Project 2: React CRUD Assessment Rubric

	10-9	8-7	6-5	4-0
Functionality	Applications demonstrate comprehensive & robust evidence on the following: • API data requested from API resource groups using Axios. • Create, update & delete API data via modal. • Incorrectly formatted form fields are handled using validation error messages. • Data automatically re-rendered after creating, updating and deleting. • API data paginated across several pages. • View API data in a table using an id and a variety of query parameters. • UI styled with Reactstrap. • Application deployed to Heroku. • Components tested using React Testing Library.	Applications demonstrate clear & detailed evidence on the following: • API data requested from API resource groups using Axios. • Create, update & delete API data via modal. • Incorrectly formatted form fields are handled using validation error messages. • Data automatically re-rendered after creating, updating and deleting. • API data paginated across several pages. • View API data in a table using an id and a variety of query parameters. • UI styled with Reactstrap. • Application deployed to Heroku. • Components tested using React Testing Library.	Applications demonstrate evidence on the following: • API data requested from API resource groups using Axios. • Create, update & delete API data via modal. • Incorrectly formatted form fields are handled using validation error messages. • Data automatically re-rendered after creating, updating and deleting. • API data paginated across several pages. • View API data in a table using an id and a variety of query parameters. • UI styled with Reactstrap. • Application deployed to Heroku. • Components tested using React Testing Library.	Applications does not, or does not fully demonstrate evidence on the following: • API data requested from API resource groups using Axios. • Create, update & delete API data via modal. • Incorrectly formatted form fields are handled using validation error messages. • Data automatically re-rendered after creating, updating and deleting. • API data paginated across several pages. • View API data in a table using an id and a variety of query parameters. • UI styled with Reactstrap. • Application deployed to Heroku. • Components tested using React Testing Library.

Code Elegance	Applications thoroughly demonstrates code elegance on the following: • Appropriate use of control flow, data structures and in-built functions. • Sufficient code modularity. • Components written as functional, not class. • Adheres to client-server architecture. • Header & in-line comments explain complex logic. • Formatted code using Prettier & npm script. • No dead or unused code.	Applications clearly demonstrates code elegance on the following: • Appropriate use of control flow, data structures and in-built functions. • Sufficient code modularity. • Components written as functional, not class. • Adheres to client-server architecture. • Header & in-line comments explain complex logic. • Formatted code using Prettier & npm script. • No dead or unused code.	Applications demonstrates code elegance on the following: • Appropriate use of control flow, data structures and in-built functions. • Sufficient code modularity. • Components written as functional, not class. • Adheres to client-server architecture. • Header & in-line comments explain complex logic. • Formatted code using Prettier & npm script. • No dead or unused code.	Applications does not or does not fully demonstrate code elegance on the following: • Appropriate use of control flow, data structures and in-built functions. • Sufficient code modularity. • Components written as functional, not class. • Adheres to client-server architecture. • Header & in-line comments explain complex logic. • Formatted code using Prettier & npm script. • No dead or unused code.
Documentation & Git Usage	README file contains thoroughly evidence of: • URL to application on Heroku. • How to setup the environment for development & deploy the application. Git branches thoroughly named with convention & contain the correct code relating to the functional requirement. Git commit messages comprehensively formatted & reflect the functionality changes in succinct detail.	README file contains clear evidence of: • URL to application on Heroku. • How to setup the environment for development & deploy the application. Git branches mostly named with convention & contain the correct code relating to the functional requirement. Git commit messages clearly formatted & reflect the functionality changes in substantial detail.	README file contains evidence of: • URL to application on Heroku. • How to setup the environment for development & deploy the application. Some git branches named with convention & contain the correct code relating to the functional requirement. Git commit messages formatted & reflect the functionality changes in detail.	README file does not or does not fully contain evidence of: • URL to application on Heroku. • How to setup the environment for development & deploy the application. Git branches are not or are not fully named with convention & do not or do not fully contain the correct code relating to the functional requirement. Git commit messages are not or are not fully formatted & do not or do not reflect the functionality changes.

Project 2: React CRUD App Marking Cover Sheet

Name:

Date:

Learner ID:

Assessor's Name:

Assessor's Signature:

Criteria	Out Of	Weighting	Final Result
Functionality	10	40	
Code Elegance	10	45	
Documentation & Git Usage	10	15	
Final Result			/100
This assessment is worth 50% of the final mark for the Introductory Application Development Concepts course.			

Feedback:

Functionality:

Code Elegance:

Documentation & Git Usage: