SOFTWARE ENGINEERING PROGRAM (PART TIME) INSTRUCTOR GUIDE

Product manager: **Ivor Bokun**

Date: **February 2024**

Introduction

Keynote: This document serves as a recommendation for instructors as to what is important for students to cover in QuickStart online Software engineering program and what is the expected outcome of the program. However, instructors have the flexibility to adjust their approach as they feel suitable for the best possible efficiency and effectiveness for delivering the training.

The **purpose of this document** is to provide more information to instructors for conducting online training for Software engineering bootcamp program (Part time option). This document consists of information related to topics that should be covered as well as details around Projects student will be working on. It is crucial that students receive the necessary knowledge prior to working on (and submitting) the projects which are part of this program.

The **goal of this program** is to prepare students for entry-level job roles in Software engineering/development industry, with emphasis on web development roles, such as Front-end, Back-end developer, and similar entry-level roles.

Table of content

1. Schedule example: Part-time bootcamp [Asynchronous learning]

Course	Week
1. Principles of Software Engineering I	1
2. Introduction to Web Development	2 - 3
3. Introduction to HTML and CSS	
4. Starting with Git & GitHub	4
5. Introduction to UX and Product Management	5
6. Introduction to Bootstrap	
Project 01: Basic Business or Personal Website	**6**
7. Fundamentals of Modern JavaScript - ES6 and Beyond	7-10
8. Dynamic and Interactive Web Pages - Beginners JavaScript DOM	
9. JavaScript Async	
Project 02: Website with a Search Engine	**11**
10. Data structures and algorithm theory	12
11. Principles of Software Engineering II: System Design	
12. JavaScript Objects and OOP Programming with JavaScript	13
13. Principles of Software Engineering III: Software design, UML	
14. Introduction to React	14 -15
Project 03: Task Management Web App	**16**
15. Scrum and Agile immersion	17
16. Querying Data with SQL	
17. Node.js - From Zero to Web Apps	18
18. Starting with REST APIs	19
Project 04: Nodejs Express	**20**
TOTAL	**20**

2. Principles of Software Engineering I

Timeframe: 1 week / 20 weeks

Materials: This course does not follow any official study guide.

Key note: Week by week schedule is a recommendation and instructor has the freedom to adjust the pace of covering those lectures as well as identifying for which courses students need more time to focus on.

This are the lesson that are covered in video content for **Principles of Software Engineering I** course, which instructors can cover in their weekly sessions with students:

- **Module 1: Software Engineering**
    - Lesson 1: Software
    - Lesson 2: Engineering
    - Lesson 3: Development process

- **Module 2: Software Development Lifecycle**
    - Lesson 1: Lifecycle Overview & Requirement
    - Lesson 2: Specification
    - Lesson 3: Design & Implementation
    - Lesson 4: Testing & Maintenance

- **Module 3: Lifecycle Models and Processes**
    - Lesson 1: Lifecycle Models
    - Lesson 2: Agile and Scrum

- **Module 4: The Project Team**
    - Lesson 1: Product/ Project Manager
    - Lesson 2: UX designer, Engineer/Architects

Learning outcomes

By the end of the course, students should be able to:

1. **Understand the Core Concepts of Software Engineering:**
    - Define software engineering and explain its importance in the development of software applications.
    - Differentiate between software engineering and other forms of engineering, understanding the unique challenges in software development.

2. **Comprehend the Software Development Lifecycle (SDLC):**
    - Describe the stages of the SDLC, including requirement gathering, specification, design, implementation, testing, and maintenance.
    - Understand the significance of each phase in the lifecycle and how they contribute to a successful software project.
3. **Identify and Apply Various Software Development Models:**
    - Compare and contrast different lifecycle models such as waterfall, iterative, and agile methodologies.
    - Demonstrate an understanding of the Agile framework and the Scrum methodology, including roles, ceremonies, and artifacts.
4. **Design and Specification:**
    - Explain the importance of proper design and specification in the development process.
    - Outline the basic principles of software design and how specifications guide the development process towards meeting user requirements.
5. **Testing and Maintenance of Software Projects:**
    - Understand the role of testing in the software development lifecycle and describe various testing methods.
    - Recognize the importance of maintenance in extending the life of software systems and ensuring their continued effectiveness.
6. **Participate Effectively in a Software Project Team:**
    - Identify the roles and responsibilities of key project team members, including product/project managers, UX designers, and engineers/architects.
    - Demonstrate an understanding of how effective collaboration and communication within a project team contribute to the success of software projects.
7. **Apply Principles of Software Engineering to Real-world Problems:**
    - Utilize knowledge of software engineering principles to analyze, design, and propose solutions to real-world software problems.
    - Critically assess the applicability of different software development models to various project types and scenarios.

3. Introduction to Web Development

Timeframe: 2 - 3 week / 20 weeks

Materials: This course does not follow any official study guide.

Key note: Week by week schedule is a recommendation and instructor has the freedom to adjust the pace of covering those lectures as well as identifying for which courses students need more time to focus on.

This are the lesson that are covered in video content for **Introduction to Web Development** course, which instructors can cover in their weekly sessions with students:

- **Module 1: What Does a Web Developer Do?**
    - Topic 1A: Why Web development and Who is it for?
    - Topic 1B: Jobs Outlook - Web Development
    - Topic 1C: Frontend and Backend Development
    - Topic 1D: Technologies in Frontend Development
    - Topic 1E: HTML, CSS, JavaScript in Action
    - Topic 1F: Bootstrap Test Drive
    - Topic 1G: Introducing TypeScript
    - Topic 1H: A Gentle Introduction to React
    - Topic 1I: Technologies in Backend Development
    - Topic 1J: SQL and NoSQL
    - Topic 1K: Python + Django
    - Topic 1L: Node.js and API Development
    - Topic 1M: How to Get Hired as a Web Developer?
    - Topic 1N: Portfolio and GitHub Profile
    - Topic 1O: Common Myths – Getting Hired as a Developer

- **Module 2: Installing Software for Web Development**
    - Topic 2A: Browsers
    - Topic 2B: Node.js and npm
    - Topic 2C: Git and Github
    - Topic 2D: Visual Studio Code

- **Module 3: Folders, Files, Terminal, Computer Networks**
    - Topic 3A: Introduction
    - Topic 3B: Operating Systems
    - Topic 3C: Using the Terminal
    - Topic 3D: Computer Networks
    - Topic 3E: Checkpoint

Learning outcomes

By the end of the course, students should be able to:

1. **Explain the Role of a Web Developer**: Describe what web developers do, differentiate between front-end and back-end development, and understand the technologies used in both.
2. **Understand the Job Market**: Summarize the job outlook for web developers and identify key skills and areas of growth within the field.
3. **Identify Key Technologies**:
    a. Front-end Development: List and describe the core technologies used in front-end development, including HTML, CSS, JavaScript, Bootstrap, TypeScript, and React.
    b. Back-end Development: Outline the primary technologies in back-end development, such as SQL, NoSQL, Python with Django, and Node.js for API development.
4. **Apply Basic Web Technologies**: Demonstrate a basic understanding of HTML, CSS, and JavaScript by creating a simple web page. Test drive Bootstrap and understand the role of TypeScript and React in development.
5. **Navigate the Hiring Process**: Discuss strategies for getting hired as a web developer, including building a portfolio, managing a GitHub profile, and debunking common myths about getting hired as a developer.
6. **Install Essential Software**: Successfully install and configure web development software, including browsers, Node.js with npm, Git with GitHub, and Visual Studio Code.
7. **Understand the Development Environment**: Describe the purpose of each software tool in the web development process and how they contribute to efficient development workflows.
8. **Understand Basic Computing Concepts**: Describe the basics of operating systems and their role in web development. Navigate the file system and utilize terminal commands for development tasks.
9. **Explain Computer Networks**: Understand the fundamental concepts of computer networks and their importance in web development, including how the internet works and the basics of server-client communication.
10. **Checkpoint**: Demonstrate the ability to set up a development environment, manage files and folders for a web development project, and apply basic terminal commands. Show an understanding of how web pages are served and accessed over the internet.

