

BRidge Analysis Web Application Computer Science NEA

Deepkumar Pawar



**Table of Contents**

[ANALYSIS 3](#_Toc73806403)

[BACKGROUND TO PROJECT 4](#_Toc73806404)

[RESEARCH 4](#_Toc73806405)

[Identification / Interview of end-user / supervisor 4](#_Toc73806406)

[Prospective Users 5](#_Toc73806407)

[Currently Available Systems 5](#_Toc73806408)

[Research around Investigation Area 6](#_Toc73806409)

[Research into Programming Languages/Hardware 8](#_Toc73806410)

[OBJECTIVES AND AIMS 10](#_Toc73806411)

[EXTENSION OBJECTIVES 11](#_Toc73806412)

[ACCEPTABLE LIMITATIONS 11](#_Toc73806413)

[DOCUMENTED DESIGN 12](#_Toc73806414)

[OVERVIEW DIAGRAMS 12](#_Toc73806415)

[Basic Solution Hierarchy Chart: 12](#_Toc73806416)

[Basic System Flowchart for Bridge Analysis 12](#_Toc73806417)

[DATA FLOW DIAGRAM 13](#_Toc73806418)

[UML DIAGRAM 14](#_Toc73806419)

[WEB PAGES 14](#_Toc73806420)

[Index 15](#_Toc73806421)

[AnalyzeBridge 15](#_Toc73806422)

[ViewMaterials 15](#_Toc73806423)

[ViewLoads 15](#_Toc73806424)

[CreateMaterial 15](#_Toc73806425)

[CreateLoad 15](#_Toc73806426)

[RemoveMaterial 15](#_Toc73806427)

[RemoveLoad 15](#_Toc73806428)

[CLASSES 15](#_Toc73806429)

[BeamBridge 15](#_Toc73806430)

[Beam 19](#_Toc73806431)

[Pillar 20](#_Toc73806432)

[Material 20](#_Toc73806433)

[Load 20](#_Toc73806434)

[BridgeAnalyzer 20](#_Toc73806435)

[DATABASE: LOADS 23](#_Toc73806436)

[DATABASE: MATERIALS 23](#_Toc73806437)

[USER INTERFACE 24](#_Toc73806438)

[SYSTEM SECURITY AND INTEGRITY OF DATA 24](#_Toc73806439)

[TECHNICAL SOLUTION 24](#_Toc73806440)

[CLASSES 26](#_Toc73806441)

[Beam.cs 26](#_Toc73806442)

[BeamBridge.cs 27](#_Toc73806443)

[Pillar.cs 31](#_Toc73806444)

[BridgeAnalyzer.cs 32](#_Toc73806445)

[MODELS 35](#_Toc73806446)

[Material.cs 35](#_Toc73806447)

[Load.cs 37](#_Toc73806448)

[PAGES 38](#_Toc73806449)

[\_Layout.cshtml 38](#_Toc73806450)

[AnalyzeBridge.cshtml 41](#_Toc73806451)

[AnalyzeBridge.cshtml.cs 43](#_Toc73806452)

[Index.cshtml 51](#_Toc73806453)

[Index.cshtml.cs 52](#_Toc73806454)

[ViewMaterials.cshtml 53](#_Toc73806455)

[ViewMaterials.cshtml.cs 55](#_Toc73806456)

[ViewLoads.cshtml 57](#_Toc73806457)

[ViewLoads.cshtml.cs 59](#_Toc73806458)

[CreateMaterial.cshtml 60](#_Toc73806459)

[CreateMaterial.cshtml.cs 62](#_Toc73806460)

[CreateLoad.cshtml 65](#_Toc73806461)

[CreateLoad.cshtml.cs 66](#_Toc73806462)

[RemoveMaterial.cshtml 70](#_Toc73806463)

[RemoveMaterial.cshtml.cs 70](#_Toc73806464)

[RemoveLoad.cshtml 72](#_Toc73806465)

[RemoveLoad.cshtml.cs 73](#_Toc73806466)

[TESTING 74](#_Toc73806467)

[UNIT TESTING 74](#_Toc73806468)

[BeamBridge.cs 75](#_Toc73806469)

[BridgeAnalyzer.cs 81](#_Toc73806470)

[FINAL SOLUTION TESTING 86](#_Toc73806471)

[Homepage (Index) 86](#_Toc73806472)

[AnalyzeBridge 87](#_Toc73806473)

[ViewMaterials and ViewLoads 91](#_Toc73806474)

[CreateMaterial 92](#_Toc73806475)

[RemoveMaterial 94](#_Toc73806476)

[SUMMARY 95](#_Toc73806477)

[EVALUATION 95](#_Toc73806478)

[REFLECTION ON EFFECTIVENESS 95](#_Toc73806479)

[EVALUATION OF OBJECTIVES 96](#_Toc73806480)

[INDEPENDENT FEEDBACK FROM END/USER AND SUPERVISOR 97](#_Toc73806481)

[EVALUATION OF FEEDBACK 98](#_Toc73806482)

[IMPROVEMENTS ON REVISITATION 98](#_Toc73806483)

[APPENDIX 99](#_Toc73806484)

[BEAM STRESS TEST 99](#_Toc73806485)

[SHEAR STRENGTH TEST 100](#_Toc73806486)

[DEFLECTION TEST 101](#_Toc73806487)

[Bibliography 102](#_Toc73806488)

# ANALYSIS

BACKGROUND TO PROJECT

The job of a bridge engineer is to design a bridge and then analyze it using various engineering techniques to determine if the bridge designed would fail or succeed. My project involves automating the analysis of a bridge design to determine whether it would fail or succeed.

There exist professional solutions in the market. These are highly sophisticated software that help model and design bridges as well as conduct a civil engineering analysis. However, the main drawback of these software is that they are expensive and bulky in nature and must be downloaded. Their licensing is also rigid in nature meaning that it is difficult for a standalone client to use them. My solution would be simple and run on an HTTP web server that can be accessed online.

RESEARCH

Identification / Interview of end-user / supervisor

For this project, I am being helped by Elisa Izquierdo-Acebes, my school physics teacher, as the supervisor for this project. She shall also act as my client.

Since the merits of this system are that it is lightweight, online and accessible to use, the end users should typically be small businesses and individuals. This is described further in the next section.

Questions for Supervisor:

Q1: What is the physics behind a bridge and how does it succeed in holding load?

A1: Bridge physics is mainly based on statics and support of loads. Bridges do this by different mechanisms based on the type of bridge.

[Has sent a helpful link which explains this in detail, and would be used for research]

Q2: What are the different environmental factors that a bridge would have to fend against to stop it from collapsing?

A2: Bridge design must factor in wind conditions, the soil, seismic conditions, weather conditions, traffic conditions, etc.

Q3: What are the different types of bridges and how does the physics change between them?

A3: Bridges occur in different types. They all have the same purpose which is to support the load, but they do this in different ways. These include beam bridges, arch bridges, suspension bridges, and more.

[Has sent a helpful link which explains this in detail, and would be used for research]

Q4: Would there be a market for this type of application and who would comprise prospective users?

A4: Since there exists a similar professional product in the market which is licensed, there must be a market for this application. Prospective users would probably be teams of such civil engineers possibly working in small teams and do all these calculations manually. A very important point to note here is that a client must have confidence in the product, which means that it must be tested to a high degree and standard before the possibility of it being used.

Prospective Users

The prospective users fall into the following groups:

1. Professional civil engineers: these could use the application for analyzing real bridge designs as they would be able to input their bridge variables and the application would return the success of the bridge.
2. Students; these could use the application for learning or experimenting with different values of the variables involved in bridge design to understand how that would affect the bridge analysis without having to pay or indulge in complicated licensing.

Currently Available Systems

Currently, there exist highly sophisticated software that help in modelling, designing and analyzing bridges. These software have a license based subscription service.

Examples of such software:

1. Bentley Systems Bridge Analysis Software
   1. Helps address complex modelling, design and analysis for all types of bridges. Features construction simulation. Features 3D modelling and visualization for bridge design. (Bentley, 2020)
   2. Pricing is subscription based and approximately $250 per month per user. (SOURCEFORGE, 2020). This is not an official price but one I found online. To get official pricing from the website, one must fill out a special form which requires to be part of an organization.
   3. Thus, a standalone client cannot use this this software
2. CivilFem for ANSYS
   1. Used for carrying out various prescribed analyses based on civil engineering codes on structures such as bridges, tunnels, buildings, etc. It is an advanced software package particularly available for civil engineering projects. (CivilFEM, 2020)
   2. Pricing is subscription based. The price for similar software by the same developer is around $1500 annually so this might give insight into what the pricing might be, but it is still unknown. (CivilFEM, 2020). To get the official pricing from the website, one must fill out a special form which requires to be part of a company.
   3. Thus, a standalone client cannot use this software.

The main problems of these software are that they are paid, and subscription based. This closes off clients that are not part of a particular company or organization that needs such a service. These are very powerful and sophisticated software. But these features might be overwhelming for a beginner and thus, discourages clients who are not strictly civil engineers that are required to use this software.

My application would help in these aspects as follows:

1. It would be free and easily accessible to all
2. It would strip unnecessary features such as 3D modelling and design and provide the core functionality of bridge analysis.

Research around Investigation Area

The investigation area is that of the physics behind how bridges function and fail to collapse under its load.

A bridge has to deal with forces produced by loads. These loads are produced by the traffic sustained by the bridge. The forces produced by loads occur in the form of compression and tension forces.

Compression acts on the top side of the bridge, at the point of contact between the traffic vehicles and the bridge. This causes the bridge to bend downwards. Since the ends of the bridge are fixed by piers and abutments, tension forces act on the bottom side of the bridge, pulling it apart. When the compression forces reach a certain threshold, the bridge buckles. When the tension forces reach a certain threshold, the bridge snaps.

To prevent this from happening, the bridge has mechanisms that either transfer these forces of compression or tension to a stronger part, or they redistribute them over a larger area which results in less pressure on the bridge.

Bridges come in different types based on how they counteract forces of compression and tension.

First type of bridge is a beam bridge. This bridge is supported by abutments which are pillar like supports in the middle of the bridge’s length. These types of bridges span very short distances but are able to carry very heavy loads.

Second type of bridge is an arch bridge. This bridge is similar to a beam bridge, but it is supported by arches. This is more effective than a beam bridge at supporting loads over a longer distance but is still used for short distances.

Third type of bridge is a suspension bridge. This type of bridge has large firmly grounded towers throughout the length of the bridge. Strong main cables run from each tower to the next and end at firm anchorages at each end of the bridge. From these main cables, further smaller strong cables of steel chains drop down and are attached to the bridge itself. These cables thus support the load of the bridge. They transfer the compression forces on the top of the bridge to tension forces in the cables. This type of bridge is able to span very long distances even over a few miles.

(Woodford, 2020)

In reference to the scope of this project, we shall only be describing a beam bridge in detail.

#### Design Concept

The beam bridge would consist of the platform or the beam which carries the load, supported by abutments or pillars. These pillars support the beam and are arbitrary in number. The bridge must have at least two pillars as the supports on either end of the bridge shall be counted as pillars as well.

The beam shall be modelled as a cuboid made of a single material whose properties shall play a role in the bridge analysis. The pillars would be of the same material as well.

The load shall be modelled as a distributed load to represent when the bridge is completely occupied. This would be the case when the load is maximum on the bridge and shall be considered for the tests.

All values used in the algorithm and input by the user shall be assumed to be in SI units.

#### Bridge Analysis

The analysis would consist of three tests, the Beam Stress Test, the Shear Strength Test and the Deflection Test.

The Beam Stress Test checks whether the bridge can manage the stress induced by the load. Stress is a measure of the intensity of an internal force in a member. For a bridge, stresses are caused due to tension and compression forces acting on the bridge as a result of the load on it as well as the beam self-weight. Thus, the test calculates the maximum stress the bridge could face, i.e. worst case scenario (where there is the maximum load on the bridge) and compares that to the maximum stress he bridge could take based on the bridge variables and properties. Using a factor of safety to ensure that these two values are considerably apart to account for unforeseen circumstances, the test returns a pass or fail value.

The Shear Strength Test checks whether the bridge is susceptible to collapse due to shear forces. Shear forces are forces that usually act horizontally within a bridge in opposing directions, causing the bridge to buckle and tear itself apart. Similar to the Beam Stress Test, this tests compares the maximum shear strength of the bridge to the maximum shear stress it will face.

The Deflection Test checks the deflection caused due to the load on the bridge on the beam. This deflection is the downward bending of the middle of the beam between two pillars or abutments.

Further detail on the physics behind these tests can be found in the appendix.

Note that these tests are to take place for the length of the bridge between two pillars. In the solution, the length of the beam is divided into numerous sections which encompass the gaps between each pair of neighbouring pillars. The tests are carried out for all of these sections. If any test fails, the entire bridge analysis returns a failure message. To pass the bridge analysis, each section must return a pass on each test.

#### Bridge Variables

Thus, to conduct bridge analysis, the variables required are the following:

* Length of the beam
* Width of the beam
* Height of the beam
* Weight of the beam
* Number of pillars for the bridge, including the ones at the end
* Material for the bridge
* The heaviest type of load to be present on the bridge (pedestrian, medium sized car or large and heavy vehicles like trucks, etc.)
* Factor of safety (see appendix)

Research into Programming Languages/Hardware

This solution should have the following core functionalities:

1. The application should be hosted on a server
2. On the client side, the user should be presented with a form. In this form the user would enter the variables of bridge design as required by the application.
3. The application would then evaluate these entered variables for analysis using an algorithm.
4. This algorithm would produce an output that would tell whether the bridge succeeded or failed the analysis.
5. This output should then be shown to the user.

A web browser communicates with the web server by sending and receiving HTTP requests. A request sent by the browser to the server describes what response it needs and can also embed data such as form data. The server processes this and sends the desired response to the browser that displays to the user as needed. This server-side processing is programmed, and this is called server-side programming. (MDN contributors, 2019). The server, in the case of this solution, would take care of returning the result of the algorithmic application and retrieving from a database as required. This server-side programming is usually done with the help of web frameworks that simplify the process of server-side development. The client-side programming typically does not need a framework and can be simply done using JavaScript. (MDN contributors, 2019). Thus, in conclusion, I shall need a web HTTP server and a web server-side framework.

In this solution, the user would enter the bridge design variables in a form on the client side. This input would then be validated and sent to the server for processing. The form itself would be designed and developed on the client side using HTML (HTML5), CSS and JavaScript. The code for form validation that would ensure that the server-side application would not break would be written in JavaScript. This would be client-side programming. (MDN contributors, 2020).

The data that would be required for the algorithmic web application would be organized using a Relational DataBase Management System (RDBMS) such as MySQL, Oracle Database, etc. (MDN contributors, 2020). Typically, a developer would have to use SQL to read and update data from a database. Instead, one can use Object-Relational Mappers (ORMs) that are code libraries that provide a higher-level abstraction that can transfer data stored in a database to objects that can be directly used in the application code. The ORM is respective to the used web framework. (Makai, 2020). This might be helpful for it eliminates having to write SQL code and thus, I can directly call data from the database through my server-side programming.

After some research, I found that there are two general approaches towards web development: traditional web applications that do most of the application logic on the server, and single page applications (SPAs) that do most of the user interface logic in the web browser. Broadly speaking, one should use a traditional web application when the application’s client -side requirements are simple and mostly read-only. If the application needs a rich client-side user interface, one should use a SPA. A hybrid approach is also possible. The SPA implementation would typically use JavaScript or TypeScript. (Microsoft, 2020). For this solution, I shall use a traditional web application approach. This is because the client-side requirements in UI are not complicated. It is simply a form. However, if I wish to enrich it or add extra features depending on extension objectives, I can always go with a hybrid approach and use JavaScript or TypeScript for the same.

Therefore, for this project, the key technologies I shall be needing are:

1. HTTP Server
2. Web framework
3. Database Management System (DBMS)

For the web framework, I have chosen to use ASP .Net Core for the following reasons:

1. I am familiar with coding in C# and thus it would be easier to work with
2. It is the language used in the A level course and it could be beneficial to use it to maintain familiarity with the language
3. Has sophisticated programming features such as automatic memory management, garbage collection, language independence, cross platform architecture and more. (Gadhavi, 2020)
4. Would be beneficial for me to practice object orientation with C# using this project
5. Supports unit testing and dependency injection (Microsoft, 2019).

The main reason why I am using ASP .Net Core rather than other frameworks such as NodeJS, etc. is the familiarity and close vicinity to object orientation and C#. ASP .Net Core also boasts multiple sophisticated features that are more than capable of helping me achieve my objectives. A major feature is the support for unit testing which would be incredibly helpful during development.

My choice for HTTP Server used is Apache HTTP Server because:

1. It is free and open source
2. Reliable and stable
3. Beginner friendly
4. Huge available support
5. Works with databases well

(G., 2020)

Since I have also decided to use ASP .Net Core as my web framework, I must also use IIS web server which is the supported web server by ASP .Net Core. However, I am able to set up Apache as a reverse proxy web server using XAMPP a free and open-source cross-platform web server solution stack package. This thus allows me to use Apache HTTP Server. For DBMS, I shall be using MySQL which is available to use through XAMPP.

The main reason to use XAMPP, apart from its simplicity and ease of use, is that it is run on Windows on the school computers, making it accessible to use.

The choice for DBMS is MySQL because:

1. Free and open source
2. Most popular, thus it would have much support
3. High performance
4. Strong data protection
5. Provides comprehensive support for all application development needs
6. Management ease (quick start capability, cross platform, self-management features) (DBQuest, Inc., 2012)
7. Supported by XAMPP and run on school computers

Although MySQL also provides Relational Database Management capabilities, I shall not be needing them for this solution. Thus, MySQL fits the role.

OBJECTIVES AND AIMS

1. Client connect to Web Server
   1. Start running the Web Server
   2. Get information such as hyperlink or IP address that will be used to connect to Web Server
   3. Connect to Server as a Client from a different computer
   4. Load and display home webpage
2. Validate values entered in Bridge Variables form
   1. Check that these values are:
      1. In the appropriate range of values
      2. In the correct format
   2. If values entered are improper, show message alerting the user that the values are improper
3. Web Server Communication
   1. Extract Bridge Variable values from HTTP request received and send to Web application
   2. Get Result of Bridge Analysis from Web Application and send to client as a HTTP response.
4. Produce Bridge Analysis Result
   1. Get Bridge Variables as input from Web Server
   2. Get relevant data from database
   3. Perform Bridge Analysis Algorithm using this input
   4. Send Bridge Analysis Result to Web Server
5. Database CRUD Operation
6. Allow user to access webpages separate to the bridge analysis webpage
7. Allow user to view the contents of the database containing relevant tables with data used by the bridge analysis algorithm
8. Allow user to add and delete items to the tables such as add new materials or load types which can then be used in the bridge analysis algorithm
9. User Communication
10. Load and view starting webpage
11. Provide links to other webpages
12. Display Bridge Variables form on the Bridge Analysis webpage
13. Be able to enter appropriate values in the form
14. Add a “Submit” values button that triggers the HTTP request
15. Display message for entering improper or inappropriate values in the form
16. Display Bridge Analysis Result on web page
17. Allow user to send different values for bridge analysis

EXTENSION OBJECTIVES

1. Upgrade Bridge Analysis Algorithm
2. Make webpage look more visually appealing and rich

ACCEPTABLE LIMITATIONS

Analysis of a bridge comprehensively is a difficult task and the development of which would be

Comprehensive analysis of a bridge would include analyzing multiple highly complex entities and factors such as different shapes of the bridge, different bridge types, various weather conditions, etc. Such a comprehensive analysis would not be feasible in terms of the time and expertise available. Thus, in discussion with my supervisor, we agreed on the following limitations:

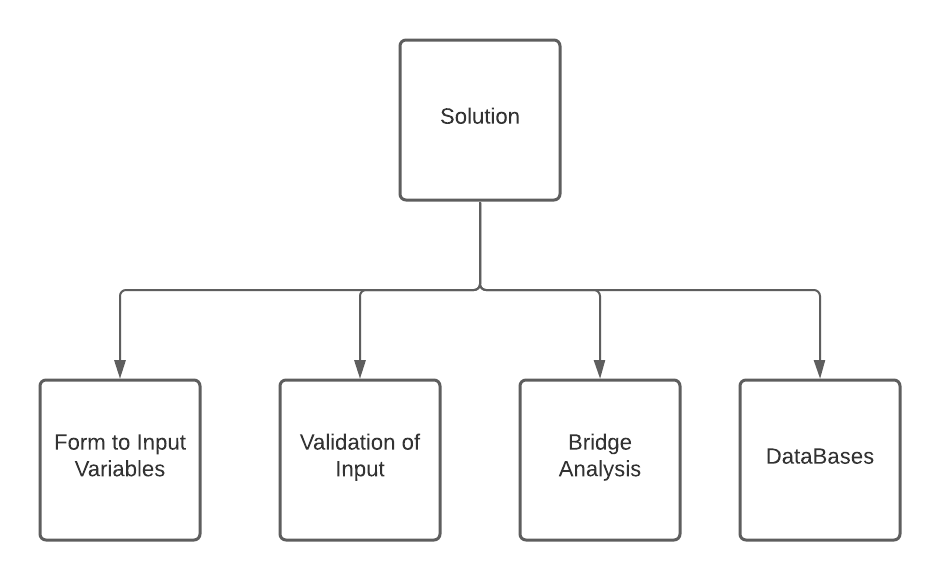
* The system would only need to consider beam bridges and no other types of bridges such as suspension bridges, etc.
* The beam would be modelled as cuboidal in shape.
* The tests that would be a part of the analysis would only need to consist of a Beam Stress Test, a Shear Strength Test and a Deflection Test
* The analysis in the solution need not factor in highly complex phenomena such as weather conditions

Although these limitations have been discussed and decided, the solution is designed so that further addition of features beyond the scope of the aims of the project is not only possible but easy to implement without breaking the structure of the solution.

