# Student Version

| Section A – Program/Course details | | | |
| --- | --- | --- | --- |
| **Qualification code:** | ICT50220 | **Qualification title:** | Diploma of Information Technology (Front End Web Development) with  Diploma of Information Technology (Back End Web Development) |
| **Subject Code:**  **Unit code:** | (DWEB6)  ICTPRG443   ICTPRG554 | **Subject Title:**  **Unit title:** | (Server-side Web Programming)  Apply intermediate programming skills in different languages  Manage data persistence using noSQL data stores |
| **Department name:** | BDIT, Computer and Information Technology | **CRN number:** |  |

| Section B – Assessment task details | | | |
| --- | --- | --- | --- |
| **Assessment number:** | 2 of 2 | **Semester/Year:** | 2/2024 |
| **Due date:** | Session 16 | **Duration of assessment:** | 4 sessions |
| **Assessment method** | Portfolio | **Assessment task results** | ☒Ungraded result (S/NS) |
| ☐Other: |

| Section C – Instructions to students |
| --- |
| **Task instructions:** |
| For this assessment task you will be required develop a solution for a client using React.js, Node.js and a noSQL database (Google Firebase).  This assessment task is divided into the following parts: Part 1 – Confirm Application Requirements  Part 2 – Plan & Develop the React.js Client  Part 3 – Design the Database  Part 4 – Develop RESTful API Server Part 5 – Refine the Client and Server Part 6 – Test and Debug the Code Part 7 – Document Activities & Hand Over  **Students are expected to submit the following documents:**   * The signed and completed Assessment Sheet; and * A ZIP file containing the client-server project file (with node modules deleted). * This task is to be completed individually. * Using this assessment document, please read all questions in this document and record your answers in the document against each question. * To be satisfactory in this assessment task all answered questions must correctly meet the marking guide criteria. * Discuss with your assessor if you feel you require special consideration or adjustment for this task. * Learners can consult class learning material via Brightspace and other softcopy information, including information from the Internet. However, all answers must be in a learner’s own words. Where a quote is used the learner must cite the information source. * You must submit all required working files, documentation, and any other assets that you feel may be required in a zipped file, including the completed and signed coversheet. The assessment must be completed and submitted electronically to Brightspace by the due date. If this is not possible, you must contact your assessor to gain written approval for an alternative arrangement for submitting the assessment. * It is expected all documents will be completed and submitted electronically but if this is not possible, make alternative arrangements for submitting the documents with your assessor. * Leaner must contribute to and abide by organisational standards including intellectual property, privacy laws, and plagiarism and academic honesty. Further information is detailed at: https://holmesglen.edu.au/Students/Student-Resources/   ***See supporting documentation for further instructions.*** |

| Section D – Conditions for assessment | |
| --- | --- |
| **Conditions:**  Student to complete and attach Assessment Submission Cover Sheet to the completed Assessment Task. | |
| * This is an individual task. However, you are required to get information, feedback and ideas from your assessor, peers or industry to help complete the assessment planning guide. * You must meet all criteria listed in the marking guide to be satisfactory in this task. * You must submit all required working files, documentation, and any other assets that you feel may be required in a zipped file, including the completed and signed coversheet. The assessment must be completed and submitted electronically to Brightspace by the due date. If this is not possible, you must contact your assessor to gain written approval for an alternative arrangement for submitting the assessment. * If not successful within the enrolment period as per Holmesglen assessment procedure, you will be requested to resubmit within 7 days of receiving feedback. You will have the opportunity to resubmit if any part of the assessment is deemed unsatisfactory (you are permitted TWO (2) resubmission per assessment task). Resubmissions must be submitted by the resubmission due date provided by your teacher. * This task is open book. You may use the internet for research purposes only. All answers must be in your own words. Where a quote is used, you must cite the information source. * If you feel you require special allowance or adjustment to this task, please discuss with your assessor within one week of commencing this assessment. Any change to assessment arrangements must be reviewed by the Education Manager and approved by the Head of Department. * You can appeal an assessment decision according to the Holmesglen Assessment Complaints and Appeals Procedure. * You are expected to dedicate time to develop this assessment task both in and out of the classroom. * Leaner must contribute to and abide by organisational standards including intellectual property, privacy laws, and plagiarism and academic honesty. Further information is detailed at: https://holmesglen.edu.au/Students/Student-Resources/ | |
| **Equipment/resources students must supply: Equipment/resources learners must supply:**  **Learners opting to BYOD laptop or intending to learn remotely will require access to:** | **Equipment/resources to be provided by the RTO:**  **This is a blended learning course and as such a remote learner will access their own computer equipment as per the specification provided. The Institute will provide the following:** |
| * A MAC or * PC/laptop with the following minimum specification:   + Quad-Core CPU, 16GB of RAM, 250GB of Storage, 2 GHz or faster processor   + Windows 10 OS, or higher * Headset with microphone and webcam (if learning remotely) * Internet access   Access to applications used in learning is available through Holmesglen MyHorizon or can be downloaded via the link provided: • Brightspace (Learning Management System) - https://holmesglen.brightspace.com/ • 365 Microsoft office suite - https://portal.office.com • WebEx - https://holmesglen.webex.com/ • Holmesglen OneDrive • Google Chrome – recommended web browser https://www.google.com/intl/en\_au/chrome/ • 7-Zip - https://www.7-zip.org/download.html • GitHub Desktop- https://desktop.github.com • Visual Studio Code - https://code.visualstudio.com/ • Figma - https://www.figma.com/ • Node.js- https://nodejs.org/en • Postman - https://www.postman.com/ • Restful API • Google Firestore - https://firebase.google.com/docs/firestore | Classroom computer with the following minimum specification: Quad-Core CPU, 16GB of RAM, 250GB of Storage, 4 GHz or faster processor Internet access  Applications access available at ZENworks and Holmesglen MyHorizon: • Brightspace (Learning Management System)  • 365 Microsoft office suite  • WebEx • Holmesglen OneDrive • Google Chrome – recommended web browser • 7Zip • Visual Studio Code  • GitHub • Figma • Node.js • Postman • Restful API • Google FirestoreFigma   * Node.js   Postman |

