


Fullstack Developer

A pragmatic course in Fullstack Development
~ FOUNDATION LEVEL ~



M. Yauri M. Attamimi

<https://yauritux.link>



What will you get from this course ?

- Who is Fullstack Developer ?
- How to become a Fullstack Developer ?
- Gain a pragmatic experience through building a mini project as a Fullstack Developer.

A close-up photograph of a person's hands typing on a laptop keyboard. The background is blurred, showing bokeh lights and a dark surface. The word 'Objectives' is overlaid in white text on the left side of the image.

Objectives

By the end of this course, you will be able to:

- Building a good enough production app as a Fullstack Developer.

About the Speaker

M. Yauri M. Attamimi

- 18 years (or so) in Software Engineering
- LOTS of project experience
- Java, Kotlin, Python, NodeJS, Golang, etc
- A Software Craftsman
- Enterprise Architect Certified (TOGAF 9.2)
- Event-Sourcing and Reactive System Provocateur
- AI Enthusiast
- TDD and Clean Code Evangelist
- Founder and CEO of bisnisin.asia
- Lead Software Architect for [DSS PropertyGuru](https://DSSPropertyGuru)

My Professional Mission :

Guiding individuals and organizations to commercial success through the application of modern technologies



<https://github.com/yauritux>



<https://yauritux.link>

A close-up, slightly blurred photograph of a person's hand resting on a desk, with their index finger pointing at a computer mouse. The background is out of focus, showing some bokeh light effects. The text 'Intended Audiences' is overlaid in white on the left side of the image.

Intended Audiences

- Someone whom passionate with programming
- Someone who's looking for a career transition
- Someone who's looking to become a Fullstack Developer



Prerequisites

- Knows how to perform basic computer operations and software installation.
- Good logical thinking (i.e., algorithms)
- Knows how to use and find information on the internet (would be nice to have a knowledge of Web protocols, etc)
- Not afraid of using a “Dark Screen”.
- **Optionally** has the basic knowledge of HTML, CSS, and JavaScript (Prior knowledge on Typescript and React will be helpful)
- **Optionally** has the basic knowledge on Git.

A close-up photograph of a person's hand holding a white marker, drawing on a whiteboard. The background is blurred, showing some bokeh lights. The word "Caveats" is overlaid in white text on the left side of the image.

Caveats

- No “Shortcuts” !!!
- Unlikely to cover all topics in short amount of time
 - ◆ Goal is to be reasonably comprehensive
 - ◆ Enable you to develop your own production (web-based) applications
 - ◆ Enable you to fill in gaps yourself once you know what does it need to be a master



Disclaimer

The information provided here is designed to provide helpful information on the subjects discussed and just my own opinion based on my proven experiences (not represent any entities)

A close-up, slightly blurred photograph of a person's hand holding a white marker, writing on a whiteboard. The background is out of focus, showing some bokeh light effects. The text 'How this Course work ?' is overlaid in white on the left side of the image.

How this Course work ?

- Slide to explain concepts
- Exercises to reinforce concepts

A laptop screen is shown at an angle, displaying a code editor with PHP code. The code includes file inclusions, database connection details, and a DELETE query. A blue gradient overlay covers the bottom half of the image, where the main title is placed. The text 'DAY 1' is in yellow, and 'FULL STACK DEVELOPMENT' is in white.

DAY 1

FULL STACK DEVELOPMENT

FULL STACK DEVELOPER



VS

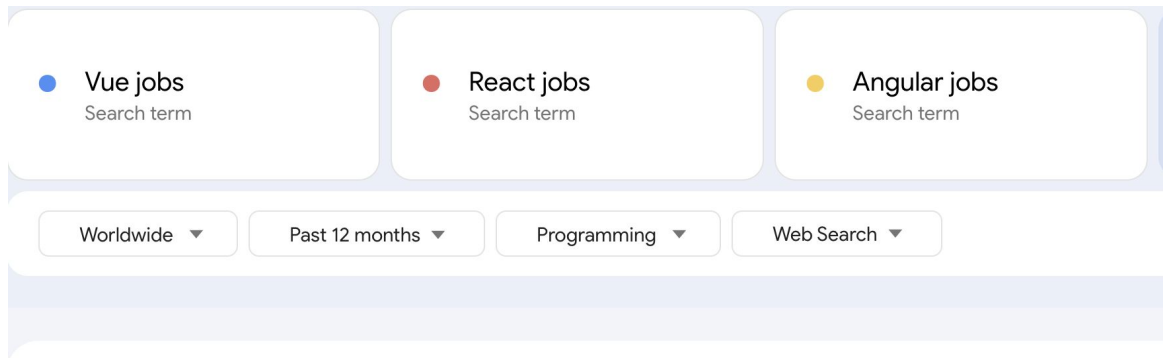


SOFTWARE ENGINEER

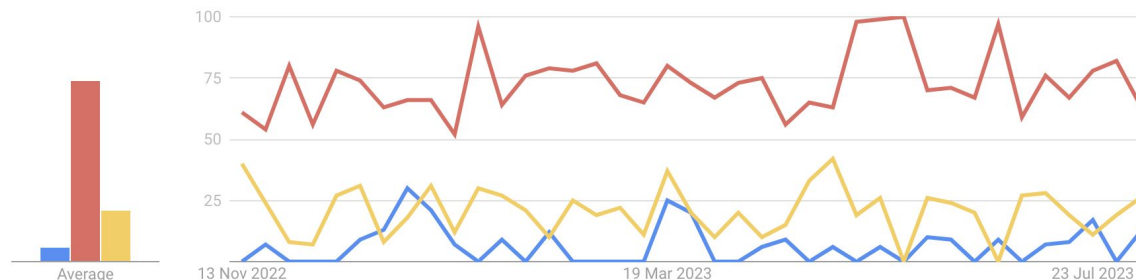
Full Stack Developer



React Job Trends (Google)



Interest over time ?



<https://trends.google.com/trends/explore?cat=31&q=Vue%20jobs,React%20jobs,Angular%20jobs>

What is React ?

- React is a JavaScript library for building fast and interactive user interfaces.
- React was developed on Facebook in 2011 and currently is the most popular JavaScript library for building interfaces

React Component (I)

- Component is the heart of all React applications
- Component essentially is a piece of the user interface
- When building a React app, we build a bunch of independent, isolated, and reusable components..., then compose them to build complex user interfaces.
- Every React application has at least one component which we refer to as the “Root” component.
- “Root” component represents the entire application and contains other children components.
- Every React application essentially is a tree of components.

React Component (II) - Example

The screenshot displays a Twitter-like interface with several distinct components:

- Header:** Navigation links for Home, Notifications, and Messages; a search bar; and a user profile icon with a 'Tweet' button.
- Left Sidebar:** A 'Singapore trends' section listing topics like #Singapore, #China, Trump, #Malaysia, #BigData, Hong Kong, #MTVHottest, #feedly, and Arsenal, each with a tweet count.
- Main Content Area:**
 - A 'What's happening?' section with a video tweet from 'Computer Vision News' about combining research and entrepreneurship.
 - A tweet from 'Coralogix' featuring a screenshot of a dashboard with various charts and graphs, titled '3rd generation logging - integrated to Heroku pipelines'.
- Right Sidebar:**
 - 'Who to follow' section with profiles like Jason K Jackson, npm, Inc., and StrongLoop.
 - 'Find people you know' section with a link to import contacts from Gmail.
 - Footer with copyright information and links to About, Help Center, Terms, Privacy policy, Cookies, Ads info, Brand, Blog, Status, Apps, Jobs, Marketing, Businesses, and Developers.

Comprises of these following components:

- Navbar
- Profile
- Trends
- Feed
 - Tweet
 - Like

And so forth.

As we can see, each component is a piece of UI.

React vs Angular ?

- Both is similar in terms of Component-Based Architecture.
- Angular is a framework with complete solution, while React is a frontend (view) library.
- React is much simpler than angular (i.e. short learning curve)

The image features the Next.js logo, which consists of the word "NEXT" in a large, white, sans-serif font, followed by ".JS" in a smaller font. A thin white diagonal line cuts through the "X" in "NEXT". The background is a dark, futuristic tunnel with glowing blue and green light bands curving along the walls, creating a sense of depth and motion. The lights are reflected on a dark surface at the bottom of the frame.

NEXT.JS

Some NextJS Features

- File-based Routing over Code-based Routing
 - **Page** Based
 - **App** Based
- SSR (pre-rendering) and CSR
- Static Site Generation (pre-build / server pre-generated)
- Fullstack Framework

React Framework Trends (Google)

● NextJS Search term

● Create-React-App Search term

● ReactBootstrap Search term

● Gatsby Search term

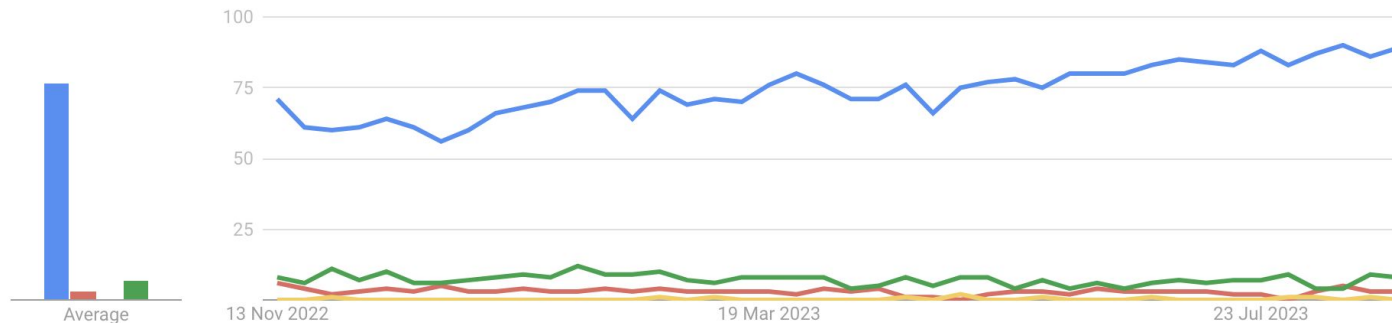
Worldwide ▼

Past 12 months ▼

Programming ▼

Web Search ▼

Interest over time ?



Lab Setup

1. Download and install the latest stable version of NodeJS.
2. Optionally install `npm`.
3. Install Code Editor (e.g. Visual Studio Code / VSCode).
4. Install these 2 extensions within your installed VSCode:
 - a. **Simple React Snippets** → developed by Burke Holland
 - b. **Prettier** → developed by Esben Petersen
5. Settings to trigger prettier on file saved.
 - a. **Accessing menu Code > Preferences > Settings**
 - b. Under **User Settings** tab, add a new pair of key-values: `"editor.formatOnSave": true`
6. Install Git.

Lab 1

First Next App

1. Open terminal.

2. Execute command:

```
pnpx create-next-app@latest
```

1. Go to the created folder.

2. Run the program.

```
pnpm run dev
```

1. Pay attention to the generated project skeleton.

2. Change the display with your own custom JSX (e.g. trying to display “Hello Next World!” on the browser page)

Lab 1.1

Static / Name Based Routing

Create a new “About” page.

Lab 1.2

Nested Paths & Routes

Create Portfolio pages grouped under a folder named “portfolio”.

Lab 1.3

Dynamic Paths & Routes

1. Create several portfolio pages (e.g., index, list, and project specific page) grouped under a folder named “portfolio”.
2. Extracting dynamic path segment data.
3. Setup dynamic path folder(s) for different Client(s).
4. Navigating with “Link” component.
5. Programmable (Imperative) Navigation.

Lab 1.4

Catch All Routes

Purpose:

Used to handle different segment naming variations regardless of the number of nested paths

(e.g., blog/[article-id], blog/[year]/[article-id], blog/[year]/[month]/[article-id], etc)

Lab 1.5

Change Default 404

- **NextJS** comes with a default 404 page to represent a “Not Found” page.
- In most cases, you would like to override this particular 404 page with your own (e.g., a standard 404 page that represent your company, etc).

Lab 1.6

Module (Component-Scoped) CSS

- CSS should have extension of `*.module.css`.
- Place it under the same package with your component.

An aerial photograph of the New York City skyline at dusk. The sky is a mix of dark purple, blue, and orange. The city is densely packed with skyscrapers, many of which are illuminated with their interior lights. The Empire State Building is prominent in the center, with its top lit in red and green. The Hudson River is visible on the right side of the image. The text "Q & A" is overlaid in the center of the image in a large, white, serif font.