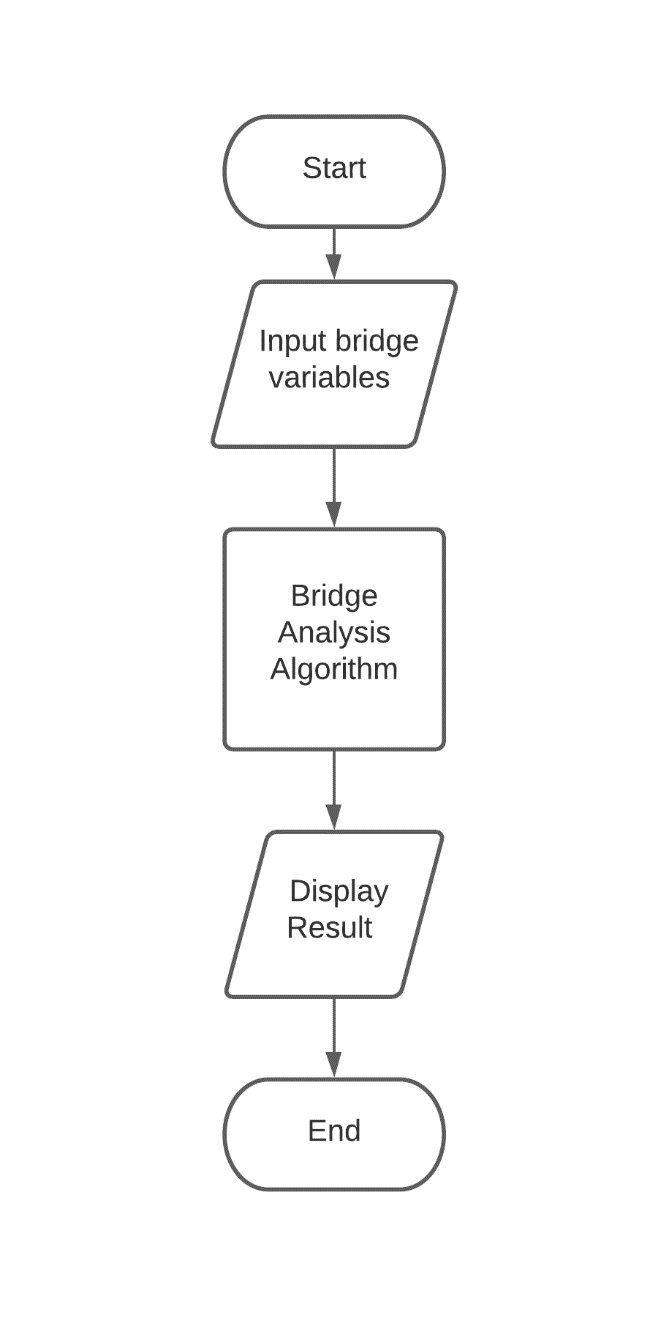
# DOCUMENTED DESIGN

## OVERVIEW DIAGRAMS

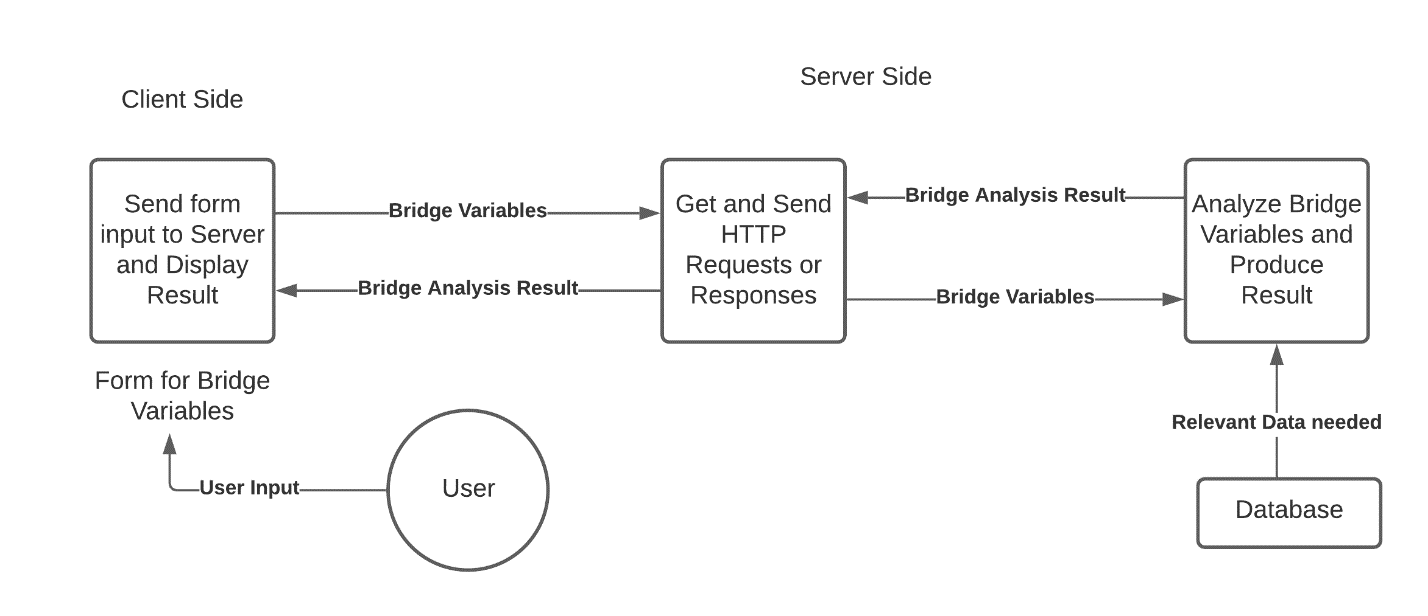
### Basic Solution Hierarchy Chart:



### Basic System Flowchart for Bridge Analysis

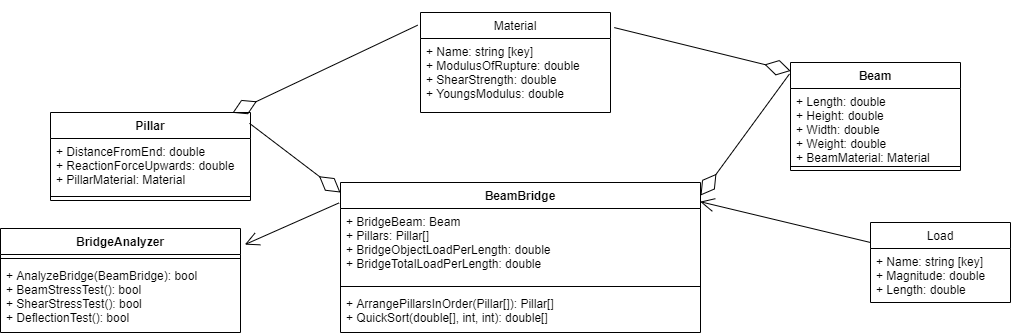


## DATA FLOW DIAGRAM

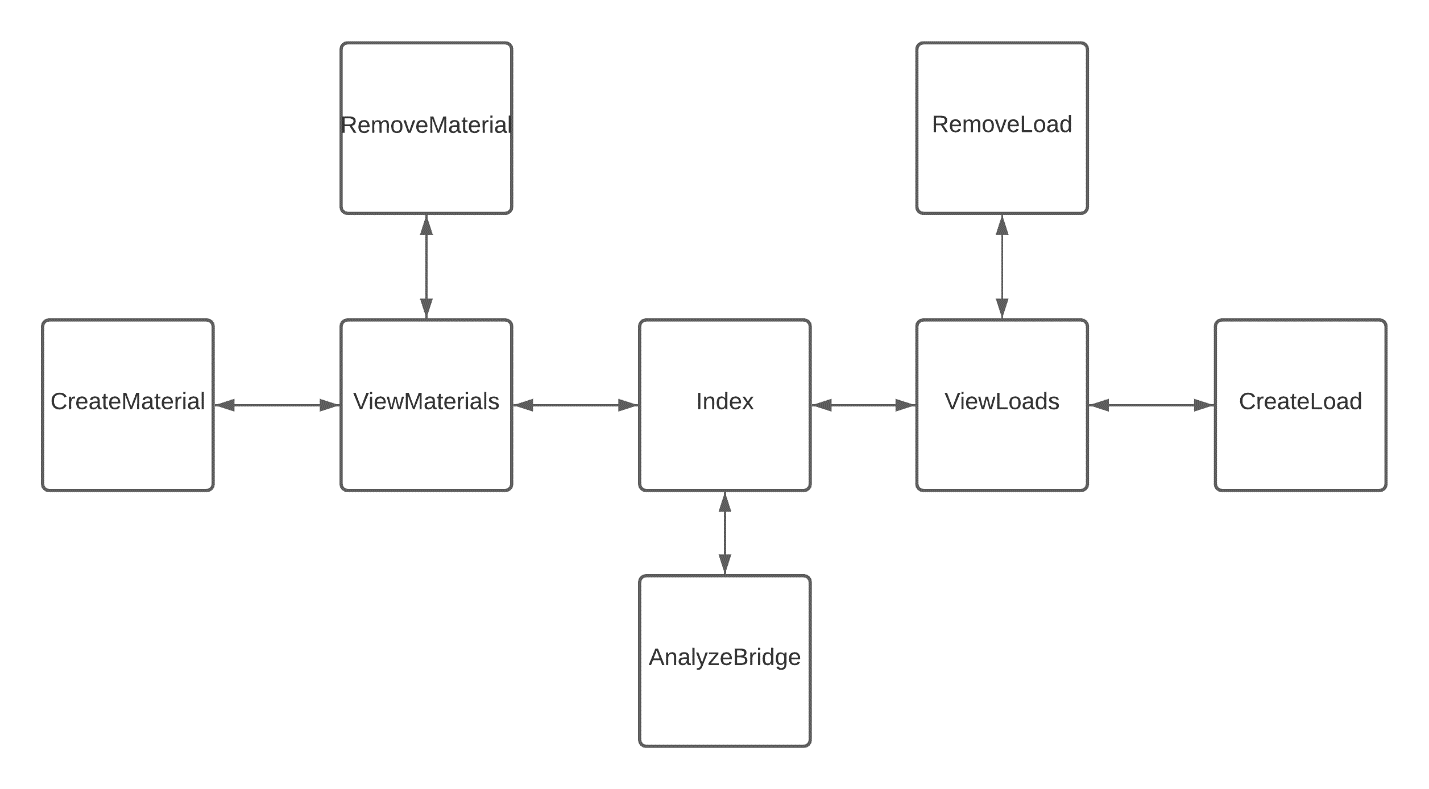


This diagram shows how data would move through the system. The arrows represent the flow of data and the boxes represent the different parts of the system where the data can go.

## UML DIAGRAM



## WEB PAGES



This diagram shows each of the web pages in the system and how they should link to each other.

### Index

This is the homepage which is the webpage the user sees first. It is called Index because of ASP .Net Core convention and design where the starting page is usually called Index.

### AnalyzeBridge

This is the page where the user finds a form where they can enter in values for bridge analysis. Once they do, they can press a button that submits the data and performs the algorithm on it. The page can then display a message to the user informing them of the result of the bridge analysis.

### ViewMaterials

This is the page where the user can view the materials and their properties stored in the database which can be used for bridge analysis.

### ViewLoads

This is the page where the user can view the different types of loads and their properties stored in the database which can be used for bridge analysis.

### CreateMaterial

This is the page where the user can create a new material by filling a form to specify the values of the properties required and then add the material to the database.

### CreateLoad

This is the page where the user can create a new load by filling a form to specify the values of the properties required and then add the load to the database.

### RemoveMaterial

This is the page where the user can delete a material from the database by specifying its name.

### RemoveLoad

This is the page where the user can delete a load from the database by specifying its name.

## CLASSES

### BeamBridge

#### Data Structures

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| BridgeBeam | Beam | Holds a reference to the Beam object for that BeamBridge. |
| Pillars | Pillar[] | Array of Pillar objects for the BeamBridge |
| BridgeObjectLoadPerLength | double | Holds the value of the load due to the weight of the objects (not including weight of beam) on the bridge, distributed across the bridge. |
| BridgeTotalLoadPerLength | double | Holds the value of the total load on the bridge, distributed across the bridge. |

#### QuickSort Function Pseudocode

SUBROUTINE

QuickSort(array, start, end)

IF start >= end THEN

RETURN array

ENDIF

pivot  start

left  start + 1

right  end

sorted  NEW ARRAY[LEN(array)]

sorted  array

index  start

FOR i  start + 1 TO end

IF array[i] < array[pivot] THEN

sorted[index]  array[i]

index  index + 1

ENDIF

ENDFOR

left  index - 1

sorted[index]  array[pivot]

index  index + 1

right  index

FOR i  start + 1 TO end

IF array[i] > array[pivot]

sorted[index] array[i]

index  index + 1

ENDIF

ENDFOR

sorted  QuickSort(sorted, start, left)

sorted  QuickSort(sorted, right, end)

RETURN sorted

ENDSUBROUTINE

#### ArrangePillarsInOrder Function Pseudocode

SUBROUTINE

ArrangePillarsInOrder(pillars)

length  LEN(pillars)

distances  NEW ARRAY[length]

FOR i  0 TO Length – 1

distances[i]  pillars[i].DistanceFromEnd

ENDFOR

distances  QuickSort(distances, 0, length – 1)

arrangedPillars = NEW ARRAY[length]

FOR i  0 TO length – 1

FOR j  0 TO length – 1

IF distances[i] = pillars[j].DistanceFromEnd THEN

arrangedPillars[i]  pillars[j]

ENDIF

ENDFOR

ENDFOR

RETURN arrangedPillars

ENDSUBROUTINE

#### EvaluateBridgeForces Function Pseudocode

Note that this function was removed from the final solution as it became redundant.

SUBROUTINE

EvaluateBridgeForces(loadDistribution)

pillarsNum  LEN(Pillars)

distances  NEW ARRAY[pillarsNum]

FOR i  0 TO pillarsNum – 1

distances[i]  Pillars[i].DistanceFromEnd

ENFOR

FOREACH loadDistance IN loadDistribution.LoadsDistancesDict.Keys

FOR i  0 TO pillarsNum – 2

IF loadDistance = distances[i] THEN

Pillars[i].ReactionForceUpwards  Pillars[i].ReactionForceUpwards + loadDistribution.LoadsDistancesDict[loadDistance]

BREAK

ENDIF

IF loadDistance > distances[i] AND loadDistance < distances[i + 1] THEN

Pillars[i].ReactionForceUpwards  Pillars[i].ReactionForceUpwards + loadDistribution.LoadsDistancesDict[loadDistance] / 2

Pillars[i + 1].ReactionForceUpwards  Pillars[i + 1].ReactionForceUpwards + loadDistribution.LoadsDistancesDict[loadDistance] / 2

BREAK

ENDIF

ENDFOR

IF loadDistance = distances[pillarsNum - 1] THEN

Pillars[pillarsNum - 1].ReactionForceUpwards  Pillars[pillarsNum – 1]. ReactionForceUpwards + loadDistribution.LoadsDistancesDict[loadDistance]

ENDIF

ENDFOREACH

beamWeightDistribution  BridgeBeam.Weight / pillarsNum

FOREACH pillar IN Pillars

pillar.ReactionForceUpwards  pillar.ReactionForceUpwards + beamWeightDistribution

ENDFOREACH

ENDSUBROUTINE

### Beam

#### Data Structures

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| Length | double | Length of the beam |
| Height | double | Height of the beam |
| Width | double | Width of the beam |
| Weight | double | Weight of the beam |
| BeamMaterial | Material | Material of the beam, from the list of stored materials in the database |

### Pillar

#### Data Structures

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| DistanceFromEnd | double | Distance of that pillar from the end of the bridge |
| ReactionForceUpwards | double | Normal reaction force of that pillar upwards |
| PillarMaterial | Material | Material of the pillar, from the list of stored materials in the database |

### Material

#### Data Structures

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| Name | string | Name of the material |
| ModulusOfRupture | double | Property of the material |
| ShearStrength | double | Property of the material |
| YoungsModulus | double | Property of the material |

### Load

#### Data Structures

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| Name | string | Name of the type of load |
| Magnitude | double | Magnitude of one instance of that load |
| Length | double | Length of load object (and extra length for space between load objects) |

### BridgeAnalyzer

#### Data Structures

No data structures.

#### BeamStressTest Function Pseuodocode

SUBROUTINE

BeamStressTest(bridge, factorOfSafety)

sectionLength 🡨 0

maxInternalMoment 🡨 0

elasticSectionModulus 🡨 bridge.BridgeBeam.Width \* (bridge.BridgeBeam.Height)2 / 6

maximumStress 🡨 0

FOR i 🡨 0 TO LEN(Bridge.Pillars) – 2

sectionLength 🡨 ABSOLUTEVALUE(bridge.Pillars[i + 1]).DistanceFromEnd - bridge.Pillars[i]).DistanceFromEnd

maxInternalMoment 🡨 bridge.BridgeTotalLoadPerLength \* (sectionLength)2 / 8

maximumStress = maxInternalMoment / elasticSectionModulus

IF maximumStress > bridge.BridgeBeam.BeamMaterial.ModulusOfRupture / factorOfSafety THEN

RETURN False

ENDIF

RETURN True

ENDFOR

ENDSUBROUTINE

#### ShearStrengthTest Function Pseudocode

SUBROUTINE

ShearStrengthTest(bridge, factorOfSafety)

maxInternalShearForce 🡨 0

shearStress 🡨 0

sectionLength 🡨 0

FOR i 🡨 0 TO LEN(Bridge.Pillars) – 2

sectionLength 🡨 ABSOLUTEVALUE(bridge.Pillars[i + 1]).DistanceFromEnd - bridge.Pillars[i]).DistanceFromEnd

maxInternalShearForce 🡨 bridge.BridgeTotalLoadPerLength \* sectionLength / 2

shearStress 🡨 maxInternalShearForce / (bridge.BridgeBeam.Width \* bridge.BridgeBeam.Height)

IF shearStress > bridge.BridgeBeam.BeamMaterial.ShearStrength / factorOfSafety THEN

RETURN False

ENDIF

RETURN True

ENDFOR

ENDSUBROUTINE

#### DeflectionTest Function Pseudocode

SUBROUTINE

DeflectionTest(bridge, factorOfSafety)

sectionLength 🡨 0

maxDeflection 🡨 0

elasticSectionModulus 🡨 bridge.BridgeBeam.Width \* (bridge.BridgeBeam.Height)2 / 6

areaMomentOfInertia 🡨 elasticSectionModulus \* bridgeBeam.Height / 2

w 🡨 bridge.BridgeObjectLoadPerLength

FOR i 🡨 0 TO LEN(Bridge.Pillars) – 2

sectionLength 🡨 ABSOLUTEVALUE(bridge.Pillars[i + 1]).DistanceFromEnd - bridge.Pillars[i]).DistanceFromEnd

maxDeflection 🡨 5 \* w \* (sectionLength)4 / (384 \* bridge.BridgeBeam.BeamMaterial.YoungsModulus \* areaMomentOfInertia)

IF maxDeflection > sectionLength / 360 THEN

RETURN False

ENDIF

RETURN True

ENDFOR

ENDSUBROUTINE

## DATABASE: LOADS

DATA DICTIONARY:

|  |  |  |  |
| --- | --- | --- | --- |
| ATTRIBUTE | TYPE | FORMAT/VALIDATION | DESCRIPTION |
| Name (Primary Key) | Varchar(25) | Should be unique and not null | Name of the type of load |
| Magnitude | Double | Should be in nonnegative and in newtons | Magnitude of one instance of that load |
| Length | Double | Should be nonnegative and in metres | Length of load object (and extra length for space between load objects) |

## DATABASE: MATERIALS

DATA DICTIONARY:

|  |  |  |  |
| --- | --- | --- | --- |
| ATTRIBUTE | TYPE | FORMAT/VALIDATION | Description |
| Name (Primary Key) | String | Should be unique and not null | Name of the material |
| ModulusOfRupture | Double | Should be in newtons per metres squared and nonnegative | Property of the material |
| ShearStrength | Double | Should be in newtons per metres squared and nonnegative | Property of the material |
| YoungsModulus | Double | Should be in newtons per metres squared and nonnegative | Property of the material |

## USER INTERFACE

The user interface would be a simple form with a submit button. Once pressed, it would display a message at the bottom stating whether the bridge has passed the tests or not.

