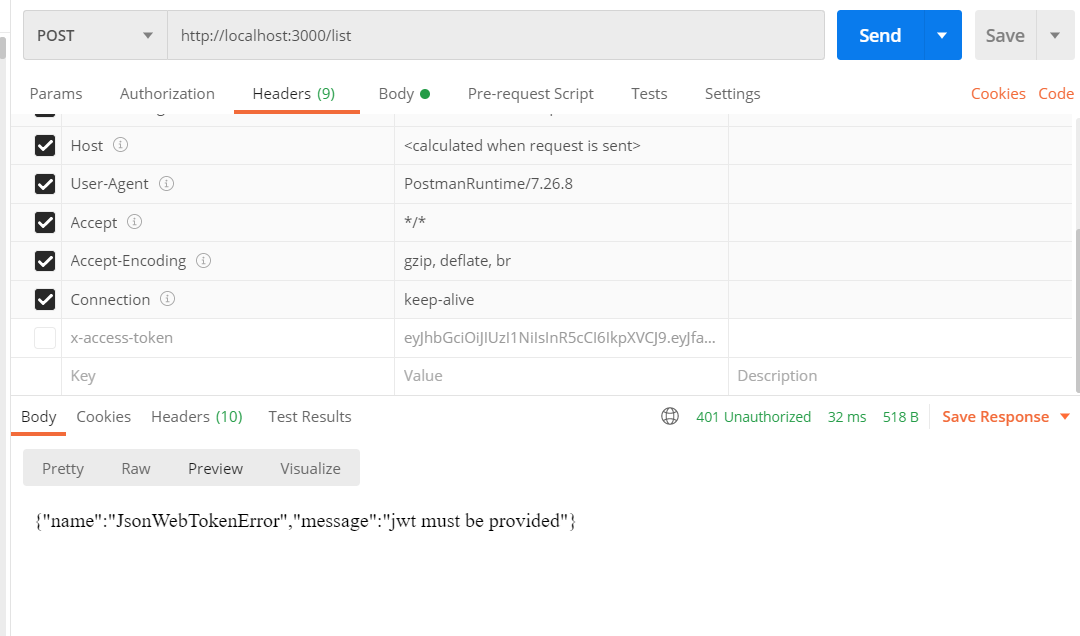
* Test login with token

Login with user already sign up



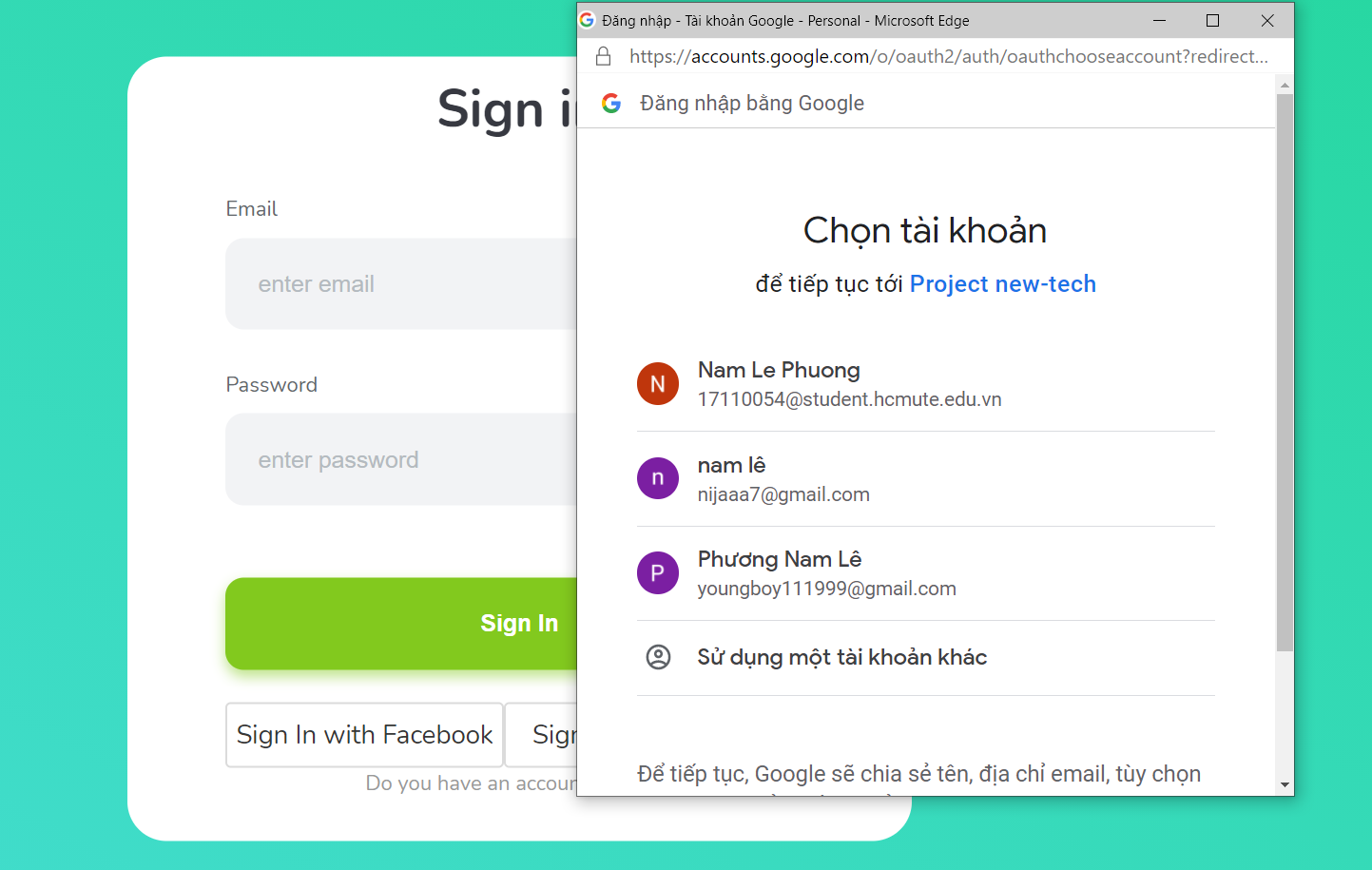
