Type something

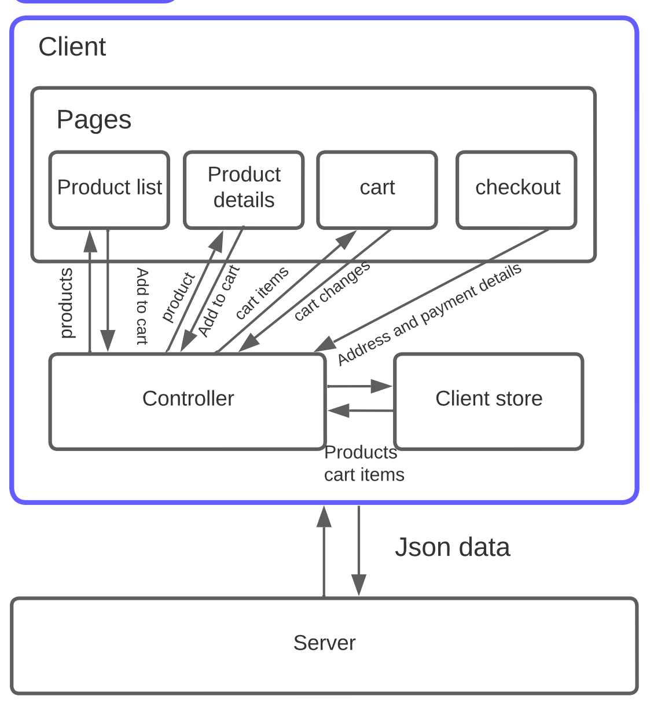
Ecommerce website

Frame 2

Requirements

1. Core features (browse, add to cart, checkout)
2. Product details shown - (product name, product image, price)
3. In the detailed view, product description will be extra
4. Non functional requirements - Page should load within 2 seconds, interaction with page elements should respond quickly.
5. Should work well on all the devices
6. Users should be able to purchase as guests, without signing in.

Frame 3



Frame 1

Component responsibilities

1. Server should provide an http apito fetch products data, cart items, modify the carts, and create orders
2. Controller controls the flow of data within the application
3. Store will have the data needed across the whole application.

SCREENS - Basic functionality
1. View all products scree, with add to cart option
2. Individual product details view page, with product description and add to cart option
3. List of items in the cart with total bill and checkout option
4. Form to add address and place order

Frame 4

SSR or CSR ?

SSR - server fetches all the data, uses them to create the final markup and sends down to html. Most of the rendering work is done on the server
- Pro- Performance is generally better
- Con - Page transitions are slower, for every request, entire page has to be constructed on the server
- SEO is important for ecommerce websites, hence SSr should be a priority.

CSR - Server sends the initial html which contains JS to bootstarp the application, client fetches the necessary data and creates the final mark up. (used for SPAs where full page refresh isnt required)

SPAs use CSR where as MPAs user SSR, the comfortable middle ground with best of both worlds is SSR with hydration - server renders the full html, but the rendering and navigation becomes client side. Hydrations means adding event handlers after initial load.

SPA --> SSR (next.js, remix, Nuxt) ----> CSR (create react app)
MPA ----> Ruby on rails, django ---> This combination isn't practical

Data Model

Frame 5

Entity	Source	Belongs to	Fields
1. Product List	1. Server	1. Product Listing page	1. Products (list), pagination (meta data)
2. Product	2. Server	2. Product listiong, product details	2. name, description, unit_price, currency, primary image, image_url
3. Cart	3. Server	3. Client store	3. items (list of cart items), total_price, currency
4. Cart Item	4. Server	4. Client Store	4. Quantity, product, price, currency
5. Address Details	5. User Input(client	5. Checkout page	5. name, country, street, city, zip etc
6. Payment Details	6. User input (client)	6. Checkout Page	6. Card_number, card_expiry, card_cvv etc

Frame 6

Interface Definition

1. Product information (fetch product listing, fetch product details)
2. Cart Modification - (Add a product to cart, Change quantity of a product in the cart, Remove the product from cart)
3. Complete order

   (we assume that the login user has only one cart, so no need to pass the cart id)

- GET/products, params {size, page, country)
- We use offset based pagination here instead of cursor based pagination because, we need to be able to jump between the pages, useful to know how many total results are there, user browsing history is not easily lost as the new results aren't that quickly added)
- GET/products/{productId, country}
- POST/cart/products/{productid}
- PUT/cart/products/{productid}/ --> params {quantity}
- DELETE/cart/product/{productid}
- POST/order params - {cartid, addressdetails, payment details}
- We might want to save the address and payment details for the returning users)

Frame 7

Optimizations

1. Performance is absolutely critical for ecommerce websites
2. Code splitting based on JS routes using bundlers
3. Split the content in to seperate sections and prioritize above the fold content while lazy loading below the fold content
4. Defer loading of non critical Javascript (Ex: code for modals, dialogs etc)
5. Prefetch the data neded for the next page like, fetch the product details, upon hovering on the product, fetch checkout page while on cart page etc
6. OPtimize the images with lazy loading and adaptive loading
7. Prefetching the top search results

Core web vitals

1. Largest contentful paint (LCP) - The render time of the largest image or text block visible within the viewport, relative to when the page first started loading.
   - Optimize performance – loading of JavaScript, CSS, images, fonts, etc.

2. First Input Delay(FID) - Quantifies the user experience while dealing with unresponsive pages. A low FID means page is usable. Reduce the amount of JS needed to be executed on page load to improve this.

3. Cumulative Layout Shift (CLS) - Visual stabilty, quantifies how often users experience layout shift. Low CLS means page is delightful. Use shimmer effect, reserve the aspect ratio while the images are loading,

SEO

1. Important because, primary search is how people discover the products

- PDPs should have proper `<title>` and `<meta>` tags for description, keywords, and open graph tags.
- Generate a `sitemap.xml` to tell crawlers the available pages of the website.
- Use JSON structured data to help search engines understand the kind of content on your page. For the e-commerce case, the `Product` type would be the most relevant.
- Use semantic markup for elements on the page, which also helps accessibility.
- Ensure fast loading times to help the website rank better in Google search.
- Use SSR for better SEO because search engines can index the content without having to wait for the page to be rendered (in the case of CSR).

Images

1. Use webp images for good quality images
2. Host the images on CDN
3. Define priority of the images and divide them into critical and non critical. Lazy load the non critical ones.
4. Adaptive loading of images, loading high-quality images for devices on fast networks and using lower-quality images for devices on slow networks

Forms
1. Use Localized address forms (country specific)
2. Optimize the auto filling the forms
3. Alternately, along with the traditional option, allow the user to search for an address and select from the results or to select from the map
4. Leverage client side validation and display the error states properly
5. Have the pages translated in the supported languages (i18n)

Accessibility

1. Use semantic html elements wherever possible
2. Use appropriate aria values for non semantic elements
3. All the image tags should have alt attribute
4. Make the form fields focussable
5. Make the currently focussed form element obvious

Security

1. use HTTPS so that all the communication with the server is encrypted and other users on the same network cannot intercept
2. Payment details submission api should not be using HTTP GET because the sensitive details will get added to the browser history, which is potentially unsafe if the browser is shared. use PUT orPOST instead.

User Experience

1. Make the check out page clean with minimum distractions
2. Persist the cart contents
3. Make the promo code fields less prominent so that people without promo code will not leave the page in search for it. Those who already have it will make the effort to look for it.