# Student Version

| Section A – Program/Course details | | | |
| --- | --- | --- | --- |
| **Qualification code:** | ICT50615 | **Qualification title:** | Diploma of Website Development |
| **Unit code:** | (D8)  ICTWEB501 ICTWEB502 ICTWEB503 ICTDBS504 | **Unit title:** | (Dynamic Web Development)  Build a dynamic website Create dynamic web pages Create web-based programs Integrate database with a website |
| **Department name:** | BDIT, Computing & Information Technology | **CRN number:** | Enter CRN number |

| Section B – Assessment task details | | | |
| --- | --- | --- | --- |
| **Assessment number:** | 2 of 2 | **Semester/Year:** | 2/2020 |
| **Due date:** | 20-Nov-20 | **Duration of assessment:** | 5 weeks |
| **Assessment method** | Project/Report/Portfolio | **Assessment task results** | Ungraded result |
| Other: Click here to enter text. |

| Section C – Instructions to students |
| --- |
| **Task instructions:** |
| This assessment task requires learners to build a server-side API that saves and stores data in a database, along with a client side application for the server-side APIs endpoints. The scripts will need to demonstrate introductory object orientated programming techniques. The teacher will be playing the role of the client for this assessment task. The server-side API the learner developers may be about or for any topic, company or community of the learner’s choice. As long as the website meets the requirements outlined in this project.    This assessment has been divided into 6 key parts:  Part 1 – Identify and Confirm Design Requirements Part 2 – Develop a server-side API Part 3 – Develop the client side application for a server-side API Part 4 – Testing and Debugging  Part 5 – Document the API  Part 6 – Approval and Feedback    REFER TO SUPPORTING DOCUMENTS FOR DETAILED INSTRUCTIONS |

| Section D – Conditions for assessment | |
| --- | --- |
| **Conditions:**  Student to complete and attach Assessment Submission Cover Sheet to the completed Assessment Task. | |
| Add conditions of assessment here. Include general instructions that are appropriate for the task, such as: - This is an individual assessment. - Discuss with your assessor if you feel you require special consideration or adjustment for this task. - Any options for negotiating assessment tasks. - Students must meet all criteria listed in the marking guide to be satisfactory in this task. - Students may resubmit this task if not successful within the enrolment period as per Holmesglen conducting  assessment procedure. - If it is open or closed book. - The learner may use the internet for research. | |
| **Equipment/resources students must supply:** | **Equipment/resources to be provided by the RTO:** |
| GitHub Account GitHub Client Code Editor Browser Computer Internet Design Software Local Server Database Database Management Tools  Requirements document Postman | Computer Internet access Microsoft word  GitHub Client Node.js Browser Design Software Local Server Database Database Management Tools  Requirements document  Postman Visual Studio Code: https:/code.visualstudio.com/Bootstrap: http:/getbootstrap.com |

|  |  |  |  |
| --- | --- | --- | --- |
| Section E – Marking Sheet - Student Answer Sheet | | | |
| **Student ID:** | **1076912** | **Student name:** | Sanjith Padmadas Das |
| **Unit code:** | (D8)  ICTWEB501 ICTWEB502 ICTWEB503 ICTDBS504 | **Unit title:** | (Dynamic Web Development)  Build a dynamic website Create dynamic web pages Create web-based programs Integrate database with a website |
| **Date:** |  | | |

