1. Web Scraping:
    - Understand the importance and applications of web scraping, including tracking competitors, analyzing product prices, and data collection for machine learning.
    - Consider using headless browsers like Puppeteer and leveraging tools like Bright Data for unblockable scraping of tough websites.
2. Product Tracking and Database Management:
    - Develop a web scraping application to track product prices and store data in a database.
    - Utilize Mongoose to interact with MongoDB for efficient database management.
    - Set up Cron jobs to automate periodic price checks and updates.
3. Full Stack Development:
    - Design responsive layouts using flexbox, create navigation bars with logos and icons, and implement modals for interactive user experiences.
    - Build a Next.js application and deploy it using Vercel for efficient and dynamic rendering.
4. Email Notifications and API Development:
    - Enable functionality to add user emails for product alerts and implement a function to send periodic email notifications based on price changes.
    - Create API routes for interacting with the database and automate tasks such as data scraping and email notifications.
5. Troubleshooting and Error Handling:
    - Address issues with package compatibility, configuration errors, and undefined values in the data to ensure smooth application functionality.
    - Utilize TypeScript for type-safe development and to catch errors at compile time.

By focusing on these key areas, you can effectively develop a comprehensive web scraping and product tracking application with efficient database management, user notifications, and error handling.

Data sources for web scraping can include various types of websites, from e-commerce platforms like Amazon and eBay to social media networks, news sites, and business directories. Here are some common sources:

1. E-commerce Websites:
   - Amazon, eBay, Walmart, Etsy, and other online retail platforms provide valuable data on product prices, descriptions, and customer reviews.

2. Social Media Platforms:
   - Twitter, Facebook, Instagram, and LinkedIn can be sources for social media data including user profiles, posts, and engagements.

3. Real Estate and Property Listings:

- Websites like Zillow, Realtor.com, and Airbnb offer data on property listings, prices, and location information.

4. Business Directories:
- Platforms such as Yelp, Yellow Pages, and Google My Business provide business data, including contact information, reviews, and operating hours.

5. Financial and Stock Market Websites:
- Financial news sites, stock market platforms, and cryptocurrency exchanges offer financial data, stock prices, and market trends.

6. News Aggregators:
- Websites aggregating news articles and blogs, such as BBC, CNN, and TechCrunch, provide valuable content for data analysis and sentiment analysis.

7. Government Databases and Public Records:
- Government websites, census data, and public records can be valuable sources for demographics, public health information, and legal data.

It's important to note that when web scraping, it's essential to comply with the terms of use and adhere to legal and ethical considerations concerning data privacy and copyright.

Websites use web scraping to monitor actions, analyze competitors, and suggest products. Learn how to leverage it in this video.

Websites use web scraping to analyze competitors and monitor their brand's reputation.:

- Web scraping is used by big websites like Amazon, eBay, Twitter, and Instagram.
- Web scraping can be used to create software that notifies users when it's the best time to make a purchase.
- In this video, you will learn what web scraping is and how to build a scraping tool.
- Web scraping can be used to develop a software as a service business.
- Developers can build a scraping bot to overcome obstacles like handling captchas and anti-scraping measures.

Build a web scraping application that tracks Amazon products.:

- Scrape product details such as image, price, and description from Amazon.
- Calculate and display average, highest, and lowest prices of the product.
- Display similar scrape products to explore.
- Implement a special track option that sends email notifications.
- Automatically run the program at specified intervals using cron.
- Keep the application up to date with recent data from Amazon.
- Combine all concepts into a functional application.

Web scraping is the process of generating a dataset for machine learning.:

- Web scrapers navigate through websites, find links, and rank them.
- Scraping targets specific types of websites and focuses on relevant information.
- Web scraping starts with sending an HTTP request to the website.

- The website's content, including HTML, CSS, and JavaScript, is received.
- The content is parsed to select the desired information.
- Parsing involves pinpointing elements like prices on specific web pages.
- The extracted data is stored in a database for training ML models or building apps.
- Web scraping requires writing code, but it is not extensive.

Web scraping can be done using headless browsers like Puppeteer:

- Puppeteer is a popular choice for web scraping
- Cheerio can be used in combination with Puppeteer for easier HTML parsing and data extraction

Build an unblockable web scraping tool using bright data:

- Utilize bright data's web unlocker feature to overcome website blocks
- Scrape data from Amazon websites within seconds using bright data
- Start building the tool by creating a new empty folder
- Initialize a Next.js application for creating full stack web applications

Learn how to install and use xjs with the help of documentation.:

- Follow the installation process mentioned in the documentation.
- Join the Next GS course to understand xjs concepts and develop applications.
- Copy and run specific commands to install the project and dependencies.
- Start developing the application with the initial file and folder structure.
- Access the app on localhost 3000 by running npm run Dev.

