COMP 2001

Contents

Introduction

This document will discuss and explore the process of carrying out both Task 1 and Task 2 of the COMP2001 70% assignment. The first part of this coursework sees through the production of a Microsoft SQL database and an accompanying ASP.NET RESTful API. Part two of the coursework is connected to the development of a Linked Data Application (LDA), for which another very simple, unrelated ASP.NET API had been created.

For both parts of the coursework, the outcomes can be found within the reference table at the end of this section. The dataset used within the Linked Data Application can also be found within the same table.

The report starts by introducing the reader to the LDA within the Background section – this section of the report aims to convey the choice of dataset and contains the vision of the application. Following this will be a section that explores the different issues that the project will consider – and potentially tackle, such as the Legal, Social, and Ethical factors of developing and deploying software. The report will then elaborate upon the Design choices for both Tasks, as well as their final implementations. The final section of the report will be the Evaluation – here the lessons learned throughout the Assignment will be discussed, and any test results will be highlighted.

GitHub Repository: Task 1	https://github.com/Plymouth-University/comp2001_assignment-ORG4N/tree/main/Task%201
GitHub Repository: Task 2	https://github.com/Plymouth-University/comp2001_assignment-ORG4N/tree/main/Task%201
Dataset	https://plymouth.thedata.place/dataset/active-library-users-by-age

Figure 1 – Resources

Background

This section of the report discusses and justifies the chosen dataset that was used within Task 2 – Linked Data Application. The original link to this dataset can be found within the Introduction section (Figure 1 – Resources).

The chosen dataset presents a collection of libraries within Plymouth and describes the library users in terms of age range. Figure 2 shows the fields within the dataset and the first 5 records. For example, the count of all book borrowers/renewers between the ages of 12 and 17 is listed for each of the named libraries. For a user to be accepted as part of the count it is stated that they must have interacted with the library within the past year, which includes borrowing/renewing a book, or using a computer (Back, 2018).

The dataset was collected by Plymouth City Council in 2018. This dataset potentially has usage in being displayed in graph format to show the differences in age between library users. The data could show whether Plymouth's libraries are generally accessed by specific age ranges or if the count of each age range is widely spread. Libraries could use this research to gain more information on the target audience they should be marketing towards. If the data indicates that if there are younger, instead of old, users of the library then that specific library might want to target events towards that highlighted demographic.

Without showing this data in graph format it is much harder to visualise and gain meaning from. The LDA resulting from this project, however, does not visualise the data in graph format and this would be the next step to further improving the application and its comprehensibility.

_id	Library	Age0-4	Age5-11	Age12-17	Age18-59	Age60-100
1	Central	621	1990	772	11616	3800
2	Crownhill	227	659	192	1335	708
3	Devonport	181	407	136	980	198
4	Efford	109	264	91	466	178
5	Estover	50	196	66	188	103

Figure 2 – Dataset table (Back, 2018)

The overall vision of this project was to take the data, which is stored in csv format, and display it on a webpage. However, a more personal goal that the developer of the LDA set for themselves was to refine their skillset and display knowledge of how to create the frontend, API and services needed to make the application.

Legal, Social, Ethical, and Professional Issues

General Considerations

The considerations discussed within this section can be applied to both Task 1 and 2 and are therefore general.

1. Deadlines

Time management is a serious Professional concern for any project – and this project was not an exception. Although not to do with the app's interact-ability/accessibility and is solely a concern of the developer, this consideration was important for both Tasks as the deliverables needed to be made and tested.

2. Data Integrity

Both apps would need to handle Data Integrity in similar ways. In the API, data integrity is handled by ensuring that the fields within each table have a data type that describes the data as accurately as possible – such as a project's year being stored as a 4-char integer.

Data consistency could have been fully integrated into the API if stored procedures had harsh input constraints – however, this had not been fully implemented into the final deliverable. Using the inbuilt primary key indexer however allowed for some consistency as primary keys for each entity are generated automatically and are always unique.

The Audit table that had been implemented further provides some integrity as it stores the old data, and the new change being applied to a programme when it is updated.

Finally, the Stored Procedures for the programme table ensure data integrity by including conditional statements that check whether the data being created, deleted, or updated already exists. Therefore, duplicate records can't exist.

The LDA ensures data integrity by making sure that the data being displayed on the website cannot be tampered with by any external parties/users. As the csv file is stored locally on the server it is impossible for end-users to manipulate. The dataset csv also had no personal information and therefore need not be tampered with for anonymity.

The Data Protection Act's principles were considered as a guideline when considering the above issues of Data Integrity within both Tasks (Data protection, n.d.).

LDA Considerations

1. Readability / Accessibility

The LDA was tested for readability and accessibility by showing the wireframes and subsequently delivered prototype to a few users. The aim of this was to grasp that they understood what the data was showing and how to find it on the webapp. More about this will be discussed within the Design section of this report.

As previously stated in the Background section, the project could be further improved by visualizing data in graphs so that it is more readable. However, the Index page as is, is coherent and clear as it uses clear colors and buttons to differentiate different elements on the page. Similarly, a navbar at the top of each page allows users to navigate between different pages, and the title of the tab updates to reflect the current page that they are on.

Another future improvement that could be made to the project is to embed the HTML with ARIA markup so that users with assistive technologies can access the website with greater ease as ARIA describes HTML elements in greater detail to screen readers etc. (ARIA - Accessibility | MDN, n.d.).

2. Copyright / Referencing

As the dataset was not personally collected by the Author and developer of the LDA, the original link has been provided both on this report and on the GitHub repository readme. Likewise, Bootstrap (a third-party application) was used to create the layout of the application and is also referenced. As the project is not being used to develop monetary value in any way, these considerations may have not been necessary, but by ensuring that the developer

kept track of all resources during development, they are preparing for a professional environment, where Copyright would be a major legal issue (How copyright protects your work, n.d.).

API Considerations

1. **Privacy and Security**

The Data Protection Act (Data protection, n.d.) requires that an individual's information be held in a way that is secure. Currently, the Database does not store any confidential information (such as passwords or health conditions), but in the future the database's scope could be increased to need these fields. Therefore, in its current state there is no functionality to stop anyone from accessing and manipulating the database via the API. However, as described in the provided COMP2001.yaml file, users could be stopped from accessing the database unless they are logged in and authorized to access that data. In the context of the Task, this could be understood from a teacher using their staff login to remove a student from their programme.