There would be links at appropriate places to other webpages.

## SYSTEM SECURITY AND INTEGRITY OF DATA

When the algorithm is executed, the bridge variables are not stored for later use. Thus, this data is deleted and not saved and does not pose security concerns or compromise the integrity of the data.

The user can store data by creating new Materials and Loads and saving them to the database. This process is done privately and cannot be accessed outside of the scope of the function that does this.

However, the process of deleting data from the database is simple and easy to do. Thus, this is something to keep in mind while using the system, to check that the required values to use from the database are correct and accurate.

The application does not use any particularly sensitive data and so system security and integrity of data is not a major issue.

# TECHNICAL SOLUTION

The following is the technical solution for this project. It consists of the C# class files as well as the HTML razor page files separated under headers of the same name. They are also organized under their folder names as in the visual studio solution.

After each code listing, there is a table that points to the interesting parts of that code listing using its line number.

To help navigate between the important files (classes and pages):

|  |  |
| --- | --- |
| INTERESTING BIT | SEE FILE |
| Lists and List Operations | AnalyzeBridge.cshtml.cs  ViewMaterials.cshtml.cs  ViewLoads.cshtml.cs  CreateMaterial.cshtml.cs  CreateLoad.cshtml.cs  RemoveMaterial.cshtml.cs  RemoveLoad.cshtml.cs  FROM PAGES |
| Database Connections and SQL | AnalyzeBridge.cshtml.cs  ViewMaterials.cshtml.cs  ViewLoads.cshtml.cs  CreateMaterial.cshtml.cs  CreateLoad.cshtml.cs  RemoveMaterial.cshtml.cs  RemoveLoad.cshtml.cs  FROM PAGES |
| Creation and use of cache object | AnalyzeBridge.cshtml.cs  FROM PAGES |
| Complex Client Server Model | Index.cshtml.cs  AnalyzeBridge.cshtml.cs  ViewMaterials.cshtml.cs  ViewLoads.cshtml.cs  CreateMaterial.cshtml.cs  CreateLoad.cshtml.cs  RemoveMaterial.cshtml.cs  RemoveLoad.cshtml.cs  FROM PAGES |
| OOP Class Aggregation | Material.cs in Pillar.cs  Pillar.cs in BeamBridge.cs  Material.cs in Beam.cs  Beam.cs in BeamBridge.cs  FROM CLASSES and MODELS |
| OOP Class Association | BeamBridge.cs with BridgeAnalyzer.cs  Load.cs with BeamBridge.cs  FROM CLASSES and MODELS |
| OOP Polymorphism (constructor overloading) | Material.cs  Load.cs  FROM MODELS |
| Recursive Algorithm | BeamBridge.cs  FROM CLASSES |
| Quick Sort Implementation | BeamBridge.cs  FROM CLASSES |
| Dynamic Object Generation | AnalyzeBridge.cshtml.cs  FROM PAGES |
| Algorithmic Calculations | BridgeAnalyzer.cs  FROM CLASSES |

## CLASSES

These are the main pure classes of the system.

### Beam.cs

1. using BridgeAnalysisWebApplication.Models;

2. using System;

3. using System.Collections.Generic;

4. using System.Linq;

5. using System.Threading.Tasks;

6.

7. namespace BridgeAnalysisWebApplication.Classes

8. {

9. public class Beam

10. {

11. public double Length { get; set; }

12. public double Height { get; set; }

13. public double Width { get; set; }

14. public double Weight { get; set; }

15. public Material BeamMaterial { get; set; }

16.

17. public Beam(double length, double height, double width, double weight, Material beamMaterial)

18. {

19. this.Length = length;

20. this.Height = height;

21. this.Width = width;

22. this.Weight = weight;

23. this.BeamMaterial = beamMaterial;

24. }

25. }

26.

|  |  |
| --- | --- |
| Line no. | Interesting bit |
| 17 | Constructor to make sure whenever a new Beam object is created, it always contains all the data required. |

### BeamBridge.cs

1. using BridgeAnalysisWebApplication.Models;

2. using System;

3. using System.Collections.Generic;

4. using System.Linq;

5. using System.Threading.Tasks;

6.

7. namespace BridgeAnalysisWebApplication.Classes

8. {

9. public class BeamBridge

10. {

11. public Beam BridgeBeam { get; set; }

12. public Pillar[] Pillars { get; set; }

13. public double BridgeObjectLoadPerLength { get; set; }

14. public double BridgeTotalLoadPerLength { get; set; }

15.

16. public BeamBridge(Beam bridgeBeam, Pillar[] pillars, double bridgeObjectLoadPerLength, double bridgeTotalLoadPerLength)

17. {

18. this.BridgeBeam = bridgeBeam;

19. this.Pillars = pillars;

20. this.BridgeObjectLoadPerLength = bridgeObjectLoadPerLength;

21. this.BridgeTotalLoadPerLength = bridgeTotalLoadPerLength;

22. }

23.

24. public Pillar[] ArrangePillarsInOrder() //Function to arrange pillars in order of their distance from end

25. {

26. int length = Pillars.Length;

27. double[] distances = new double[length];

28.

29. for (int i = 0; i < length; i++)

30. {

31. distances[i] = Pillars[i].DistanceFromEnd;

32. }

33.

34. distances = QuickSort(distances, 0, length - 1);

35.

36. Pillar[] arrangedPillars = new Pillar[length];

37.

38. for (int i = 0; i < length; i++)

39. {

40. for (int j = 0; j < length; j++)

41. {

42. if (distances[i] == Pillars[j].DistanceFromEnd)

43. {

44. arrangedPillars[i] = Pillars[j];

45. }

46. }

47. }

48.

49. return arrangedPillars;

50. }

51.

52. public double[] QuickSort(double[] array, int start, int end) //Quicksort implementation to sort an array of doubles

53. {

54.

55. if (start >= end)

56. {

57. return array;

58. }

59.

60. int pivot = start;

61.

62. int left = start + 1, right = end;

63.

64. double[] sorted = new double[array.Length];

65.

66. array.CopyTo(sorted, 0);

67.

68. int index = start;

69.

70. for (int i = start + 1; i < end + 1; i++)

71. {

72. if (array[i] < array[pivot])

73. {

74. sorted[index] = array[i];

75. index++;

76. }

77. }

78.

79. left = index - 1;

80. sorted[index] = array[pivot];

81. index++;

82. right = index;

83.

84. for (int i = start + 1; i < end + 1; i++)

85. {

86. if (array[i] > array[pivot])

87. {

88. sorted[index] = array[i];

89. index++;

90. }

91. }

92. sorted = QuickSort(sorted, start, left);

93. sorted = QuickSort(sorted, right, end);

94.

95. return sorted;

96. }

97.

98. }

99. }

100.

|  |  |
| --- | --- |
| Line no. | Interesting bit |
| 16 | Constructor to make sure whenever a new BeamBridge object is created, it always contains all the data required. |
| 92 & 93 | Instances of recursion, used to implement QuickSort |

### Pillar.cs

1. using BridgeAnalysisWebApplication.Models;

2. using System;

3. using System.Collections.Generic;

4. using System.Linq;

5. using System.Threading.Tasks;

6.

7. namespace BridgeAnalysisWebApplication.Classes

8. {

9. public class Pillar

10. {

11. public double DistanceFromEnd { get; set; }

12. public double ReactionForceUpwards { get; set; } = 0d;

13. public Material PillarMaterial { get; set; }

14.

15. public Pillar(double distanceFromEnd, Material pillarMaterial)

16. {

17. this.DistanceFromEnd = distanceFromEnd;

18. this.PillarMaterial = pillarMaterial;

19. }

20. }

21. }

22.

|  |  |
| --- | --- |
| Line no. | Interesting bit |
| 15 | Constructor to make sure whenever a new Pillar object is created, it always contains all the data required. |

### BridgeAnalyzer.cs

1. using System;

2. using System.Collections.Generic;

3. using System.Linq;

4. using System.Threading.Tasks;

5.

6. namespace BridgeAnalysisWebApplication.Classes

7. {

8. public class BridgeAnalyzer

9. {

10.

11. public bool AnalyzeBridge(BeamBridge bridge, double factorOfSafety) //Returns the aggregated result of all 3 tests

12. {

13. return BeamStressTest(bridge, factorOfSafety) & ShearStrengthTest(bridge, factorOfSafety) & DeflectionTest(bridge, factorOfSafety);

14. }

15.

16. public bool BeamStressTest(BeamBridge bridge, double factorOfSafety) //Returns the result of the Beam Stress Test

17. {

18. double sectionLength = 0;

19. double maxInternalMoment = 0;

20. double elasticSectionModulus = bridge.BridgeBeam.Width \* Math.Pow(bridge.BridgeBeam.Height, 2) / 6;

21. double maximumStress = 0;

22.

23. for (int i = 0; i < bridge.Pillars.Length - 1; i++)

24. {

25. sectionLength = Math.Abs(bridge.Pillars[i + 1].DistanceFromEnd - bridge.Pillars[i].DistanceFromEnd);

26.

27. maxInternalMoment = bridge.BridgeTotalLoadPerLength \* Math.Pow(sectionLength, 2) / 8;

28.

29. maximumStress = maxInternalMoment / elasticSectionModulus;

30.

31. if (maximumStress > bridge.BridgeBeam.BeamMaterial.ModulusOfRupture / factorOfSafety)

32. {

33. return false;

34. }

35. }

36.

37. return true;

38. }

39.

40. public bool ShearStrengthTest(BeamBridge bridge, double factorOfSafety) //Returns the result of the Shear Strength Test

41. {

42. double maxInternalShearForce = 0;

43. double shearStress = 0;

44. double sectionLength = 0;

45.

46. for (int i = 0; i < bridge.Pillars.Length - 1; i++)

47. {

48. sectionLength = Math.Abs(bridge.Pillars[i + 1].DistanceFromEnd - bridge.Pillars[i].DistanceFromEnd);

49.

50. maxInternalShearForce = bridge.BridgeTotalLoadPerLength \* sectionLength / 2;

51.

52. shearStress = maxInternalShearForce / (bridge.BridgeBeam.Width \* bridge.BridgeBeam.Height);

53.

54. if (shearStress > bridge.BridgeBeam.BeamMaterial.ShearStrength / factorOfSafety)

55. {

56. return false;

57. }

58. }

59.

60. return true;

61. }

62.

63. public bool DeflectionTest(BeamBridge bridge, double factorOfSafety) //Returns the result of the Deflection Test

64. {

65. double sectionLength = 0;

66. double maxDeflection = 0;

67. double elasticSectionModulus = bridge.BridgeBeam.Width \* Math.Pow(bridge.BridgeBeam.Height, 2) / 6;

68. double areaMomentOfInertia = elasticSectionModulus \* bridge.BridgeBeam.Height / 2;

69. double w = bridge.BridgeObjectLoadPerLength;

70.

71. for (int i = 0; i < bridge.Pillars.Length - 1; i++)

72. {

73. sectionLength = Math.Abs(bridge.Pillars[i + 1].DistanceFromEnd - bridge.Pillars[i].DistanceFromEnd);

74.

75. maxDeflection = 5 \* w \* Math.Pow(sectionLength, 4) / (384 \* bridge.BridgeBeam.BeamMaterial.YoungsModulus \* areaMomentOfInertia);

76.

77. if (maxDeflection > sectionLength / 360)

78. {

79. return false;

80. }

81. }

82.

83. return true;

84. }

85.

86. }

87. }

88.

|  |  |
| --- | --- |
| Line no. | Interesting bit |
| 23, 46 & 71 | For loop from first element to 2nd last element, this is because I am considering 2 consecutive elements at a time and so cannot consider the last element on its own. |
| 16 - 84 | These 3 functions are the tests for the beam bridge. They have a similar structure wherein they loop through the pillars, considering 2 at a time (so called sections) and find the maximum and threshold or failure values for different types of stress. These 2 values are then compared (considering a factor of safety) to yield a result. |

## MODELS

These are classes that define templates for objects held in a database.

### Material.cs

1. using System;

2. using System.Collections.Generic;

3. using System.ComponentModel.DataAnnotations;

4. using System.Linq;

5. using System.Threading.Tasks;

6.

7. namespace BridgeAnalysisWebApplication.Models

8. {

9. public class Material

10. {

11. public string Name { get; set; }

12. public double ModulusOfRupture { get; set; }

13. public double ShearStrength { get; set; }

14. public double YoungsModulus { get; set; }

15.

16. public Material(string name, double modulusOfRupture, double shearStrength, double youngsModulus)

17. {

18. this.Name = name;

19. this.ModulusOfRupture = modulusOfRupture;

20. this.ShearStrength = shearStrength;

21. this.YoungsModulus = youngsModulus;

22. }

23.

24. public Material()

25. {

26.

27. }

28. }

29. }

30.

|  |  |
| --- | --- |
| Line no. | Interesting bit |
| 16 | Constructor to make sure whenever a new Material object is created, it always contains all the data required. |
| 24 | Overloaded constructor to allow creation of a Material object without initially specifying the data. This is used later in the AnalyzeBridge.cshtml.cs file at line 49. |

### Load.cs

1. using System;

2. using System.Collections.Generic;

3. using System.ComponentModel.DataAnnotations;

4. using System.Linq;

5. using System.Threading.Tasks;

6.

7. namespace BridgeAnalysisWebApplication.Models

8. {

9. public class Load

10. {

11. public string Name { get; set; }

12. public double Magnitude { get; set; }

13. public double Length { get; set; }

14.

15. public Load(string name, double magnitude, double length)

16. {

17. this.Name = name;

18. this.Magnitude = magnitude;

19. this.Length = length;

20. }

21.

22. public Load()

23. {

24.

25. }

26. }

27. }

28.

|  |  |
| --- | --- |
| Line no. | Interesting bit |
| 15 | Constructor to allow creation of a new Load object to be simpler |
| 22 | Overloaded constructor to allow creation of a Load object without initially specifying the data. This is used later in the AnalyzeBridge.cshtml.cs file at line 50. |

## PAGES

These are the razor page files that define and make up the webpages for the application. They mostly occur in pairs of a .cshtml and a .cshtml.cs script file.

The .cshtml files contain HTML that defines the webpage. Within the HTML however, it can contain parts of appropriately marked C# code to add logic for certain HTML tags and elements. It also allows to add references to data structures or algorithms that may be present in the accompanying .cshtml.cs file.

The .cshtml.cs files contain C# code that accompanying a .cshtml web page allows to add logic and data structures towards the appearance of that webpage. These are inherently C# class files that can specify the course of action when specific buttons are pressed or when the page is loaded.

### \_Layout.cshtml

This is an auto generated file that specifies shared HTML elements between all the pages, most notably the links at the top that allow to navigate between pages. The parts that I have added and changed are marked in the adjoining table. This file does not have an accompanying .cshtml.cs file with it.

1. <!DOCTYPE html>

2. <html lang="en">

3. <head>

4. <meta charset="utf-8" />

5. <meta name="viewport" content="width=device-width, initial-scale=1.0" />

6. <title>@ViewData["Title"] - BridgeAnalysisWebApplication</title>

7. <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />

8. <link rel="stylesheet" href="~/css/site.css" />

9. </head>

10. <body>

11. <header>

12. <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">

13. <div class="container">

14. <a class="navbar-brand" asp-area="" asp-page="/Index">BridgeAnalysisWebApplication</a>

15. <button class="navbar-toggler" type="button" data-toggle="collapse" data-target=".navbar-collapse" aria-controls="navbarSupportedContent"

16. aria-expanded="false" aria-label="Toggle navigation">

17. <span class="navbar-toggler-icon"></span>

18. </button>

19. <div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">

20. <ul class="navbar-nav flex-grow-1">

21. <li class="nav-item">

22. <a class="nav-link text-dark" asp-area="" asp-page="/Index">Home</a>

23. </li>

24. <li class="nav-item">

25. <a class="nav-link text-dark" asp-area="" asp-page="/AnalyzeBridge">Analyze Bridge</a>

26. </li>

27. <li class="nav-item">

28. <a class="nav-link text-dark" asp-area="" asp-page="/ViewMaterials">View Materials</a>

29. </li>

30. <li class="nav-item">

31. <a class="nav-link text-dark" asp-area="" asp-page="/ViewLoads">View Loads</a>

32. </li>

33. </ul>

34. </div>

35. </div>

36. </nav>

37. </header>

38. <div class="container">

39. <main role="main" class="pb-3">

40. @RenderBody()

41. </main>

42. </div>

43.

44. <footer class="border-top footer text-muted">

45. <div class="container">

46. &copy; 2021 - BridgeAnalysisWebApplication

47. </div>

48. </footer>

49.

50. <script src="~/lib/jquery/dist/jquery.min.js"></script>

51. <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>

52. <script src="~/js/site.js" asp-append-version="true"></script>

53.

54. @RenderSection("Scripts", required: false)

55. </body>

56. </html>

57.

|  |  |
| --- | --- |
| Line no. | Interesting bit |
| 21 - 32 | This adds a list of links to different pages in the web application that can be accessed from a bar at the top. |

### AnalyzeBridge.cshtml

1. @page

2. @model BridgeAnalysisWebApplication.Pages.AnalyzeBridgeModel

3. @{ ViewData["Title"] = "Analyze Bridge"; }

4.

5. <div class="text-center">

6. <h1 class="display-4">Analyze Bridge</h1>

7. <p>Please enter the following inputs required:</p>

8. </div>

9.

10. <form method="post">

11. Beam Details (please enter in SI units):<br><br>

12. <div>Enter Length: <br><input type="text" name="beamLength"></div><br>

13. <div>Enter Width: <br><input type="text" name="beamWidth"></div><br>

14. <div>Enter Height: <br><input type="text" name="beamHeight"></div><br>

15. <div>Enter Weight: <br><input type="text" name="beamWeight"></div><br>

16. <div>Enter Number of Pillars for the Bridge (including ones at the ends): <input type="number" name="numOfPillars"></div><br>

17.

18. @for (int i = 0; i < Model.numOfPillars; i++)

19. {

20. <div>Enter Distance from End for this Pillar: <input type="text" name=@("pillarDistance" + i.ToString())></div><br>

21. }

22.

23. <div>

24. Enter Name of Material from list of available materials: <select class="form-control" name="beamMaterialName">

25. @for (int i = 0; i < Model.materials.Count; i++)

26. {

27. <option>@Model.materials.ToArray()[i].Name</option>}

28. </select>

29. </div><br>

30.

31. <div>

32. Enter Type of Load on the Bridge from list of available types: <select class="form-control" name="loadTypeName">

33. @for (int i = 0; i < Model.loads.Count; i++)

34. {

35. <option>@Model.loads.ToArray()[i].Name</option>}

36. </select>

37. </div><br>

38.

39. <div>Enter Factor of Safety for the Bridge (default is @Model.defaultFactorOfSafety): <input type="text" name="factorOfSafety"></div><br>

40.

41. <div><input type="submit"></div>

42. </form>

43.

