```
- Building a simple application to understand the common flow
-------------------------------------------------------------

  Introduction: Why the simple app?
  ---------------------------------
    - Wheather you are building a simple app or a complex app, there are a lot of commonalities
that you repeat again and again. We will try to understand those commonalities by building a very
basic full stack app that is used to add a new employee to the database and alow an employee to
login back. We will use git to manage our code and deploy it to the server. Confugure our server
to serve the pages. Restructure the code to follow the standard folder structure and use APIs. We
will use the simple app we use and the standards we follow as a base for the main APP we will
build.

    Steps we will follow:
      - Set up the git repository
      - Set up the dev database locally
      - Develop the Backend
      - Develop the Frontend
      - Set up our host server on Amazon AWS (EC2)
      - Deploy our code
      - Restructure our code to follow the standard folder structure
      - Update our code base and deploy it again
      * We will then use the same flow to build the Garage App

    NOTE: Let's assume we are only building two pages of the application.
      - The add employee page and the log in page
      - Design of the two pages are provided in the design folder

  Set up the git repository
  -------------------------
    - Set up the repo as a private repo on git hub
      /sampleapp
    - Clone the repo to your local machine and start working on
      - Before you clone the code, create access token on github
      - git clone  {project link} (Use the SSH link as this is a private repo)
      - If you haven't added your SSH key to your git hub account, you need to do that first
        (We will see how to do that when we later configure our server)

  Set up the dev database locally
  -------------------------------
    - Install MySql on your local machine
      - https://dev.mysql.com/downloads/installer/
      (If you have MAMP already installed, you can use that as well)

    - Create a new database and add user to the database
      username: demoapp ? So554Dgteb766@!!
      password: demoapp
      database name: demoapp

    - Let's desing abd create the tables we need to add employee
      - Let's just simplify it to the max and create a single table with the following fields in
it.
        Table Name: employee_test
          Fields:
            - id (int, auto increment, primary key)
            - first_name (varchar)
            - last_name (varchar)
            - email (varchar, unique)
            - password (varchar)
```

```
      - We can create the above tables manually on our database
        - The better approach is to write SQL script to create the tables automatically when we
run the script
        * We will do that later once we are done with this test pages

      -- Create the tables --

  Set up the Backend
  ------------------
    Note:
      Let's just create the very basic files necessary at firts. We will then come back and
restructure our files following the industry standards. That way, you will also understand how
restructuring actually makes your life easier.

    - Create the "backend" Folder:
      - To hold all your backend codes
      - package.json (file):
        Create the initial package.json file to track your dependencies
          - npm init

    - Create the app.js (file):
      - This is the main file that we use to run and manage the backend server
      - Let us now start writing our code
        - Let's start by creating our webserver
          - Install Express
            - npm install express
          // Import the express module
          // Set up the port to listen to
          // Set up the listener

        - Let's now create a simple get request handler to send a response back
          // Create a simple get request handler to send a response back
          - Test it on the browser
            - http://localhost:4000/

        - Let's now connect to the database
          - To do that we need the mysql (mysql2) bridge module to connect to the database
          - Install the mysql2 module
            - npm install mysql2
          // Import the mysql module
            - We use the commonJS module system to import the modules
            * mysql2 is faster and has more features than mysql. But they both do the same thing
          // Define the connection parameters for the database
          // Create the connection to the database
          // Connect to the database
          - Check if the connection is successful on the console

        - Let's now design and prepare the API documentation
          - Remember:
            - There is no code to write for this step. The goal of this step is to define what is
to be send to the backend server and what is to be recieved from it.
            - We are restricted by the HTTP protocol. Meaning, we can only send and recieve data
in the form of text.