If any sensitive data such as passwords were to be stored within the database, they would be encrypted using salt and hash encryption methods – NOT as plain text. However, information like medical conditions or home addresses would not be censored and instead, the database would be structured so that only users who NEED to see that information could, such as medical staff.

LSEP Conclusions

Overall, many issues were noted before and during the development of the application – yet it was not completely possible to solve all of them. Issues that were easier to combat have been focused on, whilst certain issues require complex and rigid systems to be put in place to ensure that LSEP standards are adhered to.

Design

This section of the report presents and describes each artifact that has been created to outline the system being developed. Artifacts for both Task 1 and Task 2 will be shown.

Task 1 Design Artifacts

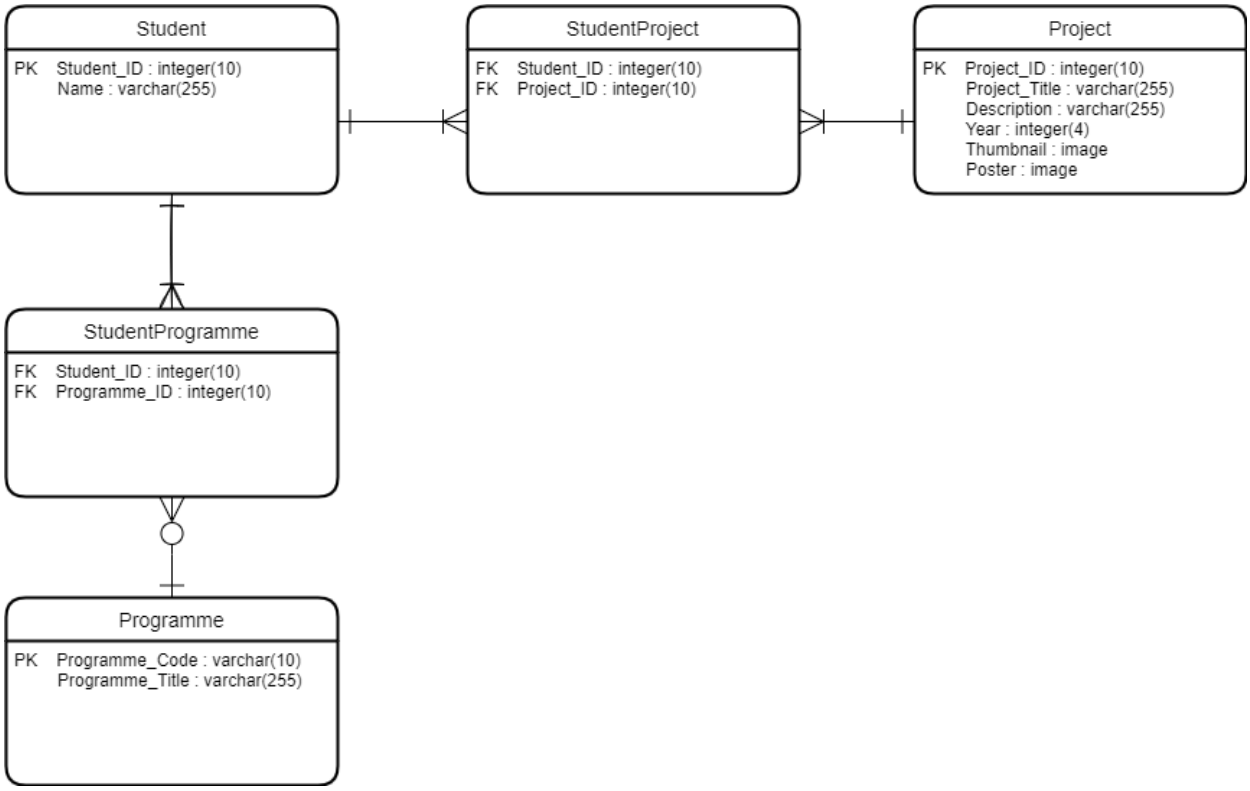


Figure 3 – Advanced Entity-Relationship Diagram

Improving upon the simple ERD that was provided within the Assessment brief, the above ERD has been developed to further describe how the database should be structured. The differences between this model and the 'conceptual' one is:

- Resolved many to many relationships with intersect entities.
- Entity fields (with datatypes) to describe properties.
- Connector notations to describe assumptions.

Firstly, two additional tables have been designed: StudentProgramme, and StudentProject. By looking at the relationships between Student and Programme with StudentProgramme, it can be interpreted that a Programme can have many Students, and a Student can be on many Programmes. Student and Project can be described in the same manner in relation to StudentProject. Resolving these many to many relationships by integrating interest entities is good practice (IBM Docs, n.d.) as it ensures each key is unique.

Secondly, the entity fields and their datatypes are self-explanatory, but it's important to note that each table has its own unique identifier.

Finally, from looking at the notation of each relationship, the following assumptions can be made:

- A student MUST have a program
- A program DOESN'T need any students
- A student MUST have a project
- A project MUST have a student

Task 2 Design Artifacts

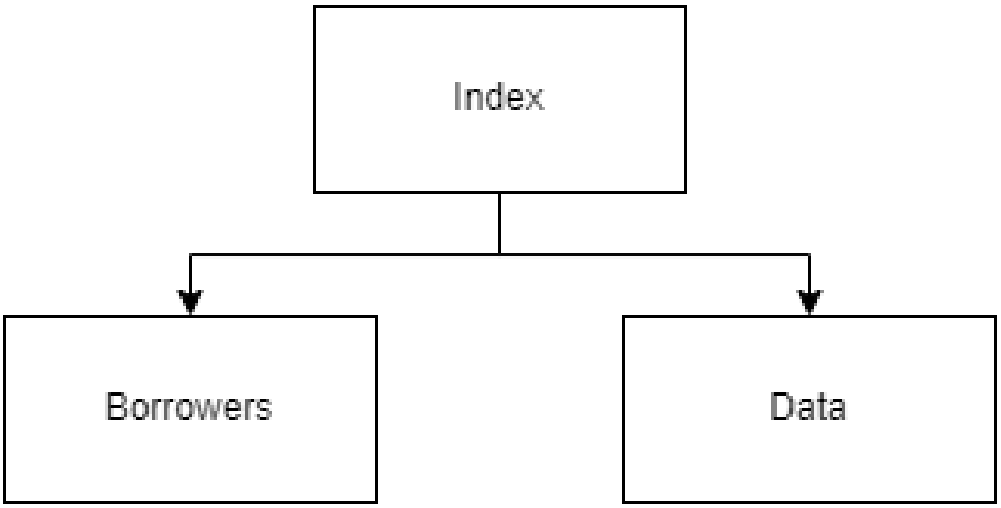


Figure 4 – Sitemap

The above model shows the sitemap for the LDA. In total, the LDA was designed to have three webpages. The aim of each page is described below:

- Index – explain the purpose of the application
- Data – display the dataset
- Borrowers – display the dataset in JSON-LD format

The user should be able to access the latter two pages via the index page. The below wireframes visualize the layout and enable an understanding of how the user would navigate the webapp.