4. Introduction to HTML and CSS

Timeframe: 2 - 3 week / 20 weeks

Materials: This course does not follow any official study guide.

Key note: Week by week schedule is a recommendation and instructor has the freedom to adjust the pace of covering those lectures as well as identifying for which courses students need more time to focus on.

This are the lesson that are covered in video content for **Introduction to HTML and CSS** course, which instructors can cover in their weekly sessions with students:

- **Module 1: HTML Fundamentals**
  - Topic: Your First HTML Page
  - Topic: Documentation, Live Server
  - Topic: Headings & Paragraphs, HTML Tag Structure
  - Topic: Lists
  - Topic: Images
  - Topic: Identifiers and Links
  - Topic: Block and Inline Elements – div and span
  - Topic: HTML5 Semantic Elements

- **Module 2: HTML Forms and Multimedia**
  - Topic: Basic Form Fields
  - Topic: Checkboxes, Radio Buttons, Select, Slider
  - Topic: Embedding Video and Audio

- **Module 3: CSS Fundamentals**
  - Topic: How does CSS work?
  - Topic: Writing CSS selectors
  - Topic: Fonts, CSS reset
  - Topic: Text decoration
  - Topic: Styling lists
  - Topic: Display modes - block, Inline, Inline-Block, Flex
  - Topic: CSS Units
  - Topic: Box model
  - Topic: Styling and designing web layouts
  - Topic: Background images
  - Topic: Styling the form

Learning outcomes

By the end of the course, students should be able to:

1. **Create Basic Web Pages**: Understand and apply the basics of HTML to create structured web pages including using headings, paragraphs, lists, images, identifiers (IDs and classes), and links.

2. **Utilize HTML Tag Structure**: Demonstrate the ability to use HTML tags correctly, including the understanding of block and inline elements as well as the implementation of div and span tags for layout control.

3. **Employ HTML5 Semantic Elements**: Apply semantic elements appropriately to improve the accessibility and SEO of web pages.

4. **Construct HTML Forms**: Create forms using basic fields, checkboxes, radio buttons, select menus, and sliders, understanding their use cases and how they contribute to user interaction on the web.

5. **Embed Multimedia Content**: Embed video and audio content into web pages, enhancing the multimedia experience of the website while considering usability and accessibility.

6. **Apply Basic CSS Styling**: Use CSS to style HTML elements including fonts, text decorations, and lists, and reset default browser styling for consistent cross-browser rendering.

7. **Write CSS Selectors**: Effectively use CSS selectors to target and style specific elements, groups of elements, or elements based on their state or attributes.

8. **Understand and Apply the CSS Box Model**: Apply the CSS box model concept to control layout through manipulation of margins, borders, padding, and content dimensions.

9. **Implement Display Modes**: Use CSS to change element display modes (block, inline, inline-block, flex) and understand the use cases for each.

10. **Use CSS Units and Display Properties**: Apply different CSS units (e.g., px, em, rem, %) appropriately for responsive design and use CSS properties to manage spacing, sizing, and other layout aspects.

11. **Design Web Layouts**: Utilize CSS to create appealing web layouts and designs, including the use of background images and styling forms to improve aesthetics and user experience.

5. Starting with Git & GitHub

Timeframe: week 4 / 20 weeks

Materials: This course does not follow any official study guide.

Key note: Week by week schedule is a recommendation and instructor has the freedom to adjust the pace of covering those lectures as well as identifying for which courses students need more time to focus on.

This are the lesson that are covered in video content for **Starting with Git & GitHub** course, which instructors can cover in their weekly sessions with students:

- Module 1: Introduction to GIT
- Module 2: What are Repositories?
- Module 3: Basic Snapshotting
- Module 4: Branching and Merging
- Module 5: Sharing and Updating Projects
- Module 6: Inspecting and Comparing
- Module 7: Stashing and Cleaning
- Module 8: Advanced Tools
- Module 9: Tagging
- Module 10: Course Conclusion

Learning outcomes

By the end of the course, students should be able to:

1. **Understand the Basics of Version Control:**
    - Explain the importance and benefits of using version control systems.
    - Identify the key features and advantages of Git as a version control system.
2. **Navigate Git Repositories:**
    - Initialize new Git repositories and clone existing ones.
    - Understand the structure and purpose of repositories in Git.
3. **Perform Basic Snapshotting:**
    - Add and modify files in a repository and stage those changes.
    - Commit changes to the repository with meaningful commit messages.
    - Explain the concept of the working directory, staging area, and repository in the context of Git.

4. **Utilize Branching and Merging:**
   - Create, list, and delete branches in a Git repository.
   - Perform merges to integrate changes from different branches.
   - Resolve merge conflicts when they occur.
5. **Share and Update Projects:**
   - Push changes to a remote repository.
   - Fetch and pull changes from a remote repository to keep the local repository up-to-date.
   - Understand the workflow of collaboration in Git and GitHub.
6. **Inspect and Compare Changes:**
   - Use Git log to inspect the history of changes.
   - Compare changes across commits, branches, and tags.
7. **Manage Stashing and Cleaning:**
   - Stash changes temporarily to switch contexts or clear the working directory.
   - Clean the working directory by removing untracked files.
8. **Employ Advanced Git Tools:**
   - Use rebases as an alternative to merging and understand its implications.
   - Leverage advanced tools and commands for managing and troubleshooting Git repositories.
9. **Implement Tagging in Git:**
   - Create, list, delete, and check out tags.
   - Understand the use of tagging for marking release points or significant changes.
10. **Conclude the Course with Practical Applications:**
    - Apply the concepts learned in a real-world or simulated project scenario.
    - Reflect on the importance of version control and collaborative development with Git and GitHub

6. Introduction to UX and Product Management

Timeframe: week 5 / 20 weeks

Materials: This course does not follow any official study guide.

Key note: Week by week schedule is a recommendation and instructor has the freedom to adjust the pace of covering those lectures as well as identifying for which courses students need more time to focus on.

This are the lesson that are covered in video content for **Introduction to UX and Product Management** course, which instructors can cover in their weekly sessions with students:

- **Module 1: Product management**
  - Topic: Software quality
  - Topic: Software development is teamwork
  - Topic: What is Product Management?
  - Topic: PM Responsibilities in the development lifecycle
  - Topic: Product lifecycle
  - Topic: Software development models
  - Topic: Managing requirements
  - Topic: Technical writing

- **Module 2: User experience design**
  - Topic: Introduction: what is user experience design?
  - Topic: The UX pyramid and its influence on UX design
  - Topic: Developing user personas and value propositions
  - Topic: UX prototypes, models, and their evaluation
  - Topic: User Stories
  - Topic: Wireframes
  - Topic: Draw.io example
  - Topic: UI Design, component libraries, design systems

Learning outcomes

By the end of the course, students should be able to:

- **Understanding Software Quality:** Students will grasp the concept of software quality and its significance in product development.
- **Team Collaboration:** Learners will appreciate the collaborative nature of software development and the importance of teamwork.
- **Product Management Definition:** Students will define what product management entails and its role within the development lifecycle.
- **Responsibilities in Development Lifecycle:** Understanding the specific responsibilities of product managers throughout the product development lifecycle.

- **Product Lifecycle:** Familiarity with the stages of a product's lifecycle and their implications for product management decisions.
- **Software Development Models:** Knowledge of different software development models and their applicability to various project scenarios.
- **Managing Requirements:** Techniques for effectively managing and prioritizing project requirements.
- **Technical Writing Skills:** Developing proficiency in technical writing for clear communication of project requirements and specifications.
- **Introduction to UX Design:** Understanding the fundamental principles and concepts of user experience design.
- **The UX Pyramid:** Grasping the hierarchical structure of user experience design and its influence on the overall user experience.
- **User Personas and Value Propositions:** Ability to create user personas and articulate value propositions to address user needs.
- **UX Prototypes and Evaluation:** Techniques for developing and evaluating UX prototypes to ensure usability and effectiveness.
- **User Stories:** Understanding the role of user stories in defining user requirements and guiding product development.
- **Wireframes:** Proficiency in creating wireframes to visualize and communicate interface designs.
- **Practical Example using Draw.io:** Application of wireframing tools such as Draw.io to create interface prototypes.
- **UI Design and Design Systems:** Knowledge of UI design principles, component libraries, and design systems for consistent and efficient design implementation.