|  |  |  |  |
| --- | --- | --- | --- |
| Section E – Marking Sheet - Student Answer Sheet | | | |
| **Subject code:**  **Unit code:** | (DWEB6)  ICTPRG443   ICTPRG554 | **Subject Title:**  **Unit title:** | (Server-side Web Programming)  Apply intermediate programming skills in different languages  Manage data persistence using noSQL data stores |

| **Portfolio task** | | | | | |
| --- | --- | --- | --- | --- | --- |
| **Criteria for assessment** | | | **Satisfactory** | | **Comment** |
| **Yes** | **No** |
| **Files:**  The following files must be submitted for this assessment: | | | | | |
| 1. | Assessment Sheet (signed) | | ☐ | ☐ |  |
| 2. | Portfolio Project file (*including Client and Server directories*) | | ☐ | ☐ |  |
| **Marking criteria:** Part 1 - Confirm Application Requirements | | | | | |
| 1. | Learner has reviewed the user requirements in the brief and Identified the core functionality needed for the development of the project | | ☐ | ☐ |  |
| 2. | Learner has confirmed the core pages required by the user | | ☐ | ☐ |  |
| 3. | Learner has confirmed all relevant technologies required for the application | | ☐ | ☐ |  |
| 4. | Learner has confirmed the “type” of database being used | | ☐ | ☐ |  |
| 5. | Learner has confirmed the primary scripting language used by the server to allow for dynamic functionality. | | ☐ | ☐ |  |
| 6. | In accordance with brief requirements, learner has confirmed that a private cloud repository has been created in GitHub as backup | | ☐ | ☐ |  |
| 7. | The learner:   * has reviewed the user requirements and needs of the project * has determined the pages needed and dynamic functionality * has determined appropriate language & technologies based on required functionality * has obtained approval from the client about the pages required and the functionality | | ☐ | ☐ |  |
| **Marking criteria:** Part 2 - Plan & Develop the React.js Client | | | | | |
| 1. | Learner has developed the hierarchy / sitemap of the content to be displayed on the website according to the user requirements | | ☐ | ☐ |  |
| 2. | Learner has created the prototype UI using wireframes for the website’s home page to show content layout | | ☐ | ☐ |  |
| 3. | Learner has built the website navigation system with the routed page links as specified by the brief requirements | | ☐ | ☐ |  |
| 4. | Learner has built the page as specified in their wireframe | | ☐ | ☐ |  |
| 5. | The application contains functioning routing to functioning pages as specified in the user brief requirements | | ☐ | ☐ |  |
| 6. | Learner has applied basic language syntax rules, including use of JSX, and best practices to structure their React.js code | | ☐ | ☐ |  |
| 7. | In accordance with the project brief, the learner has used CSS techniques to style, layout and position the document elements of webpages on the website | | ☐ | ☐ |  |
| 8. | State has been used to Create, manipulate and destroy dynamic variables, including data structures. | | ☐ | ☐ |  |
| **Marking criteria:** Part 3 - Design the Database | | | | | |
| 1. | Learner has confirmed the specific noSQL database being used | | ☐ | ☐ |  |
| 2. | Provided brief descriptions on the concepts of “horizontal scaling” and “vertical scaling” AND describe at least ONE benefit of preferring horizontal scaling databases than to vertical scaled databases | | ☐ | ☐ |  |
| 3. | Learner has compared the SQL with noSQL databases to the competency specified | | ☐ | ☐ |  |
| 4. | Listed the different data types required used by the noSQL database in your application & confirmed overall data structure of your database through use of Firestore database design diagram | | ☐ | ☐ |  |
| 5. | Learner has confirmed the TWO areas in your database where raw data is stored | | ☐ | ☐ |  |
| 6. | Created a database partition, provided a screenshot the database that includes the different partition “keys” and determine whether it is a “partition” or a “shard” | | ☐ | ☐ |  |
| 7. | Commented on the estimated distribution of data between the partitions | | ☐ | ☐ |  |
| 8. | Learner has confirmed approval of database design & supporting documentation with assessor | | ☐ | ☐ |  |
| **Marking criteria:** Part 4 - Develop RESTful API Server | | | | | |
| 1. | Confirmed the server-side API runtime environment which interacts with the application’s noSQL database | | ☐ | ☐ |  |
| 2. | Created the entry point for the routes in the API server | | ☐ | ☐ |  |
| 3. | Created the file that handles the initial noSQL database connection | | ☐ | ☐ |  |
| 4. | Created the file(s) that handle at least ONE of each CRUD operation in the application | | ☐ | ☐ |  |
| 5. | Created the file(s) that handle/” controls” the route requests and query the database as required for each CRUD operation | | ☐ | ☐ |  |
| 6. | Learner confirmed CRUD operations & queries are working as intended as follows: | | ☐ | ☐ |  |
| 6a. | 1. the GUI of the page that handles a GET request, and; 2. the successful Postman result for a GET request at the endpoint used. | | ☐ | ☐ |  |
| 6b. | 1. the GUI of the page that handles a POST request, and; 2. the successful Postman result for a POST request at the endpoint used. | | ☐ | ☐ |  |
| 6c. | 1. the GUI of the page that handles a PUT request, and; 2. the successful Postman result for a PUT request at the endpoint used. | | ☐ | ☐ |  |
| 6d. | 1. the GUI of the page that handles a DELETE request, and; 2. the successful Postman result for a DELETE request at the endpoint used. | | ☐ | ☐ |  |
| 6e. | 1. the GUI of the page that handles a GET By ID request, and; 2. the successful Postman result for a GET By ID request at the endpoint used. | | ☐ | ☐ |  |
| 7. | Learner has confirmed approval of RESTful API & supporting documentation with assessor | |  |  |  |
| **Marking criteria:** Part 5 - Refine the Client and Server | | | | | |
| 1. | | Incorporated at least ONE sort of key into a query made to the noSQL database and provide a screenshot of the code | ☐ | ☐ |  |
| 2. | | Provided TWO screenshot of a Morgan output that details the through-put for TWO CRUD request. (One Read & One Write request) | ☐ | ☐ |  |
| 3. | | Listed at least TWO methods in which we have or can optimise the through-put of the queries to database | ☐ | ☐ |  |
| 4. | | Created at least TWO composite indexes to optimise queries made to the selected noSQL database | ☐ | ☐ |  |
| 5. | | Described the function of each index created above and how they optimise the queries to the database | ☐ | ☐ |  |
| 6. | | Perform a query against at least ONE of the newly created index and provide a screenshot of both (i) the code of the index query, and (ii) the change to the GUI page based on the altered query | ☐ | ☐ |  |
| 7. | | Implemented the file(s) which handle authentication, authorisation and one admin level of authorisation for the server-side API | ☐ | ☐ |  |
| 8. | | Created the file(s) which encrypt relevant authentication data and stated the technology used to encrypt the data | ☐ | ☐ |  |
| 9. | | Learner confirmed ONE technology used in the application to secure the API’s authorisation / authentication processes describe its function | ☐ | ☐ |  |
| 10. | | Learner has configured the Time-To-Live (TTL) for the objects created by the “auth” technology and explained how TTL has been adjusted | ☐ | ☐ |  |
| 11. | | Stated ONE technology used to configure the server software to minimize potential database attacks and describe its function | ☐ | ☐ |  |
| 12. | | Implemented the file(s) that handle data validation for at least ONE route and confirmed the technology used. | ☐ | ☐ |  |
| 13. | | Learner has refined the client & server structures in accordance with the project brief & reviewed these structures with the assessor | ☐ | ☐ |  |
| **Marking criteria:** Part 6 - Test and Debug the Code | | | | | |
| 1. | Listed at least TWO debugging tools, ONE of which must be a “standalone” tool | | ☐ | ☐ |  |
| 2. | Tested the client, server and database outputs and security and record the results for the following six tests: | |  |  |  |
| 2a. | **Test:** That each CRUD operation renders the expected output at a specified route which contains all CRUD operations | | ☐ | ☐ |  |
| 2b. | **Test:** That a 404 Error Page displays when a user navigates to an undefined route and the error stack is provided to the developer in terminal | | ☐ | ☐ |  |
| 2c. | **Test:** Test that the JWT is persistent for the session when users a logged in | | ☐ | ☐ |  |
| 2d. | **Test:** Test that a GET route, with dynamically retrieved data, loads and renders correctly in at least TWO different browsers (confirm which browsers tested) | | ☐ | ☐ |  |
| 2e. | **Test:** Test that a POST route, which accepts a number, has been converted to the correct number type in the database. | | ☐ | ☐ |  |
| 2f. | **Test:** Test that the logout function removes the user from local storage along with the user token | | ☐ | ☐ |  |
| 2g. | **Test:** Designed Test 7 | | ☐ | ☐ |  |
| 3. | Learner has finalised testing & debugging of the web application, and has reviewed test completion with the assessor | |  |  |  |
| **Marking criteria:** Part 7 - Document Activities & Hand Over | | | | | |
| 1. | The learner has diagnosed bugs or potential bugs and identify resolution requirements. | | ☐ | ☐ |  |
| 2. | The Client and Server have been designed and implemented according to the user brief requirements | | ☐ | ☐ |  |
| 3. | The project has been created using an integrated development environment (IDE) that meets the needs of the brief | | ☐ | ☐ |  |
| 4. | The project has been tested, debugged and documented according to user requirements and expected outcomes | | ☐ | ☐ |  |
| 5. | Confirm that the final version of the project has been transferred to GitHub as a private repository and provide ONE screenshot of the complete private “repo” in the browser | | ☐ | ☐ |  |
| 6. | The learner has sought to improve protocols governing communications with the “client” and co-workers during the length of the project | | ☐ | ☐ |  |
| 7. | The learner has collaborated with others, sharing information to build strong work groups and interactions with the “client” | | ☐ | ☐ |  |

# Assessment Submission Cover Sheet (VET)

|  |  |
| --- | --- |
| **Student Declaration – Must be signed before submission** |  |
| By submitting this assessment task and signing the below, I acknowledge and agree that:  • This completed assessment task is my own work.  • I understand the serious nature of plagiarism and I am aware of the penalties that exist for breaching this.  • I have kept a copy of this assessment task.  • The assessor may provide a copy of this assessment task to another member of the Institute for validation and/or benchmarking purposes. | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Student ID:** | **100683785** | **Student name:** | **Vorakorn Taweetawon** |

|  |  |
| --- | --- |
| **Submission or observation date:** | **14/11/2024** |
| **Student signature**  For electronic submissions: By typing your name in the student signature field, you are accepting the above declaration. | **Vorakorn Taweetawon** |

| Section F – Feedback to Student | | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| **Has the student successfully completed this assessment task?** | | | | | **Yes** | **No** |
| **☐** | **☐** |
| **Additional Assessor comments (as appropriate):** | | | | | | |
| **Resubmission allowed:** | **Yes ☐** | **No ☐** | **Resubmission due date:** |  | | |
| **Assessor name:** | Alex Bicknell | | | | | |
| **Assessor signature:** |  | | | | | |
| **Date assessed:** |  | | | | | |

**Supporting document**

# Portfolio (Brief & Documentation)

|  |  |  |  |
| --- | --- | --- | --- |
| **Subject code:**  **Unit code**: | (DWEB6)  ICTPRG443   ICTPRG554 | **Subject title:**  **Unit title:** | (Server-side Web Programming)  Apply intermediate programming skills in different languages  Manage data persistence using noSQL data stores |

|  |
| --- |
| **Project Brief** Read through the brief below take note of the needs and requirements outlined in the brief. |
| **Overview of project**  For this assessment task you will be required to plan and develop a full stack application that involves a front-end client which passes data to and from a back-end server RESTful API, which stores data within a noSQL database. The scripts will need to demonstrate introductory object orientated programming techniques.  **NOTE**: *If you would like to create a full-stack application* ***that is NOT a mock “e-commerce shop” website****, please discuss with your facilitator in advance to determine suitability.*  **This assessment task is divided into the following parts:** Part 1 – Confirm Application Requirements  Part 2 – Plan & Develop the React.js Client  Part 3 – Design the Database  Part 4 – Develop RESTful API Server Part 5 – Refine the Client and Server Part 6 – Test and Debug the Code Part 7 – Document Activities & Hand Over  **Client Scenario**  You are a freelancer website developer and have been recently engaged by Timbertop United Pty Ltd (“TU”) to build a full-stack website prototype for their football club’s e-commerce shop web application*.* TU plans to launch the new venture late into Quarter 2 of FY2023, to hit markets just before the Christmas rush.  Broadly, TU would like the front-end UI to be built using React.js in order to allow for modular design and to allow for future iteration following their usual Agile development processes. Importantly, TU expects you to build a RESTful API server that saves and stores data in a noSQL database, as it expects to store large amounts of data on user information & currencies.  