



**Database Tables**

**Users:** This table will store information about website users such as their username, email, password, and other user-related details.

**Events:** This table will contain details about the events that users can purchase tickets for. You may include columns such as event name, date and time, location, ticket price, and other relevant information.

**Tickets:** This table will store information about each ticket sold, such as the ticket ID, event ID, user ID, and other details.

**Orders:** This table will contain information about each order placed, such as the order ID, user ID, order date and time, and other relevant information.

**Payment:** This table will store information about each payment made by users, such as the payment ID, payment method, payment status, and other details.

**Reviews:** This table will contain reviews submitted by users about events, venues, and the website in general. You may include columns such as the review ID, user ID, event ID or venue ID, rating, and review text.

**Differently priced tickets under same event id**

If you have an event that has multiple ticket prices, you can handle this in a database table by creating a separate record for each ticket type and linking them all to the same event ID.

For example, let's say you have an event with an ID of "123" and three different ticket types: General Admission, VIP, and Premium. You could create a table with the following columns:

Ticket ID: A unique identifier for each ticket type.

Event ID: A foreign key linking the ticket to the event it belongs to.

Ticket Type: A description of the type of ticket (e.g. "General Admission", "VIP", "Premium").

Price: The cost of the ticket.

Then, you could create three records in this table, one for each ticket type, with the same Event ID of "123". Each record would have a different Ticket ID, Ticket Type, and Price. This allows you to store and retrieve information about the different ticket types associated with the event.

When you query the table, you can filter by the Event ID to retrieve all the tickets associated with that event, and then display the ticket types and prices as needed.

It's worth noting that you may also want to consider storing additional information about each ticket type, such as the number of tickets available for each type, the date and time of the event, and any other relevant details. This can help you manage the event and ensure that all the necessary information is available to users who are purchasing tickets.

**Ways to make a header stick on the screen while scrolling:**

Using position: fixed: This method involves setting the position property of the header element to fixed, which will make it stay in the same place on the screen even when the page is scrolled.

Copy code

header {

position: fixed;

top: 0;

left: 0;

width: 100%;

z-index: 1;

}

Using JavaScript: You can use JavaScript to detect when the user is scrolling and change the position of the header element accordingly. This method will be useful if you want to add some additional functionality such as hiding the header when the user scrolls down and showing it when the user scrolls up.

Copy code

// Get the header

var header = document.getElementById("header");

// Get the offset position of the navbar

var sticky = header.offsetTop;

// Add the sticky class to the header when you reach its scroll position. Remove "sticky" when you leave the scroll position

function myFunction() {

if (window.pageYOffset > sticky) {

header.classList.add("sticky");

} else {

header.classList.remove("sticky");

}

}

// When the user scrolls the page, execute myFunction

window.onscroll = function() {myFunction()};

Using CSS position:sticky, this method is similar to position:fixed but it will only work when the user scrolls the page up or down. This method is more efficient than the previous one and it does not require JavaScript.

Copy code

header {

position: sticky;

top: 0;

z-index: 1;

}

It's important to note that the exact code will vary depending on the specific CSS and JavaScript frameworks and libraries you are using.

**USERS TABLE**

user\_id INT PRIMARY KEY AUTO\_INCREMENT - This line of code defines the first column in the table called "user\_id" with a data type of "INT". It is set as the primary key, which means it will be used as a unique identifier for each record in the table. The "AUTO\_INCREMENT" keyword means that the value of this column will be automatically incremented for each new record added to the table.

first\_name VARCHAR(100) NOT NULL - This line of code defines the second column in the table called "first\_name" with a data type of "VARCHAR" (variable-length string). The maximum length of this field is 100 characters, and the "NOT NULL" constraint means that this field must always have a value (i.e., it cannot be left blank).

last\_name VARCHAR(100) NOT NULL - This line of code defines the third column in the table called "last\_name" with a data type of "VARCHAR". It has the same characteristics as the "first\_name" column.

email VARCHAR(100) NOT NULL - This line of code defines the fourth column in the table called "email" with a data type of "VARCHAR". It has the same characteristics as the "first\_name" and "last\_name" columns.

phone\_number BIGINT - This line of code defines the fifth column in the table called "phone\_number" with a data type of "BIGINT" (big integer). This column is optional and can be left blank if a phone number is not available.

password\_hash VARBINARY(100) - This line of code defines the sixth column in the table called "password\_hash" with a data type of "VARBINARY" (variable-length binary data). It is used to store a hashed version of the user's password.

user\_type ENUM('admin', 'user', 'guest') NULL - This line of code defines the seventh column in the table called "user\_type" with a data type of "ENUM". The "ENUM" data type limits the possible values of this field to a specific set of values. In this case, it can be one of three possible values: 'admin', 'user', or 'guest'. The "NULL" keyword means that this field is optional and can be left blank.

CONSTRAINT unique\_email UNIQUE (email) - This line of code adds a constraint to the table called "unique\_email". This constraint ensures that the "email" field is always unique, so no two records in the table can have the same email address.

CHECK (user\_type IN ('admin', 'user', 'guest')) - This line of code adds a check constraint to the table called "user\_type". This constraint ensures that the "user\_type" field can only have one of the three values defined in the "ENUM" data type earlier in the table definition.

**crud functionalities of a system**

CRUD stands for "Create, Read, Update, and Delete" and these are the basic functions of a system that allows a user to interact with a database or other data storage system.

Create - This functionality allows a user to add new data to the system, such as creating a new record in a database.

Read - This functionality allows a user to view or retrieve existing data from the system, such as reading a record from a database.

Update - This functionality allows a user to modify existing data in the system, such as updating a record in a database.

Delete - This functionality allows a user to remove data from the system, such as deleting a record from a database.

Marshah Mungai

how can i authenticate sellers on my website

There are several ways to authenticate sellers on a website:

Email Verification: Send a verification link to the seller's email address, and once they click on it, their account will be verified.

Phone Verification: Send a verification code to the seller's phone number, and once they enter it, their account will be verified.

Social Media Authentication: Allow sellers to sign in with their social media accounts, such as Facebook, LinkedIn, or Google.

ID Verification: Require sellers to upload a copy of their government-issued ID, and then manually verify the ID to confirm their identity.

Business Verification: Require sellers to provide proof of their business, such as a business license or registration documents.

Two-factor authentication: Require sellers to provide an additional form of authentication, such as a code sent to their email or phone, or use of a token generator, to ensure that the account is being accessed by the authentic seller.

It's important to keep in mind that the level of authentication required will depend on the type of products or services being sold, as well as the level of trust and security you want to establish with your sellers.

how do i make the hover effect stay on the selected lkink when navigating numerous links

To make the hover effect stay on the selected link when navigating numerous links, you can use a combination of CSS classes and JavaScript.

First, create a CSS class for the selected link. For example:

css

Copy code

.selected-link {

background-color: #ffcc00;

color: #fff;

}

This class defines the background color and text color for the selected link.

Add an event listener to each link using JavaScript. When a link is clicked, remove the selected-link class from all links and add it to the clicked link. For example:

javascript

Copy code

var links = document.querySelectorAll('a');

links.forEach(function(link) {

link.addEventListener('click', function(event) {

event.preventDefault();

links.forEach(function(link) {

link.classList.remove('selected-link');

});

this.classList.add('selected-link');

});

});

This code selects all a elements on the page and adds an event listener to each one. When a link is clicked, it prevents the default link behavior, removes the selected-link class from all links, and adds it to the clicked link.

Finally, apply the selected-link class to the current link on page load, if applicable. For example, if the user is on the "Home" page, apply the selected-link class to the "Home" link. This can be done using server-side code or JavaScript.

With this approach, the hover effect will stay on the selected link when navigating numerous links.

**How can i ensure that sellers on my website are not fraudulent**

Here are some ways to ensure that sellers on your website are not fraudulent:

Verify seller information: Verify the seller's identity by requiring them to provide government-issued ID or business registration documents.

Use third-party verification services: Utilize third-party verification services such as credit bureaus or identity verification services to validate the seller's identity.

Monitor transactions: Monitor transactions for any suspicious activity, such as multiple transactions with the same credit card or a large number of refund requests.