Importing and defining fonts in the application:

- The fonts can be imported from Google or elsewhere
- The fonts can be defined as subsets of Latin
- Different weights can be added to the fonts

Track product prices and save money on online shopping.:

- Effortlessly track prices
- Modify SEO metadata for better search rankings
- Create a responsive navbar component

Developing navbar using semantic HTML5 tags and Tailwind CSS.:

- Adding CSS utility classes for faster and flexible customization.
- Utilizing the nav class from globals.css to style the navigation bar.
- Ensuring accessibility by using appropriate HTML tags.
- Recommended to install Tailwind intellisense for better understanding of utility class properties.

Add a navigation bar with logo and icons:

- Import and set up logo image
- Add logo and price text in navigation bar
- Create div for icons in navigation bar
- Declare and iterate over icon elements

Create a responsive layout with flexbox:

- Apply padding and border styles to a div

- Use media queries to adjust layout on different devices
- Nest divs with flexbox classes for column display
- Add text and image elements to the layout

Create a webpage layout with primary color and search bar:

- Add a span element with a class name of primary to set the primary color
- Include a paragraph tag with a class name of Mt6 to add a margin top of 6
- Copy the description from the deployed site and paste it below the paragraph
- Render a new component called search bar
- Display the hero carousel to showcase different images
- Create a second section with a class name of trending-section
- Add an H2 tag with the text 'trending' within the section

Create a form component with className flex Flex Dash wrap and onSubmit event:

- Declare a handleSubmit function within the SearchBar component
- Use the 'useClient' directive to turn the component into a client-side runner component

Use of Next.js requires understanding server-side concepts to avoid slow client-side performance.:

- Next.js course covers server-side concepts in depth.
- Without understanding server-side concepts, the app may become slow and inefficient.
- Learning server-side concepts in Next.js empowers developers to maximize its capabilities.
- It is useful to pin important tabs for easy reference.
- The page can be server-side rendered by using 'use client' with interactivity and hooks.
- Creating an input field within a form is essential.
- Styling the input field and adding a search button improves the user interface.
- The focus can now shift to the hero Carousel component.

Creating a carousel with images in next.js:

- Importing CSS and package for Carousel
- Creating an array of hero images with URLs and alt tags
- Rendering the hero images dynamically in the carousel

Implementing a carousel in React with specific props:

- Define the image tag and provide the required attributes such as source, alt, width, height, class name, and key
- Additional props can be added to the carousel component such as show thumbs, autoplay, infinite loop, interval, show arrows, and show status
- Despite proper implementation, the page still breaks due to an error coming from react-easy-swipe/react-swipe

Implementing an arrow to draw users into a link on the application:

- Insert a new image below the carousel with the source equal to 'assets/icons/hand drawn Arrow.svg'
- Name the SVG file and icon meaningfully to easily refer to them

- Set the width and height of the image to 175
- Give the image a class name of 'Max-Dash-XL' and hide it on devices with a min width of 1280
- Position the image absolutely with left: -15px, bottom: 0, and z-index: 0
- Ensure the image looks good and disappears on smaller devices
- Use Tailwind for styling and responsiveness
- Hero Carousel and home page layout are complete, now move on to scraping and saving data

Bright Data's web unlocker is a powerful solution for scraping tough websites.:

- Scraping websites for periodic updates can be difficult and blocked by captchas and checks.
- Bright Data's web unlocker offers solutions to overcome bot detection and simulate real user behavior.
- It can unlock and scrape even the toughest websites, allowing access to public websites at scale.
- One example is scraping Amazon products from their website.
- The functionality of the application starts from the search bar and requires implementing logic.

Checking if the URL is a valid Amazon product:

- Create a function that takes in a URL
- Check if the hostname contains 'Amazon.com' or 'Amazon' followed by a country code

Verify if the input is a valid Amazon link.:

- Check if the input starts with 'https://' and ends with 'amazon.com'.
- If the input is not valid, return false.
- If the input is valid, open a try-catch block.
- Turn on the loading state.
- Perform actions within the try block.
- In the finally block, stop the loading state.