QUICKSTART

7. Introduction to Bootstrap

Timeframe: week 5 / 20 weeks

Materials: This course does not follow any official study guide.

Key note: Week by week schedule is a recommendation and instructor has the freedom to adjust the pace of covering those lectures as well as identifying for which courses students need more time to focus on.

This are the lesson that are covered in video content for **Introduction to Bootstrap** course, which instructors can cover in their weekly sessions with students:

- **Module 1: Bootstrap Fundamentals**
  - 01 - Installation
  - 02 - My first Bootstrap page
  - 03 - Using Bootstrap Components
  - 04 - Installing Bootstrap locally
  - 05 - Example component customization: Carousel
  - 06 - Text color, background color
  - 07 - Bootstrap font styling classes
  - 08 - Bootstrap Themes

- **Module 2: Bootstrap Grid System and Components**
  - 01 - Web project introduction
  - 02 - Breakpoints in Bootstrap
  - 03 - Solution of Exercise 1
  - 04 - Bootstrap Containers
  - 05 - Solution of Exercise 2
  - 06 - Bootstrap Rows
  - 07 - Margins, paddings, gutters
  - 08 - Exercise 3 and its solution
  - 09 - Main area styling
  - 10 - Solution of Exercise 4
  - 11 - Map and Footer
  - 12 - Solution of Exercise 5: Overflow
  - 13 - Bootstrap Icons
  - 14 - CSS cleanup

- **Module 3: Bootstrap components and forms**
  - 01 - Bootstrap components
  - 02 - Navbar
  - 03 - Solution of Exercise 1: Navbar text
  - 04 - Modal Windows
  - 05 - Forms
  - 06 - Final touches and comparison

Learning outcomes

By the end of the course, students should be able to:

1. Understand the process of installing Bootstrap locally.
2. Create a basic Bootstrap page.
3. Utilize Bootstrap components effectively in web development.
4. Customize Bootstrap components, such as carousels, with practical examples.
5. Implement text and background color styling using Bootstrap classes.
6. Apply Bootstrap font styling classes to enhance typography.
7. Explore various Bootstrap themes and their applications.
8. Comprehending the concept of breakpoints in Bootstrap for responsive design.
9. Construct web layouts using Bootstrap containers, rows, and columns.
10. Manage margins, paddings, and gutters within Bootstrap grid system.
11. Design and style main areas of a webpage using Bootstrap utilities.
12. Incorporate Bootstrap icons into web projects.
13. Conduct CSS cleanup for improved code readability and maintainability.
14. Implement Bootstrap components like navbars and modal windows for enhanced user interaction.
15. Create and customize navigation bars with text and other elements.
16. Utilize modal windows to display content or forms dynamically.
17. Develop forms using Bootstrap form components.
18. Perform final touches on web projects and compare different design approaches

8. Project 01: Basic Business or Personal Website

Timeframe: week 6 / 20 weeks

This project will kickstart your journey and assess your understanding of previous weeks' material. It provides hands-on experience in the real world, enabling you to develop a website for your own business.

Introduction

You are tasked with crafting an informational website tailored to either a freelance professional or a business, whether it's real or imaginary. Your project may involve building a portfolio site for a photographer friend or designing a website for a fictional pizza restaurant.

While you have total control over the topic and depth of information contained in the site, you will be evaluated on your ability to meet both the technical requirements and workflow requirements.

There are screenshots of an example project included in the next few slides. Your project does not have to look identical to this. It is to be used as inspiration!

Key Areas Covered

- HTML
- CSS
- Git/GitHub workflow

Workflow Requirements

The following requirements are related to how you go about building your project

- **Planning phase** - *planning should be completed prior to beginning the development phase and/or touching any code.*

  User stories
  1) Write out at least three user stories that describe how a user will interact with your site

  Wireframes
  2) Go ahead and create wireframes of each page of your site--they should include important content and user interface elements such as images, links, buttons etc.

- *Wireframes can be done on paper or using any number of widely available applications. A free one that may be useful is* draw.io.
-

  Development phase
  3) Create a GitHub repository on Github.com (before you start coding)
  4) Clone it to your local machine (before you start coding)
  5) Make frequent commits throughout your development that are descriptive, such as "adds nav bar to home page" (throughout development/coding process)

Technical Requirements

The following requirements are related to what your ***code*** *should contain*

1) Your site should either be comprised of at least three separate .html pages or you should have at least three separate sections in your single html page, where such that a menu can navigate between these sections.
2) Your site should have a *style.css* file in your */css* folder that contains your custom css
3) All of your *.html* pages should link to and include the *style.css* , so that your stylesheet is shared site-wide
4) Your pages should all be linked together using <a> tags, by way of a navigation bar
5) Your site should include at least one image (<img> tag)
6) Your site should contain a form that includes at least three fields, at least two of which should be required.

e.g. Here is an example of how you could meet this requirement for a contact form.

Name	**Type**	**Validation**
First Name	input type of text	required
Last Name	input type of text	none
Email	input type of email	required
Comment	textarea	required
Submit	input type of submit	none

7) When a user submits your contact form, use formspree.io to ensure you receive the submissions (optional, for extra points)
8) At least one page on your site should make use of a multi-column layout. For example, you may want to use Bootstrap's grid system to layout rows and columns of images on a "gallery" page of your site
9) An implementation of either Bootstrap's carousel component or tabs component (optional, for extra points)
10) Your repository should be pushed to your GitHub account
11) Your HTML should be validated using an HTML validator and should not contain any errors. Only warnings and info items are accepted.

Deliverables

To successfully complete this project, you must perform the following.

1) Your user stories
2) A collection of wireframes - one for each view of your app
3) Your app source code should be available for viewing in your GitHub repository
4) A ***readme.md*** file in the root project folder that contains the following information about your project:

- Your name
- Overview/description of the project
- Details on how to use it or what functionality is offered
- Technologies Used (**.html** , **.css**)
- Ideas for future improvement (minimum of 3)

5) Your repository should contain at least 15 commits and should reflect a consistent commit history
6) Your code should be hosted on GitHub Pages.

You determine the topic of the application, and you may even choose to replace questions with forum topics in case you wanted to build a forum. Any changes you make to the project are intended to serve the purpose of creating something that resonates with you. Changes should not be made to simplify the project.

You don't have to stick to the exact wireframes. Just make sure you include all the necessary elements on your website.

Submit your GitHub link and hosting link (include everything on a word document and upload the document under Project Submission section). All deliverables should be included on GitHub and the site should be made publicly available using a hosting service.

Project Grading

To pass the project, Instructor should take below criteria into consideration while grading this project and decide whether to Pass or Fail the student.

- Functionality
- Robustness
- Creativity, styling, user experience
- Code quality.
- GitHub structure
- Documentation, Installation instructions, Comments

Mandatory requirements for grading instructor should consider:

1. The website should be hosted on GitHub Pages and the hosting link should be in the README.md file of your GitHub repository.

2. All pages of the hosted website should contain 0 validation errors using https://validator.w3.org/. Without meeting (1) + (2), the submitted project automatically receives a score of 0 points.

9. Fundamentals of Modern JavaScript - ES6 and Beyond

Timeframe: week 7 - 10 / 20 weeks

Materials: This course does not follow any official study guide.

Key note: Week by week schedule is a recommendation and instructor has the freedom to adjust the pace of covering those lectures as well as identifying for which courses students need more time to focus on.