They have summarised the specific features and requirements as follows:  **General Client Features**   * The Client Application returns RESTful API data to be displayed in the GUI **to at least TWO endpoints** * As the website is a prototype, **a minimum of at least FOUR routed pages** (using react-router-dom), including at least TWO dynamic pages * Use class-based components and stateless functional components * Displaying of data is to use appropriate controls, such as text boxes, dropdowns, radio buttons and so on * Format the data so it is easily understood by the user * Structuring of data (*such as using TypeScript or JavaScript methods*) * Error pages must be designed and created * *Optional:* *The developer (you) may choose to build a search component that searches the client state OR search an API data endpoint and display dynamically in the browser* * **IMPORTANT**: You are **NOT** required to build complete integrated e-commerce solution. Integrations such as payment processing & checkout completion & processing are **NOT** required.   **General API Requirements**   * A RESTful API that returns JSON at various endpoints * The RESTful API data must be saved and retrieved from a noSQL database (Google Firestore) * All 4 CRUD operations must be used in your application * At least ONE POST route and ONE PUT route must be validated using a module such as JOI * Authentication and authorisation must be implemented * At least ONE endpoint must be secured such as an admin endpoint * Structuring of data (*such as using TypeScript or JavaScript methods*) * Error pages or responses, such as 404 / 500 errors, must be created and implemented   **Database Requirements**   * The database must have at least TWO collections (*for example “users” & “products”*) * The database must use at least THREE different data types * The database must incorporate TWO composite indexes and query at least ONE of them * The database must utilise at least ONE sort key to refine the query * Images must be stored in a separate partition to the standard data types   **Security Requirements**   * At least ONE POST route and ONE PUT route must be validated before saving to the database * Authentication and authorisation must take place before any data is saved to the database * Application secrets must be stored in environmental variables or using a module such as DOTENV * It is recommended that a module such as HELMET be used to improve protection of the application and database * Tests must be conducted (*in accordance with Part 6*) to ensure no unnecessary sensitive data is returned from an endpoint * Query responses should be tracked and monitored using a module such as MORGAN * Appropriate error messages must be returned and without error stack traces   **Required Language Standards, Third-Party Libraries & Technologies**   * Mandatory   + HTML5, CSS3 & JSX languages to be used for client-side development   + React.js as a client framework **using Vite**   + The GUI must interact with at least TWO of the RESTful APIs endpoints   + Node.js (with JavaScript) with Express must be used for server-side API development   + Google Firestore should be used as the NoSQL database   + Use of a CSS-in-JS library, such as vanilla-extract or styled-components * Optional   + Use of a responsive component library like Bootstrap or unstyled/primitives component library like RadixUI   **Coding Standards**   * All code should be commented clearly:   + **Classes and Modules** - A descriptive overview should be provided for each class / script as a comment at the top of the file. Details about any parent classes should be documented at the top of the file.   + **Members** -The purpose of each member should be documented as a comment.   + **Methods** -The purpose of each method should be documented as a comment.   + **Parameters** - The purpose of each parameter should be documented as a comment. * XHTML Standards must be adhered to wherever possible. * CSS Standards must be adhered to wherever possible. * XHTML and HTML accessibility standards |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Part 1 - Confirm Application Requirements** Based on the information you have gathered from the brief complete the following: | | | | | | |
| **Question / Criteria** | | | **Answer / Satisfactory Response** | | | |
| 1. Review the user requirements *in the brief above* and confirm:  * the **purpose** of the web application, * the required **features / technologies**; and * any relevant **standards**,needed for the development of the project. | | | **Purpose of web application** | | | |
| *The web application is intended to serve as a website for an e-commerce named Garden of Eden, its primary goal is to provide a platform for users to browse and potentially purchase their products.* *It allows users to browse a product list, information about individual products, provide a clear product displays, descriptions, and user-friendly navigation.* | | | |
| **List of Required Features / Technologies** | | | |
| Back-End (Server Side):  * RESTful API built using **Node.js with Express**. * **CRUD operations** for data handling. * Must use a **noSQL database (Google Firestore)** to store user and product data. * At least **one POST and one PUT route** must include validation using a module such as **JOI**. * At least **one secured endpoint**, such as for admin access. * **Authentication and authorization** to control access and data storage.  Front-End (Client Side):  * **React.js** framework (with Vite) * **Minimum four routed pages** using react-router-dom, including at least **two dynamic pages**. * Use of **class-based components** and **stateless functional components**. * Data displayed via the UI must come from at least **two RESTful API endpoints**. * **Controls** such as text boxes, dropdowns, and radio buttons must be used for data input. * The UI should be structured using **TypeScript or JavaScript methods**. * Optionally, a **search component** may search client state or an API endpoint.  Database:  * **Two collections** (e.g., users and products) in Firestore. * **Three data types** for data storage (e.g., string, number, Boolean). * **Composite indexes** for querying with at least one indexed query. * **Image storage** in a separate partition from standard data types. * Use of a **sort key** to refine queries.  Security Requirements:  * **Validation** on POST and PUT routes before saving to the database. * **Authentication and authorization** mechanisms in place before data storage. * **Environment variables** (e.g., DOTENV) to store secrets. * Recommended use of **security modules**, such as **HELMET**. * **Error messages** should avoid showing sensitive data and stack traces. * API **query monitoring** using a module like **MORGAN**. | | | |
| **Relevant Standards** | | | |
| **Coding Standards:**   * All code should be commented clearly:   -Classes and Modules  -Members  -Methods  -Parameters   * XHTML Standards must be adhered to wherever possible. * CSS Standards must be adhered to wherever possible. * XHTML and HTML accessibility standards   **Technologies:**   * **HTML5, CSS3, JSX** for front-end development. * **CSS-in-JS libraries** (e.g., vanilla-extract or styled-components). * Optional: Responsive component libraries like **Bootstrap** or **Radix UI**. | | | |
| 1. Confirm the core pages / views required by the user in the application and highlight at least ONE page(s) which will be fed dynamic data | | | * Home Page * All Products * On Sale Product * Dashboard * Signup & Login | | | |
| 1. Confirm the FOUR relevant technologies required for the application, as per a typical “MERN Stack” (*or rather “FERN Stack”*) application | | | FERN Stacks   * Firebase * Express.js * React * Node.js | | | |
| 1. Confirm the “type” of database being used in the application stack | | | NoSQL database | | | |
| 1. Confirm the primary programming language used by the server to allow for dynamic functionality | | | JavaScript with Node.js | | | |
| 1. In accordance with brief requirements, confirm that a private cloud repository has been created in GitHub, to allow for backup transfers during project development   (*Provide ONE screenshot of GitHub “repo”)* | | | *< Screenshot of GitHub repo >../../../../../Screenshot%202024-11-13%20at%2012.11.23%20AM.p* | | | |
| **7. Approval and Feedback** After you have finished the planning, obtain approval from the assessor to confirm the user requirements | | | | | | |
| **Checklist (To be completed by the learner’s facilitator)** | | | | | **Yes** | **No** |
| The learner has reviewed the user requirements and needs of the project | | | | |  | ☐ |
| The learner has determined the pages needed and dynamic functionality | | | | |  | ☐ |
| The learner has determined appropriate language & technologies based on required functionality | | | | |  | ☐ |
| The learner has obtained approval from the client about the pages required and the functionality | | | | |  | ☐ |
| **Assessor Name** | *Alex Bicknell* | **Assessor Signature** | *Alex Bicknell* | **Date** | *14.11.2024* | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Part 2 - Plan & Develop the React.js Client** Based on the information you have gathered from the brief complete the following: | | | | | | |
| **Question / Criteria** | | | **Answer / Satisfactory Response** | | | |
| 1. Learner has developed the hierarchy / sitemap of the content to be displayed on the website according to the user requirements   (*Provide ONE image / screenshot of sitemap*) | | | *< Screenshot / image of site map >*  */var/folders/53/v0z34g8n6z95yl9ys0nfz8mc0000gn/T/TemporaryItems/NSIRD_screencaptureui_S9bEbs/Screenshot 2024-11-13 at 3.08.18 PM.png* | | | |
| 1. Learner has designed the overall UI of the website using wireframes for the website’s home page to show content layout   (*Provide ONE image of wireframe for Home page*) | | | *< Wireframe of Home page >../../../../../Downloads/Desktop%20-%2* | | | |
| 1. The learner has built the website navigation system with the routed page links as specified by the brief requirements     (*Provide ONE screenshot of Navigation Bar*) | | | *< Screenshot of Navigation Bar >/var/folders/53/v0z34g8n6z95yl9ys0nfz8mc0000gn/T/TemporaryItems/NSIRD_screencaptureui_cDnvgx/Screenshot 2024-11-13 at 12.23.50 AM.png* | | | |
| 1. The learner has built all required pages as specified in their wireframe & in **Part 1**.   ***NOTE****: Pages planned to render dynamic data are to be presented with mock/dummy static data* ***for this Part 2****.*  (*Provide* ***ONE*** *screenshot of the complete home page*) | | | *< Screenshot of Home page >*  */var/folders/53/v0z34g8n6z95yl9ys0nfz8mc0000gn/T/TemporaryItems/NSIRD_screencaptureui_vVBPMm/Screenshot 2024-11-13 at 12.25.54 AM.png* | | | |
| 1. The application contains functioning routing to functioning pages as specified in the user brief requirements & below.   (*Provide* ***ONE*** *screenshot* ***each*** *of* ***THREE*** *functioning non-home pages, such as “Products” or “Product” pages*) | | | *< Screenshots of* ***THREE*** *Routed Non-Home pages >*  *../../../../../Screenshot%202024-11-13%20at%2012.28.53%20AM.p*  *../../../../../Screenshot%202024-11-13%20at%2012.29.01%20AM.p*  */var/folders/53/v0z34g8n6z95yl9ys0nfz8mc0000gn/T/TemporaryItems/NSIRD_screencaptureui_ptbD9e/Screenshot 2024-11-13 at 12.31.15 AM.png* | | | |
| **FRONT-END “STATIC” REACT CLIENT – DUE SESSION 10** | | | | | | |
| **Build “Static” React.js Front-End**  The React client must have the following framework & UI specifications:   * React.js front-end client built using Vite * CSS-in-JS solution integrated with the React application (*recommended: vanilla-extract*) * App.jsx setup as a routed file using react-router-dom * Demo Home page AND **at least THREE** **(3)** other routed pages including Register, Login & NotFound pages.   At this point, it is strongly recommended you further complete:   * Register & Login forms are integrated with API & functional * Demo Dashboard with functional client-side private routes * Central theming & root CSS variables using vanilla-extract | | | | | | |
| **Approval and Feedback** After you have finished planning the frontend design, obtain approval from the assessor | | | | | | |
| **Checklist (To be completed by the learner’s facilitator)** | | | | | **Yes** | **No** |
| 1. The learner has applied basic language syntax rules, including use of JSX, and best practices to structure their React.js code | | | | |  | ☐ |
| 1. In accordance with the project brief, the learner has used CSS techniques to style, layout and position the document elements of webpages in the website | | | | |  | ☐ |
| 1. The learner has used state to create, manipulate and destroy dynamic variables, including data structures (*such as React.js states*). | | | | |  | ☐ |
| **Assessor Name** | *Alex Bicknell* | **Assessor Signature** | *Alex Bicknell* | **Date** | *14.11.2024* | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Part 3 - Design the Database** Based on the information you have gathered from the brief complete the following: | | | | | | |
| **Question / Criteria** | | | **Answer / Satisfactory Response** | | | |
| 1. Confirm the specific noSQL database being used in the project | | | Google Firebase | | | |
| 1. Provide the following:  * Brief descriptions on the concepts of “horizontal scaling” and “vertical scaling”; and   (*at least ONE sentence each*)   * Describe at least ONE benefit of preferring horizontal scaling databases than to vertical scaled databases | | | < Description on scaling types>  Vertical Scaling: •Scale achieved by making the machine more and more powerful by adding  more machines or nodes to distribute the load across multiple servers  Horizontal Scaling: •Scale achieved by obtaining more machines and linking them, accumulating  joint power, increasing the capacity of a single machine by adding more resources  <Benefits of horizontal scaling >  •Great scaling (scales vertically AND horizontally) •Data simplicity (data is not too complex and can be distributed among nodes for  better accessibility  *.* | | | |