Handling asynchronous actions and error messages in product creation/update:

- Use try-catch block to handle asynchronous actions
- In the catch block, throw a new error with a template string as message
- Scrape the product by developing a custom scraper function
- Implement URL existence check and Bright Data proxy configuration in the scraper function

You can sign in to the platform and access various services:

- After signing in, you will see a dashboard with different services
- One of the services is the web unlocker tool, which can be added under 'My Proxies'
- To use the platform, you need to set up the username and password in the environment variables

The code fetches the Amazon product page using axios and cheerio.:

- The code is wrapped in a try-catch block and throws an error if there is a failure in scraping the product.

- The response from the page is logged to the console.
- Axios and Cheerio are imported and installed.

Calling scrape and store product resulted in an error:

- The console shows an error message 'failed to create update product failed to scrape network error'
- The error is coming from the scrape and store product function

Extracted product title using Cheerio and verified its working.:

- Initialized Cheerio and loaded the scraped data using Cheerio.load() function
- Extracted the product title from the loaded data using Cheerio selectors
- Logged the extracted title to verify the functionality

Extracting price from multiple spans:

- Extracting data for application
- Creating utility function to extract price from elements

Scraping worldwide websites can cause issues on localhost:

- Deploying the website can solve the problem
- Check Amazon's approach for extracting data
- Keep the scraping file updated for troubleshooting

We are trying to extract the original price from a website:

- We are searching for items with discounts
- We are using different class names to extract the original price
- The regular expression used to extract the original price needs to be fixed

Retrieve and parse image URLs:

- Images can be retrieved by selecting the 'Landing image' attribute
- The 'data dynamic image' attribute contains the URL of the image
- The resulting images can be parsed using JSON.parse

Extracting currency and discount rate from text:

- Create a function to extract currency from text
- Implement code to extract discount rate from text

Extracted clean data from the website:

- Extracted values for image URLs, title, current price, original price, discount rate
- Created a new array called price history
- Defaulted category, reviews count, and number of stars to some values
- Handled out of stock variable
- Cleaned the data to create an application based on it

The text discusses the need to modify and update web scrapers when Amazon changes its website structure.:

- The provided utils.ts file in the GitHub gist can be used to keep the scraper up to date.
- The text also mentions some updates to the formatting of numbers and the extract price function.

- Specific functions like extract price, extract currency, get highest price, get lowest price, get average price, and get email notification type use values from the price history item type.
- A new types folder is created, and the index.ts file contains various types such as product, notification type user, and price history item.

We need to scrape product data from Amazon and store it in our database.:

- The data includes URL, currency, current price, original price, title, category, price history, stars, stock status, description, lowest price, highest price, and average price.
- We will periodically check for price changes and send automatic email alerts to users when the price drops.
- We need to find or create the product in the database to keep track of price changes.

The focus is on interacting with the database using Mongoose.:

- Install and import Mongoose to connect to the MongoDB database.
- Create a variable to track the connection status.
- Export an async function to connect to the database.
- Develop the function to set mongoose's strict mode.

Connect to MongoDB database using Node.js driver and fix a typo in the code:

- Add the IP address to the network access
- Copy the connection string and paste it in the .env file
- Enter the username and password in the .env file
- Rerun the application to read the updated .env file
- Fix the typo in the code to properly call the Mongoose function

Successfully connected to MongoDB and created a model for the product schema.:

- No need for console logging anymore.
- We can now find existing products or create new instances in the database.

Creating a product model and updating price history:

- Creating a product model using Mongoose
- Finding a product based on its URL
- Updating the price history of an existing product

Import 'get lowest price' from utils and update price history:

- Ensure to import 'get lowest price' from utils and pass the updated price history
- Give the updated price history a type of any
- Similar process for 'get highest price' and 'get average price'
- Handle the error for 'average price' later
- Create a new product instance in the database, updating if it already exists

Creating a Cron job to fetch and update product prices automatically:

- The Cron job will be set up to run at a specific time and interval
- The job will scrape data from Amazon to fetch new prices of products
- The fetched data will be saved in the database and displayed on the website

Server-side and client-side functionality in a full stack application:

- Server-side render pages with client components

- Server actions connect to database and perform actions

Create a function to fetch products from a database and import it into a page:

- To fetch products, use the 'get all products' function from the lib actions
- Import the 'get all products' function at the top of the page and utilize it
- In Next.js, fetching data from the database is simplified, no need for useEffect or useState
- Call the 'get all products' function by assigning the result to a constant
- Loop over the products obtained from the database

Maximum call stack exceeded error occurred due to complex properties in the database:

