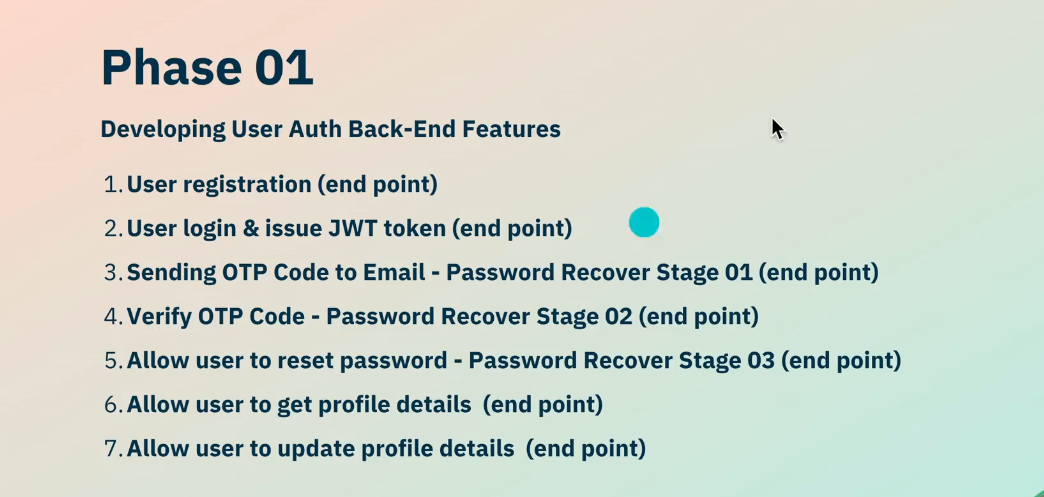
Prerecorded Video : Module 20

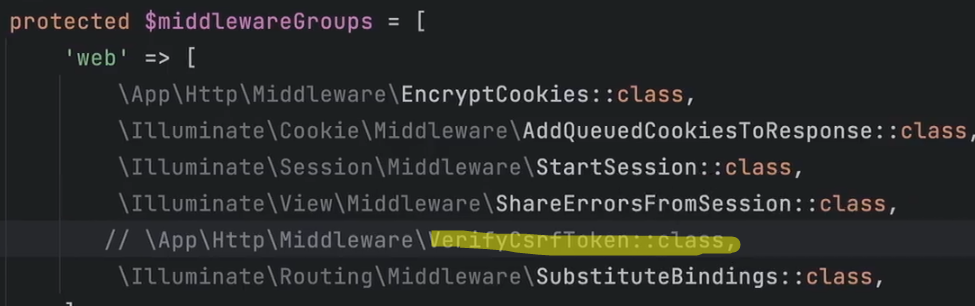
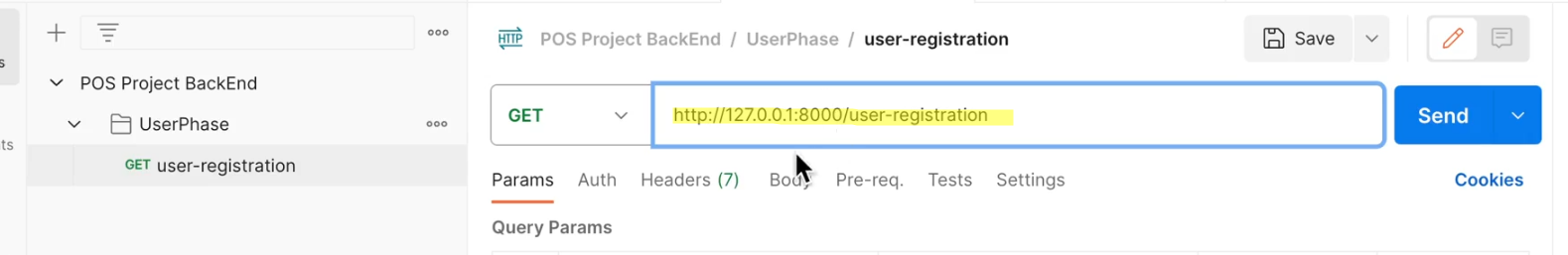
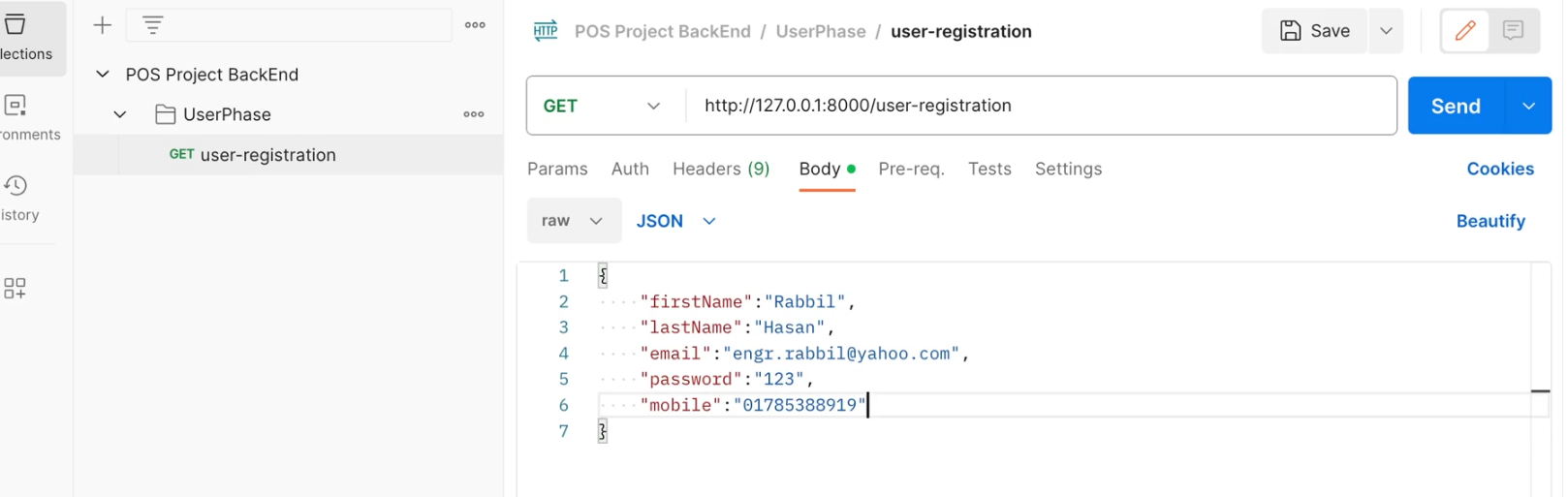
Smart Shop



