|  |  |  |
| --- | --- | --- |
| **Qualification Details** | | |
| **Training Package Code & Title** | ICT Information and Communications Technology Training Package (Release 8.1) | |
| **Qualification National Code & Title** | | **State code:** |
| ICT50220 - Diploma of Information Technology | | **AC21** |
| **Units of Competency (UoC) detailed in this DAP | Cluster: MVC and Non-relational databases** | | |
| **Unit National code and title** | | **State Code** |
| ICTPRG554 Manage data persistence using NoSQL data stores | | OBS89 |
| ICTPRG556 Implement and use a model view controller framework | | OBS87 |

*Students to sign this document when submitting an assessment*

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Assessment description** | | | AT2 Project | | | | | | |
| **Assessment date** | | | Weeks 8-18 | | | | | | |
| **Student Name** | | | Samuel Bailey | | | | | | |
| **Student ID** | | | 30106121 | | | | | | |
| **Student Declaration** | | | I have read and understood the details of the assessment.  I have been informed of the conditions of the assessment and the appeals process.  I agree to participate in this assessment.  I certify that the attached is my work.  SBailey | | | | | | |
| **Assessors Name** | | |  | | | | | | |
| **Date Due:** | | |  | | **Date Submitted:** | | | |  |
| **STUDENT FEEDBACK** | | | | | | | | | |
| **Assessment Decision** | Attempt 1 | | | ☐ Satisfactory | | | ☐ Not Yet Satisfactory | | |
| Attempt 2 | | | ☐ Satisfactory | | | ☐ Not Yet Satisfactory | | |
| Attempt 3 | | | ☐ Satisfactory | | | ☐ Not Yet Satisfactory | | |
| **Assessor Name** |  | | | | | | | | |
| **Assessor Signature** |  | | | | | **Date:** | |  | |
| **Feedback to student** | | | | | | | | | |
| Feedback will be provided to you in class or via Blackboard | | | | | | | | | |
| **Feedback from student** | | | | | | | | | |
|  | | | | | | | | | |
| **Student signature** | |  | | | | **Date:** | |  | |

|  |  |
| --- | --- |
| **INFORMATION FOR INSTRUCTORS/ASSESSORS** | |
| Type of Assessment | AT2 |
| Duration of Assessment | 10 Weeks |
| Location of Assessment | * Classroom and home |
| Conditions | * Assessment may be completed in class or at home. All documentation must be submitted via Blackboard, including proof of individual work in places where directed. * To verify the authenticity of the student’s assessment, you may ask the student to again produce an answer to an existing question. |
| **Elements and Criteria:** | |
| Unit of Competence Elements:   * Review and select NoSQL options * Determine and create storage of data types * Build and configure indexes * Use queries and retrieve objects * Confirm interaction of objects * Create an MVC project * Use the MVC framework * Finalise MVC project   **Performance Evidence**  The candidate must demonstrate the ability to complete the tasks outlined in the elements, performance criteria, and foundation skills of this unit, including evidence of the ability to:   * create at least one model view controller (MVC) project and confirm the functionality of the framework, including:   + HTTP handers and routes for getting, POST, PUT, and DELETE requests   + HTML templates, view models, and dynamic rendering   + HTTP requests, responses, and redirects. * specify partition and sort keys * optimize the data. * create at least three different queries, including updating, deleting, and creating data types * create at least two indexes.   **Knowledge Evidence**  The candidate must be able to demonstrate knowledge to complete the tasks outlined in the elements, performance criteria, and foundation skills of this unit, including knowledge of:   * benefits and functions of NoSQL database and schema-free data persistence, as well as traditional relational data models * methods and different features and functions between scaling out and scaling up (horizontal and vertical) * language used in required programming language for NoSQL applications * partitioning in a NoSQL environment and its related terms * functions and features for time-to-live (TTL) requirements * authorization and authentications procedures and levels of responsibility according to client access requirements * distribution of data storage across partitions * debugging and testing methodologies and techniques * functions and features of sort keys in NoSQL storage * features of transport encryptions, authentication, and authorization * different NoSQL data store formats, including:   + key value   + document based   + column-based   + graph based * different NoSQL data types, including:   + numeric   + string   + boolean   + complex   + date time | |

Structure and format reports in a clear manner that conforms to organizational requirements

|  |  |
| --- | --- |
| **INFORMATION FOR STUDENTS** | |
| Purpose of Assessment | This assessment is a full project which encompasses all skills learnt until this point when assessed and promotes research on upcoming trends and technologies within the NoSql/MVC field in relation to data capture, store, analyse and validate.  To verify the authenticity of your assessment, your lecturer may ask questions to substantiate it is your work. |
| Allowable materials | Weekly Readings, Class notes, Weekly Activities |
| Required resources | Computer with:   * Computer operating system; * Internet Access; * Word processing software; * Access to Blackboard |
| Assessment Presentation and Submission | The material and links to related resources are available within the Blackboard course shell created for this unit.  All questions and activities must be attempted.  Use of research tools and peers in formulating answers is acceptable – but the work submitted must be your work.  Final documentation is to be uploaded to the appropriate area in the Blackboard course shell created for this unit.  If you are marked as NYS (Not Yet Satisfactory) on your first attempt, you will be provided with another opportunity to re-attempt the assessment at the discretion of the lecturer. |
| Reasonable adjustment | In some circumstances, adjustments to assessments may be made for you.  See the DAP for more information |
| Assessment contents | This assessment consists of:  Design requirements->Project Report  TASK 1: Plan your app  Task 2: Create a Web App that will use data from our API  Task 3: Test and fix the data persistence process according to business requirements |

*Continue to the next page*

|  |  |  |  |
| --- | --- | --- | --- |
| **Student’s Name** |  | **Date marked** |  |

**ASSESSMENT SUBMISSION CHECKLIST**

Use the checklist below to ensure you have submitted all the necessary files

|  |  |
| --- | --- |
| Design requirements Project Report | **Check** |
| Answer questions A to q through a small report |  |

|  |  |
| --- | --- |
| TASK 1: Plan your app | **Check** |
| Organize your workplace and read client functional requirements. **USING MONGO DB** |  |

|  |  |
| --- | --- |
| Task 2: Create a Web App that will use data from our API | **Check** |
| Design phase, program your API CRUD functionality |  |

|  |  |
| --- | --- |
| Task 3: Test and fix the data persistence process according to business requirements | **Check** |
| Test phase, test, and document all functionalities |  |

**Introduction:**

This assessment is a report which encompasses all knowledge learned throughout the weeks of teaching and promotes analytic skills on upcoming trends and technologies within the MVC and NoSQL.

**Scenario**

You have been hired as a new programmer by ITEMS and are tasked with a project that involves analysing a business situation, building an API, developing an MVC, and creating a NoSQL database. Additionally, you will need to provide a web interface for the database and ensure secure login to the website. Be prepared to handle large volumes of data and, if needed, implement vertical or horizontal scaling.

You must adhere to coding standards, as outlined in the **organisational documentation** found at [LINK](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions) Ensure your code is properly commented, include debugging screenshots in a separate document, and provide a test table with corresponding screenshots. Additionally, you should identify the information repositories and organisational documents you have referenced.

Provide for your team: Seed\_data\_1.txt and Seed\_data\_2.txt given to be implemented through the code**. [ICTPRG554 PE1.1]**

Each student will use different data seed.

Development tools to be used: **[ICTPRG554 AC1-3]**

1. Internet connection to download all NuGet connectivity packages
2. Visual Studio For development
3. Visual Code for testing
4. Postman for code testing