44. <br>

45. <br>

46. @Model.message

|  |  |
| --- | --- |
| Line no. | Interesting bit |
| 10 - 42 | This is a form tag element where the web page shows a form which can be filled in appropriately with the bridge variables and also have a button which when pressed, executes the bridge analysis algorithm. |
| 18 | For loop to loop through the number of pillars the user has specified and have a form element to input that pillar’s distance from the end value for each pillar. |
| 25 & 33 | Have a dropdown box containing the available Material and Load objects that can be selected and used for the bridge analysis algorithm. |
| 46 | This is to display an appropriate message at the bottom containing the result or a validation message etc. depending on logic in the accompanying .cshtml.cs file. |

### AnalyzeBridge.cshtml.cs

1. using System;

2. using System.Collections.Generic;

3. using System.Linq;

4. using System.Threading.Tasks;

5. using BridgeAnalysisWebApplication.Classes;

6. using BridgeAnalysisWebApplication.Models;

7. using Microsoft.AspNetCore.Mvc;

8. using Microsoft.AspNetCore.Mvc.RazorPages;

9. using Microsoft.Extensions.Caching.Memory;

10. using MySql.Data.MySqlClient;

11.

12. namespace BridgeAnalysisWebApplication.Pages

13. {

14. public class AnalyzeBridgeModel : PageModel

15. {

16. public List<Material> materials;

17. public List<Load> loads;

18. public int numOfPillars;

19.

20. public double defaultFactorOfSafety = 1.5;

21.

22. public string message;

23.

24. private IMemoryCache \_cache;

25.

26. public void OnGet() //This is called when the page is loaded hence it is used for intialization purposes

27. {

28. GetMaterialsAndLoads();

29. \_cache.Set<int>("NumOfPillars", 0);

30. }

31.

32. public void OnPost() //This is called whenever the submit button is presses hence here, the data from the forms is converted into the right data structures and then they are passed into the bridge analysis algorithm function to yield and display a result

33. {

34. GetMaterialsAndLoads();

35.

36. numOfPillars = Int32.Parse(\_cache.Get("NumOfPillars").ToString());

37.

38. if (numOfPillars != 0 && (string.IsNullOrEmpty(Request.Form["numOfPillars"]) || Int32.Parse(Request.Form["numOfPillars"]) == numOfPillars))

39. {

40. string beamLength = Request.Form["beamLength"];

41. string beamWidth = Request.Form["beamWidth"];

42. string beamHeight = Request.Form["beamHeight"];

43. string beamWeight = Request.Form["beamWeight"];

44. string beamMaterialName = Request.Form["beamMaterialName"];

45. string loadTypeName = Request.Form["loadTypeName"];

46.

47. if (ValidateValues(beamLength, beamWidth, beamHeight, beamWeight))

48. {

49. Material beamMaterial = new Material();

50. Load beamLoad = new Load();

51.

52. foreach (Material material in materials)

53. {

54. if (material.Name == beamMaterialName)

55. {

56. beamMaterial = material;

57. break;

58. }

59. }

60.

61. foreach (Load load in loads)

62. {

63. if (load.Name == loadTypeName)

64. {

65. beamLoad = load;

66. break;

67. }

68. }

69.

70. bool pillarsAreValid = (numOfPillars >= 2);

71. Pillar[] pillars = new Pillar[pillarsAreValid?numOfPillars:0];

72. string pillarDistance = "";

73.

74.

75. for (int i = 0; i < numOfPillars; i++)

76. {

77. pillarDistance = Request.Form["pillarDistance" + i.ToString()];

78.

79. if (!ValidatePillarDistanceValue(pillarDistance, beamLength))

80. {

81. pillarsAreValid = false;

82. break;

83. }

84.

85. pillars[i] = new Pillar(Double.Parse(pillarDistance), beamMaterial);

86. }

87.

88. if (pillarsAreValid)

89. {

90. Beam bridgeBeam = new Beam(Double.Parse(beamLength), Double.Parse(beamHeight), Double.Parse(beamWidth), Double.Parse(beamWeight), beamMaterial);

91.

92. double bridgeObjectLoadPerLength = beamLoad.Magnitude / beamLoad.Length;

93. double bridgeTotalLoadPerLength = bridgeObjectLoadPerLength + (Double.Parse(beamWeight) / Double.Parse(beamLength));

94.

95. BeamBridge beamBridge = new BeamBridge(bridgeBeam, pillars, bridgeObjectLoadPerLength, bridgeTotalLoadPerLength);

96.

97. beamBridge.Pillars = beamBridge.ArrangePillarsInOrder();

98.

99. double factorOfSafety = defaultFactorOfSafety;

100.

101. if (Request.Form["factorOfSafety"] != "")

102. {

103. factorOfSafety = Double.Parse(Request.Form["factorOfSafety"]);

104. }

105.

106. BridgeAnalyzer analyzer = new BridgeAnalyzer();

107.

108. if (analyzer.AnalyzeBridge(beamBridge, factorOfSafety))

109. {

110. message = "The Bridge has passed the necessary tests.";

111. }

112. else

113. {

114. message = "The Bridge has failed the necessary tests.";

115. }

116. }

117. else

118. {

119. message = "Please ensure that the pillar distance values are non negative and that there are 2 or more pillars.";

120. }

121. }

122. else

123. {

124. message = "Please enter valid values.";

125. }

126. }

127. else

128. {

129. if ((Request.Form["numOfPillars"] != "")) { numOfPillars = Int32.Parse(Request.Form["numOfPillars"]); }

130. }

131.

132. \_cache.Set<int>("NumOfPillars", numOfPillars);

133. }

134.

135. public bool ValidatePillarDistanceValue(string pillarDistance, string beamLength) //Function to validate pillar distance from end values as doubles and non negatives

136. {

137. return (Double.TryParse(pillarDistance, out double distance) && distance >= 0 && distance <= Double.Parse(beamLength));

138. }

139.

140. public bool ValidateValues(string beamLength, string beamWidth, string beamHeight, string beamWeight) //Function to validate beam dimensional values as doubles and non negatives

141. {

142. bool areValid = true;

143.

144. if (!Double.TryParse(beamLength, out double length) || length < 0)

145. {

146. areValid = false;

147. }

148.

149. if (!Double.TryParse(beamWidth, out double width) || width < 0)

150. {

151. areValid = false;

152. }

153.

154. if (!Double.TryParse(beamHeight, out double height) || height < 0)

155. {

156. areValid = false;

157. }

158.

159. if (!Double.TryParse(beamWeight, out double weight) || weight < 0)

160. {

161. areValid = false;

162. }

163.

164. return areValid;

165. }

166.

167. public AnalyzeBridgeModel(IMemoryCache memoryCache)

168. {

169. \_cache = memoryCache;

170. }

171.

172. public void GetMaterialsAndLoads() //Function to get Material and Load data from database

173. {

174. materials = new List<Material>();

175. loads = new List<Load>();

176.

177. string cs = "Server=localhost;Database=bridgeanalysisdb;Uid=root;Pwd=;";

178.

179. using var con = new MySqlConnection(cs);

180. con.Open();

181.

182. var sqlMats = "SELECT \* FROM materials";

183.

184. using var cmdMats = new MySqlCommand(sqlMats, con);

185.

186. using MySqlDataReader rdrMats = cmdMats.ExecuteReader();

187.

188. while (rdrMats.Read())

189. {

190. materials.Add(new Material(rdrMats.GetString(0), rdrMats.GetDouble(1), rdrMats.GetDouble(2), rdrMats.GetDouble(3)));

191. }

192.

193. rdrMats.Close();

194.

195. var sqlLoads = "SELECT \* FROM loads";

196.

197. using var cmdLoads = new MySqlCommand(sqlLoads, con);

198.

199. using MySqlDataReader rdrLoads = cmdLoads.ExecuteReader();

200.

201. while (rdrLoads.Read())

202. {

203. loads.Add(new Load(rdrLoads.GetString(0), rdrLoads.GetDouble(1), rdrLoads.GetDouble(2)));

204. }

205.

206. rdrLoads.Close();

207. }

208. }

209. }

210.

|  |  |
| --- | --- |
| Line no. | Interesting bit |
| 24 | Declaration of a cache object. This is used persist values or data across different calls of the function OnPost(). The function OnPost() ordinarily wipes all of the data stored in the variables. |
| 26 - 30 | This is the OnGet() function which is called whenever the page is loaded in the web browser. Here, we initialize the List objects containing the Material and Load objects by calling GetMaterialsAndLoads() that will get this data from the database. Also, we add the numOfPillars variable value to the cache object. This data will need to persist between submit button presses so as to not lose the number of pillars the user has specified, which in turn specifies elements in the form (the number of form elements to specify distance from end values for each pillar) |
| 36 | This takes the value of the numOfPillars held in cache and puts it in the variable to use |
| 38 | This is to make sure that the bridge analysis algorithm is only onset when the number of pillars is more than 0 and the user has no changed the value of the number of pillars. |
| 40 - 45 | Get values from form and store them |
| 47 | Make sure that the values are valid by calling the appropriate function. Else make sure the appropriate message to user is to ensure values are valid |
| 52 & 61 | Loop through the Material and Load objects available and match them using their Name with the one specified by the user in the form. In this way, create the Material and Load objects to be used in the algorithm |
| 70 - 71 | Check if numOfPillars is 2 or above. This is because in the algorithm, we cannot have a section with less than 2 pillars. This particular validation is done here instead of the start so as to have a validation error message for this case. Also the array of pillars is created according to this using the ternary operator. |
| 90 - 116 | Here, the bridge analysis required objects are dynamically created and passed into the algorithm to yield a result which can then be displayed to the user. |
| 132 | Use cache object to persist the numOfPillars value. |
| 167 & 169 | Constructor which initializes the cache object for use. |
| 177 | Connection string to connect to the database. |
| 179 & 180 | Create a new MySqlConnection and open it. |
| 182 | SQL command to select all columns from materials table from database. |
| 184 - 191 | Execute the SQL command and get the values from the table and add them as Material objects in a List object. |
| 193 | Close the reader for materials to allow for the reader for loads. |
| 195 - 206 | Similar process for loads table from database to add Load objects to a List object. |

### Index.cshtml

This is the homepage of the web application. It is called Index because of the naming convention and design of ASP .Net Core.

1. @page

2. @model IndexModel

3. @{ ViewData["Title"] = "Home page"; }

4.

5. <div class="text-center">

6. <h1 class="display-4">Welcome</h1>

7. <p>This is an application designed to help you analyze and test your bridge.</p>

8. </div>

9.

10. <br>

11. <br>

12. Go to <a asp-area="" asp-page="/AnalyzeBridge">Analyze Bridge</a> to use the application to analyze your bridge.

13. <br>

14. <br>

15. Go to <a asp-area="" asp-page="/ViewMaterials">View Materials</a> to view the available materials you can use for your bridge.

16. <br>

17. <br>

18. Go to <a asp-area="" asp-page="/ViewLoads">View Loads</a> to view the available loads you can have on your bridge.

|  |  |
| --- | --- |
| Line no. | Interesting bit |
| 12 - 18 | Links to the main webpages. |

### Index.cshtml.cs

This file was auto generated

1. using System;

2. using System.Collections.Generic;

3. using System.Linq;

4. using System.Threading.Tasks;

5. using Microsoft.AspNetCore.Mvc;

6. using Microsoft.AspNetCore.Mvc.RazorPages;

7. using Microsoft.Extensions.Logging;

8.

9. namespace BridgeAnalysisWebApplication.Pages

10. {

11. public class IndexModel : PageModel

12. {

13. private readonly ILogger<IndexModel> \_logger;

14.

15. public IndexModel(ILogger<IndexModel> logger)

16. {

17. \_logger = logger;

18. }

19.

20. public void OnGet()

21. {

22.

23. }

24. }

25. }

26.

### ViewMaterials.cshtml

1. @page

2. @model BridgeAnalysisWebApplication.Pages.ViewMaterialsModel

3. @{

4. }

5.

6. <div class="text-center">

7. <h1 class="display-4">View Materials</h1>

8. </div>

9. <br>

10. These are the materials stored that can be used for bridge analysis:

11. <br>

12. <br>

13.

14. <div class="table">

15. <table>

16. <thead>

17. <tr>

18. @foreach (string columnName in Model.materialColumnNames)

19. {

20. <th>@columnName</th>

21. }

22. </tr>

23. </thead>

24.

25. <tbody>

26. @using BridgeAnalysisWebApplication.Models

27. @foreach (Material material in Model.materials)

28. {

29. <tr>

30. <td>@material.Name</td>

31. <td>@material.ModulusOfRupture</td>

32. <td>@material.ShearStrength</td>

33. <td>@material.YoungsModulus</td>

34. </tr>

35. }

36. </tbody>

37. </table>

38. </div>

39.

40. <br>

41. <br>

42. <a asp-area="" asp-page="/CreateMaterial">Create a New Material</a>

43. <br>

44. <a asp-area="" asp-page="/RemoveMaterial">Remove a Material</a>

|  |  |
| --- | --- |
| Line no. | Interesting bit |
| 15 - 37 | Table HTML tag to display a table of materials. The values are taken from the List of Material objects in the accompanying .cshtml.cs file where it gets the values from the database. |
| 42 & 44 | Links to the CreateMaterial and RemoveMaterial pages, which can only be accessed from here, as was deemed appropriate and neat. |

### ViewMaterials.cshtml.cs

1. using System;

2. using System.Collections.Generic;

3. using System.Linq;

4. using System.Threading.Tasks;

5. using BridgeAnalysisWebApplication.Models;

6. using Microsoft.AspNetCore.Mvc;

7. using Microsoft.AspNetCore.Mvc.RazorPages;

8. using MySql.Data.MySqlClient;

9.

10. namespace BridgeAnalysisWebApplication.Pages

11. {

12. public class ViewMaterialsModel : PageModel

13. {

14. public List<Material> materials;

15. public List<string> materialColumnNames;

16.

17. public void OnGet()

18. {

19.

20. materials = new List<Material>();

21. materialColumnNames = new List<string>();

22.

23. string cs = "Server=localhost;Database=bridgeanalysisdb;Uid=root;Pwd=;";

24.

25. using var con = new MySqlConnection(cs);

26. con.Open();

27.

28. var sqlMats = "SELECT \* FROM materials";

29.

30. using var cmdMats = new MySqlCommand(sqlMats, con);

31.

32. using MySqlDataReader rdrMats = cmdMats.ExecuteReader();

33.

34. materialColumnNames.Add(rdrMats.GetName(0));

35. materialColumnNames.Add(rdrMats.GetName(1));

36. materialColumnNames.Add(rdrMats.GetName(2));

37. materialColumnNames.Add(rdrMats.GetName(3));

38.

39. while (rdrMats.Read())

40. {

41. materials.Add(new Material(rdrMats.GetString(0), rdrMats.GetDouble(1), rdrMats.GetDouble(2), rdrMats.GetDouble(3)));

42. }

43.

44. rdrMats.Close();

45.

46. }

47. }

48. }

49.

|  |  |
| --- | --- |
| Line no. | Interesting bit |
| 23 - 44 | Connect to database using connection string, execute the SQL to get all columns from the materials table, add the data to a List of Material objects and close the reader. Do all this when webpage is loaded in the OnGet() function. |

### ViewLoads.cshtml

This is fairly identical to ViewMaterials.cshtml, except for the loads table and not the materials table in the database.

1. @page

2. @model BridgeAnalysisWebApplication.Pages.ViewLoadsModel

3. @{

4. }

5.

6. <div class="text-center">

7. <h1 class="display-4">View Loads</h1>

8. </div>

9. <br>

10. These are the loads stored that can be used for bridge analysis:

11. <br>

12. <br>

13.

14. <div class="table">

15. <table>

16. <thead>

17. <tr>

18. @foreach (string columnName in Model.loadColumnNames)

19. {

20. <th>@columnName</th>

21. }

22. </tr>

23. </thead>

24.

25. <tbody>

26. @using BridgeAnalysisWebApplication.Models

27. @foreach (Load load in Model.loads)

28. {

29. <tr>

30. <td>@load.Name</td>

31. <td>@load.Magnitude</td>

32. <td>@load.Length</td>

33. </tr>

34. }

35. </tbody>

36. </table>

37. </div>

38.

39. <br>

40. <br>

41. <a asp-area="" asp-page="/CreateLoad">Create a New Load</a>

42. <br>

43. <a asp-area="" asp-page="/RemoveLoad">Remove a Load</a>

### ViewLoads.cshtml.cs

This is fairly identical to ViewMaterials.cshtml.cs, except for the loads table and not the materials table in the database.

1. using System;

2. using System.Collections.Generic;

3. using System.Linq;

4. using System.Threading.Tasks;

5. using BridgeAnalysisWebApplication.Models;

6. using Microsoft.AspNetCore.Mvc;

7. using Microsoft.AspNetCore.Mvc.RazorPages;

8. using MySql.Data.MySqlClient;

9.

10. namespace BridgeAnalysisWebApplication.Pages

11. {

12. public class ViewLoadsModel : PageModel

13. {

14. public List<Load> loads;

15. public List<string> loadColumnNames;

16.

17. public void OnGet()

18. {

19. loads = new List<Load>();

20. loadColumnNames = new List<string>();

21.

22. string cs = "Server=localhost;Database=bridgeanalysisdb;Uid=root;Pwd=;";

23.

24. using var con = new MySqlConnection(cs);

25. con.Open();

26.

27. var sqlLoads = "SELECT \* FROM loads";

28.

29. using var cmdLoads = new MySqlCommand(sqlLoads, con);

30.

31. using MySqlDataReader rdrLoads = cmdLoads.ExecuteReader();

32.

33. loadColumnNames.Add(rdrLoads.GetName(0));

34. loadColumnNames.Add(rdrLoads.GetName(1));

35. loadColumnNames.Add(rdrLoads.GetName(2));

36.

37.

38. while (rdrLoads.Read())

39. {

40. loads.Add(new Load(rdrLoads.GetString(0), rdrLoads.GetDouble(1), rdrLoads.GetDouble(2)));

41. }

42.

43. rdrLoads.Close();

44.

45. }

46. }

47. }

48.

### CreateMaterial.cshtml

1. @page

2. @model BridgeAnalysisWebApplication.Pages.CreateMaterialModel

3. @{

4. }

5.

6. <div class="text-center">

7. <h1 class="display-4">Create Material</h1>

8. </div>

9. <br>

10. Enter the following values required in SI units to create a new material:

11. <br>

12. <br>

13.

14. <form method="post">

15. <div>Enter Name (should be unique): <br><input type="text" name="materialName"></div><br>

16. <div>Enter Modulus of Rupture (should be a positive real number): <br><input type="text" name="materialModulusOfRupture"></div><br>

17. <div>Enter Shear Strength (should be a positive real number): <br><input type="text" name="materialShearStrength"></div><br>

18. <div>Enter Young's Modulus (should be a positive real number): <br><input type="text" name="materialYoungsModulus"></div><br>

19.

20. <div><input type="submit"></div>

21. </form>

22.

23. <br>

24. @Model.message

25.

26. <br>

27. <br>

28. <a asp-area="" asp-page="/ViewMaterials">View Materials</a>

|  |  |
| --- | --- |
| Line no. | Interesting bit |
| 14 - 21 | This is a form tag element to allow putting values to create a new Material. |
| 28 | Link to the ViewMaterials webpage. |

### CreateMaterial.cshtml.cs

1. using System;

2. using System.Collections.Generic;

3. using System.Linq;

4. using System.Threading.Tasks;

5. using Microsoft.AspNetCore.Mvc;

6. using Microsoft.AspNetCore.Mvc.RazorPages;

7. using MySql.Data.MySqlClient;

8.

9. namespace BridgeAnalysisWebApplication.Pages

10. {

11. public class CreateMaterialModel : PageModel

12. {

13. public string message;

14.

15. public void OnGet()

16. {

17.

18. }

19.

20. public void OnPost()

21. {

22. string materialName = Request.Form["materialName"];

23. string materialModulusOfRupture = Request.Form["materialModulusOfRupture"];

24. string materialShearStrength = Request.Form["materialShearStrength"];

25. string materialYoungsModulus = Request.Form["materialYoungsModulus"];

26.

27. if (ValidateValues(materialName, materialModulusOfRupture, materialShearStrength, materialYoungsModulus))

28. {

29. string cs = "Server=localhost;Database=bridgeanalysisdb;Uid=root;Pwd=;";

30.

31. using var con = new MySqlConnection(cs);

32. con.Open();

33.

34. using var cmd = new MySqlCommand();

35. cmd.Connection = con;

36.

37. cmd.CommandText = $"INSERT INTO materials(Name, ModulusOfRupture, ShearStrength, YoungsModulus) VALUES('{materialName}', {materialModulusOfRupture}, {materialShearStrength}, {materialYoungsModulus})";

38. cmd.ExecuteNonQuery();

39. }

40. else

41. {

42. message = "Please enter valid values.";

43. }

44. }

45.