*Figure 3.4: sign in with token*

Using token veryfy



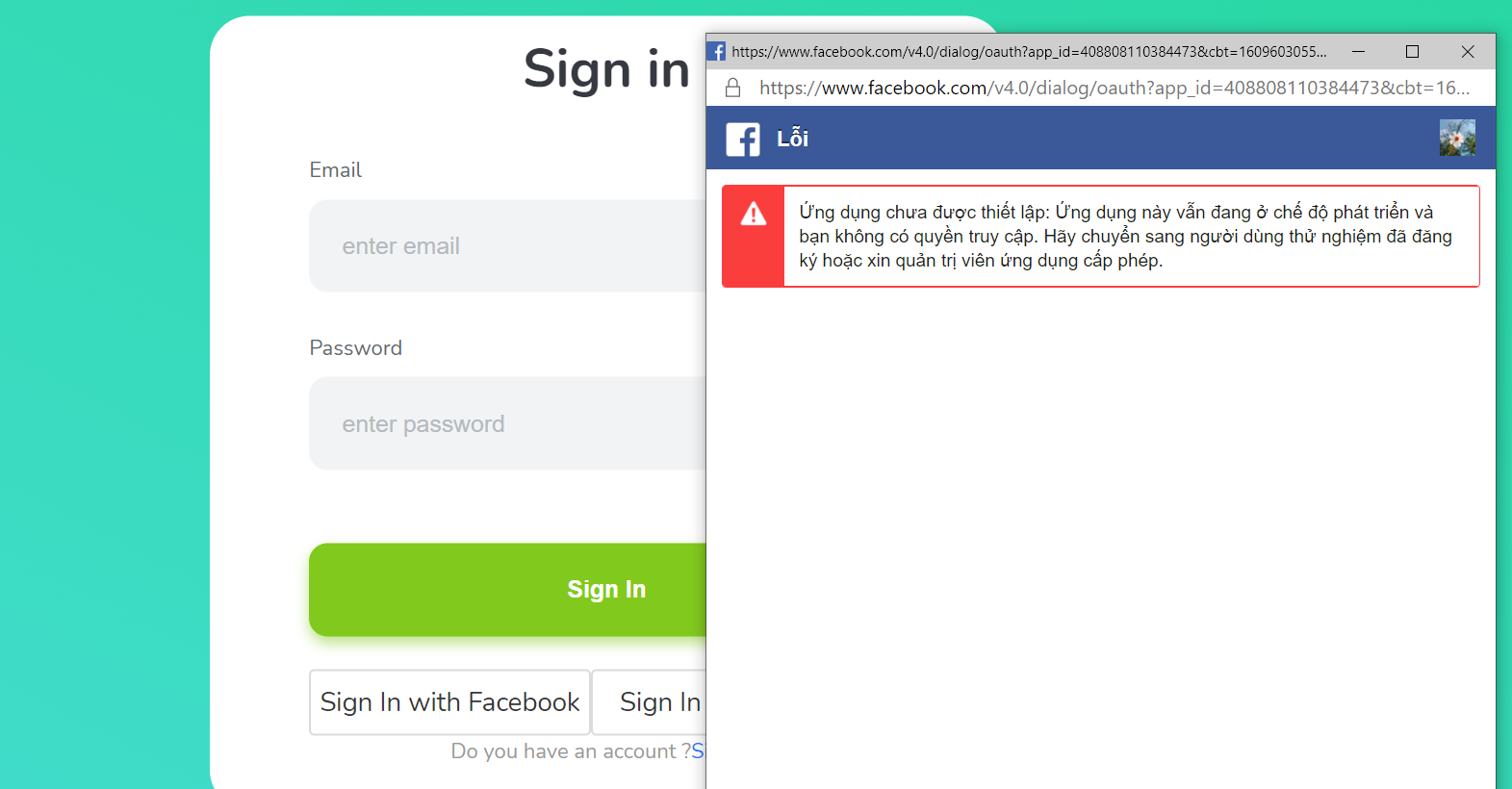
*Figure 3.5: access list page without token*