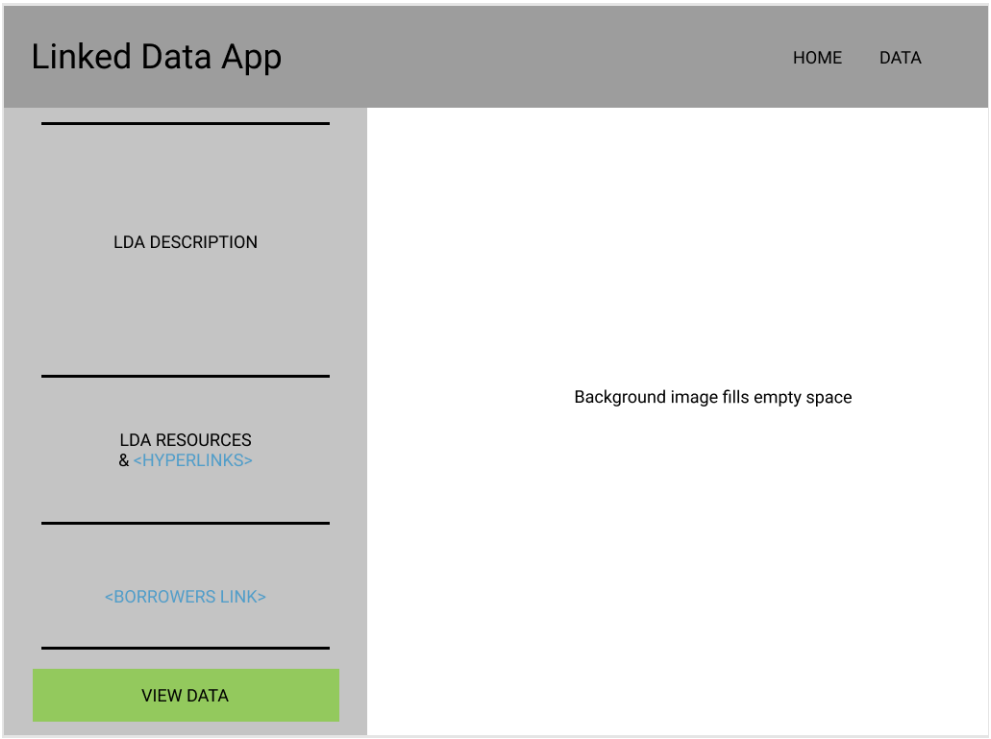


Figure 5 – Index page

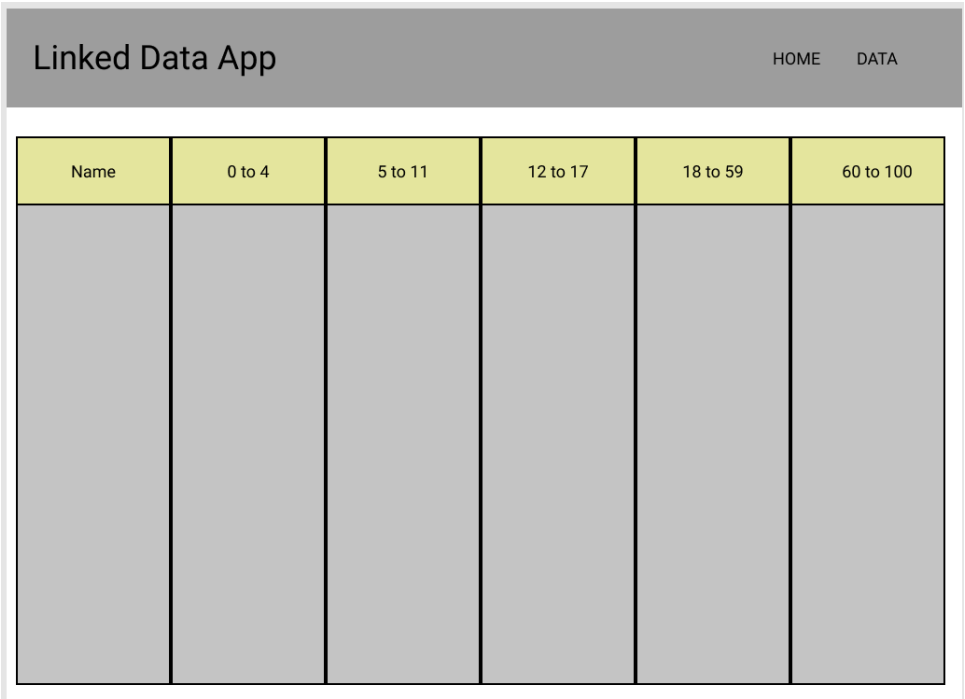


Figure 6 – Data page

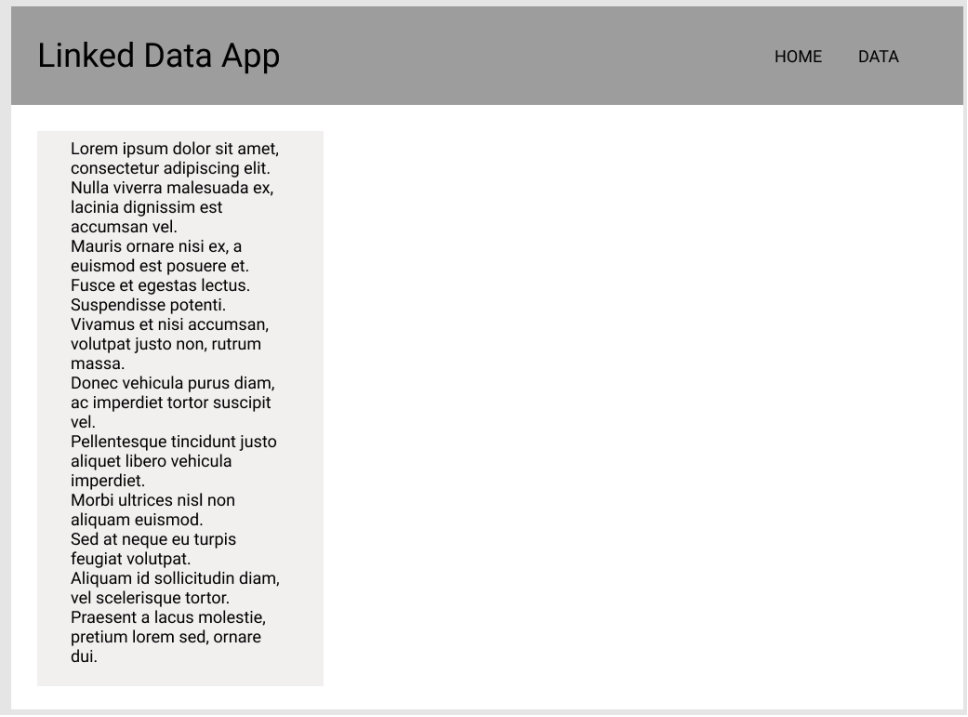


Figure 7 – Borrower (JSON-LD) page

A reoccurring element on each wireframe page is the navbar. In the wireframe prototypes it was decided that a navbar was necessary to help users navigate between the pages. As the navbar does not change between the three pages, it was noted that a reusable component could be made and thus less code is repeated in the app.

The Index page was designed to elaborate on the purpose of the project and to direct the user to the dataset source, as well as highlight a button to take them to the Data page.

The Data page was designed to be filled with a vertically scrollable table that contains all the dataset data. The table headings are differentiated from the table body via color.

Finally, the JSON-LD markup page was designed to be left as simply as possible to not inhibit on the readability of the raw text.