This are the lesson that are covered in video content for **Fundamentals of Modern JavaScript - ES6 and Beyond** course, which instructors can cover in their weekly sessions with students:

- Module 1: Datatypes and Variables
- Module 2: Sequence, Selection, and Iteration
- Module 3: Working with Functions in Modern JavaScript

Learning outcomes

By the end of the course, students should be able to:

1. **Understand JavaScript Datatypes:**
    - Identify and differentiate between primitive and reference data types in JavaScript (e.g., numbers, strings, booleans, objects, arrays).
    - Demonstrate proficiency in declaring and initializing variables using appropriate data types.
2. **Manipulate Variables and Data:**
    - Perform basic operations on variables, such as assignment, arithmetic operations, string concatenation, and comparison.
    - Utilize built-in JavaScript methods to manipulate strings and arrays effectively.
3. **Scope and Hoisting:**
    - Explain the concept of variable scope and the difference between global and local scope.
    - Understand hoisting and its impact on variable declarations and function definitions.
4. **Control Flow Structures:**
    - Implement sequence, selection (if/else statements), and iteration (loops) constructs in JavaScript programs.
    - Understand the purpose and syntax of conditional statements and loop structures.
5. **Error Handling:**

- Recognize common errors in JavaScript code and use debugging techniques to identify and fix them.
- Implement error handling mechanisms using try-catch blocks to gracefully handle exceptions.

6. **Algorithmic Thinking:**
    - Apply algorithmic thinking to solve problems using JavaScript, including designing algorithms for tasks involving iteration and decision-making.
7. **Function Declaration and Invocation:**
    - Define functions using the function declaration syntax and understand the concept of function invocation.
    - Differentiate between function parameters and arguments and demonstrate proficiency in passing arguments to functions.
8. **Function Scope and Closures:**
    - Explain the concept of function scope and lexical scoping in JavaScript.
    - Understand closures and their practical applications in maintaining state and encapsulation.
9. **Functional Programming Concepts:**
    - Explore functional programming principles such as immutability, higher-order functions, and function composition.
    - Apply functional programming techniques to solve problems and write more concise and expressive code.

10. Dynamic and Interactive Web Pages - Beginners JavaScript DOM

Timeframe: week 7 - 10 / 20 weeks

Materials: This course does not follow any official study guide.

Key note: Week by week schedule is a recommendation and instructor has the freedom to adjust the pace of covering those lectures as well as identifying for which courses students need more time to focus on.

This are the lesson that are covered in video content for **Dynamic and Interactive Web Pages - Beginners JavaScript DOM** course, which instructors can cover in their weekly sessions with students:

- Introduction to JavaScript DOM course
- Setup Developer Environment
- Create an Index Page
- Script File JS
- Console Log Options
- JS Document Object
- Selecting Elements
- Selection by Tag
- QuerySelector
- QuerySelectorAll
- Complex CSS querySelectorAll
- Update Images
- ChildNodes Children and More
- Element Style Update
- Multi Element Style Update
- Add Classes to Elements
- Set Attribute
- JavaScript Create Element
- Append and Prepend
- AddEvent Listeners
- AddEvent Multiple Listeners
- Mouse Move Events

- Fun with Images
- Form Values
- Annoying Blinker
- Animation JavaScript
- DOM Conclusion

Learning outcomes

By the end of the course, students should be able to:

1. Understand the role of JavaScript in creating dynamic and interactive web pages.
2. Set up a developer environment for JavaScript programming.
3. Create an index page and link it to a JavaScript script file.
4. Utilize console log options for debugging and displaying information in the browser console.
5. Comprehend the JavaScript Document Object Model (DOM) and its structure.
6. Select elements from the DOM using various methods, including by tag and through querySelector.
7. Apply complex CSS queries using querySelectorAll.
8. Update images dynamically on a web page using JavaScript.
9. Manipulate the DOM's structure with methods like childNodes, children, and more.
10. Update element styles dynamically using JavaScript.
11. Add and remove classes to/from DOM elements using JavaScript.
12. Set attributes dynamically for DOM elements using JavaScript.
13. Create new DOM elements dynamically using JavaScript.
14. Append and prepend elements to the DOM using JavaScript.
15. Implement event listeners to trigger JavaScript functions based on user interactions.
16. Handle multiple event listeners efficiently in JavaScript.
17. Utilize mouse move events to create interactive experiences on web pages.
18. Explore creative ways to manipulate images dynamically using JavaScript.
19. Access and manipulate form values using JavaScript.
20. Implement simple animations using JavaScript

11. JavaScript Async

Timeframe: week 7 - 10 / 20 weeks

Materials: This course does not follow any official study guide.

Key note: Week by week schedule is a recommendation and instructor has the freedom to adjust the pace of covering those lectures as well as identifying for which courses students need more time to focus on.

This are the lesson that are covered in video content for **JavaScript Async** course, which instructors can cover in their weekly sessions with students:

- Module 1: JavaScript Promises
- Module 2: Orchestrating Promises – Promise.all and allSettled
- Module 3: ES2018+ Async-await Syntax

Learning outcomes

By the end of the course, students should be able to:

1. **Understanding Promises**: Students will grasp the concept of promises in JavaScript, including their purpose and how they differ from traditional callback-based asynchronous programming.
2. **Creating and Consuming Promises**: Learners will be able to create their own promises and consume them using methods such as then, catch, and finally.
3. **Error Handling**: Students will learn how to handle errors in promise chains using the catch method and chaining multiple promises together.
4. **Promise Chaining**: Ability to chain multiple promises together to execute asynchronous operations sequentially or in parallel.
5. **Promise.all Method**: Understanding how to use Promise.all to execute multiple promises concurrently and handle the results collectively.
6. **Promise.allSettled Method**: Learning about the Promise.allSettled method and how it differs from Promise.all, specifically in handling promise rejections.
7. **Error Handling in Parallel Operations**: Ability to handle errors in parallel promise operations and ensure all promises are settled correctly.
8. **Introduction to Async-await**: Understanding the purpose and benefits of the async-await syntax introduced in ES2018 for asynchronous programming.

9. **Using Async Functions**: Ability to define and use async functions to simplify asynchronous code and make it more readable.
10. **Awaiting Promises**: Learning how to use the await keyword to pause the execution of an async function until a promise is resolved or rejected.
11. **Error Handling with Async-await**: Understanding how errors are handled in async-await functions using try-catch blocks.

12. Project 02: Website with a Search Engine

Timeframe: week 11 - 10 / 20 weeks

This project will kickstart your journey and assess your understanding of previous weeks' material. It provides hands-on experience in the real world, enabling you to develop a website with a search engine.

Introduction

This project will require you to sharpen your CSS and JavaScript expertise, alongside your proficiency in conducting research beyond the scope of what's already been covered—an essential skill for developers in their regular practice.

The goal of this project is to build out your own Giphy search engine that:

- allows users to enter and submit a keyword
- makes a request to the Giphy api based on that keyword
- receives and parses the response
- displays images on the page from that response
- uses a custom css grid to display the images
- ensures the site is responsive so that it looks good/works well on both desktop and mobile devices

Alternatively, you can create any JavaScript projects using a different API assuming that functionality of your project is comparable to the Giphy example. If you choose to implement a custom project, make sure to find an API that resonates with you and make it possible to search its contents via a form on your website.

Workflow Requirements

This project requires a bit more work with APIs than was directly covered in the material. You are encouraged to watch the following video that walks through the process of using jQuery/Ajax to request data from and API, and display images on screen.

Helpful, relevant video walk-through

The following requirements are related to how you go about building your project

The planning phase should be completed prior to beginning the development phase and/or touching any code.

Wireframes can be done on paper or using any number of widely available applications. A free one that may be useful is draw.io.