| 1. Compare the SQL with noSQL databases as follows:  * List ONE example of a database for each type; * Confirm which database types are “relational” or “non-relational” and what is meant by each concept; and * Confirm which form(s) of scaling can be utilised by each type | | | < Description of SQL/noSQL databases>   * SQL Databases: SQL (Structured Query Language) databases are relational databases that use tables with rows and columns to store data in a structured format. They are typically used when data integrity and complex querying are required. **Example: MySQL** * NoSQL Databases: NoSQL (Not Only SQL) databases are non-relational and designed to store unstructured or semi-structured data. They offer flexibility in schema design. **Example: Firebase.**   <Relational/non-relational distinction>   * Relational Databases: These are structured and organized in tables with fixed schemas, where data relationships are defined and maintained through foreign keys and structured queries. * Non-relational Databases: These are more flexible with schema, allowing data to be stored in a variety of formats.   <Which scaling is used for SQL and noSQL>   * SQL Databases: Primarily use vertical scaling, increasing resources on a single server. * NoSQL Databases: horizontal, which aligns well with distributed architectures and large-scale applications. | | | |
| 1. List the different data types (**at least THREE**) required used by your selected noSQL database in your application.   Confirm the overall data structure of your database by providing a Firestore database design diagram. | | | Here are three data types used by Firebase in this project   * String * Number * Boolean   Firestore database design diagram:  ../../../../../Screenshot%202024-11-13%20at%205.20.04%20PM.p | | | |
| 1. Confirm the TWO areas in your database where raw data is stored.   (*Provide ONE screenshot of* ***each*** *storage area with posted data in each*) | | | *< TWO screenshots of the TWO storage areas >*  *../../../../../Screenshot%202024-11-13%20at%201.09.45%20AM.p*  */var/folders/53/v0z34g8n6z95yl9ys0nfz8mc0000gn/T/TemporaryItems/NSIRD_screencaptureui_CEKk6h/Screenshot 2024-11-14 at 9.49.41 AM.png* | | | |
| 1. Create a database partition, provide a screenshot the database that includes the different partition “keys” (*i.e. collection names*) and determine whether it is a “partition” or a “shard” | | | *< Screenshots of database partition showing partition key names (“users”) >*  */var/folders/53/v0z34g8n6z95yl9ys0nfz8mc0000gn/T/TemporaryItems/NSIRD_screencaptureui_hmfQP2/Screenshot 2024-11-13 at 5.24.58 PM.png* These are **partitions**, not shards. | | | |
| 1. Comment on the estimated distribution of data between the partitions (*i.e.* “*at least 40% of all application data is in # collection, etc.*”)   (*Minimum ONE sentence*) | | | Based on our application's design, it is estimated that approximately 70% of the data in the products collection due to many data of product listings, 30% in the users collection for customer and admin information | | | |
| **Approval and Feedback** After you have finished planning the database design, obtain approval from the assessor | | | | | | |
| **Checklist (To be completed by the learner’s facilitator)** | | | | | **Yes** | **No** |
| 8. Learner has confirmed approval of database design & supporting documentation with assessor | | | | |  | ☐ |
| **Assessor Name** | *Alex Bicknell* | **Assessor Signature** | *Alex Bicknell* | **Date** | *14.11.2024* | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Part 4 - Develop RESTful API Server** Based on the information you have gathered from the brief complete the following: | | | | | | | |
| **Question / Criteria** | | | **Answer / Satisfactory Response** | | | | |
| 1. Confirm the server-side API runtime environment which interacts with the application’s noSQL database | | | Node.js is the server-side API runtime environment, interacting with NoSQL database | | | | |
|  | | | **Complete / Not Complete** | **Location of files** | | | |
| 1. Create and state the entry point for the routes in the API server | | | *Complete* | **location of entry point file:**  routes.js | | | |
| 1. Create and state the file that handles the initial noSQL database connection | | | *Complete* | **location of database connection file:**  db.js | | | |
| 1. Create and state the file(s) that handle at least ONE of each CRUD operation in the application | | | *Complete* | **location of CRUD operation files:**  productController.js | | | |
| 1. Create and state the file(s) that handle/”controls” the route requests and query the database as required for each CRUD operation | | | *Complete* | **location of database query control file:**  productRoutes.js | | | |
| 1. To confirm CRUD operations & queries are working as intended, provide ONE screenshot each for:   (*TWO screenshots in total for each section*) | | |  | | | | |
| 1. **Displaying a collection of dynamic data - GET Request** 2. the GUI of the page that handles a GET request, and; 3. the successful Postman result for a GET request at the endpoint used. | | | Screenshot of GUI handling GET request  ***/var/folders/53/v0z34g8n6z95yl9ys0nfz8mc0000gn/T/TemporaryItems/NSIRD_screencaptureui_cwv9px/Screenshot 2024-11-13 at 5.47.30 PM.png***  Screenshot of Postman GET result  */var/folders/53/v0z34g8n6z95yl9ys0nfz8mc0000gn/T/TemporaryItems/NSIRD_screencaptureui_aeov5t/Screenshot 2024-11-13 at 5.49.12 PM.png* | | | | |
| 1. **Submitting form data to API – POST request** 2. the GUI of the page that handles a POST request, and; 3. the successful Postman result for a POST request at the endpoint used. | | | Screenshot of GUI handling POST request  */var/folders/53/v0z34g8n6z95yl9ys0nfz8mc0000gn/T/TemporaryItems/NSIRD_screencaptureui_SRQ9Kh/Screenshot 2024-11-13 at 5.53.24 PM.png*  Screenshot of Postman POST result  */var/folders/53/v0z34g8n6z95yl9ys0nfz8mc0000gn/T/TemporaryItems/NSIRD_screencaptureui_8o0KMV/Screenshot 2024-11-13 at 5.52.25 PM.png* | | | | |
| 1. **Editing form data from API – PUT/UPDATE request** 2. the GUI of the page that handles a PUT request, and; 3. the successful Postman result for a PUT request at the endpoint used. | | | Screenshot of GUI handling PUT request  ***/var/folders/53/v0z34g8n6z95yl9ys0nfz8mc0000gn/T/TemporaryItems/NSIRD_screencaptureui_UwvcCU/Screenshot 2024-11-13 at 5.54.27 PM.png***  Screenshot of Postman PUT result  */var/folders/53/v0z34g8n6z95yl9ys0nfz8mc0000gn/T/TemporaryItems/NSIRD_screencaptureui_hfpTRi/Screenshot 2024-11-13 at 6.02.34 PM.png* | | | | |
| 1. **Deleting form data from API – DELETE request** 2. the GUI of the page that handles a DELETE request, and; 3. the successful Postman result for a DELETE request at the endpoint used. | | | Screenshot of GUI handling DELETE request  /var/folders/53/v0z34g8n6z95yl9ys0nfz8mc0000gn/T/TemporaryItems/NSIRD_screencaptureui_533bqB/Screenshot 2024-11-13 at 6.04.41 PM.png  Screenshot of Postman DELETE result  */var/folders/53/v0z34g8n6z95yl9ys0nfz8mc0000gn/T/TemporaryItems/NSIRD_screencaptureui_Sz2N3x/Screenshot 2024-11-13 at 6.06.45 PM.png* | | | | |
| 1. **Displaying a specific set of dynamic data – GET by ID Request** 2. the GUI of the page that handles a GET By ID request, and; 3. the successful Postman result for a GET By ID request at the endpoint used. | | | Screenshot of GUI handling GET By ID request  /var/folders/53/v0z34g8n6z95yl9ys0nfz8mc0000gn/T/TemporaryItems/NSIRD_screencaptureui_QY0yya/Screenshot 2024-11-13 at 6.08.20 PM.png  Screenshot of Postman GET by ID result  */var/folders/53/v0z34g8n6z95yl9ys0nfz8mc0000gn/T/TemporaryItems/NSIRD_screencaptureui_qC4eTm/Screenshot 2024-11-13 at 6.09.53 PM.png* | | | | |