            - The server should be programed to understand the text we send to it and send us back
the data we need. Meaning, it is our (Backend developer's) responsibility to write the code that
handles the request

          - API endpoint documentations for our test pages
            - Add employee
              - Route chosen: /add-employee
                - (This path could have been anything. It is upto us to choose path name that
describes what we want the request to do)
```

```
              - Request Method: POST
              - Request Format: JSON
              - Request Sample: JSON
              {
                "first_name": "John",
                "last_name": "Doe",
                "email": "joe@gmail.com",
                "password": "123456"
              }
              - Response Format: JSON
              - Response Sample: JSON
                {
                  status: 'success',
                  message: 'Employee added successfully'
                }

            - Log in
              - Route chosen: /login
              - Request Method: POST
              - Request Format: JSON
              - Request Sample: JSON
                {
                  "email": "joe@gmail.com",
                  "password": "123456"
                }
              - Response Format: JSON
              - Response Sample: JSON
                {
                  status: 'success',
                  message: 'Employee logged in successfully',
                }

        - Let us now develope the endpoint handlers
          - Route: /add-employee
            // Post request handler to add a new employee to the database
            - Write the sql query
              - To get the values from the request body, we need to use the express.json()
middleware to parse the request body
            - Execute the query
            - Send the response back  (Success or Error)
            - Test it by sending a POST request to the endpoint using Postman

          - Route: /login
            // Post request handler to log in an employee
            - Write the sql query
            - Execute the query
            - Send the response back (Success with employee data or Error)
            - Test it by sending a POST request to the endpoint using Postman

      - Done with the backend for now

  Set up the Frontend
  --------------------
    - frontend (Folder)
      - Run the following command to create the fronend react app from the project folder
(abegaragetest)
        - npx create-react-app frontend
        (You can use Vite in here as well. But we will use CRA for now)
          - Here is a short video on how to use Vite to create a react app
            https://www.youtube.com/watch?v=vr-I2HIVmTw
      - Go to app.js and edit the page to test if it is working
        - Keep a simple text that says "Test app" in the body
```

```
      - Based on the Wireframes, we need to create the following pages
        - Home
          Path: /
        - Add Employee
          Path: /add-employee
        - Log in
          Path: /login
        * Notes:
          - Eventhough the paths above are similar with the ones we used for tha backend, since
the webservers are different, we can use the same path names.
          - We used Express.js for the backend and the webserver.
          - Apps created using create-react-app include a built-in development server for local
development purposes (Webpack Dev Server).
        - Let us now create the pages
          - Create a folder called pages and create the following files to hold the pages
            - AddEmployee.js
            - Login.js
            - Home.js
            * Enabled the Reactjs code snippets extension on VSCode to make it easier to write the
code
              - Use rsf to create a simple placeholder component on both pages. We will come back
and add the forms later.
        - Add the routes to the pages on the App.js file
          * Note:
            - Since we are structuring our App as a single page application, where app.js is the
single page that gets called all the time, we need a mechanism to send different components for
different requests. That is why we need to use Routing.
            - Create React App tool chain doesn't include page routing by default.
            - React Router is the most popular routing library for React. Start by installing and
importing that
              - npm i react-router
              // Import the Router from react-router
            - For the routing to work based on the path provided on the browser's address bar, it
needs help from another component called BrowserRouter.
              - Install the BrowserRouter
                - npm i react-router-dom
              - Import the BrowserRouter from react-router-dom (on index.js)
                // Import the BrowserRouter from react-router
              - Wrap the App component with the BrowserRouter component (on index.js)
                - This ensures that all the child components have access to the routing features
provided by React Router
          - Import the page components on the App.js file
            (These are the different components we want to send for different requests)
            // Import the page components
          - Add the routes
            - Home: /
            - AddEmployee: /add-employee
            - Login: /login
          - Test the routes on the browser
            - http://localhost:3000/
            - http://localhost:3000/add-employee
            - http://localhost:3000/login
          - Let us now add the forms to accept user inputs on these pages

            - Add Employee
              - Add a simple form with the fields defined on the design
                function AddEmployee(props) {
                  return (
                    <div>
                      <h1>Add employee</h1>
                      <form>
                        <label htmlFor="fname">First name:</label><br />
```

```
                        <input type="text" id="fname" name="fname" /><br />
                        <label htmlFor="lname">Last name:</label><br />
                        <input type="text" id="lname" name="lname" /><br />
                        <label htmlFor="email">Email:</label><br />
                        <input type="text" id="email" name="email" /><br />
                        <label htmlFor="password">Password:</label><br />
                        <input type="text" id="password" name="password" /><br /><br />
                        <input type="submit" value="Submit" />
                      </form>
                    </div>
                  );
                }
              - First, let's see what happens if we just submit data as it is now
              - Browsers are traditionally designed to send GET requests when we submit a form.