Use fraud detection software: Implement fraud detection software that can analyze transactions and identify patterns that are commonly associated with fraud.

Have a clear policy on fraud: Have a clear policy in place that outlines what constitutes fraud, the consequences for committing fraud, and how to report it.

Reviews and Feedback: Allow buyers to leave reviews and feedback about the sellers, which can help to identify any suspicious behavior.

Keep records: Keep records of all transactions and communications with sellers, as this can be used as evidence if a seller is found to be fraudulent.

It's important to keep in mind that no system is completely fraud-proof, so it's important to have multiple layers of protection in place and to regularly review and update your fraud prevention measures.

**what layers of protection can i put in place and to prevent fraud by sellers on my website**

Here are some layers of protection you can put in place to prevent fraud by sellers on your website:

Authentication and verification: Verify the seller's identity by requiring them to provide government-issued ID or business registration documents, and use third-party verification services to validate their identity.

Monitoring and detection: Monitor transactions for any suspicious activity, such as multiple transactions with the same credit card or a large number of refund requests. Use fraud detection software that can analyze transactions and identify patterns that are commonly associated with fraud.

Risk management: Implement risk management strategies such as setting limits on the amount of sales or transactions a seller can make, or requiring sellers to provide a deposit or bond before they can start selling.

Communication and education: Communicate your fraud prevention policies to sellers and educate them on how to detect and prevent fraud.

Reviews and Feedback: Allow buyers to leave reviews and feedback about the sellers, which can help to identify any suspicious behavior.

Keep records: Keep records of all transactions and communications with sellers, as this can be used as evidence if a seller is found to be fraudulent.

Security Measures: Implement security measures such as SSL encryption to protect sensitive information, firewalls, and intrusion detection systems to protect your website and databases from hacking attempts.

Legal measures: Have a clear policy in place that outlines what constitutes fraud, the consequences for committing fraud, and how to report it, and have a legal agreement with the sellers to ensure that they understand the consequences of committing fraud.

It's important to keep in mind that no system is completely fraud-proof, so it's important to have multiple layers of protection in place and to regularly review and update your fraud prevention measures.

**Hashing Passwords**

There are many hash functions available for hashing passwords, such as bcrypt, PBKDF2, and scrypt. Each of these functions has its own strengths and weaknesses, but they are all designed to be difficult to reverse or crack.

Here's an example of how to use the bcrypt hashing function in a PHP application to hash passwords before storing them in a SQL database:

1. Install the password\_compat library, which provides an easy-to-use interface for password hashing using bcrypt. You can install it using Composer by running the following command in your project directory:

javascriptCopy code

composer require ircmaxell/password-compat

2. In your PHP code, use the password\_hash() function to hash the password:

bashCopy code

$password = 'password123'; $hashedPassword = password\_hash($password, PASSWORD\_DEFAULT);

The PASSWORD\_DEFAULT constant tells password\_hash() to use the default hashing algorithm, which is currently bcrypt.

3. Store the hashed password in the SQL database. You should use the VARBINARY data type to store the hashed password, as we discussed earlier.

4. When a user logs in, retrieve the hashed password from the database and use the password\_verify() function to compare the entered password to the hashed password:

perlCopy code

$enteredPassword = 'password123'; if (password\_verify($enteredPassword, $hashedPassword)) { // Password is correct, grant access } else { // Password is incorrect, deny access }

The password\_verify() function returns true if the entered password matches the hashed password, and false otherwise.

By using a password hashing function like bcrypt, you can ensure that your users' passwords are stored securely and are difficult to crack in the event of a data breach.