| **Approval and Feedback** After you have finished planning the API design, obtain approval from the assessor | | | | | | | |
| **Checklist (To be completed by the learner’s facilitator)** | | | | | | **Yes** | **No** |
| 7. Learner has confirmed approval of RESTful API & supporting documentation with assessor | | | | | | ☐ | ☐ |
| **Assessor Name** | *Alex Bicknell* | **Assessor Signature** | *Signature* | | **Date** | *Insert date* | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Part 5 - Refine the Client and Server** Based on the information you have gathered from the brief complete the following: | | | | | | |
| **Question / Criteria** | | | **Answer / Satisfactory Response** | | | |
| 1. Incorporate at least ONE sort of key into a query made to the noSQL database and provide ONE screenshot of the code | | | Screenshot of Sort Key query  */var/folders/53/v0z34g8n6z95yl9ys0nfz8mc0000gn/T/TemporaryItems/NSIRD_screencaptureui_7QOYyf/Screenshot 2024-11-13 at 6.12.11 PM.png* | | | |
| 1. Provide TWO screenshots of a Morgan terminal output that details the through-put for TWO CRUD requests (*ONE* ***Read*** *& ONE* ***Write***) | | | Screenshot of Morgan terminal output  */var/folders/53/v0z34g8n6z95yl9ys0nfz8mc0000gn/T/TemporaryItems/NSIRD_screencaptureui_L1YOw9/Screenshot 2024-11-13 at 6.13.41 PM.png*  ***/var/folders/53/v0z34g8n6z95yl9ys0nfz8mc0000gn/T/TemporaryItems/NSIRD_screencaptureui_ltdHsj/Screenshot 2024-11-14 at 12.03.49 PM.png*** | | | |
| 1. List at least TWO methods in which we have or can optimise the through-put of the query above | | | List of at least TWO methods that optimise queries   * Create indexes on frequently queried fields, such as color and category, Indexes allow the database to quickly locate rows that match query conditions, reducing the time spent on searches and improving the overall query speed. * Implement pagination for large datasets, such as product listings. Instead of retrieving all records at once, paginate results to return a limited number of items per page. | | | |
| 1. Create at least TWO composite indexes to optimise queries made to the selected noSQL database   (*Provide ONE screenshot of the created indexes*) | | | Screenshot of BOTH created indexes  */var/folders/53/v0z34g8n6z95yl9ys0nfz8mc0000gn/T/TemporaryItems/NSIRD_screencaptureui_8eoVOC/Screenshot 2024-11-13 at 7.23.18 PM.png* | | | |
| 1. Describe the function of each index created above and how they optimise the query(s) to the database   (*Minimum TWO sentences*) | | | Describe the how the two indexes function and optimise the queries  Color Index:   * The index on the color field allows the database to quickly locate and retrieve products based on color-specific filters, such as when users search for products in a particular color. * The index on the category field optimizes queries that retrieve products by category, allowing for faster access to items within specific categories (e.g., "Basket," "Cushion) * The index on Onsale field allows the database to quickly locate and retrieve product that are allocated as on sale products. such as when users open the on sale product page. | | | |
| 1. Perform a query against at least ONE of the newly created index and provide ONE screenshot of both: 2. the code of the index query; and 3. the change to the GUI page based on the altered query | | | Screenshot of code of index query  ***/var/folders/53/v0z34g8n6z95yl9ys0nfz8mc0000gn/T/TemporaryItems/NSIRD_screencaptureui_1SenX6/Screenshot 2024-11-13 at 7.28.44 PM.png***  Screenshot of changed GUI after index applied  */var/folders/53/v0z34g8n6z95yl9ys0nfz8mc0000gn/T/TemporaryItems/NSIRD_screencaptureui_XnqjyI/Screenshot 2024-11-13 at 7.29.54 PM.png* | | | |
| 1. Implement and state the file(s) which handle authentication, authorisation and one admin level of authorisation for the server-side API | | | Location of file handling authentication/authorisation – verifyAuth.js | | | |
| 1. Create and state the file(s) which encrypt relevant authentication data and the technology used to encrypt the data | | | Location of file handling encryption – authServices.js  Bcrypt is the technology, using to encrypt the data. | | | |
| 1. State ONE technology used in the application to secure the API’s authorisation / authentication processes and describe its function   (*Minimum ONE sentence*) | | | JWT (JSON Web Token) is used to secure the API’s authorization/authentication processes by encoding user information into a token that is sent with each request, allowing the server to verify the user's identity and permissions without maintaining session data. | | | |
| 1. Configure the Time-To-Live (TTL) for the objects created by the “auth” technology above, provide ONE screenshot of the configured code and explain how TTL has been adjusted | | | Explanation of code that has been adjusted for TTL  The jwtSignUser function creates a JSON Web Token (JWT) for a user, which is used for authentication.  Input: Takes user data as the payload.  Secret: Uses a secret key (jwtSecret) to sign the token, ensuring security.  Expiration: Sets the token to expire in 24 hours (60 \* 60 \* 24 seconds).  Token Creation: Calls jwt.sign() to generate the token.  Return: Sends the token back, which can be used by the client for future authenticated requests.  **Screenshot** of TTL configuration  */var/folders/53/v0z34g8n6z95yl9ys0nfz8mc0000gn/T/TemporaryItems/NSIRD_screencaptureui_303VWP/Screenshot 2024-11-13 at 10.26.45 PM.png* | | | |
| 1. State ONE technology used to configure the server software to minimize potential database attacks and describe its function   (*Minimum ONE sentence*) | | | **Firebase Security Rules** enforce access control directly at the database level, allowing you to define conditions based on authentication status and request parameters to restrict data access and prevent unauthorized actions, minimizing potential database attacks. | | | |
| 1. Implement and **state the file(s)** that handle data validation for at least ONE route and confirm the technology used.   Additionally, **include ONE (1) screenshot** of the code which shows how one of the policies structures the data | | | Location of file handling validation – productPolicy.js  Screenshot of the policy code structure example  */var/folders/53/v0z34g8n6z95yl9ys0nfz8mc0000gn/T/TemporaryItems/NSIRD_screencaptureui_p5Rr7p/Screenshot 2024-11-13 at 7.34.30 PM.png* | | | |
| **Approval and Feedback** After you have finished planning the fullstack design, obtain approval from the assessor | | | | | | |
| **Checklist (To be completed by the learner’s facilitator)** | | | | | **Yes** | **No** |
| 13. Learner has refined the client & server structures in accordance with the project brief & reviewed these structures with the assessor | | | | | ☐ | ☐ |
| **Assessor Name** | *Alex Bicknell* | **Assessor Signature** | *Signature* | **Date** | *Insert date* | |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Part 6 - Test and Debug the Code** Based on the information you have gathered from the brief complete the following: | | | | | | | | | | | |