- The error occurred because the product is now more complex and contains more properties
- Mongodb has a special way of creating complex documents in the database
- We need to return a card with more information about each individual product

Create a div for product details:

- Create a div with class 'flex' and 'flex-call' for product content
- Render an H3 with class 'product-title' to display the product title
- Create another div with class 'flex' and 'justify-between' for descriptive content
- Render a P tag with class 'text-black opacity-50 text-lg capitalize' to display the product category
- Render a P tag with two span elements to display currency and current price

We can get the ID of a product from params in this component or page:

- The ID is obtained through params by destructuring it
- Once we have the ID, we can fetch all the product details using the ID

Create a structured div with classes to display products and their details.:

- Use a div with class 'flex justify-between items-start gap-5 flex-wrap padding-bottom-6' to display products.
- Create a container div with class 'flex gap-3' to wrap the product details div.
- Render product title and link within the product details div.
- Style the product title to make it look like a title.
- Create a link component with 'next/link' to open the product URL in a new tab.
- Style the link to look like a button.
- Create a div with class 'flex items-center gap-3' to wrap the heart and other icons.

Render two icons in a div:

- The first icon is a red heart image with a size of 20x20
- The second icon is a circular image with a white background

Price currency is displayed:

- The original price is shown with a crossed-out amount
- The product's number of stars are displayed

Create price cards within a div:

- Div should have class name of margin-y-7 to add space
- Flex column to display cards in a column

- Flex gap-5 and flex-wrap to wrap cards
- Cards should be created as a reusable component

Create a price card with title, icon, and value:

- Define interface props with title, icon src, and border color
- Create a div with dynamic class name and border color
- Render title with styled P tag
- Render icon with self-closing image tag
- Render value with styled P tag

The cards have different icons and values, but the border color is not changing dynamically.:

- The cards display different icons and values based on the data.
- Changing the border color dynamically does not work with Tailwind CSS.
- Eliminating the border color keeps all four borders the same.

The elements will fall nicely within it:

- Create an H1 with a class name
- Create an H3 for the product name
- Render a div with class name 'flex'
- Display the product description
- Create a buy now button that renders an image

Implement a functionality to even link to multiple websites:

- Add a regular looking button with an href that points to the desired website
- Give the button a class name of 'text-base text-white'
- Implement a functionality to link to similar products to enhance the app
- Create a function 'getSimilarProducts' that accepts a product ID
- Fetch the current product using 'Product.findByID'
- If no current product is found, return null
- Fetch similar products using 'Product.find' and limit the results to three
- Return the similar products excluding the current one

Implement cron jobs to track changing prices:

- The application needs to track whether prices are changing.
- Cron jobs will be implemented to execute functions within a specific time frame.
- This will allow for the refetching of new data on demand or on schedule.

Fix the undefined values by setting a default value in the product Details page:

- Instead of providing zero in each case, set a default value using the format number function
- This ensures that the value is a number and avoids undefined values
- The average price is currently zero, but it can be calculated later
- The current price is 79, which is the only value of interest
- Fix the labels on the different values to reflect their purpose
- The lowest price is currently the same as the current price, indicating it's a good time to buy
- The highest price is 180, which can be tracked over time
- Properly scrape all data for similar products
- Implement cron jobs to periodically check if the price has changed

- Consider sending an email notification based on price changes

Implementing click functionality on a button:

- Create a function 'openModal' to set isOpen to true
- Create another function 'closeModal' to set isOpen to false
- Wrap the modal content in a transition element for a slow opening animation
- Add a dialog as a div inside the transition element
- Add an onClick event to the dialog to call 'closeModal'
- Add a 'className' property to the dialog with the value 'dialogContainer'
- Wrap everything in a div for overlaying effect

Create a dialog overlay and model for animation:

- Set class name and properties for dialog overlay
- Add additional properties for transition child
- Set properties for transition child as the actual model

Add an image with a close button and a label for email address input:

- Add an image with source='/assets/icons/x-close.svg' and ALT tag
- Set width and height of the image to 24
- Set class name='cursor-pointer' and on click property='close modal'
- Add an H4 tag with text 'Stay updated with product pricing alerts'
- Add a P tag with text 'Never miss a bargain again with timely alerts'
- Add a form element with class name='email'
- Add a label for email input with text 'Email Address'
- Add an input below the label for entering email address

Create a modal with an input field and a submit button:

- Add an image for the search icon in the modal
- Create an actual input field with required and type attributes
- Add a placeholder to the input field
- Create a submit button with a class name and type attributes
- Style the H4 and P tags
- Implement state for the modal's opening, loading, and input

Implementing add user email to product functionality.:

- This function adds a user's email to a product.
- It checks if the product exists and if the user already exists on the list of users tracking that product.
- If the user does not exist, it adds the email to the list of users for that product.
- It then saves the product and generates the email contents using the generate email body function.