**Portfolio task**

| **Criteria for assessment** | | **Satisfactory** | | **Comment** |
| --- | --- | --- | --- | --- |
| **Yes** | **No** |
| **Marking criteria:** Part 1 - Identify and Confirm Design Requirements Based on the information you have gathered from the brief complete the following:Assessment Documentation | | | | |
| 1. | The learner has identified all core business requirements and standards the project must adhere to. | ☐ | ☐ |  |
| 2. | The learner has determined the purpose of the web application based on the brief. | ☐ | ☐ |  |
| 3. | Within a minimum of 1 paragraph the learner has determined the core expectations of the project. | ☐ | ☐ |  |
| 4. | A list of all the main required functionality of the website has been compiled. (A minimum of 6 points) | ☐ | ☐ |  |
| 5. | Based on the user interface design and brief requirements the learner has created two wireframe for two of the main views of the application. | ☐ | ☐ |  |
| 6. | Based on the user interface design, brief requirements and the learners wireframes; design of one view of the learners application has been designed. | ☐ | ☐ |  |
| 7. | The learner has identified all their required distribution platforms | ☐ | ☐ |  |
| 8. | A HTML and CSS prototype for one of the views has been created by the learner based on the final design | ☐ | ☐ |  |
| 9. | A list of all architectural requirements required to build the learners application have been developed. | ☐ | ☐ |  |
| 10. | The learner has designed their database | ☐ | ☐ |  |
| 11. | The protocol used to send and receive data between the client and the server has been identified. | ☐ | ☐ |  |
| 12. | 2 limitations of HTTP when developing a web application have been identified. | ☐ | ☐ |  |
| 13. | 2 advantages of HTTP when developing a web application have been identified. | ☐ | ☐ |  |
| 14. | A List of 4 pages/components that require dynamic content for the client side application have been identified. | ☐ | ☐ |  |
| 15. | A list of all API endpoints that will need to be created to delivery dynamic content has been created. | ☐ | ☐ |  |
| **Marking criteria:**  Part 2 - Devolve a server-side API Assessment Documentation/Working files | | | | |
| 1. | The learner has create application routes and divided the routes into separate modules. | ☐ | ☐ |  |
| 2. | The application establishes a database connection | ☐ | ☐ |  |
| 3. | The learner has implement a schema as required for all CRUD operations | ☐ | ☐ |  |
| 4. | The learner has implement user data validation for each route using a module such as JOI | ☐ | ☐ |  |
| 5. | CRUD operations have been developed for each route | ☐ | ☐ |  |
| 6. | 404 error pages and 500 error pages have been implemented | ☐ | ☐ |  |
| 7. | Try catch has been used to catch errors | ☐ | ☐ |  |
| 8. | SQL statements to retrieve data from an MySQL database have been created as part of a server-side API | ☐ | ☐ |  |
| **Marking criteria:**  Part 3 - Develop the client side application for a server-side API  Assessment Documentation/Working files | | | | |
| 1. | A client side application with all necessary Vue.js/React.js components that integrate with your server-side API has been created. | ☐ | ☐ |  |
| 2. | The UI and UX for the application is logical and accessible | ☐ | ☐ |  |
| 3. | All required data from the API has been displayed in appropriate controls. | ☐ | ☐ |  |
| 4. | All Data received from the server-side API has been formatted so that it can be easily understood and the formatting aligns with the overall UI design. | ☐ | ☐ |  |
| 5. | The client side app keeps track of data between browser requests. | ☐ | ☐ |  |
| 6. | At least one example of client side app saving user interaction with the website has been incorporated into the project. | ☐ | ☐ |  |
| 7. | Code to handle session management has been created. | ☐ | ☐ |  |
| **Marking criteria:**  Part 4 - Testing and Debugging  Assessment Documentation/Working files | | | | |
| 1. | The client side application has been tested in a browser and the layout behaves as expected. | ☐ | ☐ |  |
| 2. | All the dynamic features of the client side application behave as expected. | ☐ | ☐ |  |
| 3. | All routes and CRUD operations have been tested using postman. | ☐ | ☐ |  |
| 4. | The 404 error page is displayed when a user inputs an undefined route. | ☐ | ☐ |  |
| 5. | The learner has tested the UI and UX for the application to ensure it is logical and accessible. | ☐ | ☐ |  |
| 6. | The learner has used the language debugging facilities of an integrated development environment (IDE) to examine and output results and variables. (On two occasions) | ☐ | ☐ |  |
| 7. | The learner has interpreted the compiler or interpreter messages to resolve syntax errors, and use debugging techniques to resolve logic errors. (On two occasions) | ☐ | ☐ |  |
| 8. | The learner can use the code stack trace to detect and correct errors. (On two occasions) | ☐ | ☐ |  |
| **Marking criteria:**  Part 5 - Document the API  Assessment Documentation | | | | |
| 1. | An endpoint for the learners server-side API has been documented | ☐ | ☐ |  |
| 2. | The learner has developed a Resource Description for the endpoint | ☐ | ☐ |  |
| 3. | Methods and description have been listed for the learners endpoint. | ☐ | ☐ |  |
| 4. | All parameters for the endpoint methods have been documented | ☐ | ☐ |  |
| 5. | Request examples have been provided for each of the endpoints methods | ☐ | ☐ |  |
| 6. | Response examples have been provided for each of the endpoints methods | ☐ | ☐ |  |
| **Marking criteria:**  Part 6 - Approval and Feedback  Assessment Documentation/Working files | | | | |
| 1. | The learner has demonstrated and tested the prototype with the client including errors handling. | ☐ | ☐ |  |
| 2. | The learner has incorporated feedback from the client into the prototype | ☐ | ☐ |  |
| 3. | The learner has obtained client sign-off for the prototype | ☐ | ☐ |  |
| 4. | The learner has demonstrated and tested the prototype with the client including errors handling. | ☐ | ☐ |  |