User Module : 2[POS] Registration

* Create a Project smart-shop
* - composer create-project laravel/laravel smart-shop
* Edit .env File
* - change database name
* - change DB\_PORT if necessary
* Edit user Table
* - php artisan migrate
* Create UserController
* - php artisan make:controller UserController
* - write code for User Create
* Create UserModel
* - php artisan make:model User

Create User Registration API

* Route->web.php
* Route::post('/user-registration',[UserController::class,'UserRegistration']);
* Disable ->Http->kernel.php->CSRF Security (App\Http\Middleware\VerifyCsrfToken::class
* ) for testing purpose
* 
* Test API Endpoint with Postman
* Goto Postman -> Create a Collection POS Project Backend->then create a Folder User Module
* Create a New Request : user-registration
* 
* Pass json data through Body
* 
* php artisan serve – to run the server for postman testing
* Test with postman

3[POS] Registration :

4 [POS] JWT For Authentication :

* Install JWT Package
* - composer require firebase/php-jwt
* create a folder app->helper
* create a file app->Helper->JWTToken.php
* create function CreateToken(){}
* create function VerifyToken(){}
* edit .env file & write JWT\_KEY = 123456 in the end of the file

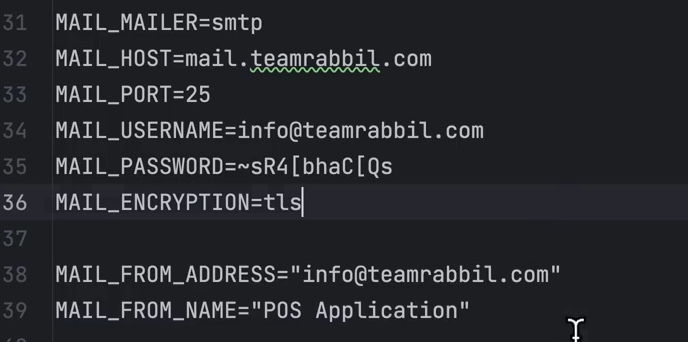
5 [POS] User Login And Issue JWT Token :