Q & A

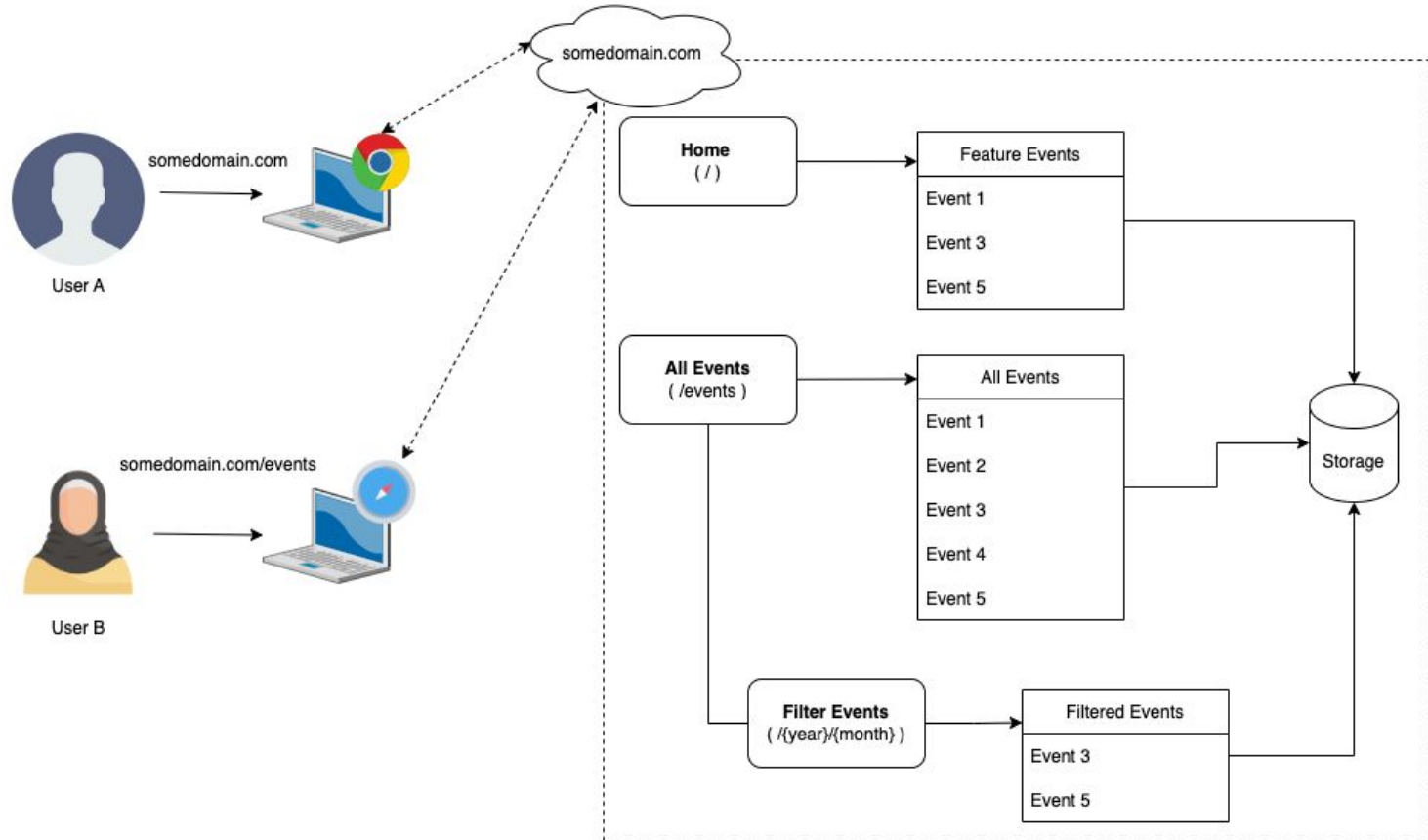
A close-up, angled view of a laptop screen. The screen shows a code editor with PHP code. The code includes comments like 'require_once __DIR__ . "/../vendor/autoload.php";', 'require_once __DIR__ . "/../config/db.php";', and a function 'delete_post(\$id)' that uses 'mysqli_real_escape_string' and 'DELETE' queries. The background is a blurred view of the laptop's keyboard and other screens in the distance. A solid blue gradient overlay covers the bottom half of the image, where the main title text is located.

DAY 2

FULL STACK DEVELOPMENT

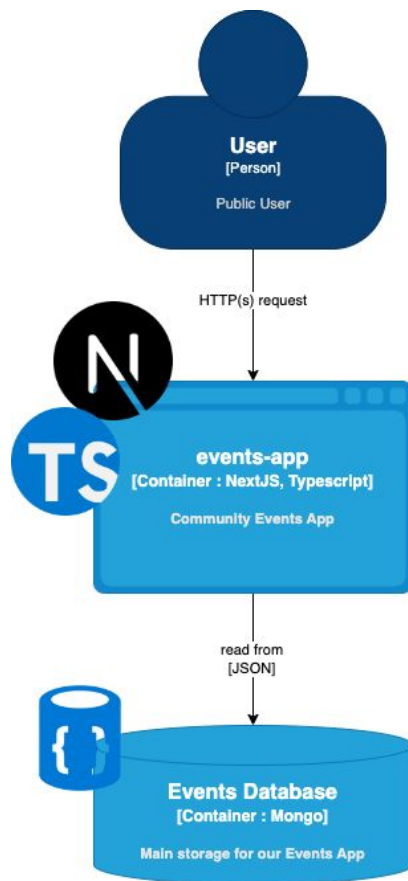
Lab 2

Community Events App (User Story Board)



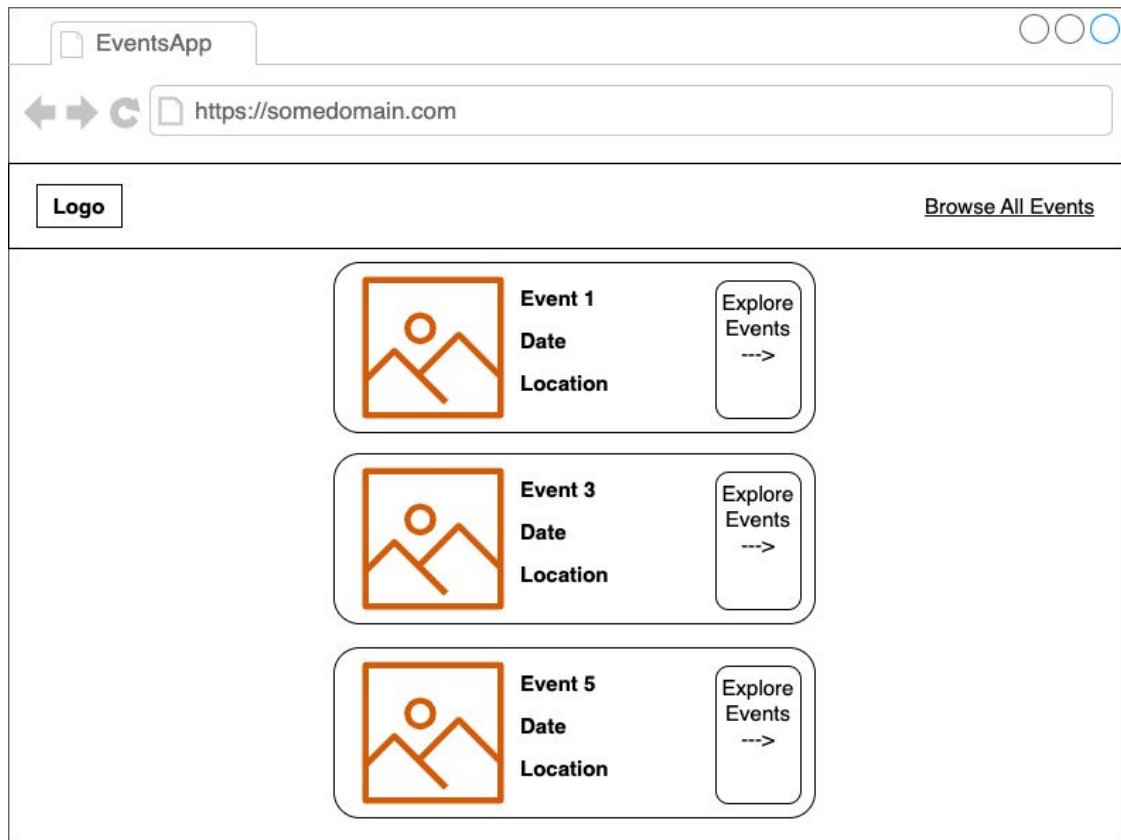
Lab 2

Community Events App (Container Level Diagram)



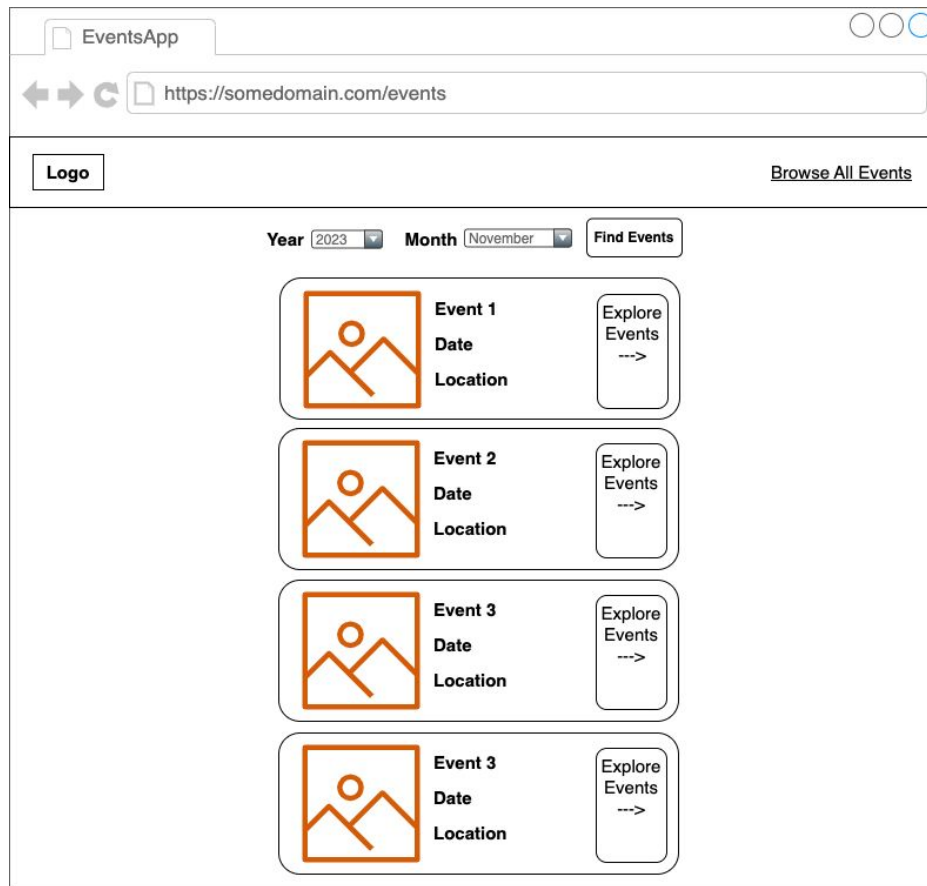
Lab 2

Wireframe (Home / Index)



Lab 2

Wireframe (All Events)



Lab 2.1

Project Setup

1. Open terminal.
2. Execute command:

```
pnpx create-next-app@latest
```

1. Name your project with "events-app"

Lab 2.2

Setup Dummy Data

1. Open your terminal.
2. Create a JSON file contains some dummy data with these following structure:
 - a. **Event ID** (attribute type: `string`, attribute name: `id`)
 - b. **Event Name** (attribute type: `string`, attribute name: `title`)
 - c. **Event Description** (attribute type: `string`, attribute name: `description`)
 - d. **Venue** (attribute type: `string`, attribute name: `location`)
 - e. **Date of Event** (attribute type: `string`, attribute name: `date`)
 - f. **Event Picture** (attribute type: `string`, attribute name: `image`)
 - g. **Event Feature Flag** (attribute type: `boolean`, attribute name: `isFeatured`)
3. Some Tips
 - a. get your free images from [unsplash](https://unsplash.com).
 - b. get your free icons from [heroicons](https://heroicons.com).

Lab 2.3

Data Fetching

1. Put all images you have downloaded under the `public/images` folder.
2. Create one folder named `providers` under the project root directory.
3. Create a new file and give it a name `EventRepository.ts` and place it directly under `providers` folder
4. Write few functions into this particular file to retrieve some events data from the JSON file you had prepared earlier.
5. Display only featured events on the Home (index) page.

Lab 2.4

Separation of Concerns (EventList Component)

1. Create one folder named `components` under the project root directory (i.e., at the same level as `providers` and `pages`).
2. Create one subfolder named `events` under the aforementioned `components` folder.
3. Create a new file named `EventList.tsx` and place it under the `events` folder.
4. Return a function component that responsible to render the events. This function-based component is supposed to receive events data from its props.
5. Call this `EventList` component from your `index.tsx`.

Lab 2.5

More Separation of Concerns (EventItem Component)

1. Create a new file named `EventItem.tsx` and place it under the `events` folder (i.e., same level as `EventList.tsx`).
2. Return a function component that responsible to render the event item. This function-based component is supposed to receive all details about the event from its props.
3. Call (embed) this `EventItem` component from within your `EventList` component.

Lab 2.6

Module / Component Scoped Stylings

1. Copy these following css files under the same folder as `EventList` and `EventItem` components (i.e., `components/events` folder):
 - a. `event-list.module.tsx`
 - b. `event-item.module.tsx`
2. Apply the stylings respectively for the `EventList` and `EventItem` components.

Caveats:

Module / Component scoped styling file should have a suffix of `*.module` after it's filename. Hence, makesure that you include that suffix.

An aerial photograph of the New York City skyline at dusk. The sky is a mix of dark purple, blue, and orange. The city is densely packed with skyscrapers, many of which are illuminated with their interior lights. The Empire State Building is prominent in the center, with its top lit in red and green. To the right, the dark, angular structure of the 111 West 57th Street is visible. The Hudson River and the New York Harbor are visible in the background, with some bridges and distant city lights. The text "Q & A" is overlaid in the center of the image in a large, white, serif font.