|  |  |  |
| --- | --- | --- |
| **Student declaration** | |  |
| By submitting this assessment task and signing the below, I acknowledge and agree that:   1. This completed assessment task is my own work. 2. I understand the serious nature of plagiarism and I am aware of the penalties that exist for breaching this. 3. I have kept a copy of this assessment task. 4. The assessor may provide a copy of this assessment task to another member of the Institute for validation and/or benchmarking purposes. | | |
| **Submission or observation date:** |  | |
| **Student signature**  For electronic submissions: By typing your name in the student signature field, you are accepting the above declaration. | **Sanjith Padmadas Das** | |

| Assessment Results and Feedback to Student | | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| **Assessment Task Result:** | | | | **Satisfactory** | | **Not Satisfactory** |
|  | |  |
| **Assessor’s Feedback:** | | | | | | |
|  | | | | | | |
| **Resubmission allowed:** | **Yes** | **No** | **Resubmission due date:** | |  | |
| **Assessor name:** | Daniel Fitzsimmons | | | | | |
| **Assessor signature:** |  | | | | | |
| **Date assessed:** |  | | | | | |

**Supporting document**

# Portfolio Instructions

|  |  |  |  |
| --- | --- | --- | --- |
| **Unit code**: | (D8)  ICTWEB501 ICTWEB502 ICTWEB503 ICTDBS504 | **Unit title:** | (Dynamic Web Development)  Build a dynamic website Create dynamic web pages Create web-based programs Integrate database with a website |
| **Student ID:** | **1076912** | **Student name:** | Sanjith Padmadas Das |

|  |
| --- |
| **Project Brief** Read through the brief below take note of the needs and requirements outlined in the brief. |
| **Introduction**  This assessment task requires learners to build a API that saves and stores data in a database, along with a client side application for the APIs endpoints. The purpose of the web application is to demonstrate a learners ability to create a API and client side application that performs operation on the APIs endpoints. The teacher will be playing the role of the client for this assessment task. The API the learner developers may be about or for any topic, company or community of the learner’s choice. As long as the website meets the requirements outlined in this project.  This assessment has been divided into 6 key parts:  Part 1 – Identify and Confirm Design Requirements  Part 2 – Develop a server side API  Part 3 – Develop the client side application for a server side API  Part 4 – Testing and Debugging  Part 5 – Document the API  Part 6 – Approval and Feedback  **Server-side API requirements**   * A API that returns JSON at various endpoints (If the API is built using the GraphQL standard then only one endpoint is required) * The API data must be saved and retrieved from a database * All 4 core CRUD operation must be used in your application * All data must be validated using a module such as JOI * A schema should be created that validates the data before sending it to the database * Authentication and Authorization must be implemented * Some of the endpoints must be secured such as an admin endpoint * Error pages or responses such as 404 & 500 errors must be created and implemented   **Client side requirements**   * A GUI for the APIs endpoints must be created using Vue or React * A responsive framework such as Bootstrap * HTML5 & CSS3 * Ensure that the UI and UX for the application is logical and accessible * Display data from the API in appropriate controls, such as text boxes, drop downs, radio buttons etc. * Format data received from the API so that it can be easily understood and digestated by the user. Ensure this formatting aligns with your overall UI design. * Ensure you client side app keeps track of data between browser requests. e.g. You application keeps the current state of the application between requests. * Save user interaction with the website. e.g. Your web application save and stores the token supplied by the API for future use.   **Security Standards**   * All data must be validated before saving to the database * Authentication and Authorization must take place before any data is saved to the database * Application Secrets must be stored in environmental variables * It is recommended that a module such as helmet be used to further protect the application and database   **Coding standards and maintainability**  All code should be commented clearly   * **Classes and Modules** - A descriptive overview should be provided for each class and 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.   **Other Standards**   * HTML Standards must be adhered to wherever possible * CSS Standards must be adhered to wherever possible * HTML accessibility standards |