Planning phase
1) Create at least 3 user stories
2) Create wireframes for desktop and mobile views

Development phase

3) Create a GitHub repository on Github.com (before you start coding)
4) Clone it to your local machine (before you start coding)
5) You will need to obtain an API key through Giphy
6) Review the documentation for the "Search Endpoint"
7) Make frequent commits throughout your development that are descriptive, such as "adds todos reducer" (throughout development/coding process)

Technical Requirements

The following requirements are related to what your code should contain

1) Your site only needs to contain one HTML page, but there should still be multiple links in your menu (even if they don't link to other pages)
2) There should be an input field (with a type of search) & a submit button
3) A user should be able to type in a search phrase, click submit, and your site should query the Giphy API based on the search expression that your users enter
4) Iterate through the returned data, and for each returned object in the array, find an image in the returned JSON and append that image to the screen
5) Your project should contain three files: ***index.html***, ***style.scss***, ***style.css*** and ***main.js*** and your project directory structure should look like this:

```
/                         (folder)
index.html
css/                      (folder)
    style.css
    style.scss
js/                       (folder)
    main.js
```

Design
5) Your styling should all be done in your *style.scss* file (SASS), and you should use a CSS Preprocessor to watch that file for changes and output it to your ***style.css*** file. (Optional – for extra

points. A solution with a simple CSS file is also accepted)

Site-wide
6) Choose a Google Font that you've never heard of before and use that to style your site title
Here is a helpful video!

Desktop
7) Use *flex* to ensure your site's name and nav bar appear aligned to the sides of the header (like in the screenshot below)
8) Use *flex;* to ensure your input field and submit button appear side by side
9) Create your own custom grid classes in css, that make use of the *flex* property, and leverage these in your code so that the images appear in rows and columns

Mobile (320px and below)
Use a @media query to ensure that:

10) Your site title and navbar stack vertically
11) Your navigation items stack vertically
12) Your images stack vertically in a single column

Deliverables

1) Your user stories
2) A collection of wireframes - one for each view of your app
3) Your app source code should be available for viewing in your GitHub repository
4) A ***readme.md*** file in the root project folder that contains the following information about your project:

- Your name
- Overview/description of the project
- Details on how to use it or what functionality is offered
- Technologies Used (*.html* , *.css*)
- Ideas for future improvement (minimum of 3)

5) Your repository should contain at least 15 commits and should reflect a consistent commit history
6) Your code should be hosted on GitHub Pages.

Submit your GitHub link and hosting link (include everything on a word document and upload the document in Project Submission section). All deliverables should be included on GitHub and the site should be made publicly available using a hosting service.

Please note, similarly to the first project, you will be evaluated on your ability to meet both the **workflow requirements** and the **technical requirements**.

Project Grading

To pass the project, Instructor should take below criteria into consideration while grading this project and decide whether to Pass or Fail the student.

- Functionality
- Robustness
- Creativity, styling, user experience
- Code quality.
- GitHub structure
- Documentation, Installation instructions, Comments

13. Data structures and algorithm theory

Timeframe: week 12 / 20 weeks

Materials: This course does not follow any official study guide.

Key note: Week by week schedule is a recommendation and instructor has the freedom to adjust the pace of covering those lectures as well as identifying for which courses students need more time to focus on.

This are the lesson that are covered in video content for **Data structures and algorithm theory** course, which instructors can cover in their weekly sessions with students:

- Module 1: Big O notation
- Module 2: Basic Data Structures: Lists, Dictionaries, Tuples, Stacks, and Queues
- Module 3: Recursion
- Module 4: Linked Lists and Binary Trees
- Module 5: Heaps and Sorting
- Module 6: Dynamic Programming

Learning outcomes

By the end of the course, students should be able to:

1. Understand the concept of algorithm efficiency and its importance in evaluating performance.
2. Analyze algorithms using Big O notation to determine their time and space complexity.
3. Apply Big O notation to compare and contrast different algorithms.
4. Identify and describe common data structures such as lists, dictionaries, tuples, stacks, and queues.
5. Implement operations (insertion, deletion, searching) on basic data structures.
6. Evaluate the suitability of different data structures for specific problem-solving scenarios.
7. Understand the concept of recursion and its applications in algorithm design.
8. Write recursive algorithms to solve problems, including base cases and recursive calls.
9. Analyze and compare iterative and recursive solutions to problems.
10. Define and implement linked lists and binary trees, including insertion, deletion, and traversal operations.

11. Understand the advantages and disadvantages of linked lists and binary trees compared to other data structures.
12. Solve problems using linked lists and binary trees, such as searching, sorting, and tree traversal algorithms.
13. Describe the properties and operations of heaps, including insertion, deletion, and heapify.
14. Implement sorting algorithms such as heap sort, merge sort, and quick sort.
15. Analyze the time complexity of sorting algorithms and their suitability for different input sizes and data distributions.
16. Understand the principles of dynamic programming and its applications in solving optimization problems.
17. Identify subproblems and overlapping subproblems in a given problem.
18. Develop dynamic programming solutions for problems such as knapsack, longest common subsequence, and Fibonacci sequence.

14. Principles of Software Engineering III: System Design

Timeframe: week 12 / 20 weeks

Materials: This course does not follow any official study guide.

Key note: Week by week schedule is a recommendation and instructor has the freedom to adjust the pace of covering those lectures as well as identifying for which courses students need more time to focus on.

This are the lesson that are covered in video content for **Principles of Software Engineering III: System Design** course, which instructors can cover in their weekly sessions with students:

- **Module 1: System Design Introduction**
    - Content
    - Topic: What is System Design? Why is it so Important?
    - Topic: What to do as an Architect?
    - Topic: Where is System Design? What is in System Design?
    - Topic: Distributed System, Distributed Features
    - Topic: Requirement, Requirements Details
    - Topic: Design steps
    - Topic: Data Flow
    - Topic: High-level design, Detailed design
    - Topic: Bottlenecks
    - Topic: CAP theorem
    - Topic: Redundancy

- **Module 2: System Design Elements**
    - Content
    - Topic: How to Deal with Requirements?
    - Topic: Scalability, Database Scaling
    - Topic: Storage, Databases
    - Topic: Distributed System Design Pattern
    - Topic: Load Balancer, OSI Model
    - Topic: Layer-4 vs. -7 Load Balancer, Algorithms
    - Topic: REST (Representational State Transfer), Service Properties
    - Topic: Caching, Layers, Logistics, Strategies, CDN
    - Topic: Containerization
    - Topic: Cloud Technology
    - Topic: Read Replicas
    - Topic: Sharding
    - Topic: Stateless and Stateful Systems
    - Topic: Message Queues

- **Module 3: System Design Details**

- Content
- Topic: Estimation, Latency
- Topic: Conversions and Data Types
- Topic: Traffic estimation
- Topic: Memory, Bandwidth & Storage
- Topic: MTBF: Mean Time Between Failure
- Topic: Availability
- Topic: Example: LMS web application for a bootcamp

- **Module 4: Case Study**
    - Content
    - Topic: Goal, Users and Scale
    - Topic: Requirements
    - Topic: Traffic estimates
    - Topic: Storage estimates
    - Topic: High-level System Design, Database Design

Learning outcomes

By the end of the course, students should be able to:

1. Define the concept of System Design and its significance in software engineering.
2. Identify the role and responsibilities of a system architect.
3. Recognize the components and scope of System Design, including distributed systems and associated features.
4. Describe the process and steps involved in system design, from requirements gathering to detailed design.
5. Understand the concepts of data flow and its importance in system design.
6. Analyze and address potential bottlenecks in system design.
7. Explain the CAP theorem and its implications for distributed systems.
8. Discuss the concept of redundancy and its role in system reliability and fault tolerance

15. JavaScript Objects and OOP Programming with JavaScript

Timeframe: week 12 / 20 weeks

Materials: This course does not follow any official study guide.

Key note: Week by week schedule is a recommendation and instructor has the freedom to adjust the pace of covering those lectures as well as identifying for which courses students need more time to focus on.