46. public bool ValidateValues(string materialName, string materialModulusOfRupture, string materialShearStrength, string materialYoungsModulus) //Validate values

47. {

48. bool areValid = true;

49.

50. string cs = "Server=localhost;Database=bridgeanalysisdb;Uid=root;Pwd=;";

51.

52. using var con = new MySqlConnection(cs);

53. con.Open();

54.

55. var sqlMats = "SELECT Name FROM materials";

56.

57. using var cmdMats = new MySqlCommand(sqlMats, con);

58.

59. using MySqlDataReader rdrMats = cmdMats.ExecuteReader();

60.

61. while (rdrMats.Read())

62. {

63. if (materialName == rdrMats.GetString(0))

64. {

65. areValid = false;

66. }

67. }

68.

69. rdrMats.Close();

70.

71. if(!Double.TryParse(materialModulusOfRupture, out double modulusOfRupture) || modulusOfRupture < 0)

72. {

73. areValid = false;

74. }

75.

76. if (!Double.TryParse(materialShearStrength, out double shearStrength) || shearStrength < 0)

77. {

78. areValid = false;

79. }

80.

81. if (!Double.TryParse(materialYoungsModulus, out double youngsModulus) || youngsModulus < 0)

82. {

83. areValid = false;

84. }

85.

86. return areValid;

87. }

88. }

89. }

90.

|  |  |
| --- | --- |
| Line no. | Interesting bit |
| 37 | SQL command to insert the values into the materials table. |
| 50 - 67 | Connect to materials database and check if the material to add is already a part of the table or not. This is to prevent the creation and addition of a Material that has an existing Name. |

### CreateLoad.cshtml

This is similar to CreateMaterial.cshtml.

1. @page

2. @model BridgeAnalysisWebApplication.Pages.CreateLoadModel

3. @{

4. }

5.

6. <div class="text-center">

7. <h1 class="display-4">Create Load</h1>

8. </div>

9. <br>

10. Enter the following values required in SI units to create a new Load:

11. <br>

12. <br>

13.

14. <form method="post">

15. <div>Enter Name (should be unique): <br><input type="text" name="loadName"></div><br>

16. <div>Enter Magnitude of the Load (should be a positive real number): <br><input type="text" name="loadMagnitude"></div><br>

17. <div>Enter Length of the Load (should be a positive real number): <br><input type="text" name="loadLength"></div><br>

18.

19. <div><input type="submit"></div>

20. </form>

21.

22. <br>

23. @Model.message

24.

25. <br>

26. <br>

27. <a asp-area="" asp-page="/ViewLoads">View Loads</a>

### CreateLoad.cshtml.cs

This is similar to CreateMaterial.cshtml.cs.

1. using System;

2. using System.Collections.Generic;

3. using System.Linq;

4. using System.Threading.Tasks;

5. using Microsoft.AspNetCore.Mvc;

6. using Microsoft.AspNetCore.Mvc.RazorPages;

7. using MySql.Data.MySqlClient;

8.

9. namespace BridgeAnalysisWebApplication.Pages

10. {

11. public class CreateLoadModel : PageModel

12. {

13. public string message;

14.

15. public void OnGet()

16. {

17.

18. }

19.

20. public void OnPost()

21. {

22. string loadName = Request.Form["loadName"];

23. string loadMagnitude = Request.Form["loadMagnitude"];

24. string loadLength = Request.Form["loadLength"];

25.

26. if (ValidateValues(loadName, loadMagnitude, loadLength))

27. {

28. string cs = "Server=localhost;Database=bridgeanalysisdb;Uid=root;Pwd=;";

29.

30. using var con = new MySqlConnection(cs);

31. con.Open();

32.

33. using var cmd = new MySqlCommand();

34. cmd.Connection = con;

35.

36. cmd.CommandText = $"INSERT INTO loads(Name, Magnitude, Length) VALUES('{loadName}', {loadMagnitude}, {loadLength})";

37. cmd.ExecuteNonQuery();

38. }

39. else

40. {

41. message = "Please enter valid values.";

42. }

43. }

44.

45. public bool ValidateValues(string loadName, string loadMagnitude, string loadLength) //Validate values

46. {

47. bool areValid = true;

48.

49. string cs = "Server=localhost;Database=bridgeanalysisdb;Uid=root;Pwd=;";

50.

51. using var con = new MySqlConnection(cs);

52. con.Open();

53.

54. var sqlLoads = "SELECT Name FROM loads";

55.

56. using var cmdLoads = new MySqlCommand(sqlLoads, con);

57.

58. using MySqlDataReader rdrLoads = cmdLoads.ExecuteReader();

59.

60. while (rdrLoads.Read())

61. {

62. if (loadName == rdrLoads.GetString(0))

63. {

64. areValid = false;

65. }

66. }

67.

68. rdrLoads.Close();

69.

70. if (!Double.TryParse(loadMagnitude, out double magnitude) || magnitude < 0)

71. {

72. areValid = false;

73. }

74.

75. if (!Double.TryParse(loadLength, out double length) || length < 0)

76. {

77. areValid = false;

78. }

79.

80. return areValid;

81. }

82. }

83. }

84.

### RemoveMaterial.cshtml

1. @page

2. @model BridgeAnalysisWebApplication.Pages.RemoveMaterialModel

3. @{

4. }

5.

6. <div class="text-center">

7. <h1 class="display-4">Remove Material</h1>

8. </div>

9. <br>

10. Enter the Name of the Material you wish to remove:

11. <br>

12.

13. <form method="post">

14. <div><input type="text" name="materialName"></div>

15. <br>

16. <div><input type="submit"></div>

17. </form>

18.

19. <br>

20. @Model.message

21.

22. <br>

23. <br>

24. <a asp-area="" asp-page="/ViewMaterials">View Materials</a>

### RemoveMaterial.cshtml.cs

1. using System;

2. using System.Collections.Generic;

3. using System.Linq;

4. using System.Threading.Tasks;

5. using Microsoft.AspNetCore.Mvc;

6. using Microsoft.AspNetCore.Mvc.RazorPages;

7. using MySql.Data.MySqlClient;

8.

9. namespace BridgeAnalysisWebApplication.Pages

10. {

11. public class RemoveMaterialModel : PageModel

12. {

13. public string message;

14.

15. public void OnGet()

16. {

17. }

18.

19. public void OnPost()

20. {

21. string materialName = Request.Form["materialName"];

22.

23. string cs = "Server=localhost;Database=bridgeanalysisdb;Uid=root;Pwd=;";

24.

25. using var con = new MySqlConnection(cs);

26. con.Open();

27.

28. using var cmd = new MySqlCommand();

29. cmd.Connection = con;

30.

31. cmd.CommandText = $"DELETE FROM materials WHERE Name = '{materialName}'";

32. cmd.ExecuteNonQuery();

33.

34. message = "Removed specified material.";

35. }

36. }

37. }

38.

### RemoveLoad.cshtml

1. @page

2. @model BridgeAnalysisWebApplication.Pages.RemoveLoadModel

3. @{

4. }

5.

6. <div class="text-center">

7. <h1 class="display-4">Remove Load</h1>

8. </div>

9. <br>

10. Enter the Name of the Load you wish to remove:

11. <br>

12.

13. <form method="post">

14. <div><input type="text" name="loadName"></div>

15. <br>

16. <div><input type="submit"></div>

17. </form>

18.

19. <br>

20. @Model.message

21.

22. <br>

23. <br>

24. <a asp-area="" asp-page="/ViewLoads">View Loads</a>

### RemoveLoad.cshtml.cs

1. using System;

2. using System.Collections.Generic;

3. using System.Linq;

4. using System.Threading.Tasks;

5. using Microsoft.AspNetCore.Mvc;

6. using Microsoft.AspNetCore.Mvc.RazorPages;

7. using MySql.Data.MySqlClient;

8.

9. namespace BridgeAnalysisWebApplication.Pages

10. {

11. public class RemoveLoadModel : PageModel

12. {

13. public string message;

14.

15. public void OnGet()

16. {

17. }

18.

19. public void OnPost()

20. {

21. string loadName = Request.Form["loadName"];

22.

23. string cs = "Server=localhost;Database=bridgeanalysisdb;Uid=root;Pwd=;";

24.

25. using var con = new MySqlConnection(cs);

26. con.Open();

27.

28. using var cmd = new MySqlCommand();

29. cmd.Connection = con;

30.

31. cmd.CommandText = $"DELETE FROM loads WHERE Name = '{loadName}'";

32. cmd.ExecuteNonQuery();

33.

34. message = "Removed specified load.";

35. }

36. }

37. }

38.

# TESTING

The testing for this web application is done in 2 sections:

* Unit Testing – to test the algorithmic parts individually and make sure they work correctly
* Final Testing – to test the final solution as a whole and check that specific parts of it work

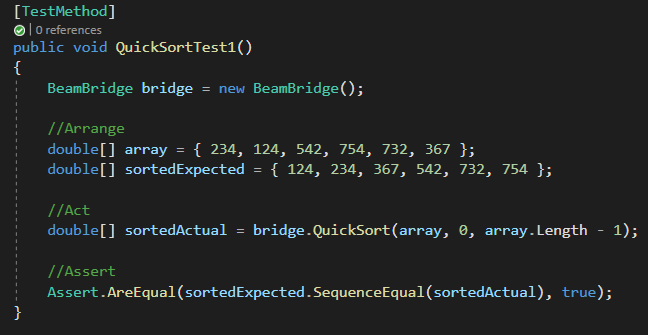
## UNIT TESTING

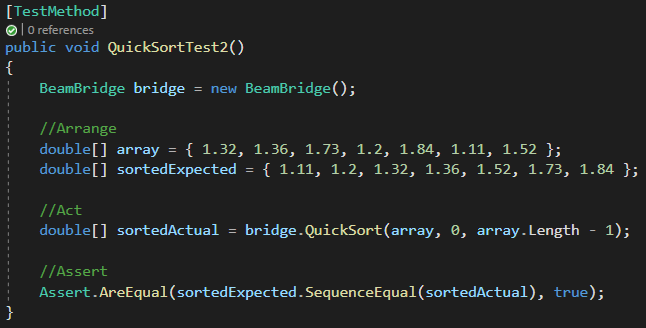
Unit testing is done on the algorithmic functions of the solution. Thus, the following functions from the following classes have been tested in this manner.

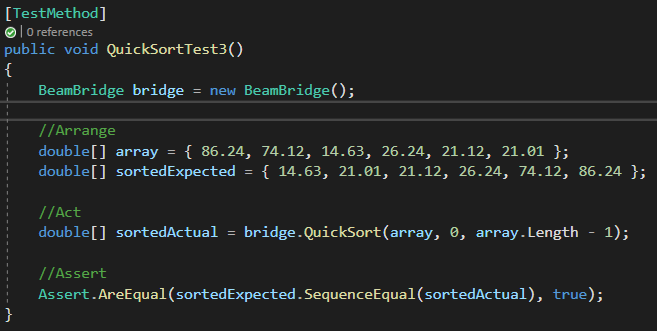
### BeamBridge.cs

#### QuickSort Testing – Unit Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test no. | Input Data | Expected Output | Actual Output | RESULT |
| 1 | { 234, 124, 542, 754, 732, 367 } | { 124, 234, 367, 542, 732, 754 } | { 124, 234, 367, 542, 732, 754 }  When test was passed | Before: FAIL  After: PASS |
| 2 | { 1.32, 1.36, 1.73, 1.2, 1.84, 1.11, 1.52 } | { 1.11, 1.2, 1.32, 1.36, 1.52, 1.73, 1.84 } | { 1.11, 1.2, 1.32, 1.36, 1.52, 1.73, 1.84 }  When test was passed | Before: FAIL  After: PASS |
| 3 | { 86.24, 74.12, 14.63, 26.24, 21.12, 21.01 } | { 14.63, 21.01, 21.12, 26.24, 74.12, 86.24 } | { 14.63, 21.01, 21.12, 26.24, 74.12, 86.24 }  When test was passed | Before: FAIL  After: PASS |

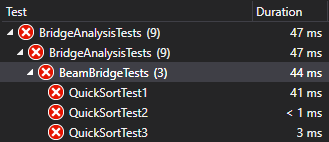




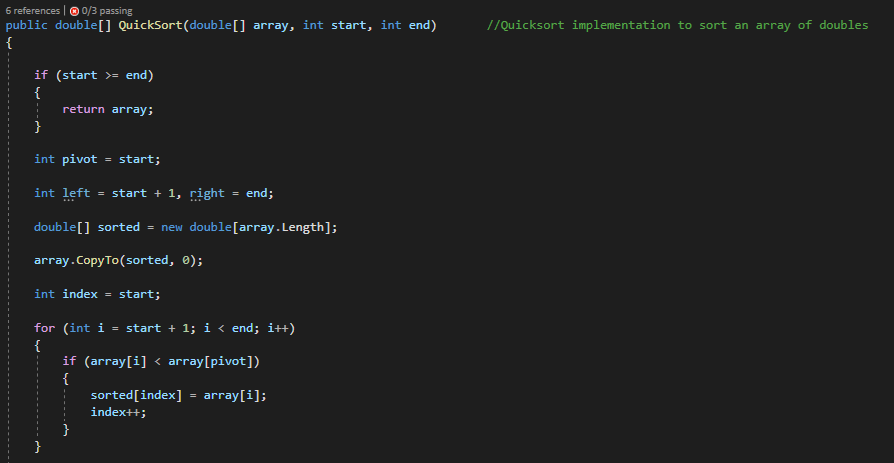


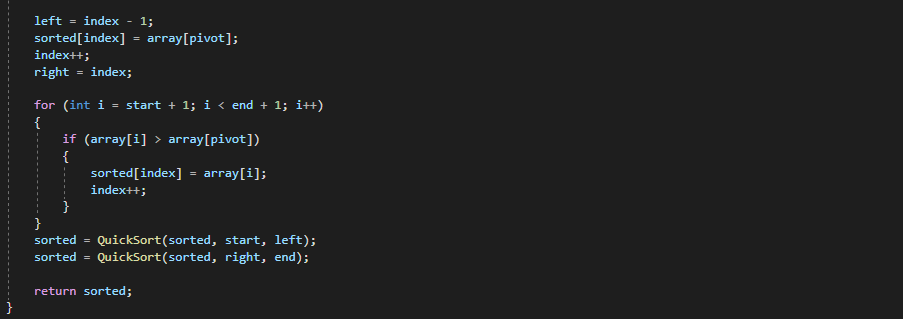
##### BEFORE:

Test Result:



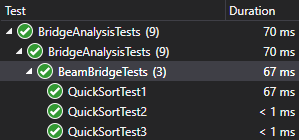
QuickSort Function:



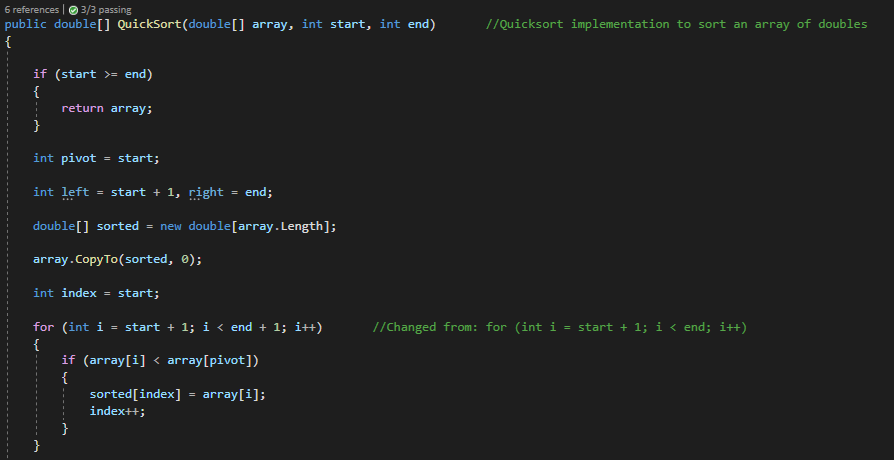


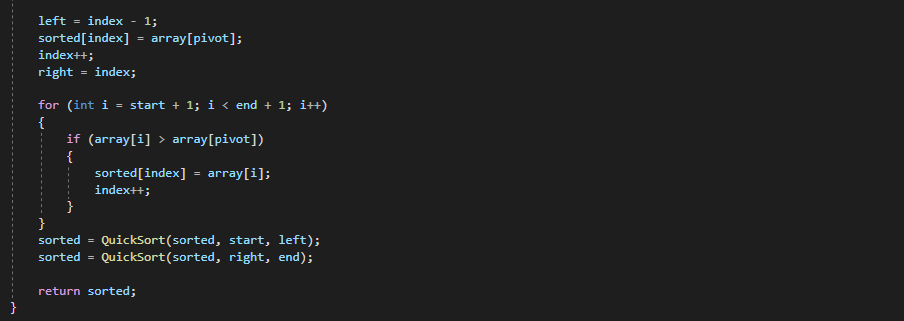
##### AFTER:

Test Result:



New QuickSort Function:

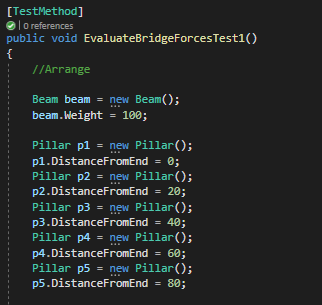


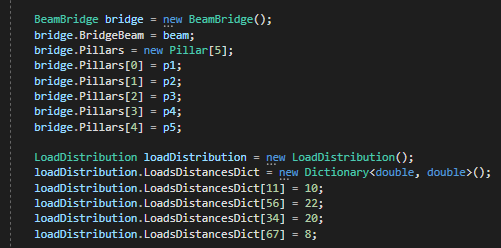


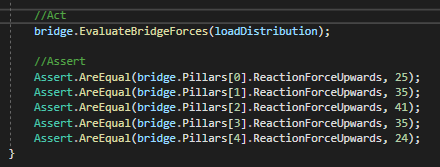
#### EvaluateBridgeForces Testing – Unit Tests

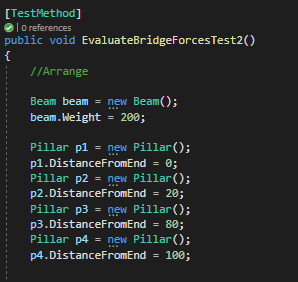
Note that this function was removed from the final solution as it became redundant.