Q & A

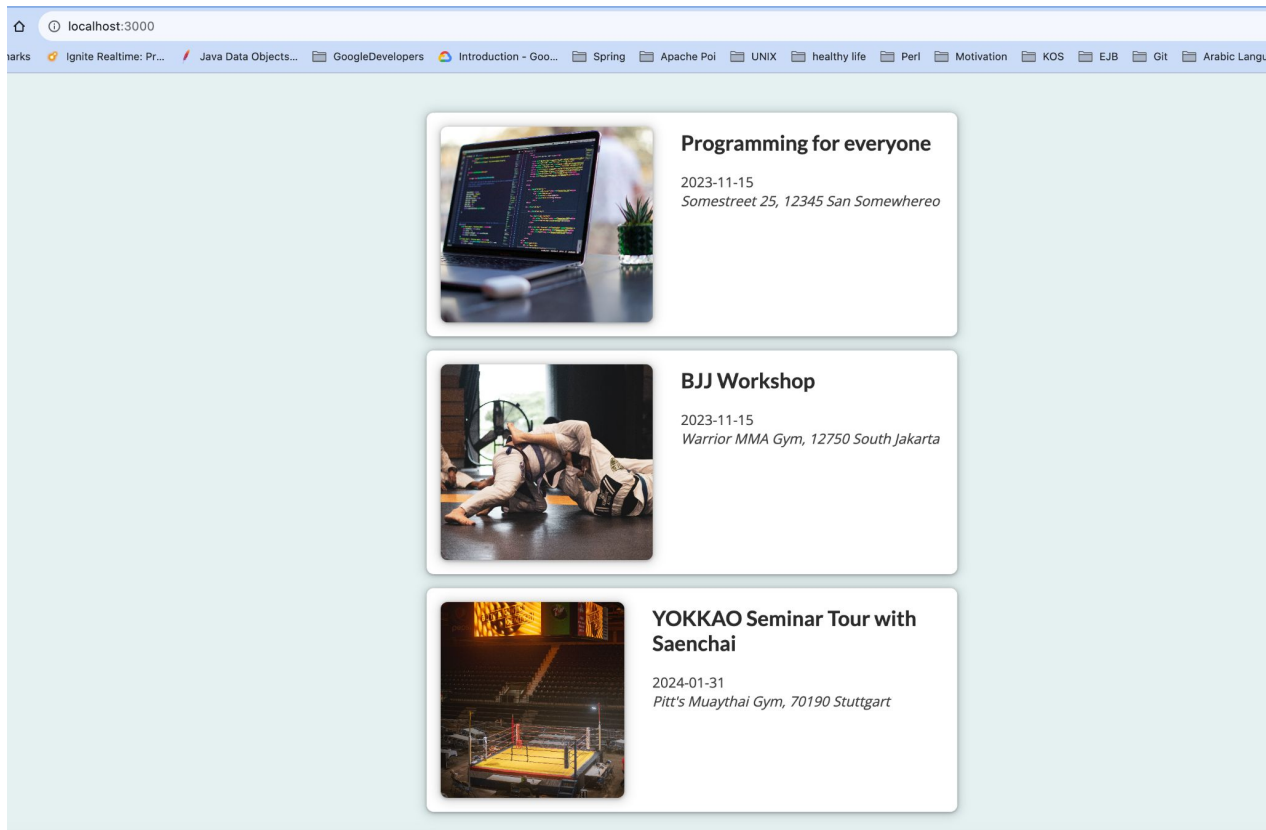
A laptop screen is shown at an angle, displaying a code editor with PHP code. The code includes file inclusions, database connection details, and a DELETE query. A blue gradient overlay covers the bottom half of the image, where the main title is placed. The text 'DAY 3' is in yellow, and 'FULL STACK DEVELOPMENT' is in white.

DAY 3

FULL STACK DEVELOPMENT

Lab 3

Our Events App So Far..



Lab 3.1

Adding a Base Layout

1. Create a new folder named `layout` under the `components` folder.
2. Copy `main-header.module.css` file into this particular `layout` folder.
3. Create a new component for `MainHeader` (i.e., `MainHeader.tsx`) and write all the necessary logics to display your common header as what depicted on the wireframe.
4. Create a new “wrapper” component named `Layout` (`Layout.tsx`). This component will be used as a main layout for binding the `MainHeader` component and all contents together, i.e. all components those are bound within the `main` section.
5. Use the aforementioned `Layout` component as a parent for the `<Component>`'s tag defined in the base `_app.tsx` file.

An aerial photograph of the New York City skyline at dusk. The sky is a mix of dark purple, blue, and orange. The city is densely packed with skyscrapers, many of which are illuminated with their lights. The Empire State Building is prominent in the center, with its top lit in red and green. The Hudson River is visible on the right side of the image. The text "Q & A" is overlaid in the center of the image in a large, white, serif font.

Q & A

A laptop screen is shown at an angle, displaying a code editor with PHP code. The code includes file inclusion, database connection, and a DELETE query. A blue gradient overlay covers the bottom half of the image, where the main title is placed. The text 'DAY 4' is in yellow, and 'FULL STACK DEVELOPMENT' is in white.

DAY 4

FULL STACK DEVELOPMENT

Lab 4.1

“/events” Page

1. Create a new page to display all events (“/events”).
2. Reuse the `EventList` component for displaying all events.

Tips:

there are 2 ways for creating this page. Choose which one suits you best.

Lab 4.2

Catch All Routes (/events/{year}/{month})

1. Create a new Page for filtering events based on the selected year and month.
(**hint:** use “catch all routes” method for creating this page)
2. Reuse the `EventList` component for displaying all of the filtered events.

Lab 4.3

Searching for Event(s)

1. Adding a “searching box” that is useful for finding any event based on your selected filter / searching criteria.
2. User should be able to filter event based on the year and month.
3. As usual, you are highly encouraged to build this particular “searching box” as a component.

(copy the `events-search.module.css` as we have provided for you and put it in the same directory as your “searching box” component’s file).

Lab 4.4

Exercise 1

1. Adding a “result title” component to display the searching result’s title. This particular `ResultsTitle` component should also contain a `Link` with title of “Show All Events” that links to the `/events` page when it’s clicked.
2. As before, utilize the `EventList` component to display all events matched your searching criteria.
3. copy the `results-title.module.css` as we have provided for you and put it in the same directory as your `ResultsTitle` component’s file).

Lab 4.5

Exercise 2

1. Replace `Link` in the `ResultsTitle` with a button.
2. Build a reusable `Button` component that can be used to handle these following functions within your page:
 - a. Searching for Event(s)
 - b. Browse All Events (the one that defined in the `MainHeader` component)
 - c. Show All Events (the one that defined in the `ResultsTitle` component)
3. Format all events date into a human readable format, for instance: “November 25, 2023” instead of “2023-11-25”.

Hint:

Put this `Button` component in the ``components/ui`` package or folder.

Lab 4.6

Exercise 3

1. Add “Explore Event” button on each `EventItem`. Once clicked, user will be redirected to a page showing all details of the selected event.

Hint: Please utilize as much as possible all of the provided css files.

Appendix

Some Useful References for your knowledge refreshment on React:

- <https://www.freecodecamp.org/news/react-hooks-fundamentals/>
- <https://react.dev/learn/referencing-values-with-refs>
- <https://react.dev/learn/referencing-values-with-refs#refs-and-the-dom>
- <https://blog.logrocket.com/usestate-vs-useref/>

Git References:

- <https://www.conventionalcommits.org/en/v1.0.0/>

App Versioning:

- <https://semver.org/>

An aerial photograph of the New York City skyline at dusk. The sky is a mix of dark purple, blue, and orange. The city is densely packed with skyscrapers, many of which are illuminated with their interior lights. The Empire State Building is prominent in the center, with its top lit in red and green. The Hudson River is visible on the right side of the image. The text "Q & A" is overlaid in the center of the image in a large, white, serif font.