The request is sent to a url provided under the "action" form property. If there is nothing
provided there, the browser sends the request to the same url as the page.
              - The form values are sent as query parameters appended on the URL.
              - But this is not what we want to happen when we submit the form. We want to send a
POST request to the backend server. Not to the frontend server.
              - First step is to prevent the default behavior of the browser when we submit the
form.
                - For that, lets write a custom function to handle our submit event
                  // Write a function to handle the form submission
                  - Call the function on form submit event
                    <form onSubmit={handleSubmit}>
                  // Prevent the default behaviour of the form submission
              - Next step is to collect all user provided data and format it in a way that is
defined on API documentation
                - We will use the useState hook to store the form data
                  // Import useState from react
                // Declare state variables for each of the form fields
                - We will use the onChange event to update the state
                  - Add the state variables to the component
                - We will use the value property to bind the state to the form field
                - We will use the name property to identify the form field
                - We will use the onSubmit event to send the request to the backend server

                // Prepare the data to be sent to the server (Should follow the format defined on
the API documentation)
                  {
                    "first_name": "John",
                    "last_name": "Doe",
                    "email": "joe@gmail.com",
                    "password": "123456"
                  }
                // Send the data to the server
                  - Get the url from the API documentation
                  - Use the fetch API to send the request
                  ** You should recieve an error message saying that the request is blocked by
CORS policy.
                    - CORS stands for Cross Origin Resource Sharing.
                    - It is a security feature built into browsers to prevent a web application
from making requests to a different domain than the one that served the web application.
                    - In our case, the frontend is served from http://localhost:3000 and the
backend is served from http://localhost:4000.
                    - The browser allows the frontend to make requests to the backend only if the
backend specifically allows it.
                  - The browser allows the frontend to make requests to the backend only if the
backend specifically allows it.
                    - To allow that, we need to add the following headers to the response from the
backend server
                      - Access-Control-Allow-Origin: http://localhost:3000
```

```
                    - Access-Control-Allow-Methods: GET, POST, PUT, DELETE, OPTIONS
                    - Access-Control-Allow-Headers: Content-Type
                    - Access-Control-Allow-Credentials: true
                  - We can do that by adding the following middleware to the backend server
                    - app.use((req, res, next) => {
                        res.setHeader('Access-Control-Allow-Origin', 'http://localhost:3000');
                        res.setHeader('Access-Control-Allow-Methods', 'GET, POST, PUT, DELETE,
OPTIONS');
                        res.setHeader('Access-Control-Allow-Headers', 'Content-Type');
                        res.setHeader('Access-Control-Allow-Credentials', true);
                        next();
                      });
                - Submit and try again now
                  - Should work now
                  - Check if the data is added to the DB

                - Display the returned message on the /add-employee page
                  - Use the useState hook to store the message
                  // Declare a state variable to store the response from the server
                  // Save the response from the server in the state variable
                  - Display the message on the page
                  {/* Display the return message in here */}

                Note: The client is directly communicating with the backend server. Eventhough
the code was written on the frontend, it is loaded on the browser and executed on the browser. You
can disable the frontend server and the application will still work as long as it is already
loaded on the browser.

            - Login
              - Very similar to the Add Employee page
              - Lets import the useState hook to store the form data
              // Declare state variables for each of the form fields
              // Declare a state variable to store the response from the server
              // Write a function to handle the form submission
              // Prevent the default behaviour of the form submission
              // Prepare the data to be sent to the server
              // Send the data to the server
              // Store the response from the server in a state variable

      - Our code is now ready to be pushed to the server. But, once it is pushed to the server,
the localhost links need to match the one on the server and not the local host. We need to modify
that before pushing the code.