**Rating Option On The User Interface**

There are many ways to implement a rating option on a user interface, but one common approach is to use a star rating system. Here's an example of how you could implement a star rating system using HTML, CSS, and JavaScript:

Create a container element for the rating, such as a div element with a unique id attribute:

html

Copy code

<div id="rating-container"></div>

Add a set of star icons to the container using HTML and CSS. You can use an icon library like Font Awesome to get the star icons. For example:

html

Copy code

<div id="rating-container">

<i class="far fa-star"></i>

<i class="far fa-star"></i>

<i class="far fa-star"></i>

<i class="far fa-star"></i>

<i class="far fa-star"></i>

</div>

<style>

#rating-container i {

font-size: 24px;

color: #ccc;

cursor: pointer;

}

#rating-container i:hover,

#rating-container i.selected {

color: #ffcc00;

}

</style>

In this example, we're using the Font Awesome star icons (far fa-star) and styling them with CSS. We've set the font size to 24 pixels, the default color to light gray, and the cursor to a pointer (which indicates that the stars are clickable). We've also added a hover effect that changes the color to a bright yellow (#ffcc00) and a selected class that applies the same color to stars that have been clicked.

Add JavaScript code to handle the user's clicks on the stars and update the rating. For example:

html

Copy code

<script>

let ratingContainer = document.getElementById('rating-container');

let stars = ratingContainer.getElementsByTagName('i');

let rating = 0;

for (let i = 0; i < stars.length; i++) {

stars[i].addEventListener('click', function() {

rating = i + 1;

updateRating();

});

}

function updateRating() {

for (let i = 0; i < stars.length; i++) {

if (i < rating) {

stars[i].classList.add('selected');

} else {

stars[i].classList.remove('selected');

}

}

}

</script>

In this example, we're using JavaScript to add a click event listener to each star icon. When the user clicks a star, we update the rating variable and call the updateRating() function to update the appearance of the stars. The updateRating() function adds the selected class to the appropriate stars based on the rating variable.

By using this approach, you can provide a user-friendly rating system that allows users to easily rate products or services with a simple click of the mouse. Note that this is just one example of how to implement a star rating system, and there are many other ways to approach this problem depending on your specific needs and preferences.

**TABLE CREATION QUERIES**

CREATE TABLE users (

user\_id INT PRIMARY KEY AUTO\_INCREMENT,

first\_name VARCHAR(100) NOT NULL,

last\_name VARCHAR(100) NOT NULL,

email VARCHAR(100) NOT NULL,

phone\_number INT,

password\_hash VARBINARY(100),

user\_type ENUM('admin', 'buyer', 'seller') NULL,

CONSTRAINT unique\_email UNIQUE (email)

);

CREATE TABLE event\_details (

event\_id INT PRIMARY KEY AUTO\_INCREMENT,

event\_name VARCHAR(100) NOT NULL;

event\_date DATE NOT NULL,

event\_time TIME NOT NULL,

location VARCHAR(100) NOT NULL

);

CREATE TABLE tickets (

ticket\_id INT PRIMARY KEY AUTO\_INCREMENT,

ticket\_type ENUM('Regular', 'VIP', 'VVIP') NOT NULL,

price INT NOT NULL

);

CREATE TABLE orders (

order\_id INT PRIMARY KEY AUTO\_INCREMENT,

order\_date DATE NOT NULL,

order\_time TIME NOT NULL

);

CREATE TABLE reviews (

review\_id INT PRIMARY KEY AUTO\_INCREMENT,

rating INT NULL,

review VARCHAR(255) NULL

);

CREATE TABLE payments (

payment\_id INT PRIMARY KEY AUTO\_INCREMENT,

payment\_method ENUM('Credit/Debit Card', 'PayPal', 'M-Pesa') NOT NULL,

payment\_status ENUM('Paid', 'Pending', 'Failed', 'Refunded', 'Canceled') NOT NULL

);

**Ticket Emporium as a business model**

A ticket selling website can work as a business model by generating revenue through the sale of tickets for events, such as concerts, sports games, theater performances, and more. Here are some key aspects of how a ticket selling website can work as a business:

Partnership with Event Organizers: The ticket selling website can partner with event organizers to obtain tickets for events. This can be done through negotiations with the organizers or through a system where the website acts as an intermediary between the organizers and the customers.

Sales Commission: The website can charge a commission on each ticket sold, which is typically a percentage of the ticket price. This commission can be negotiated with event organizers, and can vary depending on the type of event and the number of tickets sold.

Additional Revenue Streams: Ticket selling websites can also generate additional revenue through various methods, such as offering premium seats for an additional fee, providing VIP packages with special perks, or selling merchandise related to the event.