|  |  |
| --- | --- |
| **Part 1 - Identify and Confirm Design Requirements** Based on the information you have gathered from the brief complete the following:Assessment Documentation | |
| 1. Identify the business requirements and standards the project must adhere to. |  |
| 1. Determine the purpose of the web application based on the brief.  (2 Sentences minimum) | ***The purpose of this web application is to create Job portal. The modules included in this projects are***   1. ***Job Seeker Registration*** 2. ***Employer Registration*** 3. ***Job Seeker Login*** 4. ***Employer Login*** 5. ***Administrator Module*** 6. ***Listing all companies*** 7. ***Searching Jobs*** 8. ***Listing Job Categories*** 9. ***Token based authentications*** |
| 1. Determine the core expectations of the project (1 Paragraph minimum) | ***The expectation for this web application is to create Job portal. The Job Seeker can search Jobs based on their requirement’s, for e.g. they can search with Job Title, Job Type, Category etc. Once the seeker successfully registered they can apply for any jobs they want to apply. They can submit their profiles including resume and cover letter***  ***Once the Employer (Company) is authenticated, they have the provision to create the company, create the jobs, review users submission etc.***  ***Administrator user will check all the activities and should approve all the user interactions before these details will display to the user.*** |
| 1. List the main required functionality of the website (List a minimum of 6) | *1* ***Job Seeker Registration***  ***2 Employer Registration***  ***3 Job Seeker Login***  ***4 Employer Login***  ***5 Administrator Module***  ***6 Listing all companies***   1. ***Searching Jobs*** 2. ***Listing Job Categories***   *Employer has the options to create, update and delete their jobs once they are authenticated.*  *Administrator has the options to view, update, delete and create more jobs. Once the admin user is approved the listing of the Jobs will be done* |
| 1. Based on the user interface design and brief requirements create two wireframes for two of the main views of the application. | |
|  | |
|  | |
| 1. Based on the user interface design, brief requirements and your wireframes - design one view of your application using an application such as Photoshop or Adobe XD | |
| ***Please refer the attached document (******Project Guides and Screen Shots) for all the screenshots.*** | |
| 1. Identify the distribution platform and software requirements. E.g. Heroku, MongoDB, Atlas etc. | ***1. Vue***  ***2. Bootstrap***  ***3. JQuery, Java Script***  ***4. HTML, CSS***  ***5. Sequelize ORM***  ***6. MySQL Dtabase*** |
| 1. Produce a HTML and CSS prototype for one of your views based on your final design.   Included a screenshot of the desktop and mobile views. | |
| ***Please refer the attached document (Project Guides and Screen Shots) for all the screenshots.*** | |
| ***Please refer the attached document (Project Guides and Screen Shots) for all the screenshots.*** | |
| 1. List the architectural requirements required to build this application.   This would include the stack you have selected such as MERN and any application and tools needed for the application and deployment. | ***This website is designed based on the Model View Controller design pattern.***  ***Model - MySQL***  ***View - Vue***  ***Controller – Express server (Node)***  ***The request from the vue (user view) is send to the server (express), which intern process the request (by controller) , and update the database (connect to the model db) if necessary and returns the response to the user.*** |
| 1. Design the database | |
|  | |
| ***Please refer the attached document (Project Guides and Screen Shots) for all the screenshots.*** | |
| 1. What protocol will you use to send and receive data between the client and the server? | ***HTTP protocol we are using to send and receive data between client and server. Using HTTP should be the better practice for all data if possible. However it is really important when sending and receiving any confidential information such as user login credentials and personal details.*** |
| 1. List 2 limitations of HTTP when developing a web application using a stack such as MERN or MEVN. | 1. ***HTTP is a stateless protocol which does not store state between each request. So with HTTP , it is not possible identify the user among multiple users. We need to include some session handling mechanism to manage the session between requests.*** 2. ***Every time need to make an HTTP request in order to sent and retrieve data from the server*** 3. ***It also requires an active internet connection to send and receive request.*** 4. ***When a response come from the server, the user page refreshes or re-routes the user to a new page. The only way to prevent this from happening to use AJAX (Asynchronous Java Script and XML)*** |
| 1. List 2 advantages of HTTP when developing a web application using a stack such as MERN or MEVN. | 1. ***When using HTTPS, data across the network is completely secure using Secure Socket Layer encryption method.*** 2. ***Firewall friendly – Most companies allow HTTP(s) traffics through their company firewall. Reducing amount of network administration*** 3. ***Easy access – Backed up files can be easily accessed from a web server****.* |
| 1. List the pages/components of the client side application that will require dynamic content | 1. ***Home*** 2. ***Seeker*** 3. ***Employer*** 4. ***Company*** 5. ***Jobs (Create and Edit)*** 6. ***Category*** |
| 1. List the API endpoints that will need to be created to delivery dynamic content | 1. ***Fetching all the jobs from the database***   ***Method: GET***  ***Endpoint:*** [**http://localhost:8081/api/alljobs**](http://localhost:8081/api/alljobs)   1. ***Fetching all the company details***   ***Method: GET***  ***Endpoint:*** [**http://localhost:8081/api/allcompanies**](http://localhost:8081/api/allcompanies)   1. ***Retrieving the user of a particular company***   ***Method: GET***  ***Endpoint:*** [**http://localhost:8081/api/admin/comapny/user/{companyId}**](http://localhost:8081/api/admin/comapny/user/%7bcompanyId%7d)   1. **Get all categories**   **Method: GET**  **Endpoint:** [**http://localhost:8081/api/alljobcategory**](http://localhost:8081/api/alljobcategory)   1. **Search a job (conditional searching based on user input)**   **Method: GET**  **Endpoint:** [**http://localhost:8081/api/jobs/{search**](http://localhost:8081/api/jobs/%7bsearch)**}** |

|  |  |  |
| --- | --- | --- |
| **Part 2 - Develop a server-side API**  Working Files | | |
| It is now time to start building our database and server-side API. Make sure you complete each of the stages outlined below and ensure that your database and server-side API align with the brief. | | |
|  | **Complete/ Not Complete** | **Location of files that demonstrate if relevant.** |
| 1. Create application routes and divide the routes into separate modules. |  | routes/ |
| 1. Establish database connection |  | index.js |
| 1. Implement Schemas as required for each route |  | models/ |
| 1. Implement user data validation for each route using a module such as JOI |  | models/ |
| 1. Develop CRUD operations for each route as required and ensure these CRUD operations are updating the database as expected. |  | routes/ |
| 1. Implement 404 error pages and 500 error pages |  | Index.js |
| 1. Handle errors using try catch |  | routes/index.js routes/user.js |
| 1. As a small separate project develop and build a simple server-side API using MySQL and Node.js |  | RESTful\_API\_MySQL/ |

|  |  |  |
| --- | --- | --- |
| **Part 3 - Develop the client side application for a server-side API** Working Files | | |
| It is now time to start building our Client Side application using Vue.js. Make sure you complete each of the stages outlined below and ensure that your client side app aligns with the brief and your project plan. | | |
|  | **Complete/ Not Complete** | **Location of files that demonstrate if relevant.** |
| 1. Build a client side application and necessary Vue.js components that integrate with your server-side API |  | src/components |
| 1. Ensure that the UI and UX of the application is logical and accessible |  | NA |
| 1. Display data from the API in appropriate controls, such as text boxes, drop downs, radio buttons etc. |  | src/components |
| 1. Format data received from the server-side API so that it can be easily understood and digested by the user. Ensure this formatting aligns with your overall UI design. |  | src/components public/index.html |
| 1. Ensure you client side app keeps track of data between browser requests. e.g. You application keeps the current state of the application between requests. |  | src/components/ |
| 1. Write code that saves user interaction with the website. e.g. Your web application save sand stores the token supplied by the API for future use. |  | src/components/login |
| 1. Create code to handle session management |  | src/components/login |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Part 4 - Testing and Debugging**Assessment Documentation | | | | | | | | | | | | |
| Test the database and application security and record the results. Below you will need to design and record tests for your server-side API and Database. The result must be recorded. | | | | | | | | | | | | |
| **Testing Requirements** | | | | **Comment if required** | | | **Functions as Expected** | | | | | |
| **Yes** | | | | **No** | |
| 1. Test your client side application in browsers and ensure the layout behaves as expected. | | | |  | | |  | | | |  | |
| 1. Ensure the dynamic features of your client side application behave as expected.   (Your application loads and send data to and from your API as intended) | | | |  | | |  | | | |  | |
| 3. Test all routes and CRUD operations using postman. | | | | E.g. All CRUD operation and routes behave as expected | | |  | | | |  | |
| 4. Test that the 404 error page is displayed when a user inputs an undefined route. Input several undefined routes and check response. | | | | E.g. The 404 error page was displayed for each undefined route | | |  | | | |  | |
| 5. Ensure that the UI and UX of the application is logical and accessible. Have 2-3 people test the layout and design, while observing them and gathering feedback. | | | | E.g. Users were intuitively able to navigate the UI and UX with no issues discerning how to perform key operations. | | |  | | | |  | |
| **Debug the code** You will need to organise with your facilitator to observe you debug your code and demonstrate the following skills on two separate occasions.  Assessment Documentation | | | | | | | | | | | | |
| **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 two separate occasions. These may occur on the same day. | | | | | | **Date 1** | | | | **Date 2** | | |
| --/--/-- | | | | --/--/-- | | |
| **Satisfactory** | | | | **Satisfactory** | | |
| **Yes** | | **No** | | **Yes** | | **No** |
| 1. The learner has used the language debugging facilities of an integrated development environment (IDE) to examine and output results and variables. | | | | | |  | |  | |  | |  |
| 1. The learner has interpreted the compiler or interpreter messages to resolve syntax errors, and use debugging techniques to resolve logic errors | | | | | |  | |  | |  | |  |
| 1. Use the code stack trace and to detect and correct errors | | | | | |  | |  | |  | |  |
| **Assessor Name** | *Daniel Fitzsimmons* | **Assessor Signature** | *Signature* | | **Date** | | | | --/--/-- | | | |

|  |  |
| --- | --- |
| **Part 5 - Document the API**  Assessment Document | |
| Develop technical documentation for one of the endpoints of your API | |
| 1. Server-side API URL endpoint *The URL for this endpoint.* | E.g.   **ENDPOINT URL**  **1.** [**http://localhost:8081/api/alljobs**](http://localhost:8081/api/alljobs)  **Method: GET**  **Description: This endpoint connect to the MYSQL database using express server to fetch all the jobs.**   1. **Get all categories**   **Method: GET**  **Endpoint:** [**http://localhost:8081/api/alljobcategory**](http://localhost:8081/api/alljobcategory)   1. **Search a job (conditional searching based on user input)**   **Method: GET**  **Endpoint:** [**http://localhost:8081/api/jobs/{search**](http://localhost:8081/api/jobs/%7bsearch)**}** |
| 1. Resource description   *“Resources” refers to the information returned by an API.*  *Typically, an API will have a number of endpoints grouped under the same resource. In this case, you describe both the general resource and the individual endpoints.*  Example for URL param  **GET:** /users/{user\_id} | **RESOURCE DESCRIPTION**  **1. Endpoint:** [http://localhost:8081/api/jobs/{search}](http://localhost:8081/api/jobs/%7bsearch%7d)  Method: GET  Parameter (Path Param) : Search query  Description: Search job based on the parameter provided.   1. **Endpoint:** [http://localhost:8081/api/admin/comapny/user/{companyId}](http://localhost:8081/api/admin/comapny/user/%7bcompanyId%7d)   **Method:** GET  **Parameter (Path) :** Company Id  **Description:** Retrieve the user details of a particular company |
| 1. Endpoint descriptions and Methods  (Methods refer to POST, GET etc.) *If you have path parameters in your endpoint, represent them through curly braces.  /campaigns/{campaign\_id}/actions/send* | **GET Endpoint**   1. ***Fetching all the jobs from the database***   ***Method: GET***  ***Endpoint:*** [**http://localhost:8081/api/alljobs**](http://localhost:8081/api/alljobs)   1. ***Fetching all the company details***   ***Method: GET***  ***Endpoint:*** [**http://localhost:8081/api/allcompanies**](http://localhost:8081/api/allcompanies)   1. ***Retrieving the user of a particular company***   ***Method: GET***  ***Endpoint:*** [**http://localhost:8081/api/admin/comapny/user/{companyId}**](http://localhost:8081/api/admin/comapny/user/%7bcompanyId%7d)  **Parameter (Path) :** Company Id   1. **Get all categories**   **Method: GET**  **Endpoint:** [**http://localhost:8081/api/alljobcategory**](http://localhost:8081/api/alljobcategory)   1. **Search a job (conditional searching based on user input)**   **Method: GET**  **Parameter (Path): search**  **Endpoint:** [**http://localhost:8081/api/jobs/{search}**](http://localhost:8081/api/jobs/%7bsearch%7d)  **POST Endpoints**   1. **Create/ register a user**   **Method: POST**  **Endpoint:** <http://localhost:8081/api/user/create>   1. **Create a new job**   **Method: POST**  **Endpoint:** <http://localhost:8081/api/jobs/create>   1. **Create a new Category**   **Method:Post**  **Endpoint:** <http://localhost:8081/api/category/create>  **PUT Endpoints**   1. **Edit/Update a job with a given id**   **Method: PUT**  **Endpoint:** [http://localhost:8081/api/edit/job/{id}](http://localhost:8081/api/edit/job/%7bid%7d)  **Delete Endpoint**   1. **Delete a particular job from the database**   **Method: DELETE**  **Endpoint:** http://localhost:8081/api/delete/jobs/{id} |