This are the lesson that are covered in video content for **JavaScript Objects and OOP Programming with JavaScript: System Design course**, which instructors can cover in their weekly sessions with students:

- Module 1: JavaScript Objects
- Module 2: Using Objects to Encapsulate Data and Operations
- Module 3: Prototype Extensions and Inheritance
- Module 4: JavaScript Class Syntax

Learning outcomes

By the end of the course, students should be able to:

1. Understand the concept of objects in JavaScript.
2. Learn how to create and manipulate objects using object literals.
3. Explore the use of object properties and methods.
4. Understand the principles of encapsulation in object-oriented programming.
5. Learn how to use objects to encapsulate both data and functionality.
6. Explore how to create constructor functions and instantiate objects using the **new** keyword.
7. Understand the concept of prototypes in JavaScript.
8. Learn how to extend built-in JavaScript objects using prototypes.
9. Explore inheritance in JavaScript and how it can be achieved using prototypes.
10. Understand the new class syntax introduced in ECMAScript 2015 (ES6).
11. Learn how to define classes in JavaScript using the class keyword.
12. Explore how to create class constructors, methods, and use inheritance with classes.

16. Principles of Software Engineering III: Software design, UML

Timeframe: week 13 / 20 weeks

Materials: This course does not follow any official study guide.

Key note: Week by week schedule is a recommendation and instructor has the freedom to adjust the pace of covering those lectures as well as identifying for which courses students need more time to focus on.

This are the lesson that are covered in video content for **Principles of Software Engineering III: Software design, UML**, which instructors can cover in their weekly sessions with students:

- Module 1: Software Design
- Module 2: UML Diagrams

Learning outcomes

By the end of the course, students should be able to:

1. **Understanding Software Design Principles:** Students will grasp fundamental principles of software design, including abstraction, modularity, and encapsulation.
2. **Design Patterns:** Learners will be able to identify and apply common design patterns such as Singleton, Factory, and Observer to solve software design problems effectively.
3. **Architectural Styles:** Students will learn about different architectural styles (e.g., client-server, MVC) and their applicability in software design.
4. **Component-Based Design:** Understanding the concept of component-based design and its advantages in building scalable and maintainable software systems.
5. **Design Documentation:** Ability to create and interpret design documents, including class diagrams, sequence diagrams, and flowcharts.

1. **Introduction to UML:** Students will gain a comprehensive understanding of Unified Modeling Language (UML) and its significance in software engineering.
2. **UML Diagram Types:** Ability to create and interpret various UML diagrams, including class diagrams, object diagrams, sequence diagrams, state diagrams, and activity diagrams.
3. **Modeling Software Systems:** Learners will be able to model complex software systems using appropriate UML diagrams to capture different aspects of system behavior and structure.

4. **Communication and Collaboration:** Understanding how UML diagrams facilitate communication and collaboration among software development teams by providing a standardized notation.
5. **Analysis and Design:** Applying UML diagrams as tools for both analysis and design phases of the software development lifecycle, aiding in requirements gathering, system modeling, and architectural design.

17. Introduction to React

Timeframe: week 14 - 15 / 20 weeks

Materials: This course does not follow any official study guide.

Key note: Week by week schedule is a recommendation and instructor has the freedom to adjust the pace of covering those lectures as well as identifying for which courses students need more time to focus on.

This are the lesson that are covered in video content for **Introduction to React**, which instructors can cover in their weekly sessions with students:

- Module 1: Getting Started with React
- Module 2: Functional Components
- Module 3: Class Components
- Module 4: API, lifecycle methods, useEffect
- Module 5: Deployment to GitHub Pages

Learning outcomes

By the end of the course, students should be able to:

1. **Understand the basics of React:** Students will grasp the fundamental concepts of React, including its purpose, benefits, and core features.
2. **Set up a React environment:** Students will learn how to set up a development environment for React projects, including installing necessary dependencies and tools.
3. **Create a simple React application:** Students will be able to create a basic React application from scratch, including creating components and rendering them to the DOM.
4. **Understand functional components:** Students will comprehend the concept of functional components in React and their role in building user interfaces.
5. **Create functional components:** Students will learn how to create functional components using JSX syntax and how to pass props to them.
6. **Implement reusable components:** Students will be able to create reusable functional components that can be used across different parts of their React application.
7. **Understand class components:** Students will understand the concept of class components in React and how they differ from functional components.
8. **Create class components:** Students will learn how to create class components, including defining state and handling lifecycle methods.

9. **Manage component state:** Students will be able to manage component state within class components, including updating state and handling user interactions.
10. **Fetch data from an API:** Students will learn how to fetch data from an external API using built-in browser APIs or third-party libraries.
11. **Understand lifecycle methods:** Students will understand the concept of React lifecycle methods and how they can be used to manage component behavior.
12. **Use useEffect hook:** Students will learn about the useEffect hook in React and how it can be used to perform side effects in functional components.
13. **Understand deployment concepts:** Students will understand the concept of deploying a React application to a web server and the different deployment options available.
14. **Deploy to GitHub Pages:** Students will learn how to deploy a React application to GitHub Pages, including configuring settings and deploying the application.
15. **Publish and share React applications:** Students will be able to publish their React applications to GitHub Pages and share them with others.

18. Project 03: Task Management Web App

Timeframe: week 16 / 20 weeks

This project offers hands-on experience in creating a web application, serving as a test of your current skills and a valuable asset for your future endeavors.

Introduction

Your task is to create a fully operational web application for managing "to-do" lists using React. Optionally, you may integrate Redux for state management and React Router for navigation.

In practice, React & Redux is a popular combination of tools to use on the frontend of a web app --- the part the user sees and interacts with. Most web-apps hinge on the users' ability to create/read/update/delete data, also known as performing "CRUD" operations.
Some examples of CRUD operations offered in popular React web apps are:

- Spotify allows users to create/read/update/delete playlists of songs.
- Facebook allows users to create/read/update/delete posts.
- What'sApp allows users to create/read/update/delete contacts

In production apps, all that data is typically stored in a database, and the web-app reads and writes data to that database by making calls through an Application Programming Interface (API) -- code that lives on the server and can communicate directly with the database. A visual representation of this looks something like this:

Workflow Requirements

The following requirements are related to how you go about building your project

Planning phase - should be completed prior to beginning the development phase and/or touching any code.

Wireframes can be done on paper or using any number of widely available applications. A free one that may be useful is draw.io.

User stories
1) Write out at least three user stories

Wireframes
2) Create wireframes for each view of your app

3) Write out your app's state tree (remember the contact form will manage its own state for the form fields)
4) Write out a list of the container and presentational components you intend to use in your app

Development phase

1) Create a GitHub repository on Github.com (before you start coding)
2) Clone it to your local machine (before you start coding)
3) Make frequent commits throughout your development that are descriptive, such as "adds todos reducer" (*throughout development/coding process*)

Technical Requirements

The following requirements are related to what your **code** should contain:

1) This should be a React app based on Create React App
2) The app should contain at least two views: ***/todos*** and ***/contact***
3) When a user navigates to /todos, they should be presented with a view that:

- Display a list of the todo items
- Displays a form (text input and submit button) that allows users to add a new item to the list
- Offers a way for a task to be marked as "completed" and clearly indicates this status visually (e.g. strike-through effect)
- Offers a way for a task to be removed from the list
- Offers a way to view either
    - all todos
    - completed todos
    - incomplete todos
- When todos are added/updated/marked as complete, these changes should immediately be reflected in your internal state

4) When a user navigates to ***/contact***, they should be presented with a view that:

- Displays a contact form that displays the following fields
    - first name field
    - last name field
    - email field
    - comments field
    - Renders a the form as a controlled component such that after entering text into any of the fields, the form's state has changed

5) There should be at least 10 custom CSS rules used throughout the components of your React app
6) You must have a horizontal nav bar at the top of your site
7) You must have at least one example of content side-by-side (e.g. the "new todo" form in the example screenshot)

Deliverables

1) Your user stories
2) A collection of wireframes - one for each view of your app
3) Your state tree
4) Your list of container and presentational components from the planning phase
5) Your app source code should be available for viewing in your GitHub repository
6) A ***readme.md*** file in the root project folder that contains the following information about your project:

- Your name
- Overview/description of the project
- Details on how to use it or what functionality is offered
- Technologies Used (***.html, .css***)
- Ideas for future improvement (minimum of 3)

7) Your repository should contain at least 15 commits and should reflect a consistent commit history
8) Hosting on GitHub pages using the gh-pages package (https://www.npmjs.com/package/gh-pages)

Submit your GitHub link and hosting link (include everything on a word document and upload the document under Project Submission section). All deliverables should be included on GitHub and the site should be made publicly available using a hosting service.