Create a node mailer function to generate email body:

- Import the node mailer package
- Define different types of notifications
- Create a function to generate email body
- Prepare content for the email

Create a new function 'sendEmail' which uses the node mailer package to send emails.:

- Pass the user email as a parameter when calling the function.
- Declare 'mailOptions' object with 'from', 'to', 'HTML', and 'subject' properties.
- Define the 'transporter' using 'nodemailer.createTransport' and specify pool configuration.
- Refer to the node mailer documentation for more options and configurations.

Use Hotmail credentials for sending emails through node mailer:

- Gmail is not a preferred solution for sending emails
- Authenticating with Gmail requires more complex setup
- Create an Outlook account to get Hotmail credentials
- Use the Outlook account credentials for node mailer authentication

Troubleshooting error with transporter.send email function:

- The error occurs when calling the send email function
- Transporter is defined using nodemailer.createTransport
- Node mailer package is installed, but may not be compatible with next.js

Converted generate email body function to async to fix error:

- Changed generate email body function to async to match requirements
- Added await before calling the generate email body function
- Set threshold percentage as a variable within the generate email body function
- Removed unnecessary export from the object
- No errors or warnings in the console

Implementing periodic email alerts and data scraping in the application:

- Create a new API route called 'cron' in the 'API' folder
- Implement the logic for periodic email alerts and data scraping in the 'route.ts' file

Implementing a single API route to periodically modify and interact with the database:

- The route should handle a GET request
- If there is an error, throw a new error and log it
- Connect to the database
- Fetch all products from the database
- If no products are found, throw a new error
- Scrape the latest product details and update the database
- Access and update all products in the database simultaneously

Updated product:

- The product is equal to a weight product that is found based on the URL provided
- If the product already exists, it is updated instead of creating a new one
- We need to figure out if we want to exit out of the map at this point
- The next step is to check each product's status and send email notifications accordingly
- The notification type is obtained using the get email notif type function
- The notification is then generated using the scrape product and the current product
- The lowest price does not exist on type new title storing options

Fixing errors in the script:

- Identified and fixed the misspelled word 'stock'

- Corrected the variable name from 'average' to 'average price'
- Appreciation for TypeScript for detecting the errors
- Updated logic to send email notifications to users if there are updates

Deploying Next.js applications with Vercel:

- Go to vercel.com and click Start deploying
- Login to your dashboard and select the project you want to deploy
- Push the code to GitHub and create a repository
- Copy and execute the necessary git commands to initialize the repository

Publish code on GitHub using Git commands and push to origin master:

- Run 'git init', 'git add .', 'git commit -m 'first commit', 'git branch main', 'git remote add origin', and 'git push -u origin main' to publish code on GitHub
- The code gets published on GitHub in a matter of seconds
- To add environment variables, copy the contents from '.env.local' and paste it in Versel's environment variables
- Deploy the code and wait for the deployment to complete
- If there are build errors, fix them by resolving module not found and pre-rendering errors

Learn about different options for Force Dynamic rendering and uncache data fetching of layout or Page by disabling all caching:

- Force Dynamic is going to force Dynamic rendering and uncache data fetching of layout or Page by disabling all caching
- Set revalidate to zero to improve performance
- Fix build errors and push changes for deployment
- Visit the latest deployment and go live with the updated changes
- Add a new product to the deployed price-wise website

Implementing cron jobs on Versel for serverless and Edge functions.:

- Cron jobs are time-based scheduling tools that automate repetitive tasks.
- Use kron-job.org for free and easy scheduling and logging of cron jobs.

Successfully tested and verified the new Cron job:

- Verified the fix for the undefined price issue by adding the necessary change and pushing to main
- Checked the live deployed website and ran another track request to ensure it is working
- Tested the Cron job by starting a test run and checking the logs, which showed a successful connection and execution
- Confirmed that the product information and message are being processed correctly
- Ready to create and schedule the Cron job to automatically execute at 7am tomorrow

The application allows you to track Amazon products and get their price details.:

- You can set up cron jobs to run events based on a specific schedule.
- The application is powered by bright data for web scraping.
- The project involves various concepts to make things work.