| 1. Parameters *Parameters are options you can pass with the endpoint (such as specifying the response format or the amount returned) to influence the response. There are four types of parameters: header parameters, path parameters, query string parameters, and request body parameters.*   *It is good practice to organize your parameters into these 4 categories in your documentation.* | E.g.  **POST PARAMETERS POST:** <http://localhost:8081/api/user/create>  **HEADER PARAMS**  None  **PATH PARAMS**  None  **QUERY PARAMS**  None  **BODY PARAMS**   |  |  | | --- | --- | | username | required | | string | The supplied name must have a character length between 5 and 50 |  |  |  | | --- | --- | | email | required | | string | The email must be unique.  The email must be valid.  The supplied name must have a character length between 5 and 255. |  |  |  | | --- | --- | | dob | required | | Date | Date of birth should be in the date format. |  |  |  | | --- | --- | | password | required | | string | The password must be between 10 and 50 characters.  The password must contain uppercase, lowercase letters and at least one special character. |  |  |  | | --- | --- | | cpassword | required | | string | The confirm password must match with the password |   **BODY Params**  {       "username" :"Ryan",        "email":"dryan@gmail.com",        "dob":"12/10/2011"  ,        "password": "Sanjithdas123@",        "cpassword" :"Sanjithdas123@"    }  **GET PARAMETERS GET:** http://localhost:8081/api/user/{myid}  **HEADER PARAMS**   |  |  | | --- | --- | | x-auth-token | required | | string | Encrypted JWT token supplied by the API on successful login. |   **PATH PARAMS**   |  |  | | --- | --- | | myid (userid) | required | | number | Indicates to the endpoint that the user would like to view their personal info. | | Example http://localhost:8081/api/user/2 | |   **QUERY PARAMS**  None  **BODY PARAMS**  None |
| 1. Request example | E.g.  **POST REQUEST EXAMPLE** 1. http://localhost:8081/api/user/create  **BODY PARAMS**  {       "username" :"SanjithDas",        "email":"sdas@gmail.com",        "dob":"12/10/1988"  ,        "password": "Sanjithdas@gmailcom",        "cpassword" :"Sanjithdas@gmailcom"    }   1. <http://localhost:8081/api/company/create>   {              "user\_id": 5,              "cname": "DataProbe",              "slug": "data-probe",              "address": "255 Right Street",              "phone": "25455554",              "websites": "dprobe.com",               "logo": "DProbe.jpg",              "cover\_photo": "DataProbe.jpg",              "slogan": "We deliver best products and supports",              "description": "We are one of the leading IT firm in worldwide"    }  **GET REQUEST EXAMPLE GET:** http://localhost:8081/api/user/{myId}  **HEADER PARAMS**  eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey  JpZCI6NiwidXNlcm5hbWUiOiJEcmlzaHlhIiwid  XNlcl90eXBlIjoic2Vla2VyIiwiZW1haWwiOiJk  YWRhc2RAcmlzaC5jb20iLCJlbWFpbF92ZXJpZml  lZF9hdCI6bnVsbCwicGFzc3dvcmQiOiJzYW5qaX  RoZGFzMTIzIiwicmVtZWJlcl90b2tlbiI6bnVsb  CwiY3JlYXRlZEF0IjoiMjAyMC0xMC0wMVQwMToy  MjowMS4wMDBaIiwidXBkYXRlZEF0IjoiMjAyMC0  xMC0wMVQwMToyMjowMS4wMDBaIiwiaWF0IjoxNj  A1MTM3NzM4LCJleHAiOjE2MDU3NDI1Mzh9.Dz  EOm-PgRM-CMsomkktJL8MkA6nAEQFs5shJ7uYI72I |