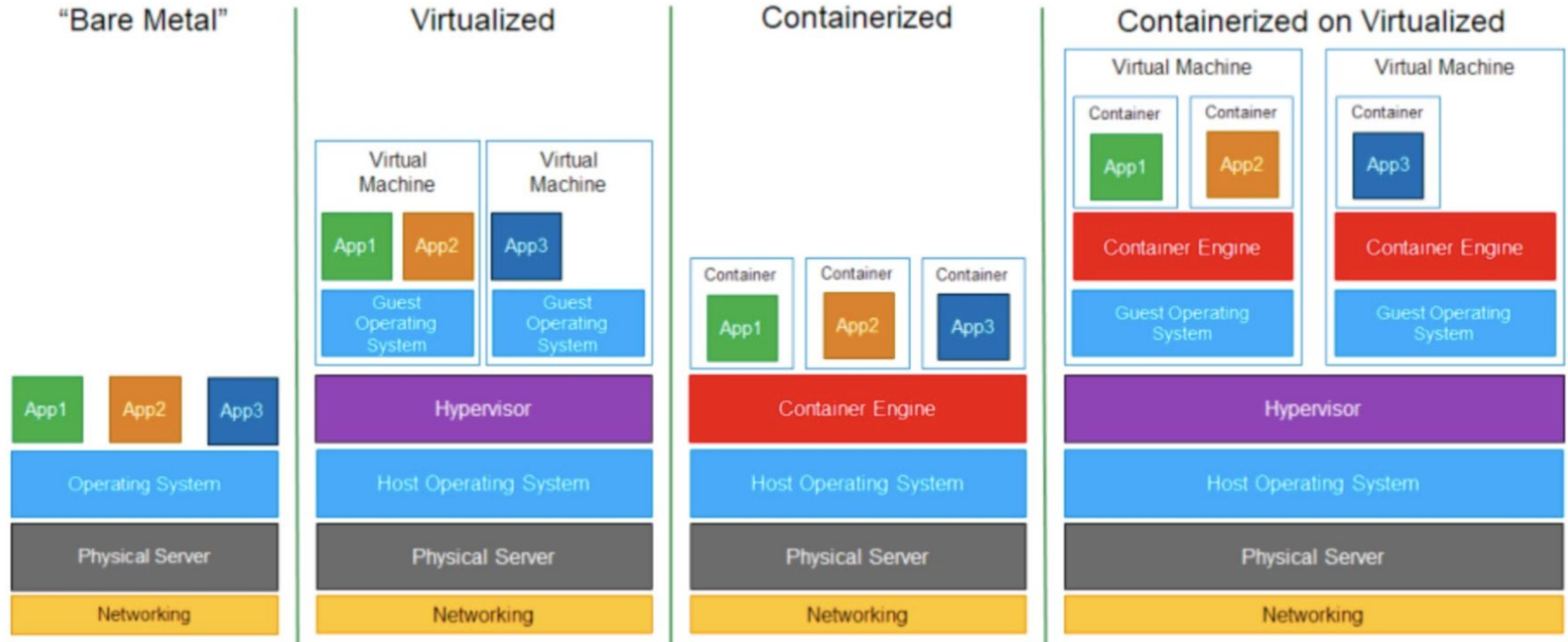
Q & A

A laptop screen is shown at an angle, displaying a code editor with PHP code. The code includes file inclusions, database connection details, and a DELETE query. A blue gradient overlay covers the bottom half of the image, where the main title is placed. The text 'DAY 5' is in yellow, and 'FULL STACK DEVELOPMENT' is in white.