* UserController.php
* create a function UserLogin
* edit web.php & create a Route for User Login
* test with Postman
* to transfer a project from one pc to another
* - following problem may happen in .env file
* - php artisan key:generate – if no key found in .env file
* - write JWT\_KEY = 664644 in .env file
* - install composer to update vendor file

6 [POS] Working With OTP Email :

* to send an OTP we need to create a email class
* - php artisan make:mail OTPMail(mail name)
* - a mail class has been created in folder app-> mail
* create a folder naming email in resource->views folder
* - create a file name OTPmail.blade.php

7 [POS] Working With OTP Email :

* search OTP email template in google
* copy from google & paste it to OTPmail.blade.php file
* configure SMTP mail in .env file
* 
* goto UserController -> create a function for OTP send mail

8 [POS] Working With OTP Email :

* create API Endpoint for OTP Send send-otp
* - test with postman

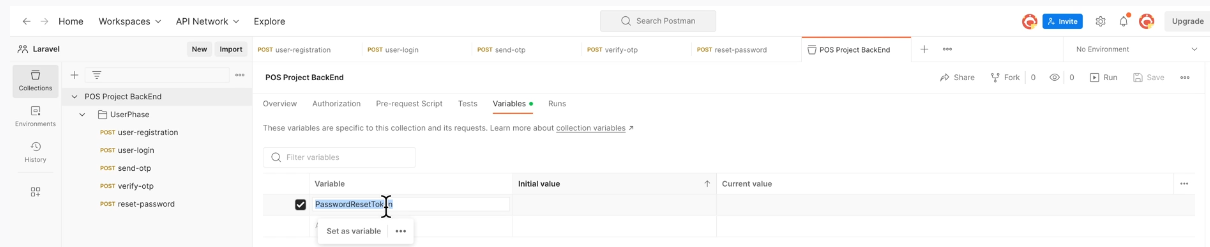
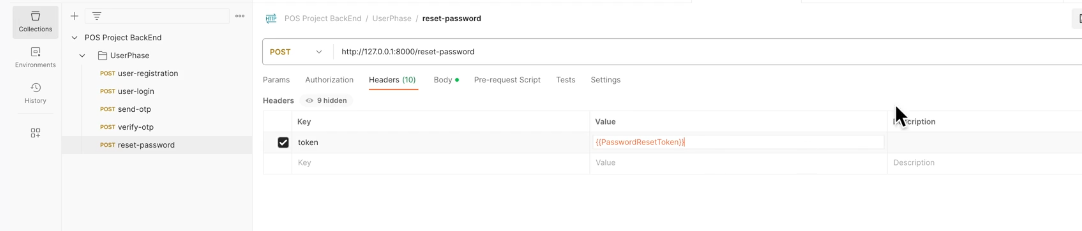
9 [POS] Verify OTP And Issue JWT Password Reset Token :

* in UserController create a function for VerifyOTP
* in JWTToken.php create a function CreateTokenForSetPassword for reset password token
* create API Endpoint for OTP Verification Route::post('/verify-otp',[UserController::class,'VerifyOTP']);
* Back End Test with Postman : verify-otp

10 [POST] Working With Token Verification Middleware And Password Reset :

* create a middleware for Token Verification
* - php artisan make:middleware TokenVerificationMiddleware : it creates a file in app-> http ->middleware->TokenVerificationMiddleware.php
* We will receive the token through header instead of body
* open TokenVerificationMiddleware.php
* goto userController.php : create function ResetPassword for password reset
* add Route API Endpoint for reset password

11 [POS] Completing Password Reset :