| 1. Response example and schema | E.g. **POST RESPONSE EXAMPLE** http://localhost:8081/api/user/create  **BODY PARAMS**   |  |  | | --- | --- | | Field | Description | | User | If true, return the created user as well | | message | Message specifying the status of the creation of the user account | | Error | Error messages in relation to user input. Overwise null |   **EXAMPLE**  {  "message": "User created successfully",      "user": {          "id": 60,          "username": "SanjithDas",          "email": "sdas@gmail.com",          "password": "$2b$10$4BKT/iJfZqjeNKMgdKxqHu4MZat1wOWPbXQR1OgtbLADQF1XtDZ.m",          "updatedAt": "2020-11-11T23:21:39.588Z",          "createdAt": "2020-11-11T23:21:39.588Z"      },  }  **HEADER PARAMS**   |  |  | | --- | --- | | Field | Description | | x-auth-token: | JWT token supplied by the API |   **EXAMPLE**  x-auth-token:  eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey  JpZCI6NiwidXNlcm5hbWUiOiJEcmlzaHlhIiwid  XNlcl90eXBlIjoic2Vla2VyIiwiZW1haWwiOiJk  YWRhc2RAcmlzaC5jb20iLCJlbWFpbF92ZXJpZml  lZF9hdCI6bnVsbCwicGFzc3dvcmQiOiJzYW5qaX  RoZGFzMTIzIiwicmVtZWJlcl90b2tlbiI6bnVsb  CwiY3JlYXRlZEF0IjoiMjAyMC0xMC0wMVQwMToy  MjowMS4wMDBaIiwidXBkYXRlZEF0IjoiMjAyMC0  xMC0wMVQwMToyMjowMS4wMDBaIiwiaWF0IjoxNj  A1MTM3NzM4LCJleHAiOjE2MDU3NDI1Mzh9.Dz  EOm-PgRM-CMsomkktJL8MkA6nAEQFs5shJ7uYI72I  **GET RESPONCE EXAMPLE**  <http://localhost:8081/api/allcategory>  Fetching all categories and the jobs belongs to that categories **(Object Relational Mapping - Sequelize)**  **BODY PARAMS**  **EXAMPLE**  {      "Category" { "id": 3,          "name": "Electronics",          "createdAt": "2020-09-23T05:41:01.000Z",          "updatedAt": "2020-09-23T05:41:01.000Z", }          "Jobs": [              {                  "id": 1,                  "title": "Software Engineer fULL STACK",                  "slug": "software-enginner",                  "description": "SOftware Full Stack",                  "roles": "Software Engineer",                  "position": "Software Full Stack Developer",                  "address": "96 Green DRive",                  "type": "parttime",                  "status": 1,                  "last\_date": "0000-00-00",                  "number\_of\_vacancy": 2,                  "experience": 3,                  "salary": "negotiable",                  "createdAt": "2020-09-30T00:00:00.000Z",                  "updatedAt": "2020-10-14T05:26:09.000Z",                  "CategoryId": 3,                  "CompanyId": 1,                  "UserId": 1              }          ]      },  ***Please refer the attached document (Project Guides and Screen Shots) for more API Routes.*** |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Part 6 Approval and Feedback** It is now time to present your final project to the client for feedback and approval. Organise a time to showcase and demonstrate your project to your client. | | | | | | |
| **Checklist (To be completed by the learner’s facilitator)** | | | | | **Yes** | **No** |
| 1. The learners project dynamic content functions according to the requirements outlined in the brief. | | | | |  |  |
| 1. The learner has demonstrated and tested the prototype with the client including errors handling. | | | | |  |  |
| 1. The learner has incorporated feedback from the client into the prototype | | | | |  |  |
| 1. The learner has obtained client sign-off for the prototype | | | | |  |  |
| **Assessor Name** | *Daniel Fitzsimmons* | **Assessor Signature** | *Signature* | **Date** | *Date* | |