* Login with google



*Figure 3.6: login with google*

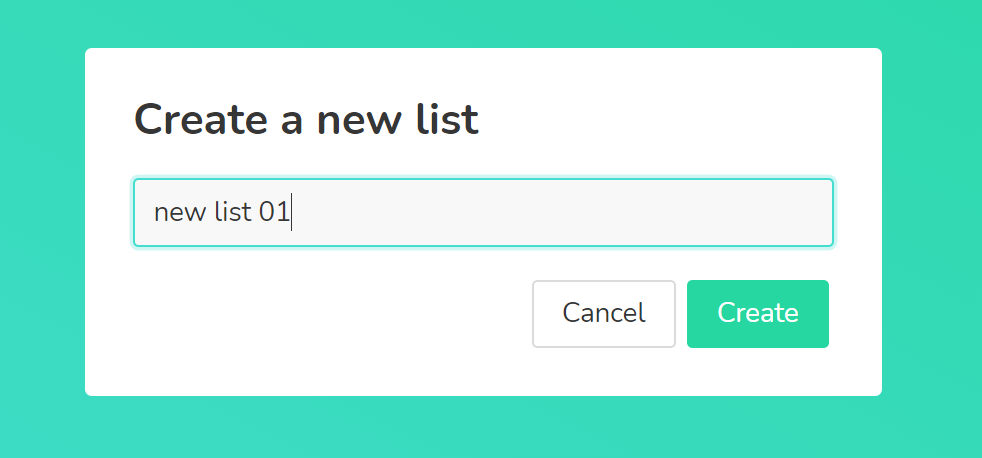
* Login with facebook



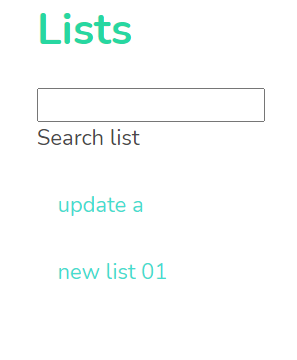
*Figure 3.7: login with facebook*

# 3.3 Add, delete, update list

* Add list



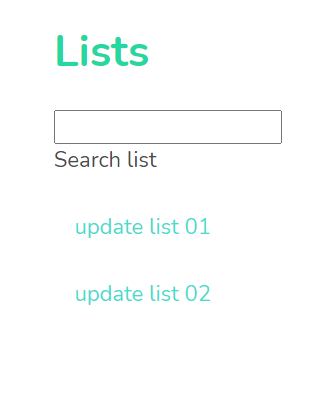
*Figure 3.8: Add new list*



*Figure 3.9: Added list*

* Update list

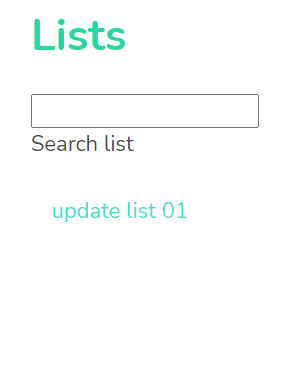
+ Click setting -> update list



*Figure 3.10: updated list*

* Delete list

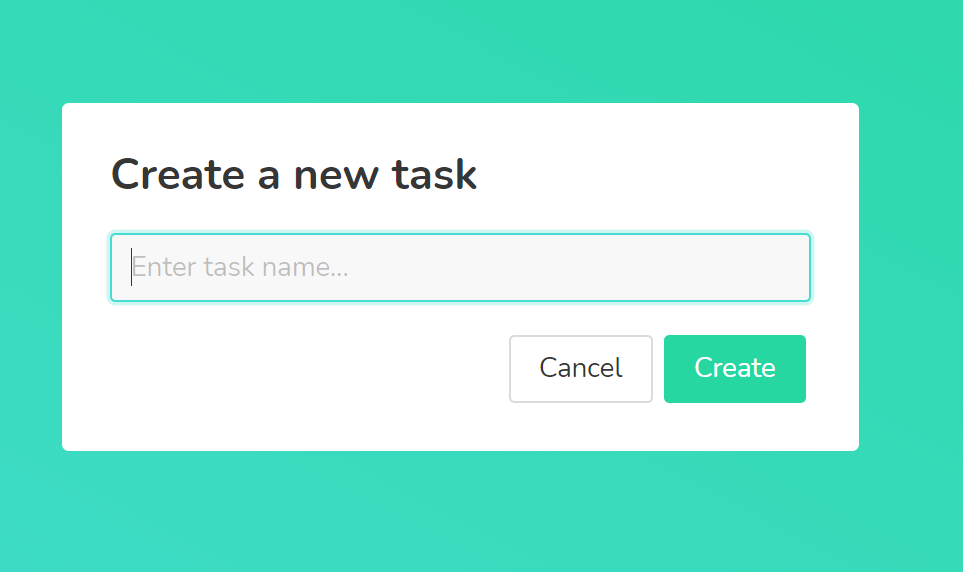
+ click setting -> delete list : delete update list 02



*Figure 3.11: deleted list*

# 3.4 Add, delete, update task

* Add task

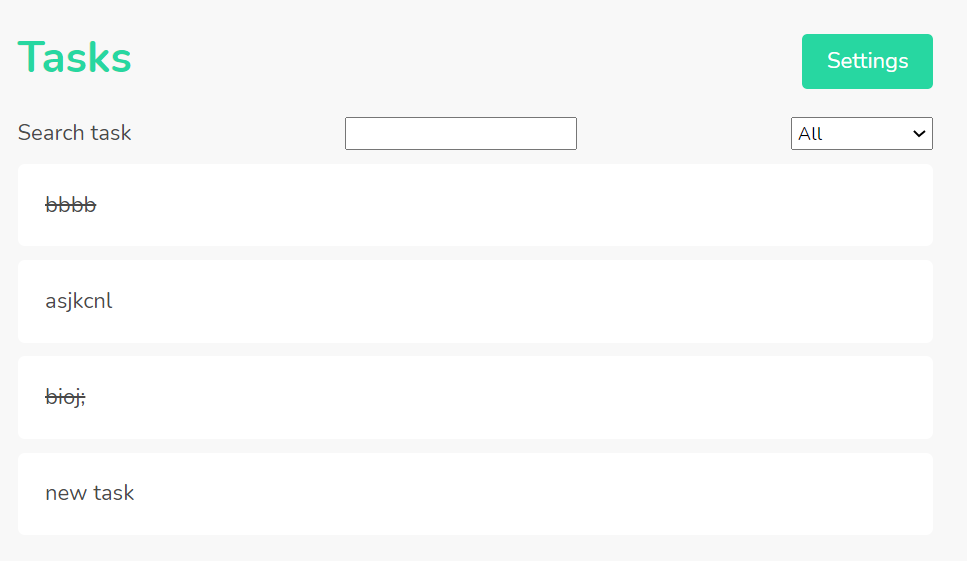


*Figure 3.12: add task*

* Delete and update
  + - Hover on task choose update or delete

# 3.5 Complete task

+ Double click on task



# 

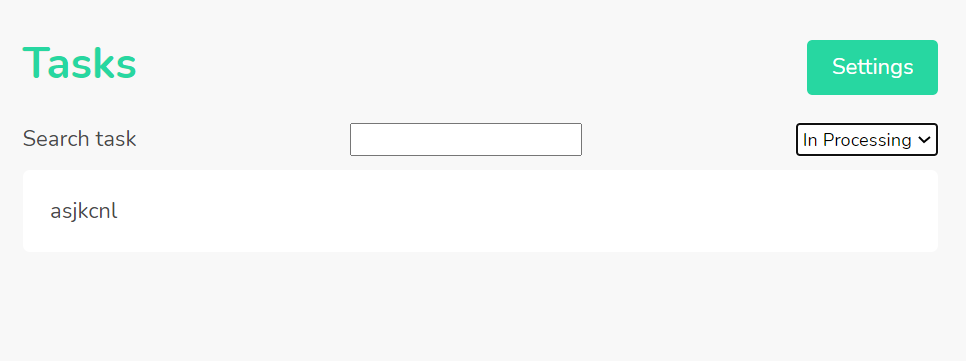
*Figure 3.13: Completed new task*

# 3.6 Filter list, task

# 

*Figure 3.14: Search list*

* Filter task follow in progessing



*Figure 3.15: Filter task follow task in processing*