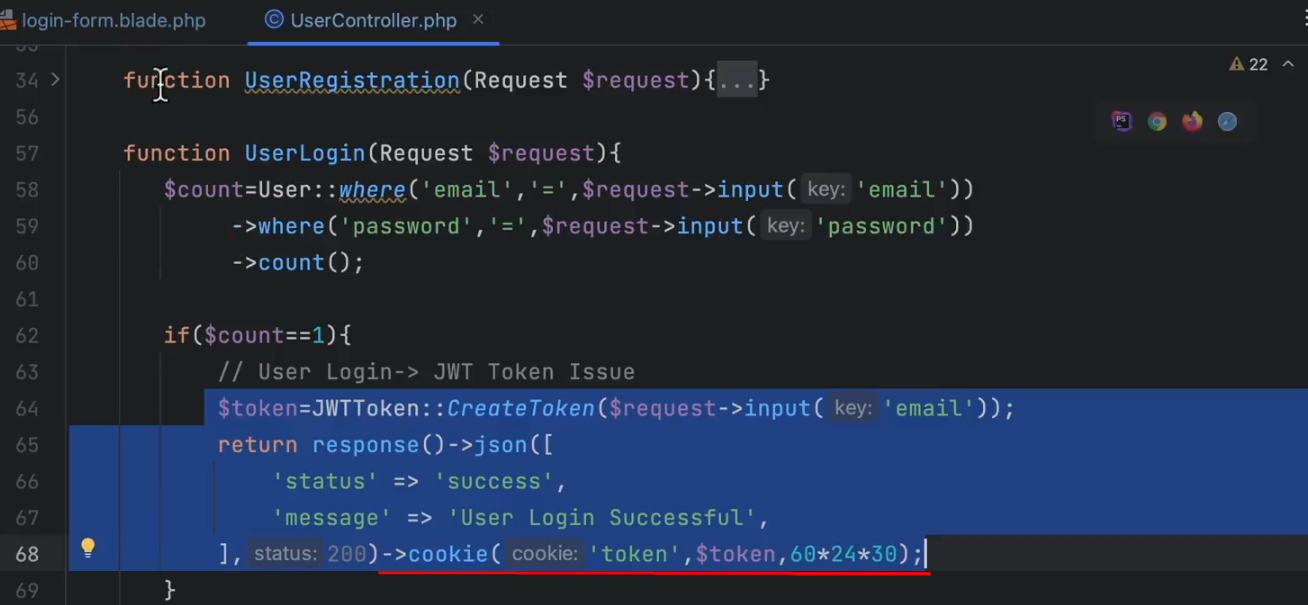
* Back End Test with Postman : reset-password
* goto postman documentation variable section
* set a variable PasswordResetToken
* 
* 

12 [POS] Front End Explain Front End Assets :

* Front End Development
* create page for UserLogin, UserRegistration, SendOTP, VerifyOTP, ResetPassword, Dashboard etc
* Add page routes in routes->web.php
* create resources->views->layout->app.blade.php and sidenav.blade.php
* HTML, CSS, Javascript strategy etc.

13 [POS] Front End User Login :

* login-form.blade.php
* create function for SubmitLogin
* goto User Controller -> User Login
* store token in browser : cache, cookies, index db, local storage & session storage, but recommended is COOKIE.
* to set Token in the Cookie we should open userController.php & do the following :



14 [POS] Front End User Registration :

* registration form : registration-form.blade.php
* set id in input fields
* insert <form action="" onsubmit="return false"> to restrict page reloading
* do necessary coding
* & test

15 [POS] Front End Send OTP : for Password Reset

* xx

16 [POS] Front End Verify OTP :

* set token in cookie
* set email in session Storage

17 [POS] Front End Reset password :

* reset-pass-form.blade.php
* TokenVerificationMiddleware.php
* web.php
* UserController.php

Module : 21

18 [POS] User Logout And Take Agile Advantages

* goto UserController, create function UserLogout, write return redirect('/UserLogin')->cookie('token','',-1);
* in web.php - Route::get('/user-logout',[UserController::class,'UserLogout']);
* sidenav-layout.blade.php

19 [POS] Auth Restriction In Page And End Point :

* Keep user ID
* change in JWTToken in VerifyToken function string|object
* $decode
* set middleware in all necessary routes through web.php

20 [POS] Working With User Profile :

* show profile detail & update
* create a file in views->pages->dasboard->profile-page.blade.php
* copy all from dashboard to profile page
* create profile-form.blade.php in resources->views->components->dashboard
* copy all from user-registration-form to profile-form.php file
* goto UserController & create a routing end point for profile-page
* create a routing endpoint for profile by web.php
* give linke with sidenav-layout.blade.php
* now test

21 [POS] Working With User Profile :

* create a function in UserController naming userProfile
* we will update all except email address

22 [POS] Working With User Profile :

* Show Profile
* Profile information edit & update

23 [POS] Working With Category Back End :

* Create Category Table
* Manage Backend
* Frontend Manage
* Create a Migration File : php aritsan make:migration create\_categories\_table
* Do necessary change in categories\_table.php file
* php artisan migrate
* to manage category table with Eloquent ORM we have create a model
* create a model name Category.php (app->models)
* php artisan make:model Category
* create a Category Controller – php artisan make:controller CategoryController
* create page for frontend http->resources->views->pages->dashboard->category-page.blade.php
* All action related to category like list/update/delete/create & read with this controller
* goto CategoryController for creating backend features
* create methods for category list/ create / delete & update
* now goto web.php
* Test with Postman

24 [POS] Test Category API In Postman :

* uses of jquery Data Table plugin to show the data from <https://datatables.net>

25 [POS] Set Up JavaScript Data table For Category :

* add datables in front end

26 [POS] Fix the Part 25 Error :

* dataList show some options

27 [POS] Working With Category Create And Refresh List :

* work with views->components->category-create.php
* details on onClick() and EventListener

28 [POS] Understanding id pulling from specific row object :

* Work with Edit & Delete
* edit button
* delete button

29 [POS] Working With Delete In Smart Way :

* Delete & Update / Edit
* Delete button successfully configured

30 [POS] Customer Module As Previous :

* Planning customer table
* Customer back-end development
* Managing front-end development
* Create Customer Table : php artisan make:migration create\_customers\_table
* Insert required fields
* command line : php artisan migrate
* Create Model Customer : php artisan make:model Customer
* Create Customer Controller : php artisan make:controller CustomerController

31 [POS] Update Category By Getting Details :

* Category Update / Edit category-update.blade.php
* create a function in CategoryController naming CategoryById
* sleep(5) / wait for 5 seconds

32 [POS] Product Module Back End :

* Create products Table : php artisan make:migration create\_products\_table
* open products.php from database->migrations, write necessary fields
* create table : php artisan migrate
* create controller : php artisan make:controller ProductController
* write function for productList, productCreate, productUpdate, productDelete etc.

33 [POS] Product Create Back End :

* write function for productList, productCreate, productUpdate, productDelete etc.
* Test with postman
* create a folder uploads within public folder
* create a model : php artisan make:model Product
* Test with postman

34 [POS] Product Listing Delete Details Back End :

* work with product delete
* product-by-id : both get / post can b used in routing end point

35 [POS] Product Update With And Without Image :

36 [POS] Product Front End List And Delete :

* Work with Product Front End Delete product-delete.blade.php

37 [POS] Product Front End Create Product :

* enable multipart/form-data with headers
* use form-data instead of json data
* add Files / Img

38 [POS] Product Front End Update Product :

* goto product-update.blade.php

39 [POS] Invoice Database Design :

* Planning invoice table
* Product Invoice Back-end development
* Front-end development
* Create Invoices Table : php artisan make:migration create\_invoices\_table
* open invoices.php from database->migrations, write necessary fields
* create invoice\_products table : php artisan make:migration create\_invoice\_products\_table
* create table : php artisan migrate

40 [POS] Invoice Back End :

* create controller : php artisan make:controller InvoiceController
* create required functions within controller for managing routing
* create model for invoice & invoiceProducts : php artisan make:model invoice
* php artisan make:model InvoiceProduct
* create blade file ‘pages.dashboard.sale-page.blade.php’
* DB::beginTransaction() and DB::rollBack() function is necessary for multiple table transaction
* Create api endpoint in web.php

41 [POS] Test Invoice Back-End :

* Back-end testing with Postman
* Invoice Create
* Invoice Delete
* Invoice Details
* Invoice List

42 [POS] Working With Invoice Front End :

* Create a page sale-page.blade.php resources->views->pages->dashboard

43 [POS] Working With Invoice Front End :

* Invoice creation completed

43 (B) [POS] Working With Invoice List And Print :

* create invoice-page.blade.php page
* create a folder resources->views->components->invoice
* create file invoice-list.blade.php, invoice-details.blade.php & invoice-delete.blade.php in invoice folder

fscanf(STDIN, “%s %s”, $a, $b);

$line = trim($line, “ \r\n”);



Code Mama : Mile & Pound