

Stripe Portal

Q: I'm building a customer billing portal on react, with express as the api. It is all deployed from a single repository using aws amplify. I am using stripe as the credit card processor. Stripe will store the credit card data. Our customers are billed based on monthly subscription to various services. The front end will allow customers to manage credit cards, view invoices, see the billing history, check next billing date, and view active subscriptions. Give me a list of the pages and a description of its purpose, that I need on the frontend to facilitate this.

Here is a list of pages and their purposes that you may need on the frontend of your customer billing portal:

- 1 Homepage:** This page will serve as the entry point for the portal and will provide an overview of the customer's billing history, active subscriptions, and next billing date.
- 2 Manage Credit Cards:** This page will allow customers to add, edit, or delete credit cards on file for billing purposes.
- 3 View Invoices:** This page will allow customers to view and download their past invoices.
- 4 Billing History:** This page will provide a detailed view of the customer's billing history, including past payments, subscriptions, and any adjustments or credits.
- 5 Active Subscriptions:** This page will allow customers to view and manage their active subscriptions, including the ability to upgrade or downgrade plans and cancel subscriptions.
- 6 Payment Settings:** This page will allow customers to manage their payment settings, such as updating their billing address and email.
- 7 Contact Us:** This page will provide a way for customers to contact you with billing-related questions or issues.

Note that this is a base list, and you may need additional pages depending on your requirements.

Q: Give me a list of the endpoints I should setup to support these pages.

Here is a list of endpoints that you may need to set up to support the pages on your customer billing portal:

- 1 GET /home:** This endpoint will retrieve the customer's billing overview information such as active subscriptions, next billing date and recent billing history.
- 2 GET /credit-cards:** This endpoint will retrieve the customer's credit card information.
- 3 POST /credit-cards:** This endpoint will add a new credit card for the customer.

- 4 PUT /credit-cards/:id:** This endpoint will update an existing credit card for the customer.
- 5 DELETE /credit-cards/:id:** This endpoint will delete a credit card for the customer.
- 6 GET /invoices:** This endpoint will retrieve the customer's past invoices.
- 7 GET /invoices/:id:** This endpoint will retrieve a specific invoice for the customer.
- 8 GET /billing-history:** This endpoint will retrieve the customer's detailed billing history.
- 9 GET /subscriptions:** This endpoint will retrieve the customer's active subscriptions.
- 10 POST /subscriptions:** This endpoint will add a new subscription for the customer.
- 11 PUT /subscriptions/:id:** This endpoint will update an existing subscription for the customer.
- 12 DELETE /subscriptions/:id:** This endpoint will cancel a subscription for the customer.
- 13 GET /payment-settings:** This endpoint will retrieve the customer's payment settings.
- 14 PUT /payment-settings:** This endpoint will update the customer's payment settings.
- 15 POST /contact-us:** This endpoint will send a contact us form.

Note that this is a base list, and you may need additional endpoints depending on your requirements. Also note that you need to handle the security and validation of the data that goes in and out of your api, and also make sure to store the credit card information in a secure and compliant way.

Q: Give me a directory and file structure to support these endpoints and pages.

Here is a suggested directory and file structure to support the endpoints and pages on your customer billing portal:

```
my-app/
|
+-- src/
|   |
|   +-- api/
|       |   |
|       |   +-- index.js // Main entry point for the API
|       |   |
|       |   +-- controllers/
|       |       |   |
|       |       |   +-- billing.js // Contains controllers for billing endpoints
|       |       |   +-- creditCards.js // Contains controllers for credit card endpoints
|       |       |   +-- invoices.js // Contains controllers for invoices endpoints
|       |       |   +-- subscriptions.js // Contains controllers for subscriptions
|       |       |   endpoints
```

```

| | | +-- paymentSettings.js // Contains controllers for payment settings
endpoints
| | | +-- contactUs.js // Contains controllers for contact us form endpoint
| | |
| | +-- models/
| | | |
| | | +-- billing.js // Contains database models for billing
| | | +-- creditCards.js // Contains database models for credit card
| | | +-- invoices.js // Contains database models for invoices
| | | +-- subscriptions.js // Contains database models for subscriptions
| | | +-- paymentSettings.js // Contains database models for payment settings
| | |
| | +-- routes/
| | | |
| | | +-- index.js // Contains all the routes for the API
| | | +-- billing.js // Contains routes for billing endpoints
| | | +-- creditCards.js // Contains routes for credit card endpoints
| | | +-- invoices.js // Contains routes for invoices endpoints
| | | +-- subscriptions.js // Contains routes for subscriptions endpoints
| | | +-- paymentSettings.js // Contains routes for payment settings
endpoints
| | | +-- contactUs.js // Contains routes for contact us form endpoint
| |
| +-- client/
| | |
| | +-- public/
| | | |
| | | +-- index.html // HTML template for the app
| | | +-- css/ // Contains CSS files
| | | +-- js/ // Contains JS files
| | |
| | +-- src/
| | | |
| | | +-- components/ // Contains React components
| | | | |
| | | | +-- Home.js // Homepage component
| | | | +-- ManageCreditCards.js // Manage Credit Cards component
| | | | +-- ViewInvoices.js // View Invoices component
| | | | +-- BillingHistory.js // Billing History component

```

Q: Give me a shell script that will stub out these files using mkdir and touch. Write the script without comments or carriage returns. Combine the commands to reduce the size of the script.

```
#!/bin/bash
mkdir my-app && cd my-app && mkdir src && cd src && mkdir api && cd api && mkdir
controllers && touch controllers/billing.js controllers/creditCards.js controllers/
invoices.js controllers/subscriptions.js controllers/paymentSettings.js controllers/
contactUs.js && mkdir models && touch models/billing.js models/creditCards.js
models/invoices.js models/subscriptions.js models/paymentSettings.js && mkdir routes
&& touch routes/index.js routes/billing.js routes/creditCards.js routes/invoices.js
routes/subscriptions.js routes/paymentSettings.js routes/contactUs.js
```

This script will create the main directory, go into it and create the api directory, controllers, models and routes directories, and create the files associated with each endpoint, all in one command.

It's more efficient in term of execution time and more readable.

Q: Give me the code for src/api/routes/index.js

Here is an example of what the code for **src/api/routes/index.js** could look like:

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

// Import controllers
const billingController = require('../controllers/billing');
const creditCardsController = require('../controllers/creditCards');
const invoicesController = require('../controllers/invoices');
const subscriptionsController = require('../controllers/subscriptions');
const paymentSettingsController = require('../controllers/paymentSettings');
const contactUsController = require('../controllers/contactUs');
```

```

// Billing routes
router.get('/billing/overview', billingController.getOverview);
router.get('/billing/history', billingController.getHistory);

// Credit card routes
router.get('/credit-cards', creditCardsController.getCreditCards);
router.post('/credit-cards', creditCardsController.addCreditCard);
router.put('/credit-cards/:id', creditCardsController.updateCreditCard);
router.delete('/credit-cards/:id', creditCardsController.deleteCreditCard);

// Invoices routes
router.get('/invoices', invoicesController.getInvoices);
router.get('/invoices/:id', invoicesController.getInvoice);

// Subscriptions routes
router.get('/subscriptions', subscriptionsController.getSubscriptions);
router.post('/subscriptions', subscriptionsController.addSubscription);
router.put('/subscriptions/:id', subscriptionsController.updateSubscription);
router.delete('/subscriptions/:id', subscriptionsController.cancelSubscription);

// Payment settings routes
router.get('/payment-settings', paymentSettingsController.getPaymentSettings);
router.put('/payment-settings', paymentSettingsController.updatePaymentSettings);

// Contact us route
router.post('/contact-us', contactUsController.sendContactUsForm);

module.exports = router;

```

This code sets up the main routing for the API by importing the controllers for each endpoint and defining the routes for each endpoint. Each route is associated with a specific controller method to handle the request. This file is the main entry point for the API, and it serves as a central location for all routes. Please note that this is just an example and you may need to adjust it according to your requirements.

Q: Give me the code for `src/api/routes/billing.js`

Here is an example of what the code for **`src/api/routes/billing.js`** could look like:

```

const express = require('express');
const router = express.Router();

// Import the billing controller
const billingController = require('../controllers/billing');

// Define routes
router.get('/overview', billingController.getOverview);
router.get('/history', billingController.getHistory);

module.exports = router;

```

This code defines the routes for the billing endpoints of your API. The routes are handled by the corresponding method of the billing controller, which is imported at the top of the file.

This approach helps to keep the routes and controllers separate, making it easier to maintain and understand the code.

Please note that this is just an example and you may need to adjust it according to your requirements.

Q: Give me the code for `src/api/controllers/billing.js`

Here is an example of what the code for **`src/api/controllers/billing.js`** could look like:

```

const Billing = require('../models/billing');

exports.getOverview = async (req, res, next) => {
  try {
    const billingOverview = await Billing.findOne({ customerId: req.user._id });
    return res.status(200).json({ billingOverview });
  } catch (error) {
    return res.status(500).json({ error });
  }
}