Where appropriate, color and different text sizes have been used to establish a difference between headings, body text, hyperlinks, buttons.

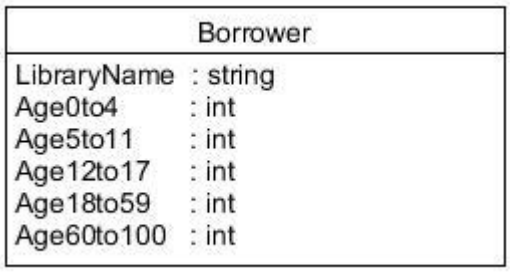


Figure 8 – Class diagram/data model

The data model above shows each field that will need to be displayed in the table, and its datatype.

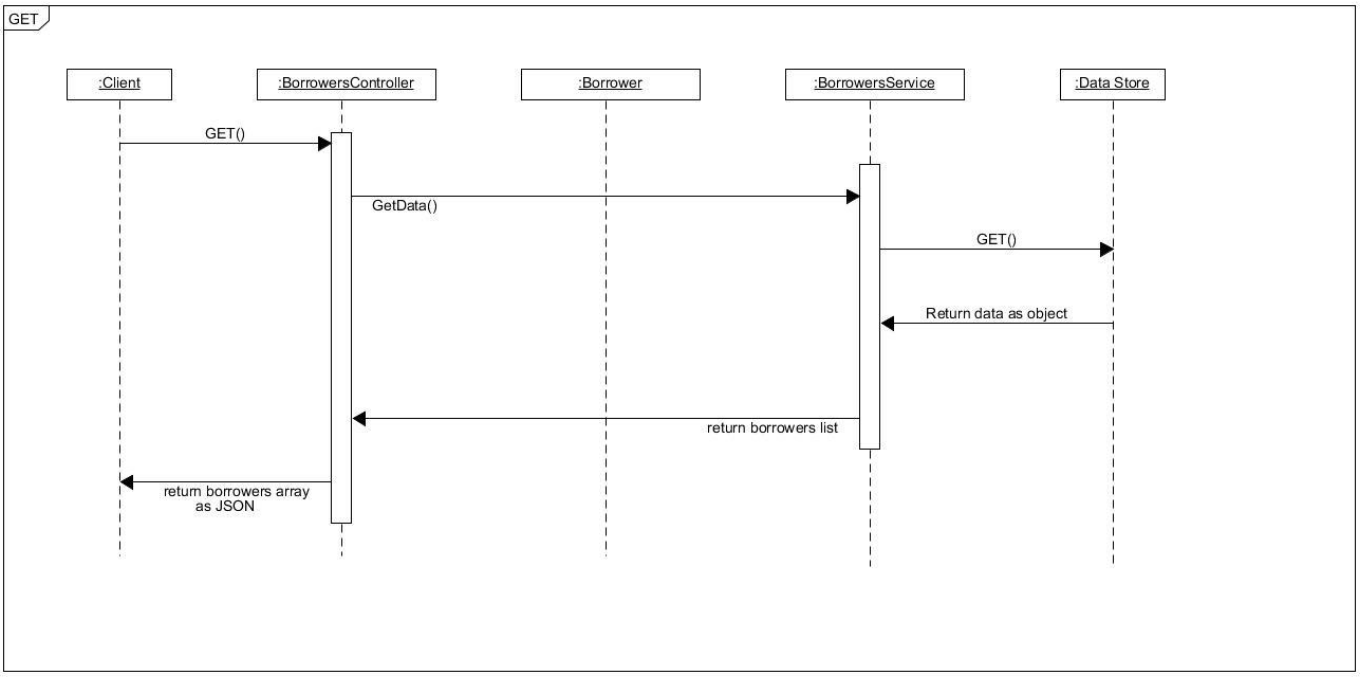


Figure 9 – Sequence diagram for HTTP GET request

The sequence diagram is showing the following:

1. Client requests to GET the data when on Data page
2. Controller receives get and performs the GetData() function
3. Service reads data from CSV and returns a List to the controller
4. Controller converts List to an array and passes the array back to the Client
    a. At this point, the data will be written to a table using client-side Javascript

Implementation

API

The tables for the database that the API would access were made exactly as designed within Figure 3. Within the Programme entity, according to the assignment specification, three stored procedures were made: Create, Update, and Delete.

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [CW2].[Create_Programme](
    @Programme_Code as VARCHAR(10),
    @Title as VARCHAR(255),
    @ResponseMessage NVARCHAR(250) OUTPUT
)
AS
BEGIN

    IF NOT EXISTS(SELECT * FROM CW2.Programme WHERE ProgrammeCode = @Programme_Code)
    BEGIN
        INSERT INTO CW2.Programme (ProgrammeCode, ProgrammeTitle)
        VALUES (@Programme_Code, @Title)
            SET @ResponseMessage = 'Programme created:' + @Programme_Code
    END

    ELSE
        SET @ResponseMessage = 'Programme already exists:' + @Programme_Code


    PRINT @ResponseMessage

END;
GO
```

```
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [CW2].[Delete_Programme](
    @Programme_Code as VARCHAR(10)
)
AS
BEGIN
    IF EXISTS (SELECT * FROM CW2.Programme WHERE ProgrammeCode = @Programme_Code)
    BEGIN
        DELETE FROM CW2.Programme
        WHERE ProgrammeCode = @Programme_Code

        DELETE FROM CW2.StudentProgramme
        WHERE ProgrammeCode = @Programme_Code
    END
END
GO
```

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [CW2].[Update_Programme](
    @Programme_Code as VARCHAR(10),
    @Title as VARCHAR(255)
)
AS
BEGIN
    IF EXISTS (SELECT * FROM CW2.Programme WHERE ProgrammeCode = @Programme_Code)
    BEGIN
        UPDATE CW2.Programme
        SET ProgrammeTitle = @Title
        WHERE ProgrammeCode = @Programme_Code
    END
END
GO
```

Figure 10 – Stored Procedures

Each procedure performs a query to check whether the programme does or doesn't exist, and then performs the desired outcome that the name of the procedure states.

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TRIGGER [CW2].[AuditChanges] ON [CW2].[Programme] AFTER UPDATE
AS
    INSERT INTO CW2.Audit(ProgrammeCode, OldTitle, NewTitle)
    SELECT i.ProgrammeCode, d.ProgrammeTitle, i.ProgrammeTitle
    FROM Inserted i
    INNER JOIN Deleted d ON i.ProgrammeCode = d.ProgrammeCode
GO
ALTER TABLE [CW2].[Programme] ENABLE TRIGGER [AuditChanges]
GO
```

Figure 11 – Trigger on programme update

The above trigger states that when the Update_Programme stored procedure is run, the old and new data should be stored to the Audit table.

(https://github.com/Plymouth-University/comp2001_assignment-ORG4N/blob/main/Task%201/Controllers/ProgrammesController.cs)

Within the specification, an API that could call these stored procedures, and manipulate the Programmes entity, was desired. The following HTTP requests were configured within the ProgrammesController:

- GET
- PUT - uses the DataAccess model to Update
- POST - uses the DataAccess model to Create
- DELETE - uses the DataAccess model to Delete

```
// GET: api/Programmes
[HttpGet]
public async Task<ActionResult<IEnumerable<Programme>>> GetProgramme()
{
    return await _context.Programme.ToListAsync();
}

// GET: api/Programmes/5
[HttpGet("{id}")]
public async Task<ActionResult<Programme>> GetProgramme(string id)
{
    var programme = await _context.Programme.FindAsync(id);

    if (programme == null)
    {
        return NotFound();
    }

    return programme;
}
```

```
[HttpPost]
public IActionResult Post([FromBody] Programme prog)
{
    string responseMessage = "";
    try
    {
        _context.Create(prog, out responseMessage);
    }
    catch (Exception e)
    {
        responseMessage = e.ToString();
        return Ok(new string[] { "Error", responseMessage });
    }
    return StatusCode(201);
}
```

```
[HttpPut("{id}")]
public IActionResult PutProgramme(string id, [FromBody] Programme programme)
{
    string responseMessage = "";

    try
    {
        programme.ProgrammeCode = id;
        _context.Update(programme);
    }
    catch (Exception e)
    {
        responseMessage = e.ToString();
        return Ok(new string[] { "Error", responseMessage });
    }

    return NoContent();
}
```

```
// DELETE: api/Programmes/5
[HttpDelete("{id}")]
public IActionResult DeleteProgramme(string id)
{
    _context.Delete(id);
    return NoContent();
}
```

Figure 12 – ProgrammesController

https://github.com/Plymouth-University/comp2001_assignment-ORG4N/blob/main/Task%201/Models/DataAccess.cs

Within the DataAccess model, each of the related stored procedure's parameters were declared.

The LDA was implemented using an ASP.NET backend that comprised of 3 components:

- Borrower (model)
- BorrowersController
- BorrowersService

The service provides a file reading method that stores each record of the dataset into an object, and then stores each object into a List which is returned to the Controller. The Controller then converts this List to an array, and it is fetched by the Client via Javascript code. This array is then extrapolated into a table. Screenshots of the code are below:

https://github.com/Plymouth-University/comp2001_assignment-ORG4N/blob/main/Task%202/Controllers/BorrowersController.cs

```
[Route("api/[controller]")]
[ApiController]
public class BorrowersController : ControllerBase
{
    [HttpGet]
    public IEnumerable<Borrower> Get()
    {
        return BorrowersService.GetData().ToArray();
    }
}
```

Figure 13 – BorrowerController

```
public static void Init()
{
    BorrowersService service = new BorrowersService();

    List<string> lines = new List<string>();

    try
    {
        using (FileStream fs = File.Open("wwwroot/data/library-users.csv", FileMode.Open, FileAccess.Read))
        {
            using (StreamReader sr = new StreamReader(fs))
            {
                while (!sr.EndOfStream)
                {
                    lines.Add(sr.ReadLine());
                }
            }
        }
    }

    catch (IOException) { }

    // Create each object by reading each field from each line in from the users.csv file
    foreach (string field in lines)
    {
        string[] split = field.Split(',');

        Borrower borrower = new Borrower();

        borrower.Library = split[0];
        borrower.Age0to4 = split[1];
        borrower.Age5to11 = split[2];
        borrower.Age12to17 = split[3];
        borrower.Age18to59 = split[4];
        borrower.Age60to100 = split[5].Replace("/r", "");


        data.Add(borrower);
    }
}
```

```
public static List<Borrower> GetData()
{
    return data;
}
```

Figure 13 – BorrowersServices

https://github.com/Plymouth-University/comp2001_assignment-ORG4N/blob/main/Task%202/Services/BorrowersService.cs

This Init() method is called once when the server is run and it is used to read the csv file contents.