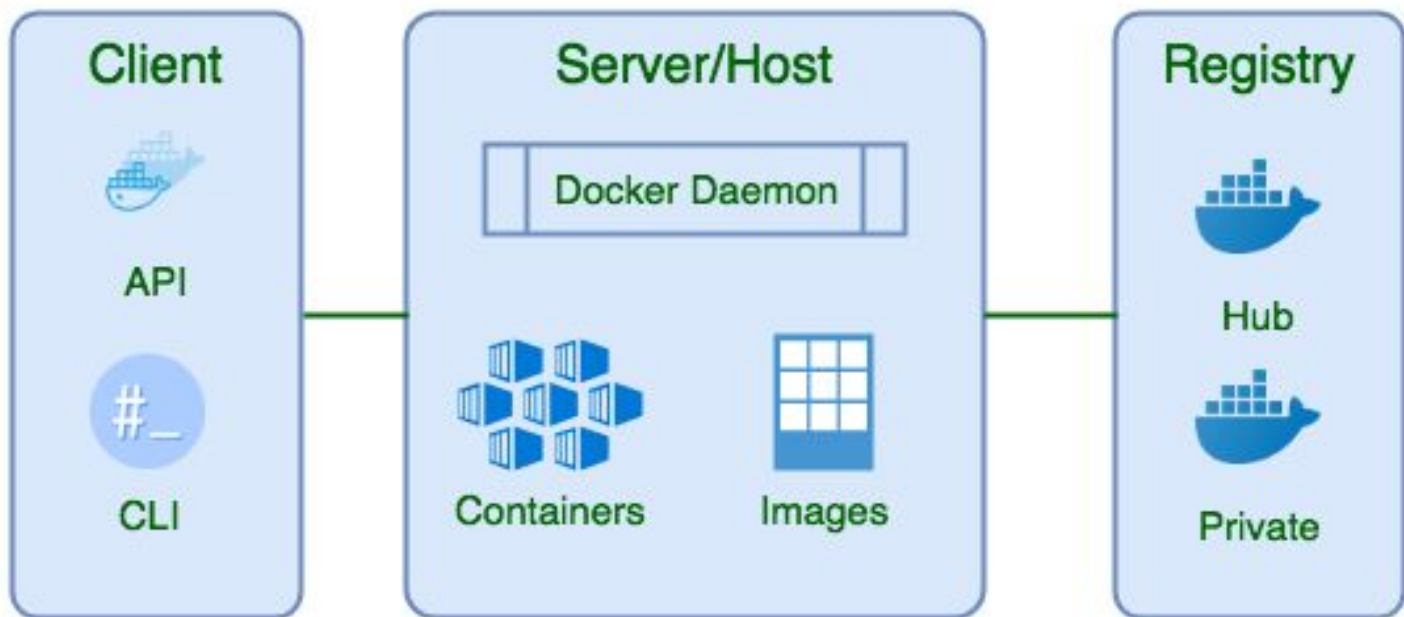
DAY 5

FULL STACK DEVELOPMENT

On-Premise vs Virtual Machine vs Container



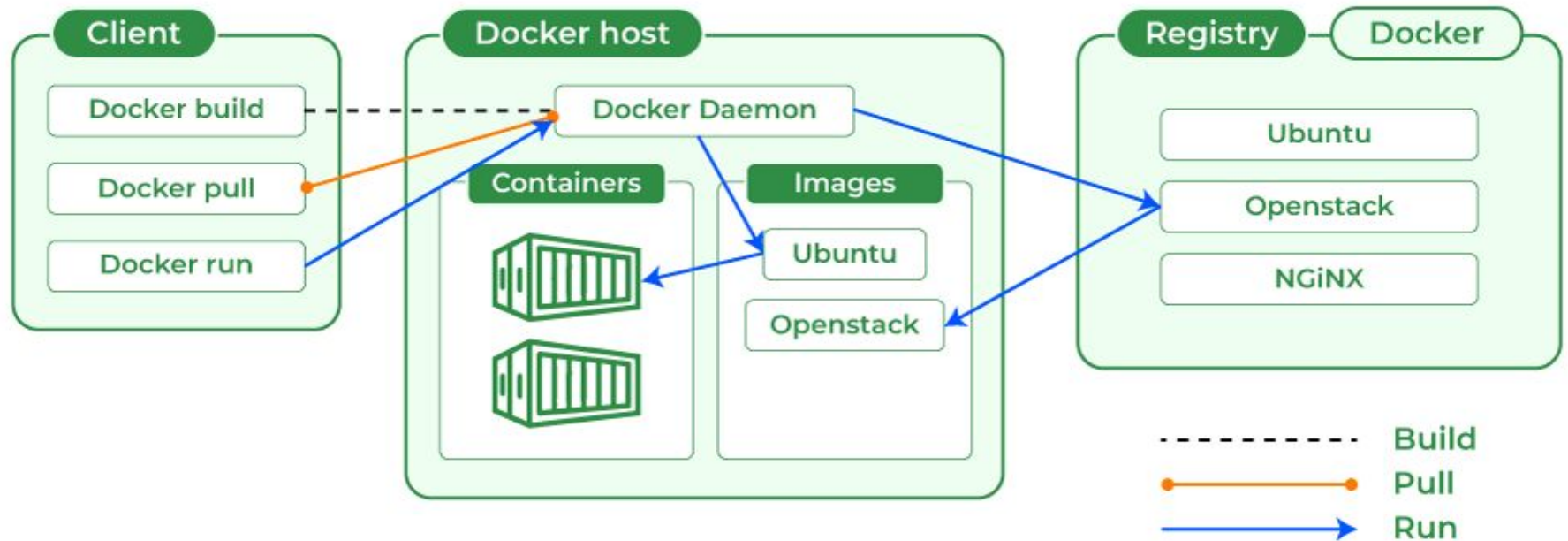
Docker Architecture



Lab Setup

1. Download and install Docker from : <https://docs.docker.com/get-docker/>

Docker Basic Commands



Other Useful Docker Commands

1. `docker image ls`
2. `docker container ls`
3. `docker container run` / `docker run`
4. `docker container exec <container_id>`
5. `docker container log <container_id>`

Tips:

Use `--help` flag to get more details for each command.

An aerial photograph of the New York City skyline at dusk. The sky is a mix of dark purple, blue, and orange. The city is densely packed with skyscrapers, many of which are illuminated with their lights. The Empire State Building is prominent in the center, with its top lit in red and green. The Hudson River is visible on the right side of the image. The text "Q & A" is overlaid in the center of the image in a large, white, serif font.

Q & A

A laptop screen is shown at an angle, displaying a code editor with PHP code. The code includes file inclusions, database connection details, and a DELETE query. A blue gradient overlay covers the bottom half of the image, where the main title is placed. The text 'DAY 6' is in yellow, and 'FULL STACK DEVELOPMENT' is in white.

DAY 6

FULL STACK DEVELOPMENT

Lab 6.1

1. Run MongoDB container in your local environment.
2. Setup new database and name it as “eventsdb”.
3. Create new collection called `events` with this following structure (a.k.a. **schema**) :
 - a. **Event ID** (attribute type: `string`, attribute name: `id`)
 - b. **Event Name** (attribute type: `string`, attribute name: `title`)
 - c. **Event Description** (attribute type: `string`, attribute name: `description`)
 - d. **Venue** (attribute type: `string`, attribute name: `location`)
 - e. **Date of Event** (attribute type: `string`, attribute name: `date`)
 - f. **Event Picture** (attribute type: `string`, attribute name: `image`)
 - g. **Event Feature Flag** (attribute type: `boolean`, attribute name: `isFeatured`)
4. Insert some data into the `events` collection we’ve just created.

Ref:

<https://www.mongodb.com/docs/manual/reference/method/db.createCollection/>

Lab 6.2

Refactoring:: Use Mongo as Database

1. Refactor your **events-ap** project to get all events data from the database (mongo) you've created.

Lab 6.3

Exercise

1. Complete all remaining event repository functions (i.e, `Get Event By ID`, `Searching for Event`, etc).
2. Adding a new form that can be used to entry a new event !
3. Refactor your Event Database Repository to ensure there's no hardcoded value anymore.

An aerial photograph of the New York City skyline at dusk. The sky is a mix of dark purple, blue, and orange. The city is densely packed with skyscrapers, many of which are illuminated with their interior lights. The Empire State Building is prominent in the center, with its top lit in red and green. The Hudson River is visible on the right side of the image. The text "Q & A" is overlaid in the center of the image in a large, white, serif font.

Q & A



DAY 7

FULL STACK DEVELOPMENT

Page Pre-Rendering & Data Fetching

1. Pre-Rendering Behavior

This will be the default behavior for any NextJS App which comprises of:

- **Static Site Generation (SSG)**
- Server Side Rendering (SSR)

2. **getStaticProps** (`export async function getStaticProps(context) { ... })`

- Add this function to pre-generate a page (*Static Site Generation / SSG*) with data prepared on the server-side during a build time.
- a.k.a. AOT (Ahead Of Time) where pages are prepared ahead of time and can be cached by the server / CDN serving the app.

3. Benefits:

- SEO Friendly
- CDN Friendly
- **Ref:** <https://www.techtarget.com/searchnetworking/definition/CDN-content-delivery-network>
- Low spec clients.

4. Drawbacks:

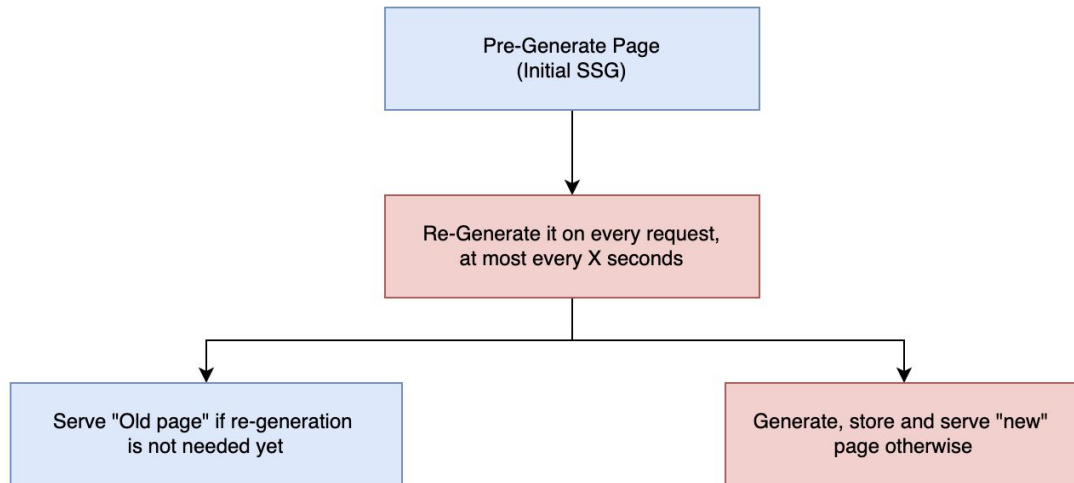
- You have no access to the actual incoming request

Lab 7.1

1. Short demo on Static Site Generation (SSG).

Incremental Site Generation (ISR)

1. **getStaticProps** can also be utilized for what being known as **ISR** (*Incremental Site Rendering / Incremental Site Generation*).
2. **ISR** might occur several times for every incoming request, at most every X seconds, e.g. every 60 seconds (i.e., not limited to the build time).