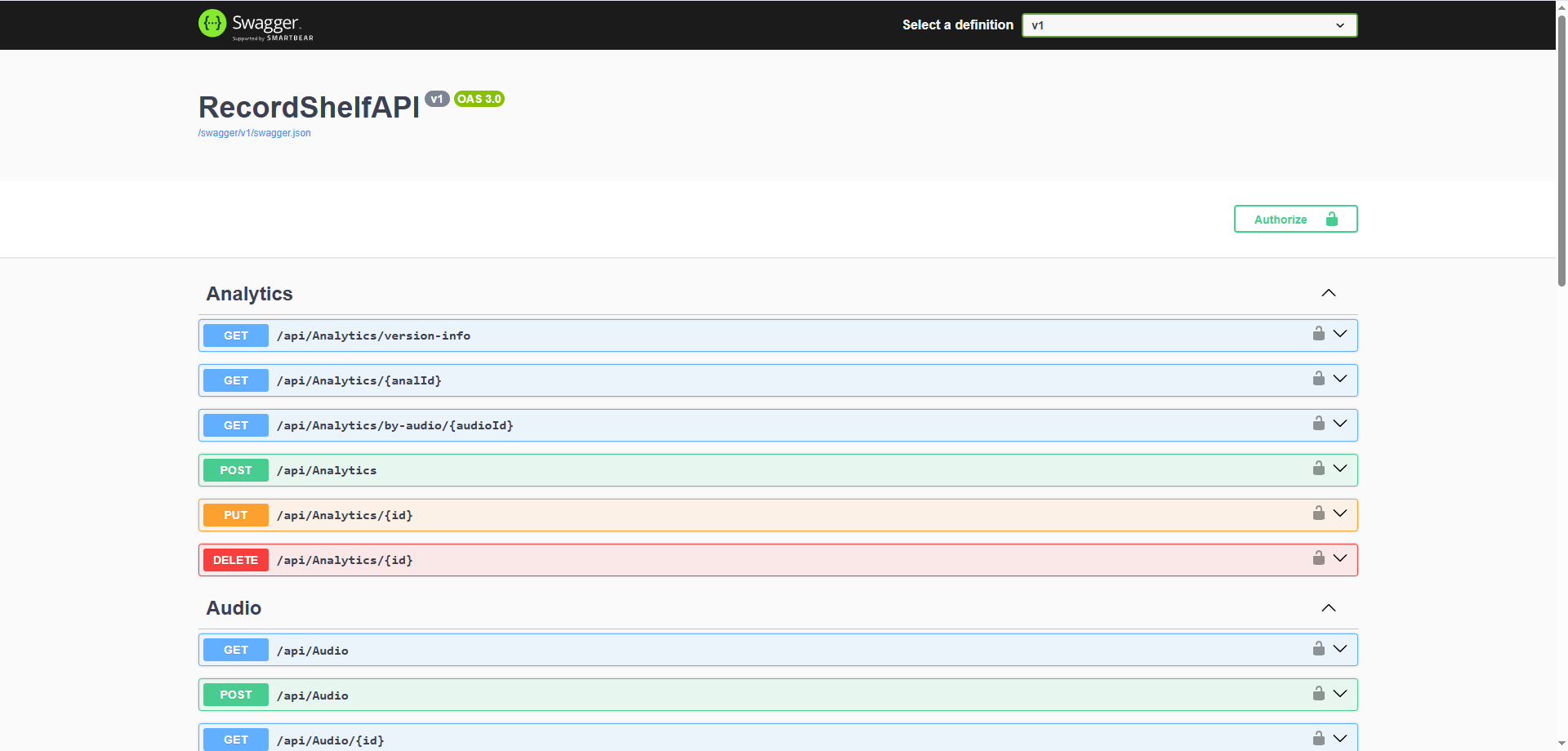
Design requirements

**Your final design at the end should look like this:**

# Project Report: RecordShelf Web API using NoSQL

## API Proof

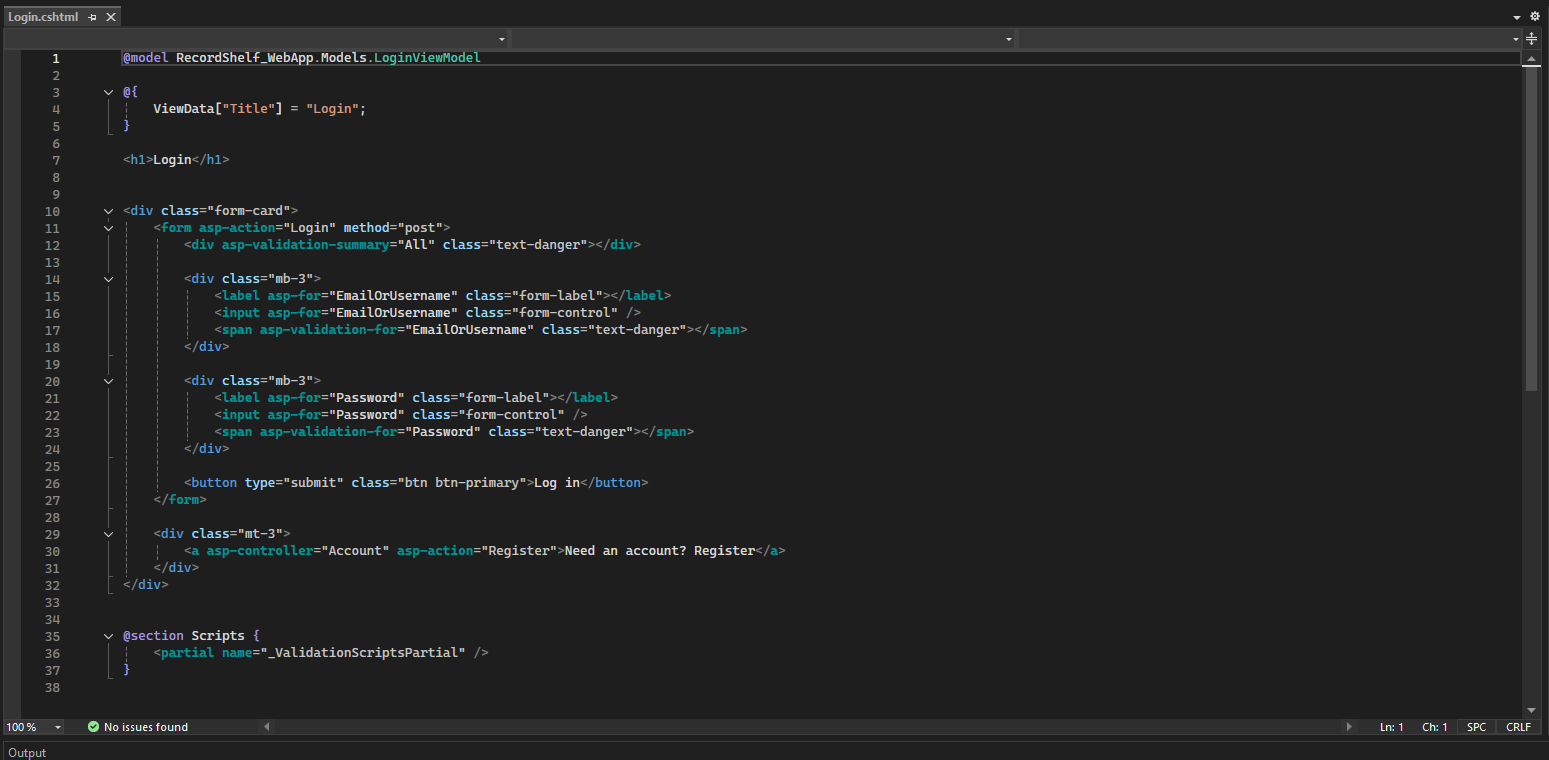
*API: Name it xxxxSales.API. The example is given for the CarSalesAPI. You will replace XXXX with the product you are selling [ICTPRG556 PE 2 ] [ICTPRG556 PE 2.1 ] {ICTPRG556 AC 2.1]*



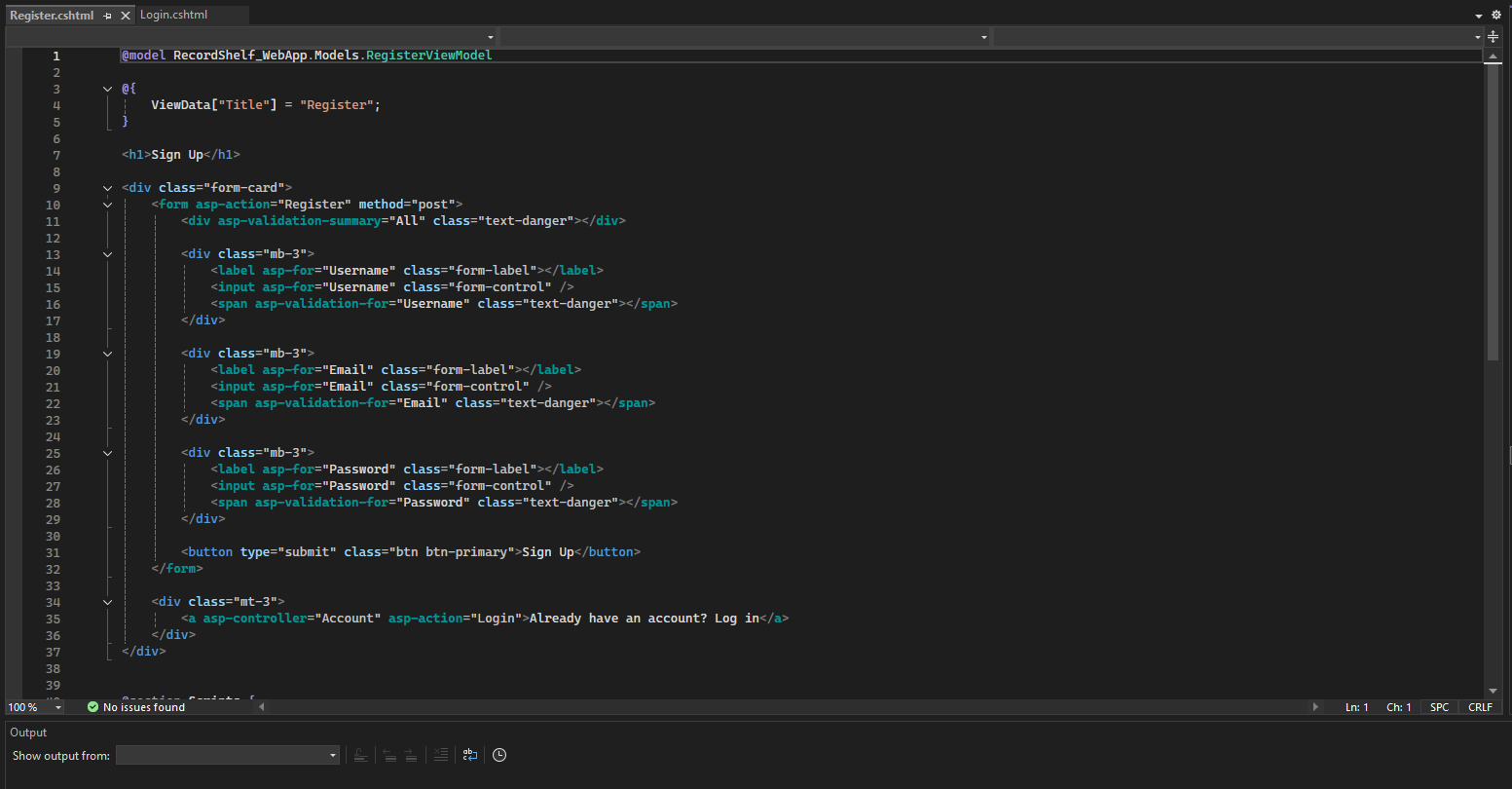
## App Proof

*Web App should have a login and register page.*

**Login**



**Register**



## Vertical and Horizontal Scaling

*Explain the implementation of vertical or horizontal scaling. [ICTPRG554 PE1.2]*

Vertical scaling means upgrading a single server by giving it more CPU, RAM, or storage so it can handle more work.  
 Horizontal scaling means adding more servers and spreading the load between them, often using load balancing and database sharding so the system can grow without relying on one machine.  
 Horizontal scaling is usually better for the client because it keeps the system running smoothly as traffic increases and avoids the limits of a single server.  
 In the RecordShelf Web API, using MongoDB on a virtual device helps simulate this horizontal approach, since MongoDB is designed for sharding and can easily be distributed across multiple servers in a real deployment.

## NoSQL Approach

*Explain why using NoSQL in this project is better than the traditional data model. [ICTPRG554 PE1.3]*

NoSQL is a better fit for this project because it uses a flexible schema, which works well when audio uploads, user data, and analytics can all have slightly different structures without breaking the database. It also handles large amounts of data easily, which is important for a SoundCloud-style site where users constantly upload new audio files and metadata. NoSQL databases scale horizontally, making it simple to grow the system by adding more servers as the user base increases. Lastly, MongoDB is commonly used for product catalogues and media libraries, which makes it a strong match for managing audio records, tags, and user-generated content in this project.

## Proposed Vendors

*Propose one vendor for our client to use, and why this one should benefit their business. [ICTPRG554 PE1.4]*

A good vendor for the client to use is MongoDB Atlas. It provides fully managed cloud hosting, which means the database is maintained, updated and secured without the client needing to handle the infrastructure. MongoDB Atlas also offers automatic scaling, making it easy for the system to grow as more users upload and listen to audio. Lastly, it integrates smoothly with web APIs, since it works natively with JSON, making development faster and simpler for a project like this.

## Data Table

*Create one data table with all data types you will use in the project.*

|  |  |  |
| --- | --- | --- |
| **Field** | **Object Type** | **Description** |
| AudioID | string | Unique ID for the audio record, generated by the database. |
| AudioTitle | string | The title or name of the audio track. |
| UserID | string | The artist name displayed for the audio. |
| Artist | string | The artist name displayed for the audio. (User – Username) |
| Tags | List<string> | A list of tags used for categorising or searching audio. |
| DurationSeconds | int | The total length of the audio file in seconds. |
| FilePath | string | The stored location of the audio file within the root files. |
| UploadDate | DateTime | The date and time the audio was uploaded. |

TASK 1: Plan your app

1. Build an app where users can purchase products. This app should have one running API which will enable users to search for products using different criteria (most important your Web/app and Web/API are written in MVC framework). **MONGO NOSQL MUST BE USED to**:
   1. Get all products.

|  |  |
| --- | --- |
|  | **[ICTPRG554 PE1.1] [ICTPRG556 PE2.1]** **[ICTPRG554 AC 2.1]** |

* 1. Return a list of items

|  |
| --- |
|  |
| **[ICTPRG554 PE1.1] [ICTPRG556 PE 2.1 ]** |

* 1. Returning multiple items

|  |
| --- |
| See Above |
| [**ICTPRG554** PE1.1] [**ICTPRG556** pe2.3] |

* 1. Error handling

|  |
| --- |
|  |

* 1. Making all calls to your API Async

|  |
| --- |
|  |
| **[ICTPRG554 PE1.2]** |

* 1. Write data using HTTP methods.

|  |
| --- |
|  |
| **[ICTPRG554 PE1.1]** [**ICTPRG556** PE2.1] |

* 1. Add data using the POST method.

|  |
| --- |
| Same as above |
| **[ICTPRG554 PE1.1] [ICTPRG556 PE2.1]** |

* 1. Update one record using the PUT method.

|  |
| --- |
|  |
| **[ICTPRG554 PE1.1] [ICTPRG554 PE1.2] [ICTPRG556 PE2.1]** |

* 1. Delete one or many records using DELETE.

|  |
| --- |
|  |
| **[ICTPRG554 PE1.1] [ICTPRG554 PE1.2] [ICTPRG556 PE2.1]** |

1. For this application you will receive DUMMY data from your lecturer (many versions/one version per student)
2. This application should be designed to Search and retrieve complex data.

|  |
| --- |
|  |
| **[ICTPRG554 PE1.1] [ICTPRG554 PE1.2] [ICTPRG556 PE2.1]** |

1. Filtering data statements should be used like %oyo% where you will find all the results which contain those letters.

|  |
| --- |
|  |
| **[ICTPRG554 PE1.1] [ICTPRG554 PE1.2]** |

1. Searching the web with null values or by some criteria

|  |
| --- |
|  |
| **[ICTPRG554 PE1.1] [ICTPRG554 PE1.2] [ICTPRG556 PE2.1]** |

1. Sorting data is important, and you can choose how to sort your data

|  |
| --- |
|  |
| **[ICTPRG554 PE1.1] ICTPRG554 [PE1.2]** |

1. Versioning your API to run only available products.

|  |
| --- |
|  |
| [**ICTPRG554 PE1.3] [ICTPRG554 PE1.4] [ICTPRG556 PE2.2]** |

1. Decide how your HTTP Header will be called and implement that (one of those below):
   1. APIVersionReader or

|  |
| --- |
|  |
| **[ICTPRG554 PE1.3] [ICTPRG554 PE1.4] [ICTPRG554 PE2.2]** |

* 1. Using Query String

|  |  |
| --- | --- |
| **NA** | **[ICTPRG554 PE1.3] [PE1.4] [PE 2.2]** |

1. Securing your API is crucial, implement your solution.

|  |
| --- |
|  |

1. If the data storage you have selected a volatile one, what is the next step, this business need data consistency too?

MongoDB is technically a volatile, schema-less data store unless configured for persistence. To ensure data consistency, we enable features such as replica sets, and automated backups so the data is never lost, even if a node fails.