```
<script>
    fetchBorrowers();

    async function fetchBorrowers() {
    const url = "http://localhost:24474/api/borrowers";
    const raw = await fetch(url);
    const data = await raw.json();

    var count = 0;

    data.forEach(({ library, age0to4, age5to11, age12to17, age18to59, age60to100 }) => {
        if (count == 0) {
            $("#businesses").find('thead').append(`<tr>
                    <th>${library}</th>
                    <th>${age0to4}</th>
                    <th>${age5to11}</th>
                    <th>${age12to17}</th>
                    <th>${age18to59}</th>
                    <th>${age60to100}</th>
                    </tr>`);
            count++;
        }
        else {
            $("#businesses").find('tbody').append(`<tr>
                    <td>${library}</td>
                    <td>${age0to4}</td>
                    <td>${age5to11}</td>
                    <td>${age12to17}</td>
                    <td>${age18to59}</td>
                    <td>${age60to100}</td>
                    </tr>`);
            }
        });
    }
</script>
```

https://github.com/Plymouth-University/comp2001_assignment-ORG4N/blob/main/Task%202/Pages/Data.cshtml

```
<script>

    fetchBorrowers();


    async function fetchBorrowers() {
        const url = "http://localhost:24474/api/borrowers";
        const raw = await fetch(url);

        const data = await raw.json();

        document.getElementById("json").textContent = '{ "@@context" : { "Place" : "http://schema.org", "borrower" : "http://web.socem.plymouth.ac.uk" }, "Place" : [ ';

        data.slice(1).forEach(({ library, age0to4, age5to11, age12to17, age18to59, age60to100 }) => {

            document.getElementById("json2").textContent += `{
                "name" : "${library}",
                "borrower:AgeRange" : {
                    "0-4" : ${age0to4},
                    "5-11" : ${age5to11},
                    "12-17" : ${age12to17},
                    "18-59" : ${age18to59},
                    "60-100" : ${age60to100}
                }
            },`

        });

        var str = document.getElementById("json2").textContent.slice(0, -1);
        document.getElementById("json2").textContent = str;
        document.getElementById("json2").textContent += "]}";