Marketing: The website can also generate revenue through marketing efforts, such as paid advertising, social media marketing, and email marketing. This can help increase visibility and drive traffic to the website, resulting in more ticket sales.

User Experience: Providing a user-friendly experience on the website, including a simple and easy-to-use interface, can also increase sales and revenue. This can be achieved through effective design, navigation, and features such as mobile optimization, real-time ticket availability, and personalized recommendations.

Overall, a ticket selling website can generate revenue by providing a platform for customers to purchase tickets for events, while also providing additional value through various features and services.

**Log In/Sign Up button and user/cart icons**

You can achieve this by dynamically changing the content of the navigation bar based on whether the user is logged in or not. Here's an example of how you can implement this:

**HTML code:**

html

Copy code

<nav>

<ul>

<li><a href="#">Home</a></li>

<li><a href="#">Products</a></li>

<li><a href="#">About Us</a></li>

<li class="nav-right"></li>

</ul>

</nav>

In the example above, we have created a navigation bar with four items. The fourth item, with a class of nav-right, is where we will dynamically display the "Log In/ Sign Up" button, user icon, or cart icon.

**JavaScript code:**

javascript

Copy code

const navRight = document.querySelector(".nav-right");

function updateNav() {

const loggedIn = false; // Replace with your own logic to check if the user is logged in

let content = "";

if (loggedIn) {

content = `<a href="#">User Icon</a> <a href="#">Cart Icon</a>`;

} else {

content = `<a href="#">Log In/ Sign Up</a>`;

}

navRight.innerHTML = content;

}

// Call updateNav initially to set the content based on the initial state

updateNav();

In the example above, we first select the nav-right element using document.querySelector(). We then define a function called updateNav() which checks whether the user is logged in or not, and dynamically generates the HTML content for the nav-right element based on that. Finally, we call updateNav() initially to set the content based on the initial state, and the function will automatically update the content whenever the user logs in or logs out.

**CSS code:**

css

Copy code

nav {

background-color: #333;

color: white;

}

nav ul {

list-style: none;

margin: 0;

padding: 0;

display: flex;

}

nav li {

margin-right: 10px;

}

nav li a {

color: white;

text-decoration: none;

}

.nav-right {

margin-left: auto;

}

In the example above, we have defined some basic styles for the navigation bar and the nav-right element. We have set the background color of the navigation bar to #333, and the text color to white. We have also set the display property of the ul element to flex, and used the margin-left: auto property on the nav-right element to align it to the right of the navigation bar. You can adjust these styles to match your own design.

**Conditional rendering**

Conditional rendering is a programming technique used in front-end web development to selectively display or hide elements on a webpage based on certain conditions. It allows developers to control the appearance and behavior of a webpage dynamically based on user input, system events, or other factors.

Conditional rendering typically involves using logical operators and control flow statements in programming languages such as JavaScript to determine whether to display or hide a particular element or component. For example, a developer might use a conditional statement to show a certain section of a webpage only if the user is logged in or if a certain condition is met. The logic can be as simple or complex as needed, depending on the requirements of the application.

In frameworks such as React and Vue.js, conditional rendering is a core feature that allows developers to create dynamic user interfaces that respond to user input and system events. These frameworks provide various built-in tools and methods for implementing conditional rendering, such as if/else statements, ternary operators, and logical operators.

**Here's how you can preview your website on GitHub:**

1. Create a new repository on GitHub or use an existing repository.
2. In your repository, create a new branch named gh-pages.
3. Add your website files to this branch. Your website should have an index.html file in the root directory.
4. Commit your changes and push the gh-pages branch to your repository on GitHub.
5. Wait a few minutes for GitHub Pages to build your website. You can check the status of the build by going to the "Settings" tab of your repository and scrolling down to the "GitHub Pages" section.
6. Once the build is complete, you can view your website by going to the URL https://username.github.io/repository-name/, where "username" is your GitHub username and "repository-name" is the name of your repository.

By previewing your website using GitHub Pages, you can make sure that everything looks and works as expected before making your website live to the public.