        - If we had already setup our server, it would have been easier. Since we dont have it set
up, it will be the IP address of the server. But we don't even know that yet. So, lets leave this
as is and prepare our server now

  Set up our host server on Amazon AWS (EC2)
  ----------------------------------------
    - Create new AWS account
      https://portal.aws.amazon.com/billing/signup#/start/email

      - Once it is done, let's choose the basic support plan and sign in

      - Click on the EC2 Icon (You can search and find it if it is not on the home page)
        https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#Home:

    - Create an instance
      - Steps:
        - Create a key pair (Or you can use one that exists)
        - Choose Amazon Machine Image (AMI)
          - Ubuntu Server 20.04 LTS (HVM), SSD Volume Type - ami-0dba2cb6798deb6d8 (64-bit x86) /
ami-0e2ff28bfb72a4e87 (64-bit Arm)
```

```
    - t2.micro (Free tier eligible)
    - Configure Instance Details (Default)
    - Add Storage (Default)
    - Add Tags
      - Key: Name
      - Value: NodeHello
    - Configure Security Group (Leave for now)
    - Review and Launch
    - Select the key pair
    - Launch Instances
    * Wait until status check completes

- Connect to an EC2 instance
  - Configure the security group to allow port accesses (If you didn't allow all)
    - Add custom TCP port 4000 to the secutity group
    - Add http port 80 to the security group
    - Add https port 443 to the security group
    - Add ssh port 22 to the security group

  - Using EC2 Connect (SSH)
    - This connects you to the server using the browser

  - Through the Terminal (SSH)
    - This connects you to the server using the terminal
    - Open your terminal and go to the directory where you saved the .pem file
    - Change the permissions of the .pem file
      - chmod 400 <name of the .pem file>
    - Connect to the instance
      - You can copy the command from the EC2 instance connect
      ssh -i "sampleproject.pem" ubuntu@ec2-44-210-90-78.compute-1.amazonaws.com

    - Connect using Cyberduck
      - Download Cyberduck
        https://cyberduck.io/download/
      - Open Cyberduck
      - Click on Open Connection
      - Choose SFTP (SSH File Transfer Protocol)
      - Enter the following information
        - Server: <Public DNS (IPv4)>
        - Username: ubuntu
        - SSH Private Key: <Choose the .pem file that you downloaded>
      - Click on Connect
      - Update the instance
        - sudo apt-get update

    - Generate SSH Key for github and add it
      sudo apt-get update  (If not done already)
      ssh-keygen -t ed25519 -C "support@evangadi.com"
      - Generating public/private ed25519 key pair.
      - Enter file in which to save the key (/home/ubuntu/.ssh/id_ed25519):
      - Enter passphrase (empty for no passphrase):AbeGaragePMA
      - Enter same passphrase again:AbeGaragePMA
      - Your identification has been saved in /home/ubuntu/.ssh/id_ed25519.
      cat ~/.ssh/id_ed25519.pub
      - Copy and add it to the github account
        - Go to the GitHub account
        - Go to Settings
        - Go to SSH and GPG keys
        - Click on New SSH key
        - Paste the SSH key
      - This will allow you to communicate with the private git repo
```

```
    - Install Mysql on our server
      - Install MySql
        sudo apt-get install mysql-server
        (Restart allows)
      - Change the default password for the root user
        sudo mysql_secure_installation
          - For now - No to everything
      - Check mysql status
        sudo systemctl status mysql
      - You can also change the root password like this
        Log in to MySql
          - sudo mysql
          alter user 'root'@'localhost' identified with mysql_native_password by 'AbeGaragePMA';
          (You should see Query OK, 0 rows affected (0.01 sec))
          - u = root
          - p = AbeGaragePMA
        exit
      - Log in to MySql
        mysql -u root -p

    - Install phpmyadmin to be able to manage our databases easily (What you already know)
      - Install Apache first
        sudo apt-get install apache2
      - Start the server
        sudo systemctl start apache2
        or
        sudo service apache2 start
      - Check if the server is running
        sudo systemctl status apache2
        or
        sudo service apache2 status
      - Open in browser
        http://<Public DNS (IPv4)>
        (Make sure the security group allows http port 80, 443, 3000 & 4000)
          - Set from anywhere for now
        http://44.211.126.111/ (Make sure to remove the https://)
          - You should see the apache2 Ubuntu Default Page
      - Install php as it is required by phpmyadmin
        sudo apt-get install php
      - Install PHPMyAdmin
        sudo apt-get install phpmyadmin
          - Choose apache
          - When asked if you want to configure the database for phpmyadmin with dbconfig-common,
select no
      - Configure PHPMyAdmin
        ** Vim cheat sheet
          https://devhints.io/vim
        sudo vim /etc/apache2/apache2.conf
        - Add the following line at the end of the file. (This allows Apache to include and use
the configuration settings specific to phpMyAdmin, which are stored in the /etc/phpmyadmin/
apache.conf file.)
          Include /etc/phpmyadmin/apache.conf
      - Access PHPMyAdmin
        http://44.211.126.111/phpmyadmin/
        - Username: root
        - Password: AbeGaragePMA
      - Import our local database to the production server
        - Create the database with the same user name and passcode
        - Export the database from your local machine
        - Import the database to the production server
        * If in case you don't have access to phpmyadmin, you can use the terminal to import the
database
```

```
          - Open the terminal
          - Go to the directory where you saved the database
          - mysqldump -u root -p <database name> > <database name>.sql
            - Enter the password

    - Install NodeJS
      - Ubuntu usually comes with node already installed on it. But, it is usually an older
version. We need to remove it and install the latest version.
      - Remove the old version of node
        sudo apt purge nodejs npm
      - We can install node in multiple ways. One of the ways is to use the node version manager
(nvm). So we will start by installing nvm first
        curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.33.8/install.sh | bash
      - Reload your terminal
        source ~/.bashrc
      - Check if nvm is installed
        nvm --version
      - Check the available versions of node
        nvm list-remote
      - Choose the one you want to install (Check the latest version on the node website)
        nvm install v18.17.0
      - Install npm
        sudo apt-get install npm

  Commit and push the code to github
  ----------------------------------
    - Update the API URL first
    - Before we move on and add the other pages, let's set up our git and remove files that we
don't want to track on git.
    - set up .gitignore
    - Change the localhost url to the AWS url on the frontend code
    - Comment out the socketPath used to connect to the local MAMP
    - Commit and push the code to github

  Deploy our code to the server
  -----------------------------
    - SSH into the server
    - Git clone with SSH
      - git clone

    Deploy the backend
    ------------------
      - cd to the backend folder
      - We normally do node app.js or nodmon app.js to start the server
        - For production, we will use a process manager instead
          - The reseaon is, if the server goes down for some reason, the process manager will
restart it
        - We will use PM2 for that

        ** Just to test if it is working properly, let's just run it with node app.js and see if
it works
          - If it works, then we can move on to PM2
      - Install PM2
        - sudo npm install pm2 -g
      - To start our backend app
        - pm2 start app.js (From the backend folder)
        ** Test again
        * You can also provide it a specific name
          - pm2 start app.js --name "sampleapp"
        - To stop a process
          - pm2 stop app.js (Or the id of the process)
        - To restart a process
```

```
          - pm2 restart app.js (Or the id of the process)
        - To check the status of the process
          - pm2 status
        - To check the logs of the process
          - pm2 logs (Or the id of the process)
        - To delete a process
          - pm2 delete app.js (Or the id of the process)
      - To configure PM2 to start the app automatically when the server is restarted
        - Type this command
          - pm2 startup
        - Copy the command that is generated
          - sudo env PATH=$PATH:/usr/bin /usr/lib/node_modules/pm2/bin/pm2 startup systemd -u
ubuntu --hp /home/ubuntu
        - Paste it in the terminal and hit enter
        - To save the current process list
          - pm2 save
            - To tell the server to save the list of processes that will be started automatically
      - Test the Backend
        - Open Postman
          - Add a new employee and test it
          - Log in and test it
```