Lab 7.2

1. Short demo on the ISR.

More “getStaticProps” Config Options

1. `notFound : boolean`

e.g.

```
if (data.products.length === 0) {  
    return { notFound: true };  
}
```

2. `redirect : { destination: string, permanent: boolean }`

e.g.

```
if (!data) {  
    return {  
        redirect: {  
            destination: "/no-data",  
            permanent: false,  
        }  
    };  
}
```

Working with Dynamic Page (Dynamic Path Param)

`useRouter` alternative

1. Leverage the `getStaticProps` context to extract the dynamic path

E.g.

```
async function getStaticProps(context) {  
  const { params } = context;  
  const productId = params.pid  
  // code to filter product based on the productId  
}
```

2. Implement `getStaticPaths` since it's required for dealing with any dynamic SSG pages (i.e., page with dynamic path param).

NB: By default, a dynamic page will be following JIT (Just In Time), i.e. page will be provided at runtime when it is requested which is different from the AOT (Ahead On Time) / Pre-generated Page as the default behavior for any NextJS app.

Ref: <https://www.vitamindev.com/next-js/getstaticprops-getstaticpaths-typescript/>

Lab 7.3

Working with Fallback Pages

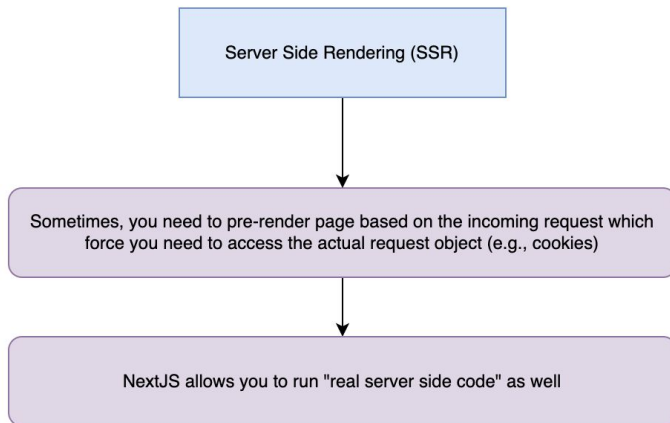
1. Useful to postpone pre-rendered (pre-generated) page, particularly for some pages those are not frequently visited.
2. Enable by setting the fallback config to either “true” or “blocking”.

Ref:

<https://stackoverflow.com/questions/67787456/what-is-the-difference-between-fallback-false-vs-true-vs-blocking-of-getstaticpa>

Server Side Rendering (SSR)

1. Another form of pre-rendering (i.e., generated on the server side).
2. Has access to the actual request object (e.g., cookies). This is what make it different from the Static-Site Generation (SSG) / Incremental Side Generation (ISG / ISR)
3. Runs on every incoming request.
4. Implemented through an async function named `getServerSideProps``



“getServerSideProps” Context Object

1. We can extract valuable information the server side props Context object such as HTTP request object and HTTP response object for further process such as:
 - Get some details information about the user that triggered the request.
 - Adding more headers into the response object before it's returned from the functional component object.

Ref:

- https://nodejs.org/api/http.html#http_class_http_incomingmessage

- https://nodejs.org/api/http.html#http_class_http_serverresponse

2. **E.g.:**

```
export const getServerSideProps: GetServerSideProps =  
  async (context) => {  
    const { params, req, res } = context;  
  }
```


Lab 7.4

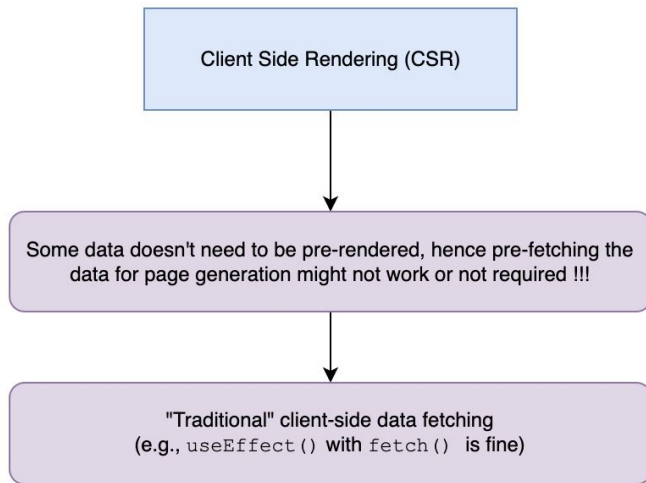
Working with “Server Side Rendering”

1. Short demo on the `getServerSideProps` usage.

Client Side Data Fetching

Client Side Rendering (CSR)

1. Happening on Client (Web) Browser, i.e. no server pre-rendering.
2. Suitable for any of these following type of applications:
 - Application with frequently changed data (e.g. Stock Data).
 - Application highly user-specific data (e.g. last orders in an online shop).
 - Application with partial data (e.g. data that's merely used on a part of a page such as dashboard).
3. Drawbacks :
 - Not SEO Friendly
 - Might depends on the client internet speed and device specs.



Ref: <https://nextjs.org/docs/pages/building-your-application/rendering/client-side-rendering>

Lab 7.5

CSR with “Traditional” approach

1. Short demo on CSR with `useEffect()` and `fetch()` methods.

CSR with SWR (“useSWR” hook)

1. Derived from “stale-while-revalidate”.
2. Similar like `useEffect` and `fetch` yet with better performance and some additional built-in features such as:
 - Caching
 - Automatic Revalidation (i.e. Optimistic update that ensure user gets the most updated data)
 - Retries on Error

Lab 7.6

CSR with “useSWR” hook

1. Short demo on CSR with `useSWR()` hook.

An aerial photograph of the New York City skyline at dusk. The sky is a mix of dark purple, blue, and orange. The city is densely packed with skyscrapers, many of which are illuminated with their interior lights. The Empire State Building is prominent in the center, with its top lit in red and green. The Hudson River is visible on the right side of the image. The text "Q & A" is overlaid in the center of the image in a large, white, serif font.

Q & A