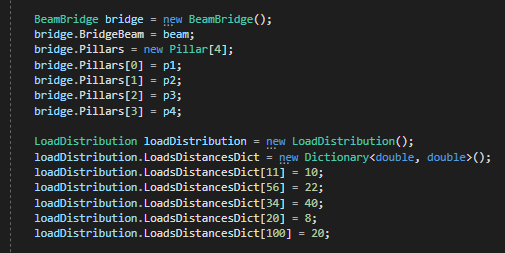
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test no. | Input Data | Expected Output | Actual Output | RESULT |
| 1 | See screenshot below | See screenshot below | See screenshot below | PASS |
| 2 | See screenshot below | See screenshot below | See screenshot below | PASS |

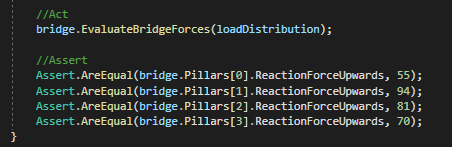


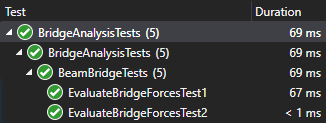








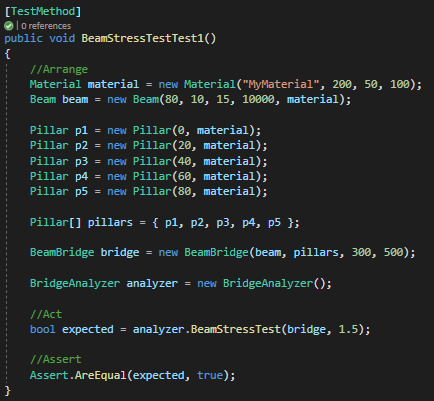


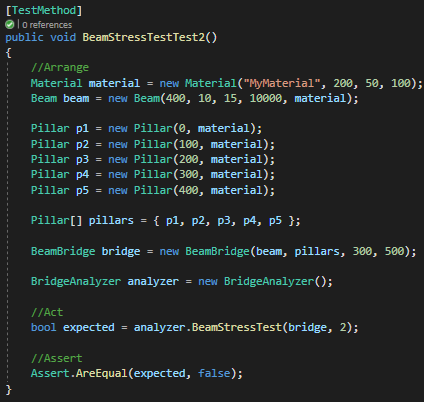


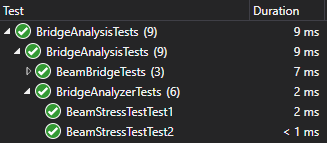
### BridgeAnalyzer.cs

#### BeamStressTest Testing – Unit Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test no. | Input Data | Expected Output | Actual Output | RESULT |
| 1 | See screenshot below | true | true | PASS |
| 2 | See screenshot below | false | false | PASS |

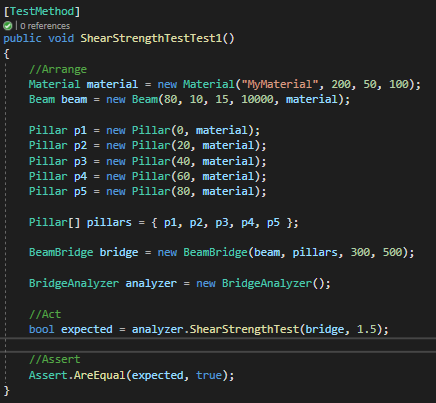


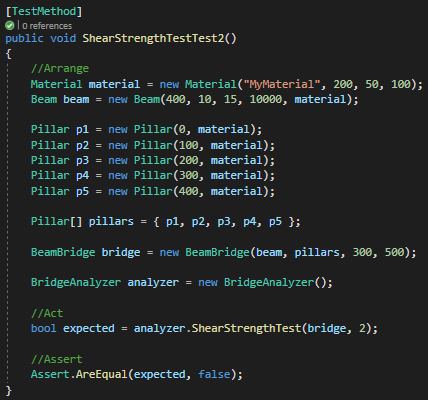


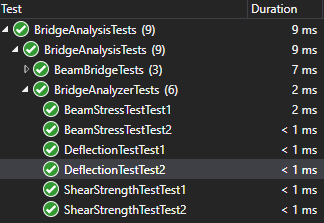


#### ShearStrengthTest Testing – Unit Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test no. | Input Data | Expected Output | Actual Output | RESULT |
| 1 | See screenshot below | true | true | PASS |
| 2 | See screenshot below | false | false | PASS |

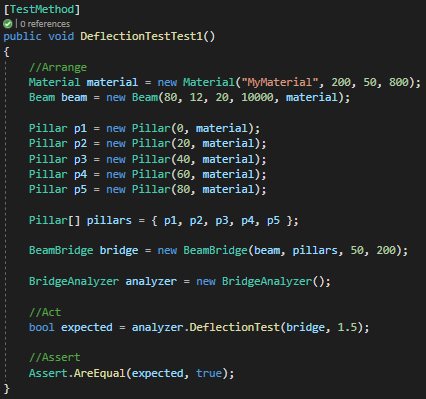


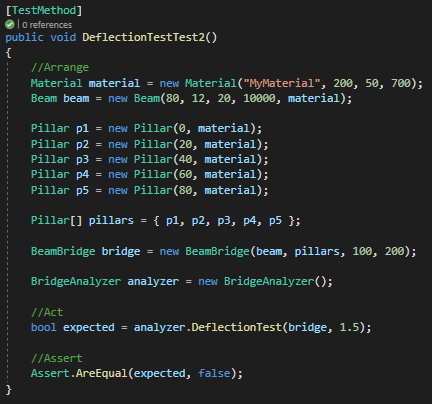


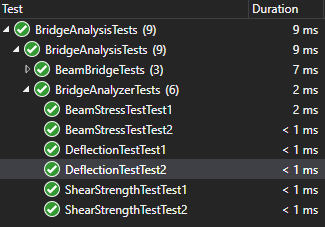


#### DeflectionTest Testing – Unit Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test no. | Input Data | Expected Output | Actual Output | RESULT |
| 1 | See screenshot below | true | true | PASS |
| 2 | See screenshot below | false | false | PASS |





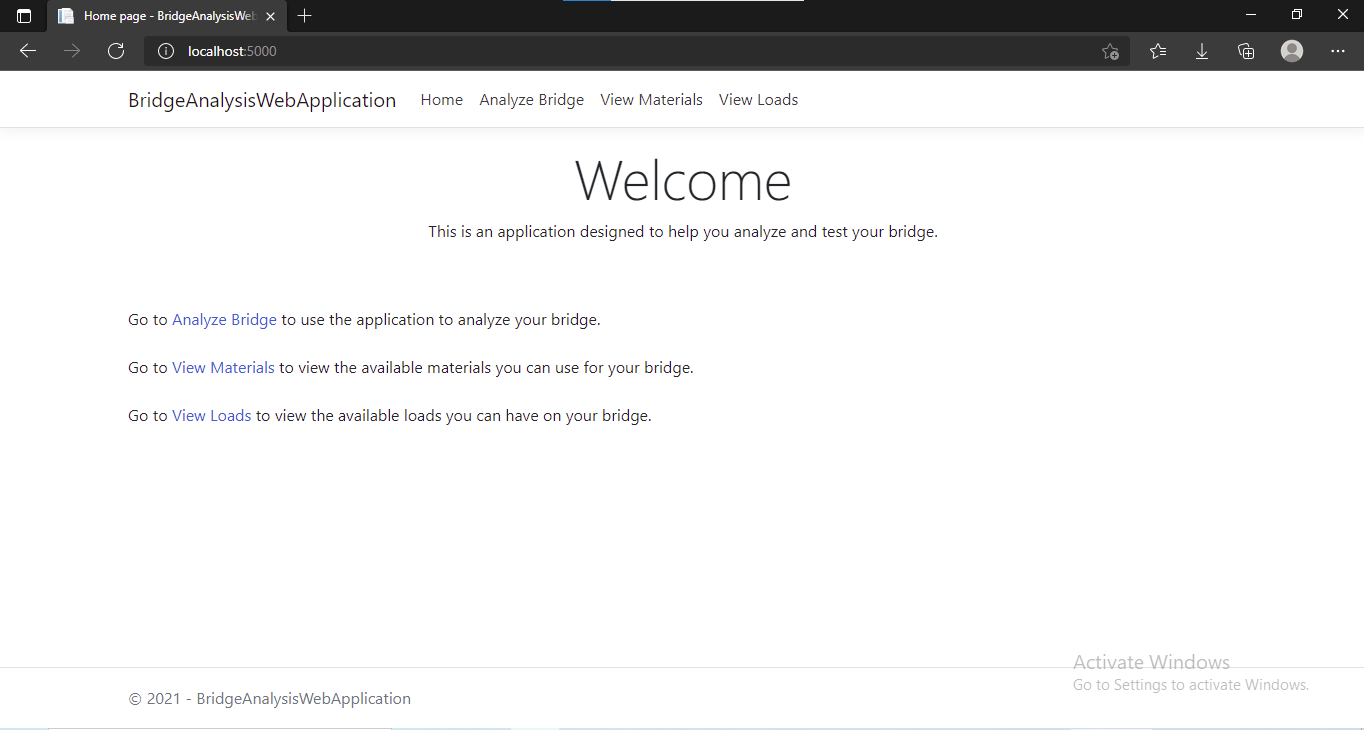


## FINAL SOLUTION TESTING

The following functionalities of the application have been tested, organized based on their webpages

### Homepage (Index)

#### Navigate to other webpages using links

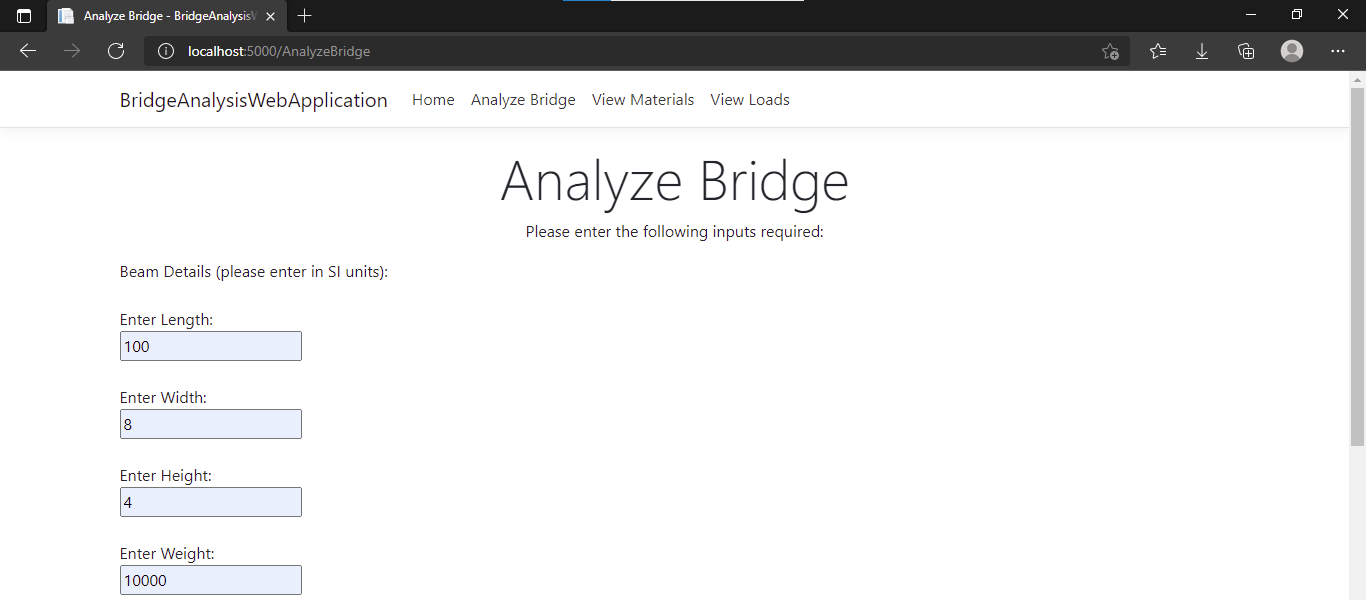


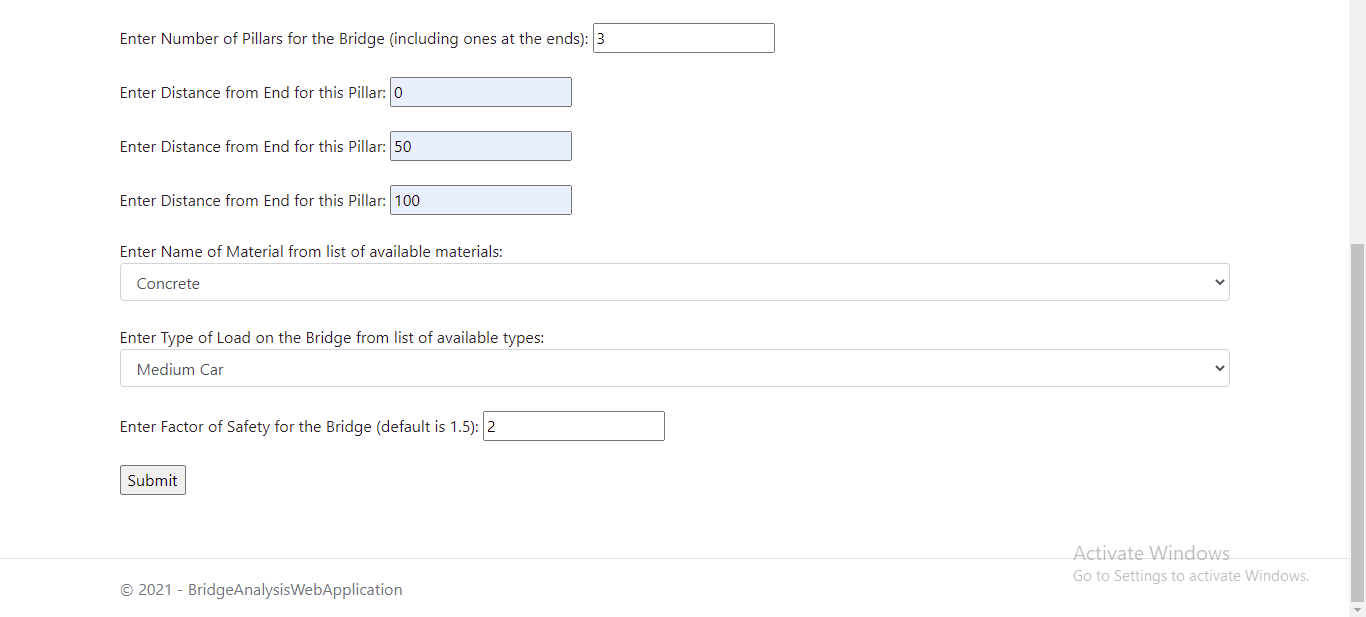
Thus, this is the homepage, where using the given links, one can navigate to the other webpages.

### AnalyzeBridge

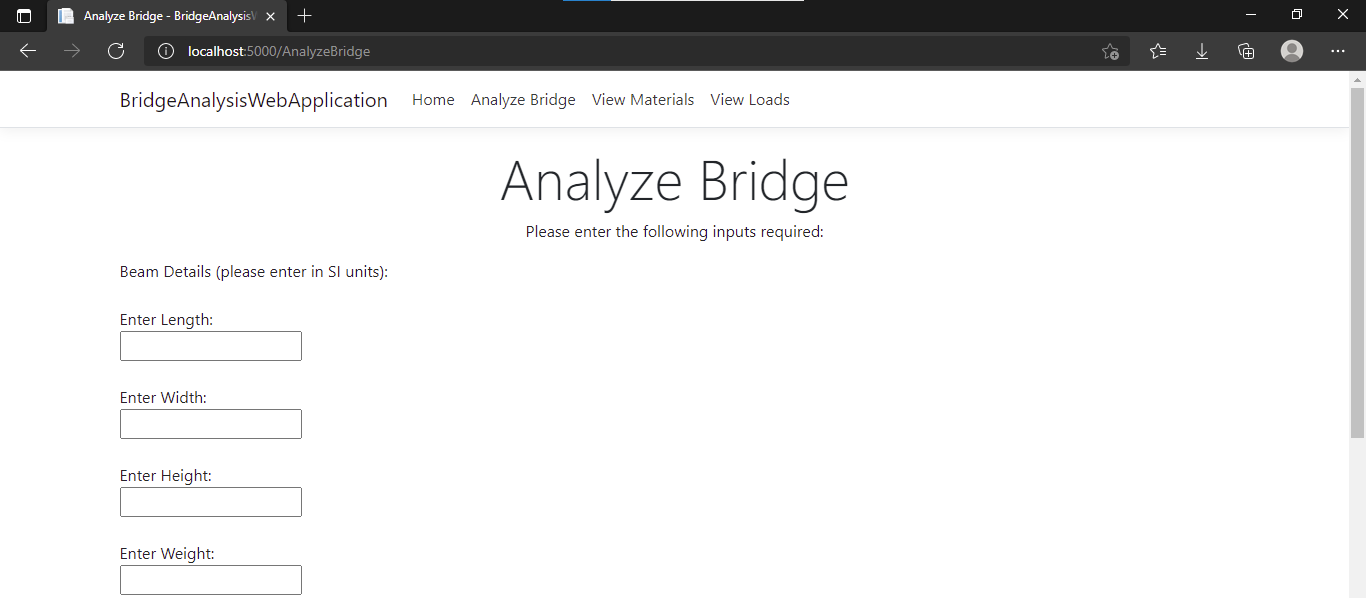
#### Fill values in and receive a result of bridge analysis

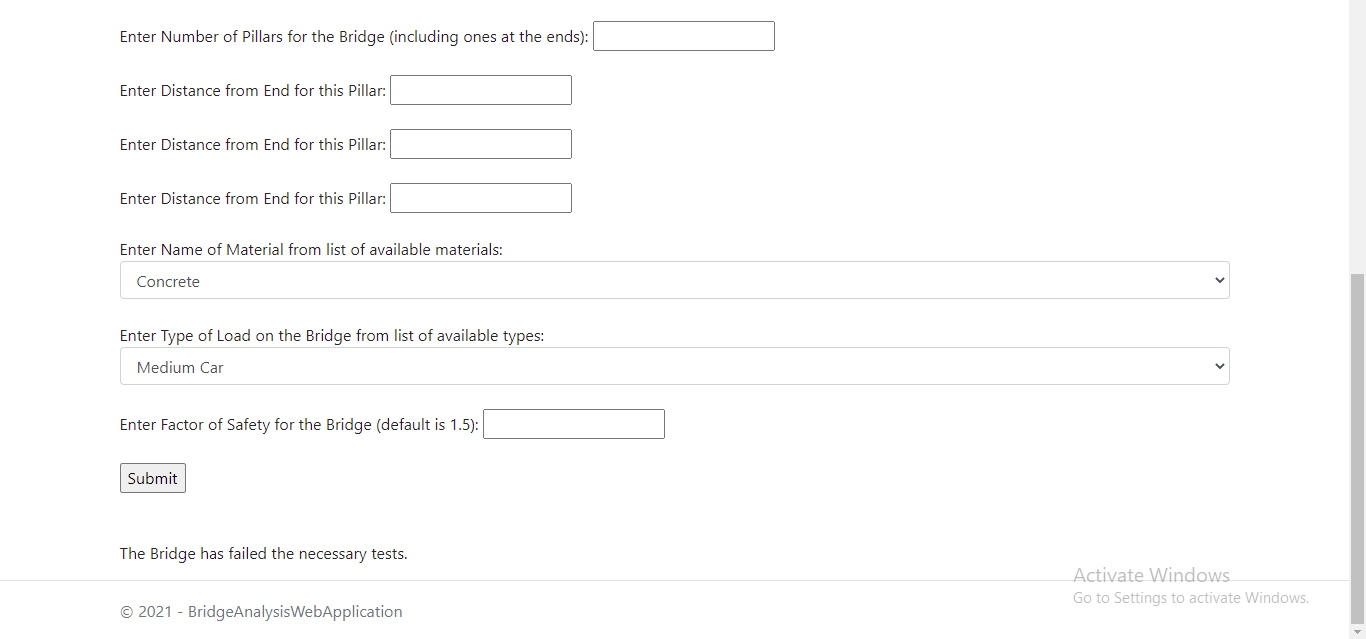
Input values:





Output after pressing the Submit button:

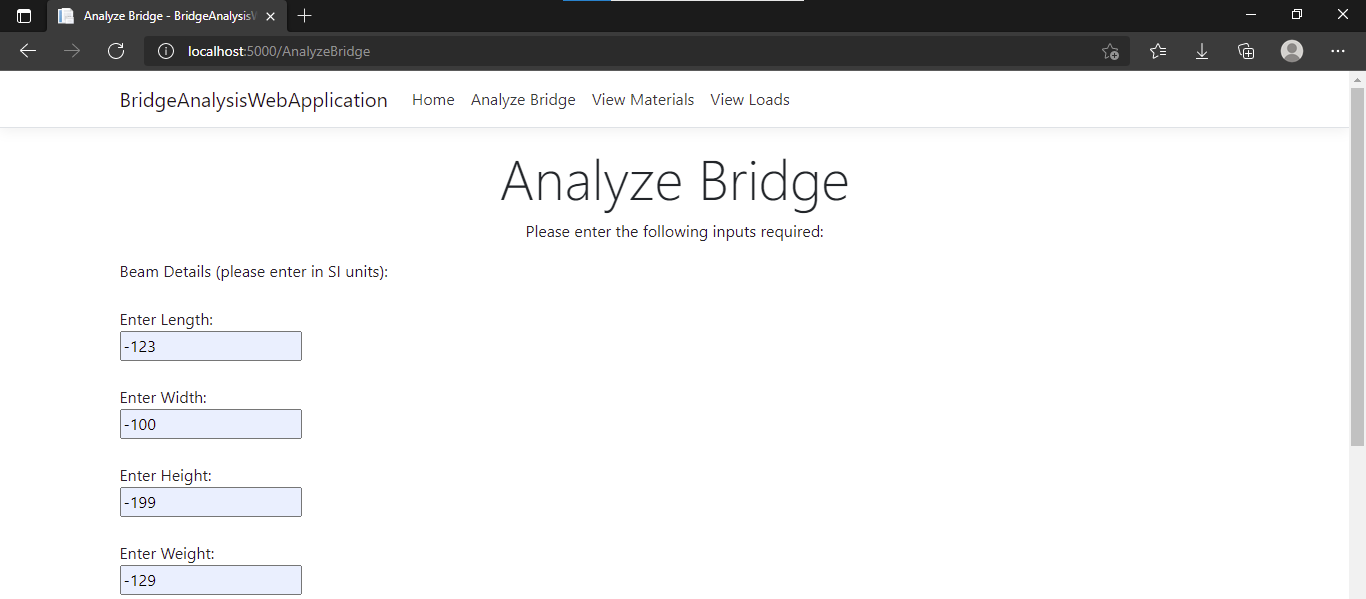


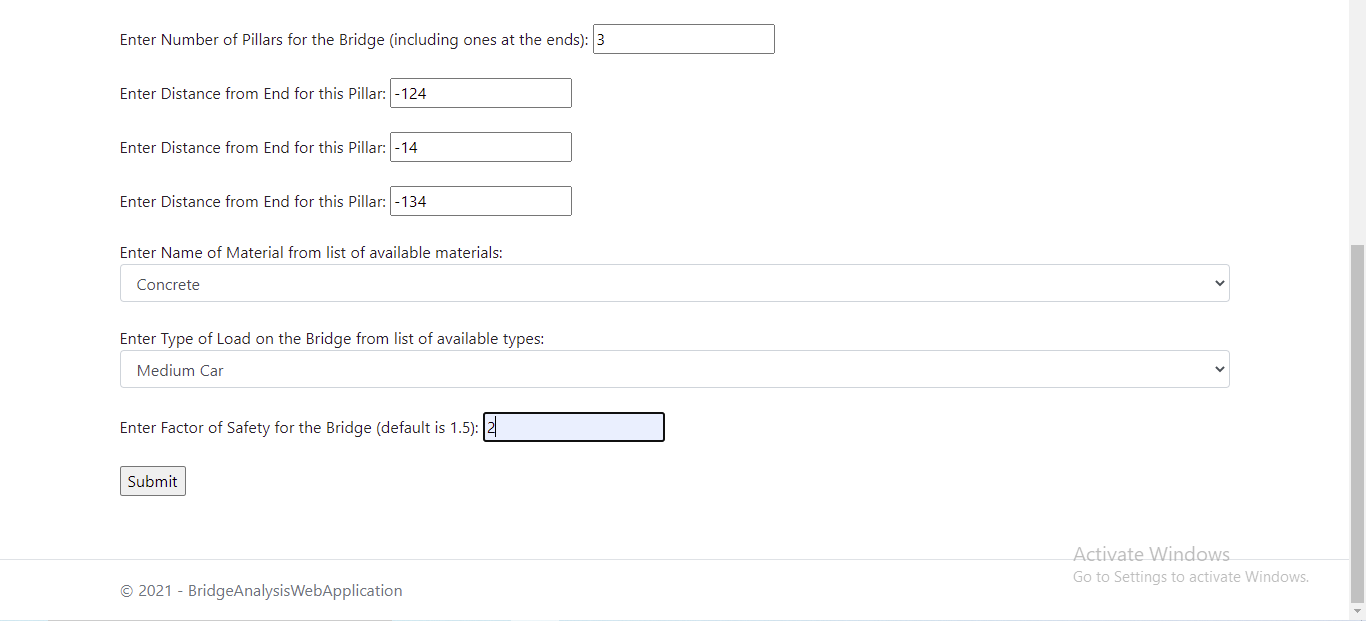


Thus, as can be seen from the message at the bottom of the page, this functionality works.

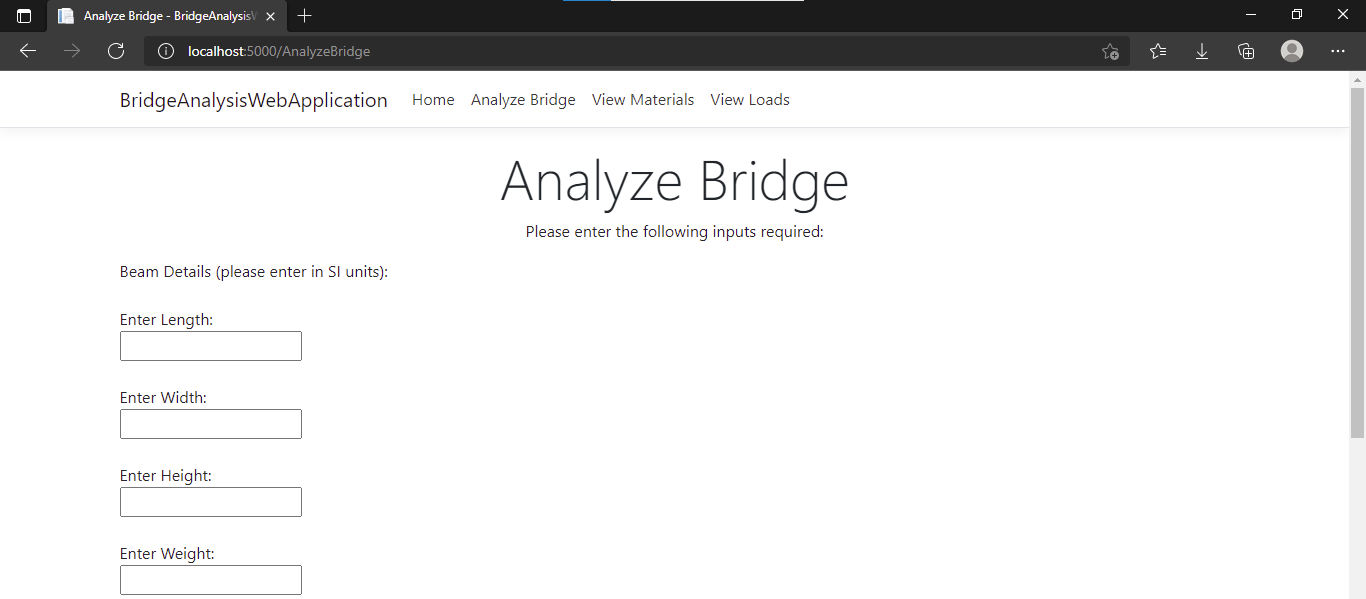
#### The page sends a message when invalid values are input

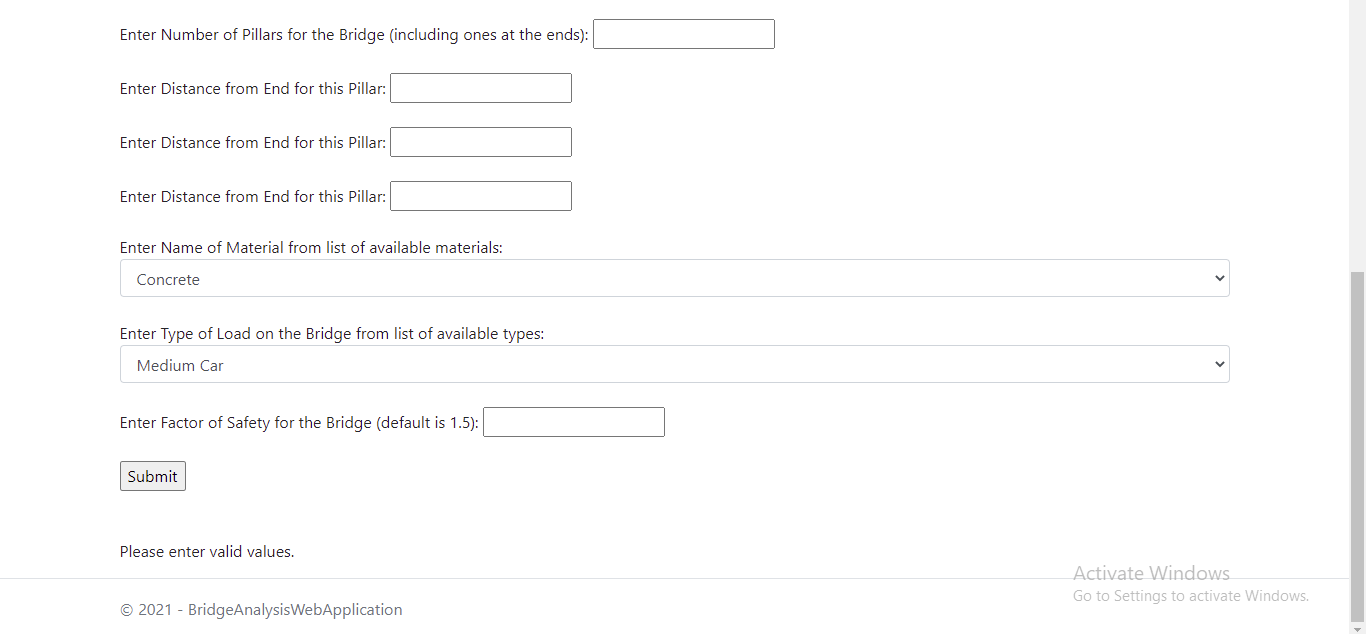
Input values:





Output after pressing the Submit button:

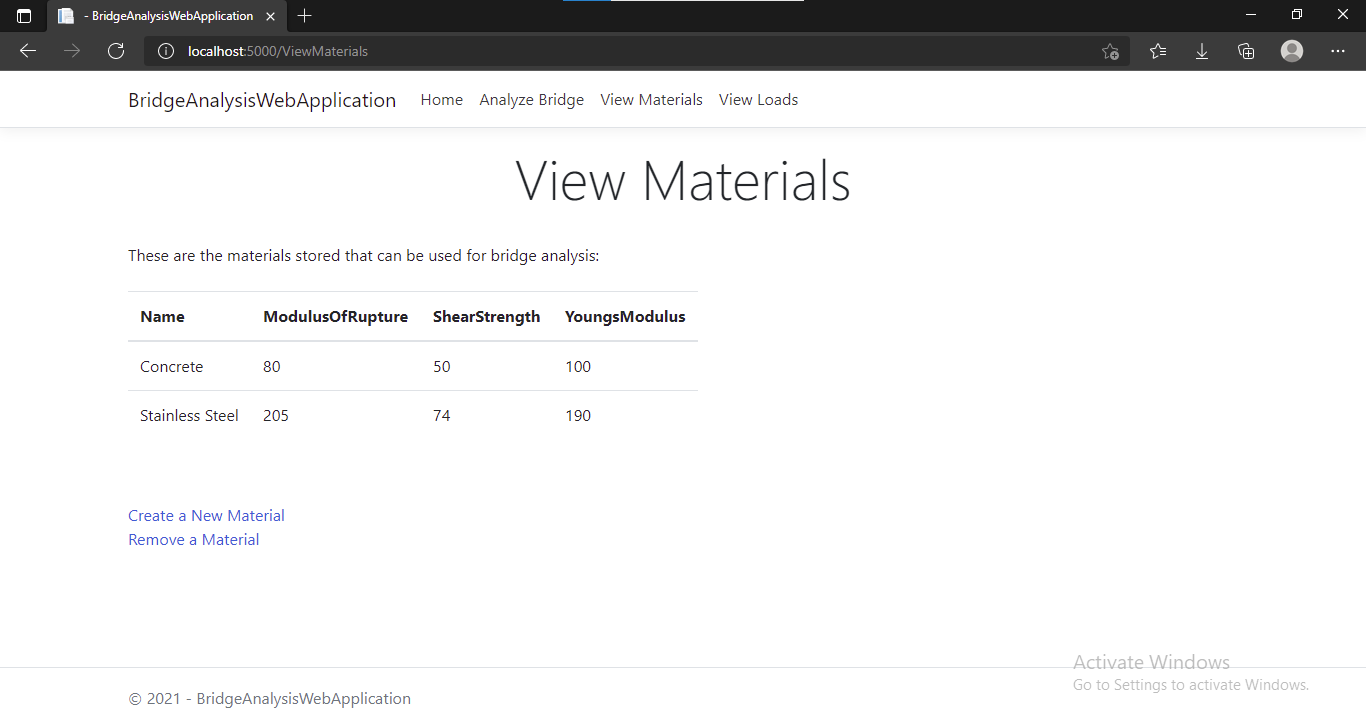


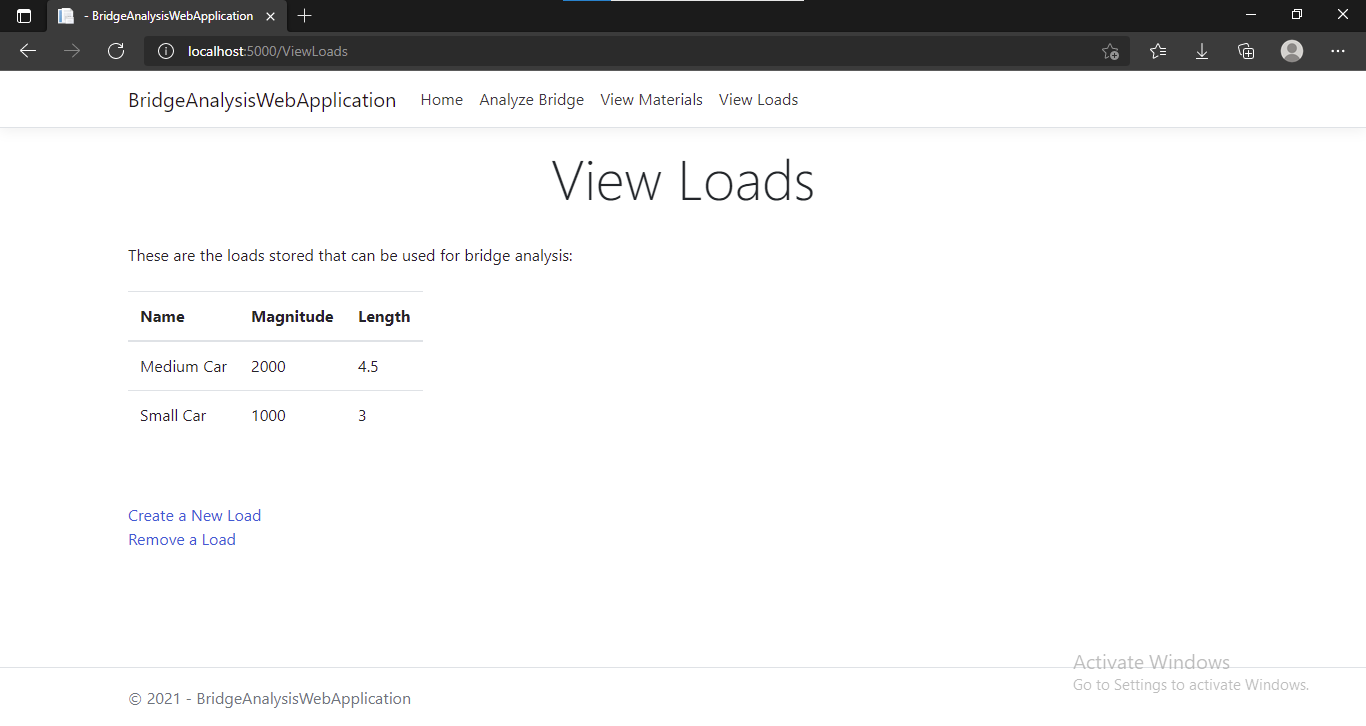


Thus, as can be seen from the message at the bottom, this functionality works.

### ViewMaterials and ViewLoads

#### Webpage shows the contents of the tables from database as a table



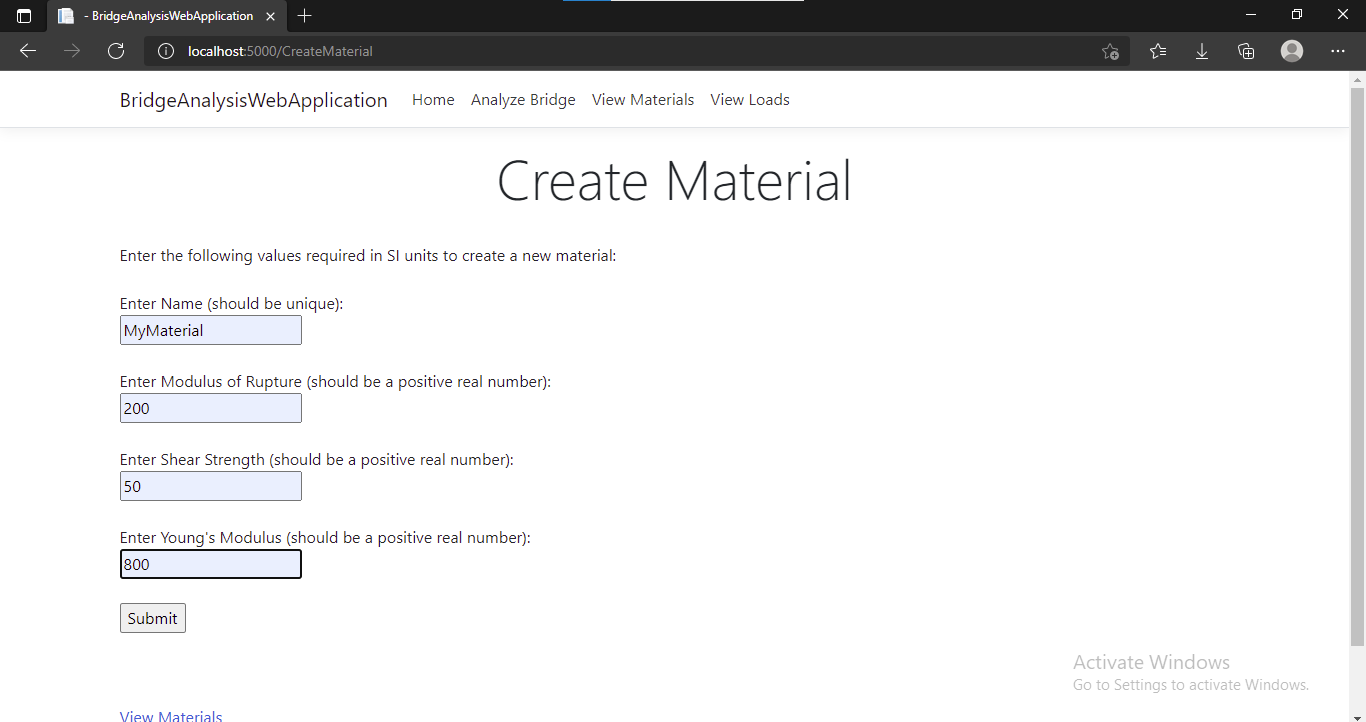


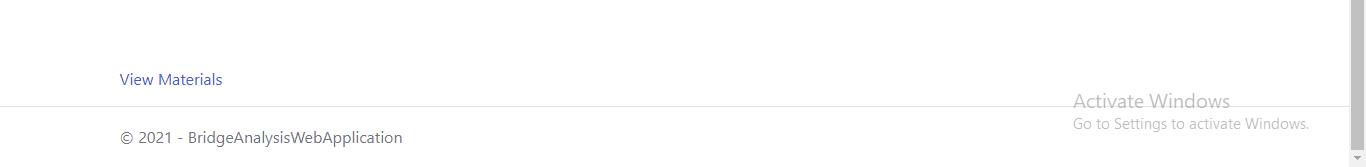
Thus, these webpages show the contents of the tables in the database as a table.