    }
</script>
```

https://github.com/Plymouth-University/comp2001_assignment-ORG4N/blob/main/Task%202/Pages/Borrowers.cshtml

The above Javascript assembles the JSON-LD format of the dataset, which provides the following RDF graph when input into the JSON playground 3rd party app: https://json-ld.org/playground/

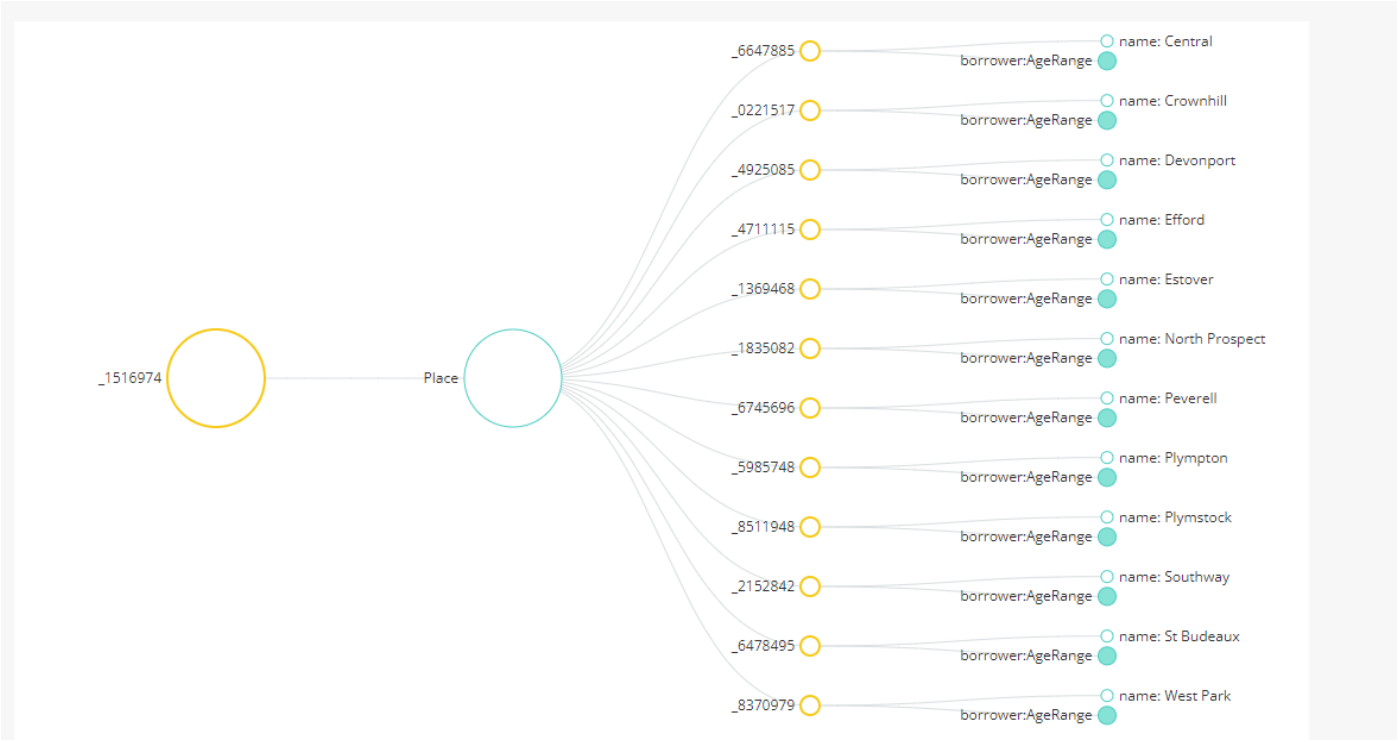
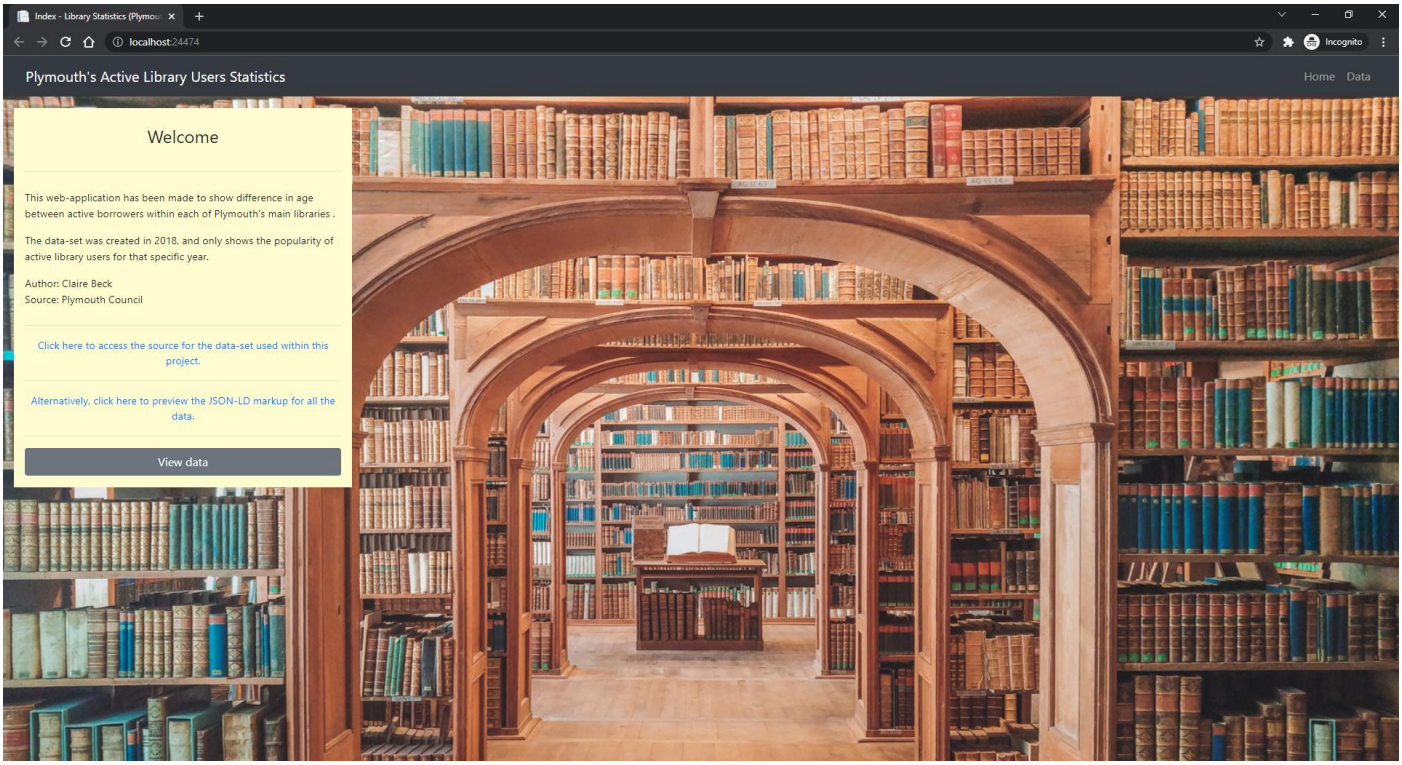
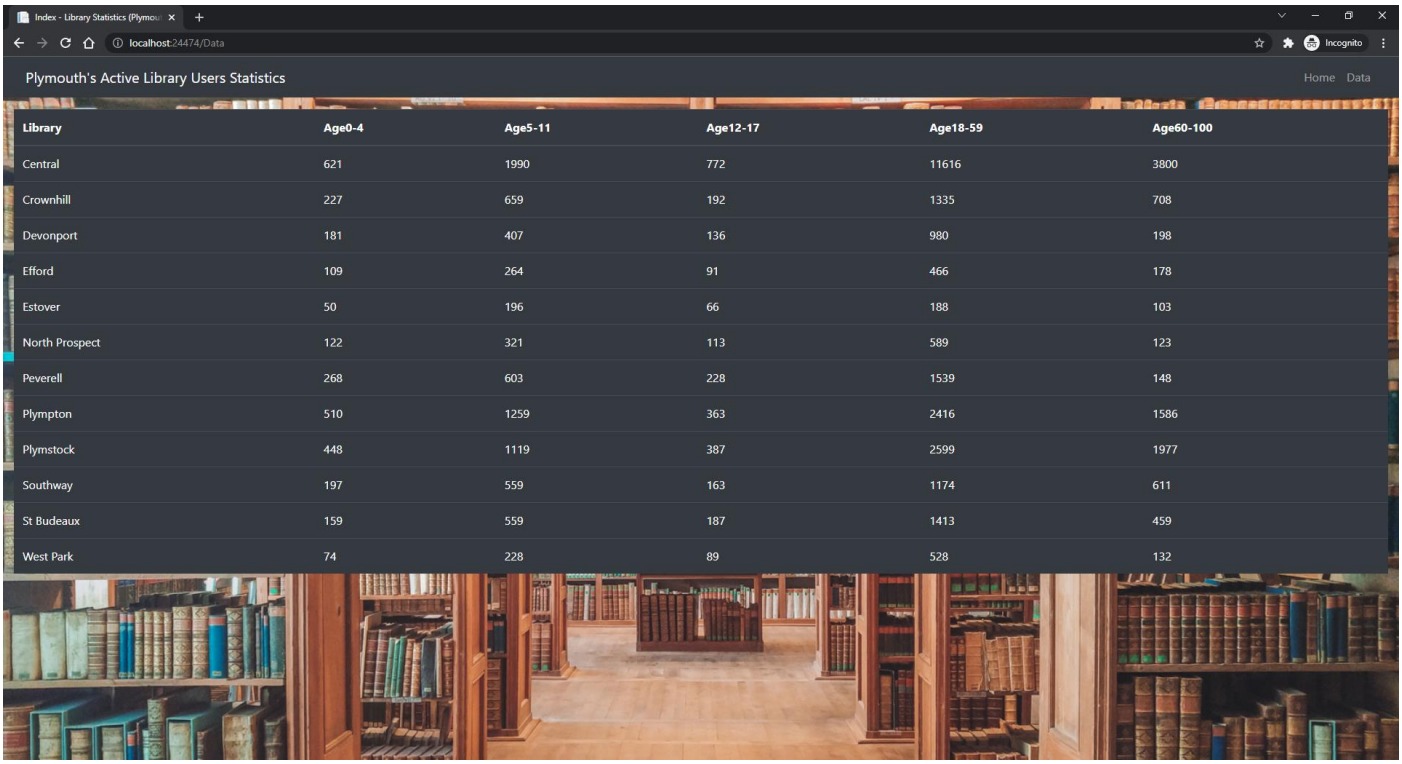


Figure 15 – RDF Graph

Evaluation

The final project looks like these following screenshots:





Library	Age0-4	Age5-11	Age12-17	Age18-59	Age60-100
Central	621	1990	772	11616	3800
Crownhill	227	659	192	1335	708
Devonport	181	407	136	980	198
Efford	109	264	91	466	178
Estover	50	196	66	188	103
North Prospect	122	321	113	589	123
Peverell	268	603	228	1539	148
Plympton	510	1259	363	2416	1586
Plymstock	448	1119	387	2599	1977
Southway	197	559	163	1174	611
St Budeaux	159	559	187	1413	459
West Park	74	228	89	528	132

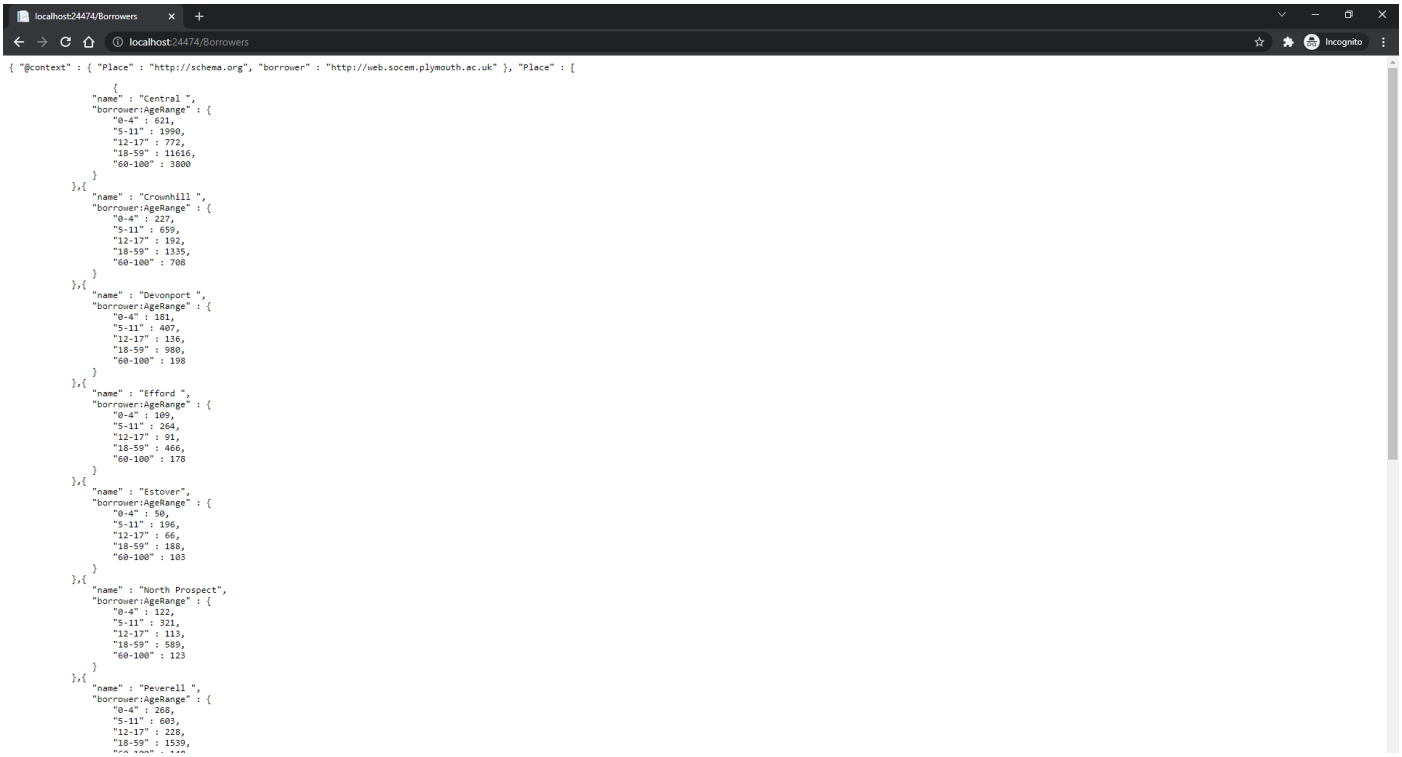


Figure 15 – Website screenshots (Index, Data, Borrowers)

Throughout this report, improvements and issues have been clearly stated. But, to reiterate some of these points: the LSEP considerations of both the API and LDA could be improved to ensure that data is completely secure and that in the instances where data can be accessed by users, these users are ensured to be authorized.

Finally, due to lack of time it is important to state that a hosted implementation of Task 2 could not be procured, and that the version hosted on the GitHub and shown within the screenshots is being run through Visual Studio's LocalHost. The lesson learnt from this is that time needs to be scheduled more effectively so that code can be tested and deployed successfully – it was this deployment stage that unfortunately was sacrificed to make sure that the code that was made, met the project specification.

References:

Developer.mozilla.org. n.d. ARIA - Accessibility | MDN. [online] Available at: <https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA> [Accessed 20 January 2022].

Back, C., 2018. Active library users by age - Data Place Plymouth. [online] Plymouth.thedata.place. Available at: <https://plymouth.thedata.place/dataset/active-library-users-by-age> [Accessed 20 January 2022].

GOV.UK. n.d. Data protection. [online] Available at: <https://www.gov.uk/data-protection> [Accessed 20 January 2022].

GOV.UK. n.d. How copyright protects your work. [online] Available at: <https://www.gov.uk/copyright> [Accessed 20 January 2022].

Ibm.com. n.d. IBM Docs. [online] Available at: <https://www.ibm.com/docs/en/informix-servers/12.10?topic=SSGU8G_12.1.0/com.ibm.ddi.doc/ids_ddi_186.htm> [Accessed 20 January 2022].