You will be evaluated on your ability to meet both the **workflow requirements** and the **technical requirements**.

Project Grading

To pass the project, Instructor should take below criteria into consideration while grading this project and decide whether to Pass or Fail the student.

- Functionality
- Robustness
- Creativity, styling, user experience
- Code quality.
- GitHub structure
- Documentation, Installation instructions, Comments

19. Scrum and Agile immersion

Timeframe: week 17 / 20 weeks

Materials: This course does not follow any official study guide. However, Product department from QuickStart is recommending following books:

1. The Scrum Guide, by Ken Schwaber & Jeff Sutherland
2. Agile Practice guide, by Project management institute

Key note: Week by week schedule is a recommendation and instructor has the freedom to adjust the pace of covering those lectures as well as identifying for which courses students need more time to focus on.

This are the lesson that are covered in video content for **Scrum and Agile immersion**, which instructors can cover in their weekly sessions with students:

- Module 1: Introduction
- Module 2: Agile Overview
- Module 3: Scrum Overview
- Module 4: Roles and Responsibilities
- Module 5: Project Lifecycle
- Module 6: Building High Performing Teams
- Module 7: Scaling Agile
- Module 8: Closure

Learning outcomes

By the end of the course, students should be able to:

1. Understand the origins and principles of Agile methodology.
2. Recognize the need for agility in modern project management.
3. Identify common challenges in traditional project management approaches.
4. Define Agile methodology and its core values and principles.
5. Compare and contrast Agile with traditional waterfall project management.
6. Explain the iterative and incremental nature of Agile development.
7. Describe the Scrum framework, including its roles, events, and artifacts.

8. Understand the importance of transparency, inspection, and adaptation in Scrum.
9. Identify the key differences between Scrum and other Agile frameworks.
10. Define the roles and responsibilities of the Scrum Master, Product Owner, and Development Team.
11. Explain how these roles collaborate to deliver value in an Agile project.
12. Understand the importance of self-organization and cross-functional teams in Agile.
13. Describe the Agile project lifecycle, including planning, execution, and delivery.
14. Identify the different phases of an Agile project, such as product backlog refinement, sprint planning, and sprint review.
15. Understand how Agile principles guide the entire project lifecycle, from inception to closure.
16. Recognize the characteristics of high-performing Agile teams.
17. Understand the importance of collaboration, communication, and trust in Agile teams.
18. Identify strategies for fostering teamwork and continuous improvement in an Agile environment.
19. Explain the challenges of scaling Agile in large organizations or complex projects.
20. Identify different scaling frameworks, such as SAFe, LeSS, and Nexus.
21. Understand how to tailor Agile practices to fit the needs of larger teams or organizations

20. Querying Data with SQL

Timeframe: week 17 / 20 weeks

Materials: This course does not follow any official study guide.

Key note: Week by week schedule is a recommendation and instructor has the freedom to adjust the pace of covering those lectures as well as identifying for which courses students need more time to focus on.

This are the lesson that are covered in video content for **Querying Data with SQL**, which instructors can cover in their weekly sessions with students:

- Introduction to SQL
- More SQL: Intermediate Functions
- SQL: Output Control
- SQL: Table Modification
- SQL: Common Table Expressions

Learning outcomes

By the end of the course, students should be able to:

1. **Understanding SQL Fundamentals:** Students will grasp the fundamental concepts of SQL, including its purpose, syntax, and basic query structure.
2. **Applying Intermediate SQL Functions:** Learners will become proficient in using intermediate SQL functions to manipulate and retrieve data from databases effectively.
3. **Controlling Output in SQL:** Students will learn techniques to control the output of SQL queries, such as sorting, filtering, and limiting results.
4. **Modifying Tables with SQL:** Understanding how to modify tables using SQL commands like **INSERT**, **UPDATE**, and **DELETE**, and the implications of these modifications on the database structure.
5. **Utilizing Common Table Expressions (CTEs):** Mastery of common table expressions for creating temporary result sets, enhancing query readability, and simplifying complex queries

21. Node.js - From Zero to Web Apps

Timeframe: week 18 / 20 weeks

Materials: This course does not follow any official study guide.

Key note: Week by week schedule is a recommendation and instructor has the freedom to adjust the pace of covering those lectures as well as identifying for which courses students need more time to focus on.

This are the lesson that are covered in video content for **Node.js - From Zero to Web Apps**, which instructors can cover in their weekly sessions with students:

- Introduction to Node.js and NPM
- Concurrency and Asynchronous JavaScript: Promises, Async and Await
- Building Web Servers using Express
- Express Routing
- Integrating Gulp into a Web Server
- NoSQL Databases Using Mongo and Mongoose
- SQL Databases Using MySQL
- Error Handling with Express
- Authentication, Authorization and Security
- Full stack Web development using React, Node, Express and SQL

Learning outcomes

By the end of the course, students should be able to:

1. **Introduction to Node.js and NPM**
   - Define Node.js and explain its importance in server-side development.
   - Understand the role of NPM (Node Package Manager) and how to manage dependencies in a Node.js project.
2. **Concurrency and Asynchronous JavaScript**
   - Explain the concepts of asynchronous programming in JavaScript.
   - Utilize Promises and Async/Await for handling asynchronous operations effectively.
3. **Building Web Servers Using Express**
   - Set up a basic Express.js server for handling HTTP requests and responses.
   - Develop RESTful APIs using Express.js.

- Understand middleware and its role in web development.

4. **Express Routing**
    - Implement basic routing patterns in Express.
    - Handle route parameters and query strings in web applications.
5. **Integrating Gulp into a Web Server**
    - Install and configure Gulp.js for automating development tasks.
    - Define and execute tasks like file compilation and live reloading with Gulp.
6. **NoSQL Databases Using Mongo and Mongoose**
    - Set up a connection between Node.js and a MongoDB database.
    - Perform CRUD (Create, Read, Update, Delete) operations using Mongoose.
7. **SQL Databases Using MySQL**
    - Connect Node.js applications to a MySQL database.
    - Perform CRUD operations using SQL queries from a Node.js application.
8. **Error Handling with Express**
    - Identify and handle basic runtime errors in Express applications.
    - Implement simple error-handling middleware for consistent error management.
9. **Full-Stack Web Development Using React, Node, Express, and SQL**
    - Build a basic full-stack web application using React for the frontend and Node.js, Express, and SQL for the backend.
    - Integrate frontend and backend to handle dynamic data and user interactions.

22. Starting with Rest APIs

Timeframe: week 19 / 20 weeks

Materials: This course does not follow any official study guide.

Key note: Week by week schedule is a recommendation and instructor has the freedom to adjust the pace of covering those lectures as well as identifying for which courses students need more time to focus on.

This are the lesson that are covered in video content for **Starting with REST APIs**, which instructors can cover in their weekly sessions with students:

- Learning REST
- REST Defined
- REST & HTTP
- REST & CRUD ops
- Designing for REST
- Developing REST API's
- NodeJS Environment
- NodeJS & REST
- REST Routes
- A REST Controller
- Completing the REST Server
- Testing our REST API

Learning outcomes

By the end of the course, students should be able to:

1. **Understand the Fundamentals of REST**
    - Define REST and explain its role in modern web applications.
    - Understand the relationship between REST and the HTTP protocol.
2. **Work with HTTP Methods and CRUD Operations**
    - Map HTTP methods to CRUD operations.
    - Demonstrate how REST APIs handle data creation, retrieval, updating, and deletion.
3. **Consume REST APIs in Client Applications**
    - Evaluate and integrate public REST APIs into applications.
    - Develop an application flow that interacts with RESTful services.
4. **Design and Build RESTful APIs**

- Develop a REST API with multiple endpoints.
- Implement a REST API using Node.js and Express.
- Connect a REST API to a database for persistent data storage.