### CreateMaterial

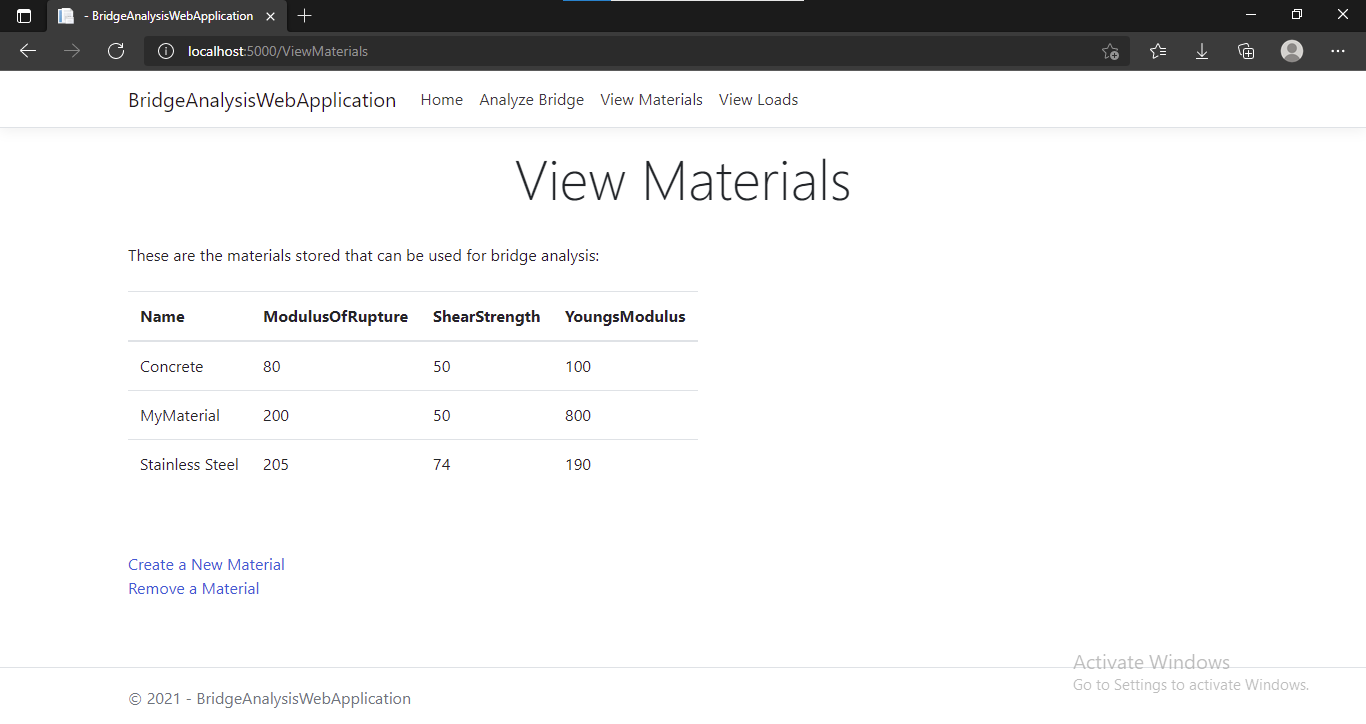
#### Create a new material and find it updated on the ViewMaterials webpage.

Input values:





ViewMaterials page after pressing Submit button:

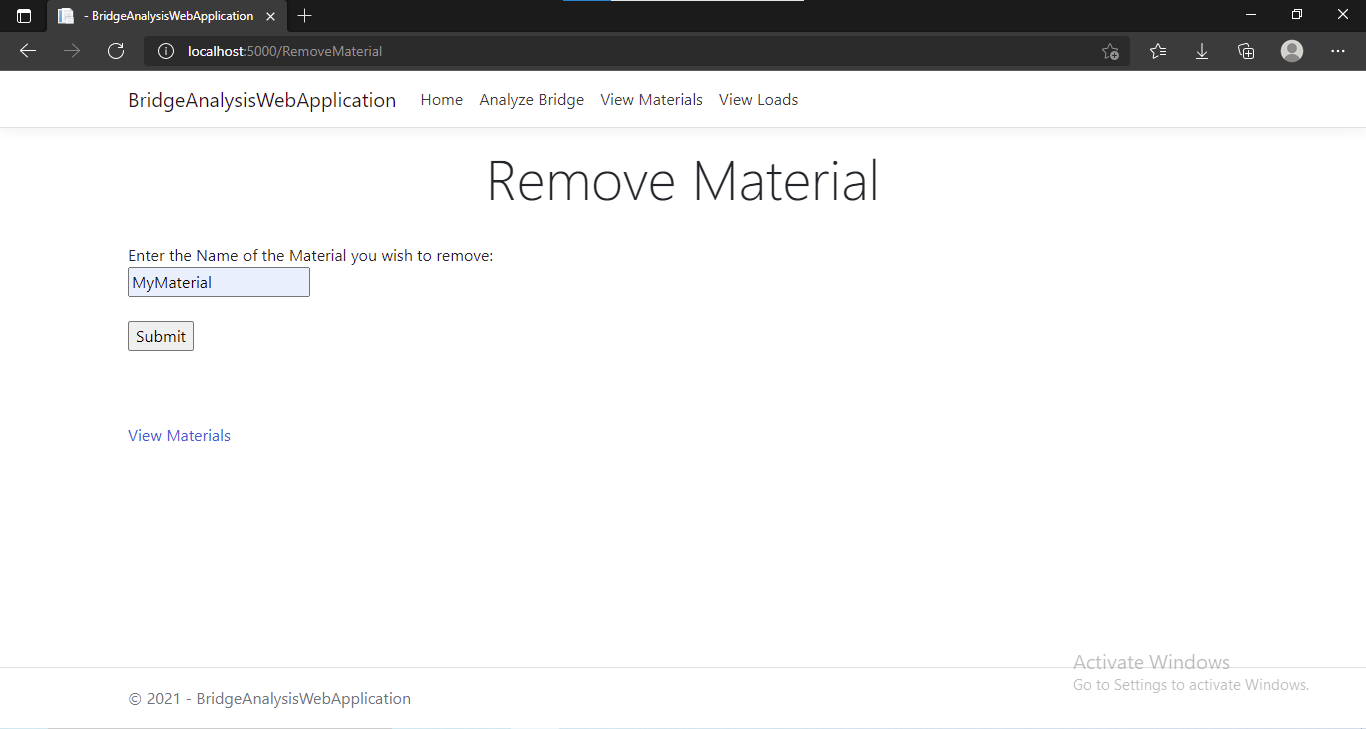


Thus, a new material has been added with the same properties as were specified.

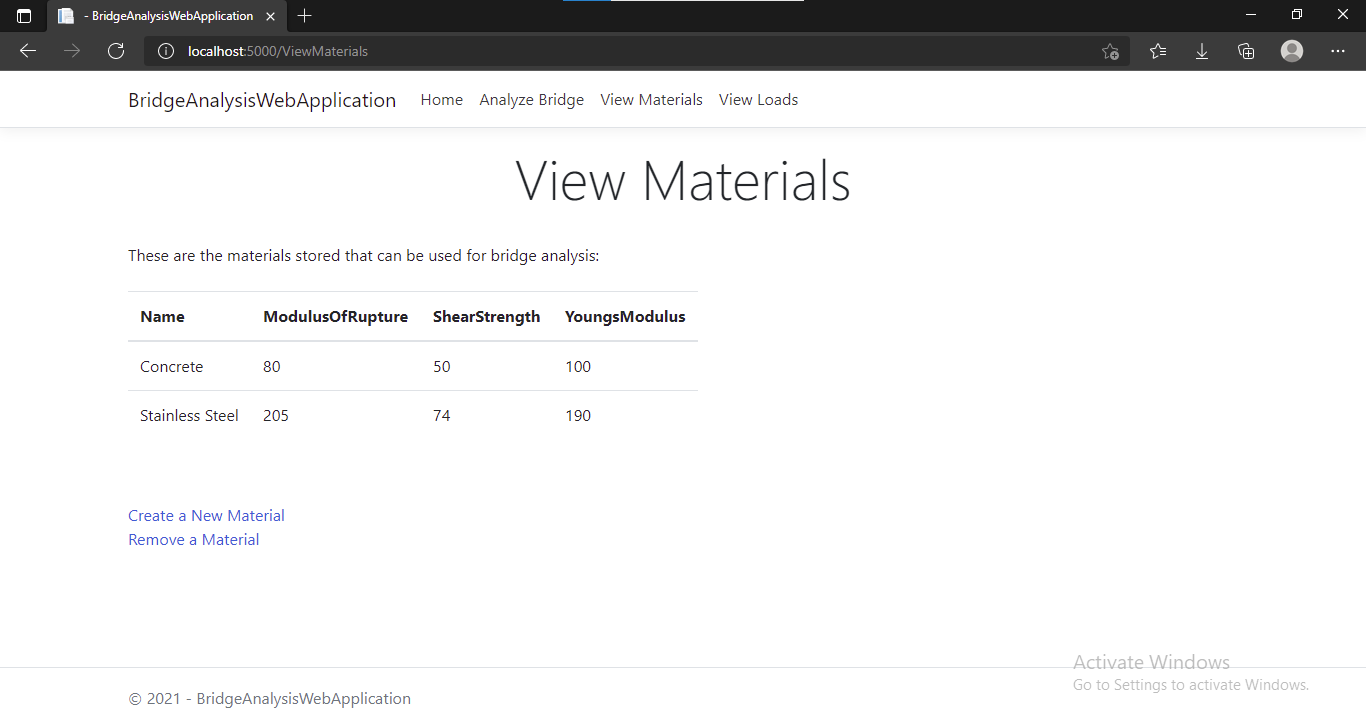
### RemoveMaterial

#### Remove a material and find it removed on the ViewMaterials page

Input Values:



ViewMaterials page after pressing Submit button:



Thus, the material was removed.

The pages CreateLoad and RemoveLoad work in the same manner

## SUMMARY

In summary, the core algorithms of the application as well as the user interface and final solution have been tested as can be seen in the sample above.

# EVALUATION

## REFLECTION ON EFFECTIVENESS

This section reflects on how the final outcome meets the original problem requirements.

The original problem requirements called for a sophisticated client server model, a web application which allowed the user to input the required variables for defining a bridge and then algorithmically determining whether the bridge would fall or not. Along with this, it required the solution to allow the user access to any database tables used and perform CRUD operations on it in the interest of providing a better user experience and flexibility in using the core bridge analysis functionality of the application.

In terms of this, as demonstrated in the Testing part of this documentation, the solution meets these requirements satisfactorily.

## EVALUATION OF OBJECTIVES

This is an evaluation of the objectives set in the Analysis part of the documentation.

The objectives set in the Analysis part of the documentation were:

1. Client connect to Web Server
   1. Start running the Web Server
   2. Get information such as hyperlink or IP address that will be used to connect to Web Server
   3. Connect to Server as a Client from a different computer
   4. Load and display home webpage
2. Validate values entered in Bridge Variables form
   1. Check that these values are:
      1. In the appropriate range of values
      2. In the correct format
   2. If values entered are improper, show message alerting the user that the values are improper
3. Web Server Communication
   1. Extract Bridge Variable values from HTTP request received and send to Web application
   2. Get Result of Bridge Analysis from Web Application and send to client as a HTTP response.
4. Produce Bridge Analysis Result
   1. Get Bridge Variables as input from Web Server
   2. Get relevant data from database
   3. Perform Bridge Analysis Algorithm using this input
   4. Send Bridge Analysis Result to Web Server
5. Database CRUD Operation
6. Allow user to access webpages separate to the bridge analysis webpage
7. Allow user to view the contents of the database containing relevant tables with data used by the bridge analysis algorithm
8. Allow user to add and delete items to the tables such as add new materials or load types which can then be used in the bridge analysis algorithm
9. User Communication
10. Load and view starting webpage
11. Provide links to other webpages
12. Display Bridge Variables form on the Bridge Analysis webpage
13. Be able to enter appropriate values in the form
14. Add a “Submit” values button that triggers the HTTP request
15. Display message for entering improper or inappropriate values in the form
16. Display Bridge Analysis Result on web page
17. Allow user to send different values for bridge analysis

|  |  |
| --- | --- |
| Objective | Evaluation |
| 1 | Final solution meets this objective and its subparts. However, one thing to note is that objective 1c was met at an early stage of the project. I was unable to explicitly test it with the final solution as it required the use of school computers whereas I completed the latter parts at home on my personal computer. However, since it was tested at school, no changes were made to the how the application connected to the web server and other computers and hence it should work nevertheless. |
| 2 | Final solution meets this objective as in the .cshtml.cs files, wherever there is user input involved, the system validates it and display an appropriate message in case the validation fails. |
| 3 | Final solution meets this objective as it is able to communicate by sending the bridge variable values entered by the user into the form to the web application to perform the algorithm and also send the appropriate result or response which is displayed to the user. |
| 4 | Final solution meets this objective as the solution is able to get the bridge variables, use them to create the appropriate objects and data structures, call the bridge analysis function which then conducts the bridge analysis algorithm on it, yielding a result, which is sent and displayed to the user over the web server. Within this process, the application is able to connect to the database and get the data from the tables (materials and loads) which is used to select the Material and Load by the user, to use in the bridge analysis algorithm. |
| 5 | Final solution meets this objective as the solution is able to let the user view the contents of the tables in the database, add new records as well as delete records as the user sees fit. This goes for both the tables in use (materials and loads). |
| 6 | Final solution meets this objective as the solution allows the user to load and navigate between the webpages, thereby being able to access the AnalyzeBridge.cshtml webpage. Here the user can enter values into the form and send them using the submit button. If the values are invalid, the user is appropriately alerted and if they are valid, the user receives the result of the bridge analysis. After this, the user is able to input different values and conduct the bridge analysis on these new different values. |

## INDEPENDENT FEEDBACK FROM END/USER AND SUPERVISOR

This is a paraphrase of the feedback received for the solution.

The final solution is well and complete. However, there are more features that could have been added to the solution, including support for different types of bridges, as well as a more sophisticated bridge analysis algorithm that would take into account factors such as weather conditions and combinations of materials to have the bridge comprise of. Apart from that it is definitely a good solution that is designed well and functions effectively.

## EVALUATION OF FEEDBACK

I agree with the feedback. The solution could have contained a more sophisticated algorithm. However, the main challenge of this project was understanding that a very realistic model of the problem and algorithm was unfeasible to implement and some models went beyond A Level standard. Some models were extremely complex and difficult to understand. However, a simplification of the algorithm that was still based in realistic physics was difficult to find and research. Most simplifications either oversimplified to the point where it was useless or had major holes and gaps in the algorithms. However, given more time and dedication, it would not be impossible to upgrade the algorithm to include more sophisticated processes and factors.

The point about the different types of bridges is interesting because it was possible to have an implementation of a bridge analysis algorithm designed for suspension bridges compared to beam bridges which is in the solution. This would also add concepts of inheritance to the solution as an interface for a Bridge would be used which the other types of bridges would inherit core functionality and properties from. This was implemented in the initial UML, however, I later removed it due to time constraints and considerations.

## IMPROVEMENTS ON REVISITATION

If I were to revisit this project, I would add a more sophisticated bridge analysis algorithm. As described above in the Evaluation of Feedback part, there are multiple ways to improve this solution.

Most notably:

* Add support for different types of bridges.
* Upgrade the bridge analysis algorithm to factor in weather conditions, soil conditions, stability of bridge supports such as the pillars in a beam bridge or the cables in a suspension bridge, different materials, etc.
* Automate the filling of the materials and loads tables in the database by using online databases or tables or by using web scraping. Thus, the user is not burdened with the task of finding the right values for Material properties and Load properties.

The upgrade of the bridge analysis algorithm would ideally consist of adding more tests to the BridgeAnalyzer.cs class, similar to the existing BeamStressTest, ShearStrengthTest and the DeflectionTest. This would also lead to more bridge variables being needed for input as well as more tables in the database. Thus, I have designed this solution such that the addition of more features is relatively simple and would not require to restructure the solution completely.

These are the ways I would improve the solution if it was revisited.

# APPENDIX

Here, the physics behind the tests in the bridge analysis algorithm would be described. For this we consider a distributed maximum load on the bridge, meaning that there is an equal force acting downwards on the beam at all points on the bridge.

The general mathematical requirement for the design of any structure is as follows:

The factor of safety is to ensure that the actual value never reaches the failure value. This is to make sure that unforeseen increases in the actual value do not pose a risk to the safety of the structure. Factor of safety is typically around 1.8 to 2.5 but can vary.

This is the principle on which these tests are based.

## BEAM STRESS TEST

Stress is a measure of the intensity of internal force in a member. It is always expressed in units of force per area. If a structural member is subjected to pure tension or pure compression, the internal stress is uniformly distributed on the cross‑section. This results in flexure, or the bending behaviour of the beam.

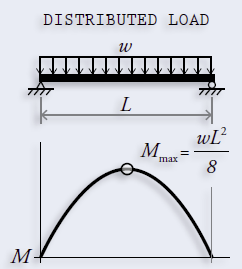
Thus,

The relevant failure stress is a material property called the Modulus of Rupture (σr), defined as the stress at which a material fails in flexure.

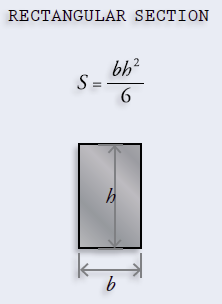
To find the maximum stress, we use the flexure formula:

Here, M is the internal moment, which can be conceptualized as the tendency of external loads to cause bending in a structural member. S is the elastic section modulus, a measure of the member’s resistance to bending.

For a distributed load as shown in the diagram below, we are able to find the value of maximum M.



Also, for a rectangular section, we can find S as shown in the diagram below.



These are given inputs, allowing us to find maximum stress. The Modulus of Rupture is a property of the material and can be taken from the database. Using this we check for the given inequality to see if the bridge passes this test.

## SHEAR STRENGTH TEST

The stress due to shear forces is called the shear stress (). This should follow a similar inequality,

Failure shear stress is a property of the material and is called the Shear Strength of that material.

The maximum shear stress for a rectangular section is calculated as follows:

These values are given as inputs and the Shear Strength is taken from the database. Using this we check for the given inequality to see if the bridge passes this test.

## DEFLECTION TEST

Deflection is the vertical distance a beam bends in response to its applied load.

For a simple beam with span length, subjected to a uniform loading, the maximum deflection, , can be calculated with the following equation,

Where is Young’s Modulus or the Modulus of Elasticity, a property of the material and is the area moment of inertia for the cross-section. Area moment of inertia is calculated as follows,

Where S is the elastic section modulus and is the height of the beam.

Note that here, the uniform loading should only consider the weight of the load applied on the bridge and not the weight of the beam.

Similar to before,

is specified by most current design codes as the following,

These values are given as inputs and Young’s Modulus is taken from the database. Using this we check for the given inequality to see if the bridge passes this test.

Thus, these are the three tests used in the solution to check whether a beam bridge would stand or fall.

(Ressler, 2017)

Bibliography

Bentley. (2020). *Bridge Analysis Software*. Retrieved September 29, 2020, from https://www.bentley.com/en/products/product-line/bridge-analysis-software

CivilFEM. (2020). *CivilFEM Apps for ANSYS Workbench*. Retrieved September 29, 2020, from https://www.civilfem.com/products/civilfem-apps-for-ansys-workbench/

CivilFEM. (2020). *CivilFEM for ANSYS*. Retrieved September 29, 2020, from https://www.civilfem.com/products/civilfem-for-ansys/

DBQuest, Inc. (2012). *Top Reasons to Use MySQL*. Retrieved October 6, 2020, from http://www.databasequest.com/index.php/product-service/mysql-dbquest

G., D. (2020). *Hostinger Tutorials*. Retrieved September 29, 2020, from https://www.hostinger.co.uk/tutorials/what-is-apache#:~:text=Apache%20is%20one%20of%20the,web%20server%20without%20any%20customization

Gadhavi, M. (2020). *Top 8 Reasons Why ASP.NET Core is the Best Framework for Web Application Development*. Retrieved October 6, 2020, from https://radixweb.com/blog/8-reasons-asp-dot-net-core-is-best-framework

Makai, M. (2020). *Object-relational Mappers (ORMs)*. Retrieved October 6, 2020, from https://www.fullstackpython.com/object-relational-mappers-orms.html

MDN contributors. (2019). *Introduction to the server side*. Retrieved October 6, 2020, from https://developer.mozilla.org/en-US/docs/Learn/Server-side/First\_steps/Introduction

MDN contributors. (2020). *Database*. Retrieved October 6, 2020, from https://developer.mozilla.org/en-US/docs/Glossary/Database

MDN contributors. (2020). *Your first form*. Retrieved October 6, 2020, from https://developer.mozilla.org/en-US/docs/Learn/Forms/Your\_first\_form

Microsoft. (2019). *Characteristics of Modern Web Applications*. Retrieved October 6, 2020, from https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/modern-web-applications-characteristics

Microsoft. (2020). *Choose Between Traditional Web Apps and Single Page Apps (SPAs)*. Retrieved October 6, 2020, from https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/choose-between-traditional-web-and-single-page-apps

Microsoft. (2020). *Host ASP.NET Core on Linux with Apache*. Retrieved October 6, 2020, from https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/linux-apache?view=aspnetcore-3.1

Ressler, S. (2017). Do-It-Yourself Engineering. In S. Ressler, *Do-It-Yourself Engineering* (pp. 38-55). THE GREAT COURSES.

SOURCEFORGE. (2020). *MicroStation Bentley Systems*. Retrieved September 29, 2020, from https://sourceforge.net/software/product/MicroStation/#:~:text=MicroStation%20is%20engineering%20software%2C%20and,at%20%24250.00%2Fmonth%2Fuser.

Woodford, C. (2020). *Explain That Stuff*. Retrieved September 29, 2020, from https://www.explainthatstuff.com/bridges.html