|  |
| --- |
|  |

Task 2: Create a Web App that will use data from our API

1. Our Web App should be able to read all the data from our API and present all products on the landing page: (make sure all your controllers are in the controllers section, models in the model section and views in the view section:

|  |
| --- |
|  |
| [**ICTPRG556** PE2.3] **[ICTPRG554 PE1.1] [ICTPRG554 PE1.2]** |

2.After enabling CORS and migrating your database adds identity to your application. Registration and login must be fully functional.

|  |
| --- |
| Register  Login    Success |
| **[ICTPRG554 AC1.1] [PE2.3] [ICTPRG554 PE1.1] [ICTPRG554 PE1.2] [ICTPRG554 PE1.3] [ICTPRG554 PE1.4]** |

Task 3: Test and fix the data persistence process according to business requirements

NOTE: YOUR TEST DATA WILL BE DIFFERENT

1. After completing all the above we need to document and finalize testing:
   1. Use POSTMAN and Visual Code to test your API

|  |
| --- |
|  |
| **[ICTPRG554 PE1.1] [ICTPRG554 PE1.2] ICTPRG554 [PE1.3] [ICTPRG554 PE1.4] [ICTPRG554 AC 1.1]** |

* 1. Open the web console and find any error which happened during your design:

|  |
| --- |
|  |
| **[PE1.1] [PE1.2] [PE1.3] [PE1.4]** [PE2.3 ] |

1. Provide testing evidence that you have received a payload from the API in a JSON format:

|  |
| --- |
|  |
| **[ICTPRG554 PE1.1]** |

1. In existing data try to search for one record which does not exist. Your code shouldn’t be 204 ==No Content

|  |
| --- |
|  |
| **[PE1.1] [PE1.2]** |

1. Show a screenshot of a get request to receive all products:

|  |
| --- |
|  |
| **[ICTPRG556 PE2.1] [ICTPRG554 PE1.1] [ICTPRG554 PE1.2] [ICTPRG554 PE1.3] [ICTPRG554 PE1.4]** |

1. Show a screenshot of a post request to create a new product:

|  |
| --- |
|  |
| **[ICTPRG556 PE2.1] [ICTPRG554 PE1.1] [ICTPRG554 PE1.2] [ICTPRG554 PE1.3] [ICTPRG554 PE1.4]** |

1. Show a screenshot of a put request to update a new product:

|  |
| --- |
|  |
| **[ICTPRG556 PE2.1] [ICTPRG554 PE1.1] [ICTPRG554 PE1.2] [ICTPRG554 PE1.3] [ICTPRG554 PE1.4]** |

1. Show 2 screenshot of filtering data through web parameters:

|  |
| --- |
|  |
|  |
| **[ICTPRG556 PE 2.1 ] [ICTPRG554 PE1.1] [ICTPRG554 PE1.2] [ICTPRG554 PE1.3] [ICTPRG554 PE1.4]** |

1. Show a get request from version 1 and version 2 of your API.

|  |
| --- |
|  |
|  |
| **[ICTPRG554 PE1.3] [ICTPRG554 PE1.4]** |

1. Show a screenshot of a get request from the version 2 API using a header.

|  |
| --- |
|  |
| **[ICTPRG556 PE2.1][ ICTPRG556 PE2.2] ICTPRG556 PE2.3] [ICTPRG554 PE1.1] [ICTPRG554 PE1.2] [ICTPRG554 PE1.3] [ICTPRG554 PE1.4]** |

1. Show a screenshot of get request using query strings or a action result.

|  |
| --- |
|  |
| **[ICTPRG554 PE1.1] [ICTPRG554 PE1.2] [ICTPRG554 PE1.3] [ICTPRG554 PE1.4]** |

# There are a number of repeat screenshots due to repeated requests.