exports.getHistory = async (req, res, next) => {

```

```
try {  
    const billingHistory = await Billing.find({ customerId: req.user._id });  
    return res.status(200).json({ billingHistory });  
} catch (error) {  
    return res.status(500).json({ error });  
}  
}
```

This code exports two methods that handle the requests for the billing endpoints. The first method **getOverview** retrieves the billing overview for the customer from the database and returns it as a JSON response. The second method **getHistory** retrieves the billing history for the customer from the database and returns it as a JSON response.

It is using a **Billing** model that is imported at the top of the file, it's supposed to handle the database operations.

It's also using **req.user._id** to identify the customer, you may use other methods to identify the user.

Please note that this is just an example and you may need to adjust it according to your requirements and your database structure.

Q: I need to create a jira ticket for the BillingHistory.js component. This ticket needs the following information in order to help the developer complete the task. A descriptive title. An overview of the task. A description of the form and UI/UX of the page. Technical details on what endpoints they will be using. An overview of the class and method structure the component should follow. A list of npm packages they need to use. A todo list to help guide the developer. A list of tests that our quality assurance team will perform once the ticket is completed in order to verify if it meets the requirements. Please write this and structure it in markup with a descriptive title and each section using sub headers.

Create BillingHistory.js Component

Overview of the Task

Create a new component for the customer billing portal called **BillingHistory.js** that allows customers to view their detailed billing history, including past payments, subscriptions, and any adjustments or credits.

UI/UX of the Page

- The component should display a table with the following columns: date, description, amount, and status.
- The table should be sortable by date, description, amount, and status.
- The table should be paginated to display a maximum of 10 records per page.
- The component should have a search bar that allows customers to search for specific transactions by description or amount.
- The component should have a download button that allows customers to download their billing history in CSV format.
- The component should have a responsive design that adapts to different screen sizes.

Technical Details

- The component will use the following endpoints: **GET /billing-history**: Retrieve the customer's detailed billing history.

Class and Method Structure

- The component should be a class-based component and should use the **componentDidMount** lifecycle method to fetch the billing history data from the API.
- The component should have a **state** object that holds the billing history data and any other necessary data for the component.
- The component should have a **handleSearch** method that updates the state with the search query and filters the data accordingly.
- The component should have a **handleSort** method that sorts the data based on the selected column.
- The component should have a **handleDownload** method that formats the data as a CSV file and prompts the user to download it.

Packages (npm)

- react-table
- react-csv

Todo List

- Create the component file and import the necessary dependencies.
- Define the state object and initialize it with the data from the **GET /billing-history** endpoint.
- Create the table component using the **react-table** package.

- Implement the search bar and the **handleSearch** method.
- Implement the sorting functionality and the **handleSort** method.
- Implement the download button and the **handleDownload** method.
- Test the component on different screen sizes and make necessary adjustments for a responsive design.
- Test the component against the API endpoints and make sure data is displayed correctly.
- Add necessary styling to the component to match the overall look and feel of the application.

Tests

- ☐ Verify that the table displays the correct data when the page is loaded.
- ☐ Verify that the search bar filters the data correctly.
- ☐ Verify that the sorting functionality sorts the data correctly.
- ☐ Verify that the download button prompts the user to download the data in CSV format.
- ☐ Verify that the component is responsive and adapts to different screen sizes.
- ☐ Verify that the data displayed in the table matches the data from the API.
- ☐ Verify that the component follows the overall style guide of the application.

Please note that this is just an example and you may need to adjust it according to your requirements.

Q: Give me an updated version of the code for `src/api/controllers/billing.js` that supports the search, sorting, pagination, and download needed in the `BillingHistory.js` component.

Here is an example of what the code for **`src/api/controllers/billing.js`** could look like with additional functionalities :

```
const Billing = require('../models/billing');
const json2csv = require('json2csv');

exports.getHistory = async (req, res, next) => {
  try {
    let query = { customerId: req.user._id };
    // Search
    if (req.query.search) {
      query = {
        customerId: req.user._id,
```

```

        $or: [
            { description: { $regex: req.query.search, $options: 'i' } },
            { amount: { $regex: req.query.search, $options: 'i' } },
        ]
    }
}

// Sorting
let sort = {};
if (req.query.sortBy) {
    sort[req.query.sortBy] = req.query.sortOrder === 'asc' ? 1 : -1;
}

// Pagination
const limit = parseInt(req.query.limit) || 10;
const skip = (req.query.page - 1) * limit || 0;

const billingHistory = await Billing.find(query)
    .sort(sort)
    .skip(skip)
    .limit(limit);

// Download
if (req.query.download === 'csv') {
    const fields = ['date', 'description', 'amount', 'status'];
    const opts = { fields };
    const csv = json2csv.parse(billingHistory, opts);
    res.setHeader('Content-disposition', 'attachment; filename=billing-
history.csv');
    res.set('Content-Type', 'text/csv');
    res.status(200).send(csv);
} else {
    res.status(200).json({ billingHistory });
}
} catch (error) {
    return res.status(500).json({ error });
}
}

```

This code now supports search, sorting, pagination, and download functionalities. It is using

some query parameters in the request, like **search**, **sortBy**, **sortOrder**, **limit**, **page**, **download**.

It's using the **json2csv** package to convert the data to csv and set the necessary headers to make the browser download the file.

It's also using **mongoose**'s built-in pagination and sorting method to retrieve the paginated and sorted data from the database.

Please note that this is just an example and you may need to adjust it according to your requirements and your database structure.