5. **Apply REST API Best Practices**
   - Identify key principles of RESTful API design.
   - Understand the flexibility of REST as a design style rather than a strict set of rules.

23. Project 04: Nodejs Express

Timeframe: week 20 / 20 weeks

This is the moment you may be waiting for. After many hours of studies on node.js, JavaScript, HTML, CSS, and many other technologies, it is time to put the puzzle pieces together, and create a portfolio project that you can proudly present in front of your prospective future employers.

Introduction

The final project will help you demonstrate the skills you have learned throughout the bootcamp, and some of its components may be reused in your future projects. For instance, you will implement a signup and login procedure that is part of most web APIs and web-applications.

You will build a system according to the principles of 3-tier architecture:

1. You will create a **data layer** using a database management system, which can either be MySQL, or MongoDB.
2. You will build an **application layer** that implements the below described business logic using node.js and Express.
3. You will build a **presentation layer** using HTML, CSS, JavaScript, and the client-side framework of your choice, using some of the following: React, React Router, Redux, Angular.

Besides following the guidelines presented in this document, you have the freedom of choice in terms of technologies used and implementation details. Keep your mind and creative thinking completely free. Don't feel bound to anyones thoughts or ideas of how to complete this project.

Make sure your implementation will be rock solid, as if your were working for real employers. As you know, in the world of software-engineering, even one mistake may cost you a lot. Think of it this way. A software engineer is like a surgeon. When implementing an application, verify your work as if lives were at stake. This kind of discipline will bring you forward in your career.

Project Details

Create a solution using the 3-Tier architecture that lets users register and log in to a forum, where users may ask and answer questions. Your solution should include:

1. A database schema and some example data. For demoing purposes, choose a free online database service, or create example files that load an empty database with example data.

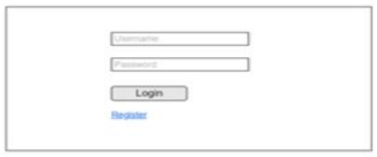
2. An application layer written using node.js and express.js. I recommend creating a JSON API that accesses the database fields and prepares a JSON string for the client
3. A single page application (client-side rich web application) that communicates with the application layer by sending and receiving JSON data. You may choose the corresponding frameworks and libraries in the solution.

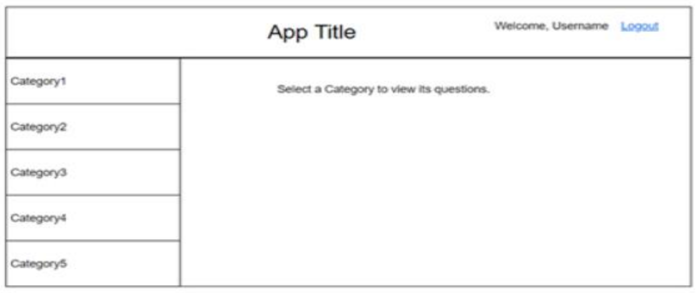
Instead of creating an application that would store any question on any topic, invent a theme for your application. Create a relatively small hierarchy of topics, and name your application after the organizing principle of your application. For instance, you may create a project called PetLand, and store questions on dogs, cats, and rabbits. Alternatively, you can create a project called FrontEnd frameworks, and store questions on JavaScript, Angular, and Vue.

Login Screen

The login screen has two fields to enter the user name and the password.

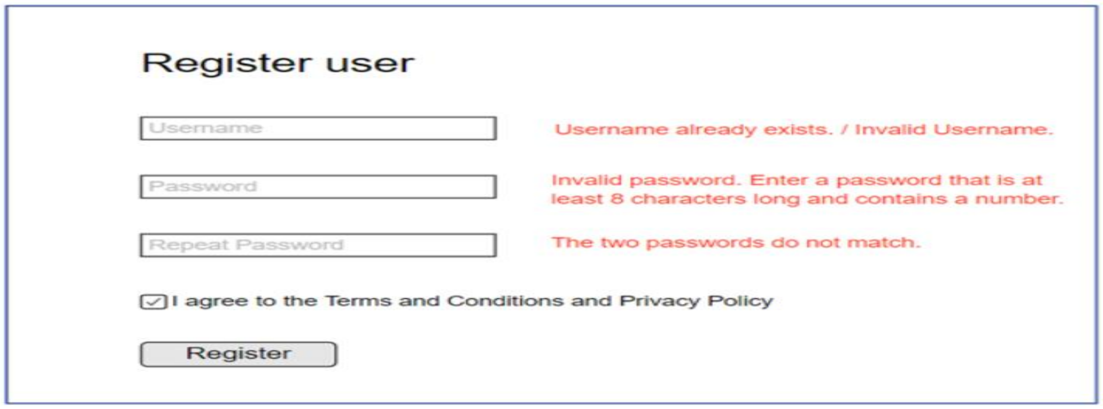
When the username-password combination is invalid, an error message informs the user on the invalid login. If the server finds the login to be successful, the Dashboard page is shown.





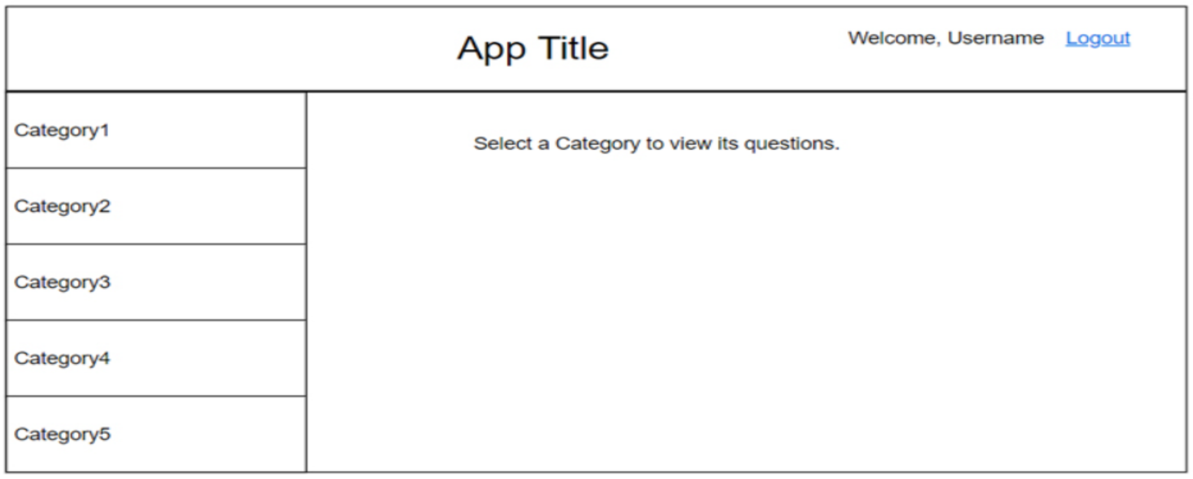
Registration Screen

The error messages appear on the right if the corresponding form field has an invalid value. The checkbox context should also be red in case its value is not checked upon pressing the Register button. Each error message should disappear once the value of the corresponding form field changes.


Register user
Username
Username already exists. / Invalid Username.
Password
Invalid password. Enter a password that is at least 8 characters long and contains a number.
Repeat Password
The two passwords do not match.
I agree to the Terms and Conditions and Privacy Policy
Register

Dashboard

The main page contains the title of the app, displays your username, and a Logout link. The Logout link transfers you back to the Login page.
On the left, there is a menu of the existing categories. If there are more categories than what fits on screen, make sure you can scroll in this menu.
Upon selecting a category, the "Select a Category to view its questions" area is replaced by questions in chronological order:



Deliverables

Submit your GitHub link and hosting link (include everything on a word document and upload the document under Project Submission section). All deliverables should be included on GitHub and the site should be made publicly available using a hosting service.

Project Grading

To pass the project, Instructor should take below criteria into consideration while grading this project and decide whether to Pass or Fail the student.

- Functionality
- Robustness
- Creativity, styling, user experience
- Code quality.
- GitHub structure
- Documentation, Installation instructions, Comments