| **Question / Criteria** | | | | | **Answer / Satisfactory Response** | | | | | | |
| 1. List at least TWO debugging tools, ONE of which must be a “standalone” tool | | | | | List of applicable debugging tools/methods   * Postman (Standalone Tool):   Postman is a standalone API testing tool that allows developers to send requests to API endpoints and inspect responses. It helps in debugging by letting test authorization, query parameters, outside the application environment.   * Chrome Developer Tools:   Integrated within the Chrome browser, this tool provides a suite of features for debugging web applications, including inspecting HTML/CSS, monitoring network requests, and debugging JavaScript code. It’s useful for identifying layout issues, analyzing performance, and finding JavaScript errors directly in the browser. | | | | | | |
| 1. Test the client, server and database outputs and security and record the results. Below you will need to **conduct at least SEVEN tests**, each providing the (i) test conducted, (ii) expected output of the test, (iii) success / failure and (iv) comments on test result and debugging.   **The first SIX tests to be conducted have been designed for you – you must design the SEVENTH test:** | | | | | | | | | | |
| **Test Design** | | **Expected Output** | | | **Correct Function** | | | **Comment on Test** | | |
| **Yes** | **No** | |
| 1. **Test:** That each CRUD operation renders the expected output at a specified route which contains all CRUD operations   (***Refer to P4 Q6***) | | *Expect Product List page to be dynamically populated with entries in the products database collection (GET request)* | | |  |  | | State whether the CRUD operations built for **P4 Question 6** are working as required  -getAll, GET ALL – Product list  -getOnsale, GET ALL ONSALE – Product Sale  -getById, GET BY ID – Product Detail  -post, POST – Add Product  -put, PUT – Edit Product  -delete, DELETE – Product Delete | | |
| 1. **Test:** That a 404 Error Page displays when a user navigates to an undefined route and the error stack is provided to the developer in terminal | | *Expect when user navigates to undefined route, to render the custom routed 404 page (see NotFound.jsx)* | | |  |  | | Provide screenshot of rendered 404 Error webpage /var/folders/53/v0z34g8n6z95yl9ys0nfz8mc0000gn/T/TemporaryItems/NSIRD_screencaptureui_EOuJx5/Screenshot 2024-11-13 at 7.42.22 PM.png | | |
| 1. **Test:** Test that the JWT is persistent for the session when users a logged in | | *Expect when logged in, to find the JWT token located in the request headers and for the JWT token to be valid* | | |  |  | | screenshot of JWT token saved to local storage in web browser /var/folders/53/v0z34g8n6z95yl9ys0nfz8mc0000gn/T/TemporaryItems/NSIRD_screencaptureui_lt3GwG/Screenshot 2024-11-13 at 7.44.09 PM.png | | |
| 1. **Test:** Test that a GET route, with dynamically retrieved data, loads and renders correctly in at least **TWO** different browsers, as indicated | | *Expect that the Home page loads correctly in both Google Chrome and Firefox browsers* | | |  |  | | screenshot of webpage in Chrome  /var/folders/53/v0z34g8n6z95yl9ys0nfz8mc0000gn/T/TemporaryItems/NSIRD_screencaptureui_Tb32Xp/Screenshot 2024-11-13 at 7.45.49 PM.png  screenshot of webpage in Firefox /var/folders/53/v0z34g8n6z95yl9ys0nfz8mc0000gn/T/TemporaryItems/NSIRD_screencaptureui_i8YKMf/Screenshot 2024-11-13 at 7.46.55 PM.png | | |
| 1. **Test:** Test that a POST route, which accepts a number, has been converted to the correct number type in the database. | | *Expect the “price” field in the “Add Product” page to be listed as a number data type in Firebase collection* | | |  |  | | screenshot of Firestore database, confirming one dataset saved to a number type  /var/folders/53/v0z34g8n6z95yl9ys0nfz8mc0000gn/T/TemporaryItems/NSIRD_screencaptureui_RVdC5r/Screenshot 2024-11-13 at 7.50.25 PM.png | | |
| 1. **Test:** Test that the logout function removes the user from local storage along with the user token | | *Expect the browsers local storage user data to be cleared and token removed* | | |  |  | | screenshot of JWT token in local storage set to falsy value on logout /var/folders/53/v0z34g8n6z95yl9ys0nfz8mc0000gn/T/TemporaryItems/NSIRD_screencaptureui_rUodbx/Screenshot 2024-11-13 at 8.30.37 PM.png | | |
| 1. **Test:** Test the colour filter | | When user using colour filter picking a colour choice , then Product list is filtered to show just colour users had picked. | | |  |  | | < filtered to show just beige colour>/var/folders/53/v0z34g8n6z95yl9ys0nfz8mc0000gn/T/TemporaryItems/NSIRD_screencaptureui_i2sI6Q/Screenshot 2024-11-13 at 8.37.17 PM.png | | |
| **Approval and Feedback** After you have finished planning the testing & debugging process, obtain approval from the assessor | | | | | | | | | | | |
| **Checklist (To be completed by the learner’s facilitator)** | | | | | | | | | **Yes** | **No** | |
| 3. Learner has finalised testing & debugging of the web application, and has reviewed test completion with the assessor | | | | | | | | |  | ☐ | |
| **Assessor Name** | *Alex Bicknell* | | **Assessor Signature** | *Alex Bicknell* | | | **Date** | | *14.11.2024* | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Part 7 - Document Activities & Hand Over** Based on the information you have gathered from the brief complete the following: | | | | | | | |
| **Skills to be observed during this task to the required standard.** Checklist (To be completed by the learner’s facilitator)The following tasks are to be completed in relation to the brief for this project. Each of the skills must be observed on at least one occasion. | | | | **Date 1** | | **Date 2** | |
| ##.##.2024 | | ##.##.2024 | |
| **Satisfactory** | | **Satisfactory** | |
| **Yes** | **No** | **Yes** | **No** |
| 1. The project has been built using maintainable code and internal documentation (in at least ONE file) according to organisational guidelines and coding standards | | | |  |  |  |  |
| 1. The Client and Server have been designed and implemented according to the user brief requirements | | | |  |  |  |  |
| 1. The project has been created using an integrated development environment (IDE) that meets the needs of the brief | | | |  |  |  |  |
| 1. The project has been tested, debugged and documented according to user requirements and expected outcomes | | | |  |  |  |  |
| 1. Confirm that the final version of the project has been transferred to GitHub as a private repository and provide ONE screenshot of the complete private “repo” in the browser | | | |  |  |  |  |
| 1. The learner has sought to improve protocols governing communications with the “client” and co-workers during the length of the project | | | |  |  |  |  |
| 1. The learner has collaborated with others, sharing information to build strong work groups and interactions with the “client” | | | |  |  |  |  |
| **Assessor Name** | *Alex Bicknell* | **Assessor Signature** | *Signature* | **Date** | | *Insert date* | |