Fall 2021

Guitar Store Inventory

Alexander DiDonato
ad212@uakron.edu

Follow this and additional works at: https://ideaexchange.uakron.edu/honors_research_projects

Part of the Data Storage Systems Commons

Please take a moment to share how this work helps you through this survey. Your feedback will be important as we plan further development of our repository.

Recommended Citation
https://ideaexchange.uakron.edu/honors_research_projects/1443

This Dissertation/Thesis is brought to you for free and open access by The Dr. Gary B. and Pamela S. Williams Honors College at IdeaExchange@UAkron, the institutional repository of The University of Akron in Akron, Ohio, USA. It has been accepted for inclusion in Williams Honors College, Honors Research Projects by an authorized administrator of IdeaExchange@UAkron. For more information, please contact mjon@uakron.edu, uapress@uakron.edu.
Guitar Inventory System

Alex DiDonato

Senior Programming Projects

Spring 2021
What is the problem
The problem I want to solve is that I want to make a better program to track our inventory and give our customers a chance to order even if it is out of stock. I also wanted a priority list so people can still order an item even out of stock. I am trying to find a way to integrate a database into a website so customers can see when an item would go out of stock and how many are left. I chose this project because I have a profound interest in guitars and doing inventory as I work at Drug Mart. I also might want to open up a guitar business in the future and this will give me an idea on how to do my website. I intend to solve it by making a website that implemented a database to track the amount of stock left.

Databases

Integration

Employee side

Website
<table>
<thead>
<tr>
<th></th>
<th>A Name</th>
<th>Status</th>
<th>A Headline</th>
<th>Sub-headline</th>
<th>Creator</th>
<th>Due date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Meet With Planners</td>
<td>~1 Day</td>
<td>Phase 1 Meet with Planners</td>
<td></td>
<td>Alex DiDonato</td>
<td>1/18/2021</td>
</tr>
<tr>
<td>2</td>
<td>Meet with Manager</td>
<td>~1 Day</td>
<td>Phase 1 Meet with Planners</td>
<td></td>
<td>Alex DiDonato</td>
<td>1/18/2021</td>
</tr>
<tr>
<td>3</td>
<td>Meet with Agents</td>
<td>~1 Day</td>
<td>Phase 1 Meet with Planners</td>
<td></td>
<td>Alex DiDonato</td>
<td>1/20/2021</td>
</tr>
<tr>
<td>4</td>
<td>Define Use Cases</td>
<td>~3 Days</td>
<td>Phase 2 Lay out a plan</td>
<td></td>
<td>Alex DiDonato</td>
<td>1/25/2021</td>
</tr>
<tr>
<td>5</td>
<td>Define Information Requirements</td>
<td>~2 Days</td>
<td>Phase 2 Lay out a plan</td>
<td></td>
<td>Alex DiDonato</td>
<td>1/31/2021</td>
</tr>
<tr>
<td>6</td>
<td>Develop workflows</td>
<td>~4 Days</td>
<td>Phase 2 Lay out a plan</td>
<td></td>
<td>Alex DiDonato</td>
<td>2/8/2021</td>
</tr>
<tr>
<td>7</td>
<td>Define Classes</td>
<td>~3 Days</td>
<td>Phase 2 Lay out a plan</td>
<td></td>
<td>Alex DiDonato</td>
<td>2/11/2021</td>
</tr>
<tr>
<td>8</td>
<td>Design overall architecture</td>
<td>~7 Days</td>
<td>Phase 2 Lay out a plan</td>
<td></td>
<td>Alex DiDonato</td>
<td>2/19/2021</td>
</tr>
<tr>
<td>Task</td>
<td>Duration</td>
<td>Phase</td>
<td>Team Member</td>
<td>Date</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>----------</td>
<td>--------</td>
<td>------------------</td>
<td>-----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9 Design Screens</td>
<td>~5 Days</td>
<td>Phase 3</td>
<td>Alex DiDonato</td>
<td>2/26/2021</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 Design and Build Database</td>
<td>~7 Days</td>
<td>Phase 3</td>
<td>Alex DiDonato</td>
<td>3/4/2021</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11 Design program details</td>
<td>~5 Days</td>
<td>Phase 3</td>
<td>Alex DiDonato</td>
<td>3/11/2021</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12 Integrate the database into the program</td>
<td>~4 Days</td>
<td>Phase 3</td>
<td>Alex DiDonato</td>
<td>3/17/2021</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13 Code and test logic layer</td>
<td>~6 Days</td>
<td>Phase 4</td>
<td>Alex DiDonato</td>
<td>3/25/2021</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14 Perform user acceptance test</td>
<td>~7 Days</td>
<td>Phase 4</td>
<td>Alex DiDonato</td>
<td>4/2/2021</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15 Perform functional tests</td>
<td>~5 Days</td>
<td>Phase 4</td>
<td>Alex DiDonato</td>
<td>4/6/2021</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Analysis

A

Use Cases

Using the event decomposition technique, a use case will be when an item gets to 3 left, the system will trigger an alert to send to the supplier to get more of the stock in as soon as possible. Another possible use case is when the item gets down to 10 because that is when the information system will put out a message saying there is 10 left. The second use case will be if the amount left gets under 3, that will be when the program puts a notice out to have the employees put out an order for said item. The final use case would be if a user put in an order on an out of stock item, the system will put the order on a priority list that says that person will get it as soon as it arrives.

<table>
<thead>
<tr>
<th>Customer</th>
<th>Order an item</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Place an order in the Priority List</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Supplier</th>
<th>Puts in order to get item restocked</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gets notified if an item is 10 or less</td>
</tr>
<tr>
<td></td>
<td>Gets notice to place order if stock drops below 3</td>
</tr>
</tbody>
</table>

I used this to analyze how users and suppliers would act in some scenarios and how events would play out. The customer would buy an item and the supplier would note that the amount of a certain guitar would decrease. I put in things that both would do and create possible scenarios that this application would be used for.
Brainstorming

Customer interacts with the website
Customer sees if item is in stock or out of stock
Customer places order
Supplier puts order on priority list
Supplier checks stock of item
Supplier sees if the item needs restocked

Nouns
Customer
Supplier
Order
Guitar Inventory
List

<table>
<thead>
<tr>
<th>Extract Things</th>
<th>Guitar Inventory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employee</td>
<td>Guitar Inventory</td>
</tr>
<tr>
<td>EmployeeId</td>
<td>Id</td>
</tr>
<tr>
<td>FirstName</td>
<td>Brand</td>
</tr>
<tr>
<td>LastName</td>
<td>Model</td>
</tr>
<tr>
<td>Username</td>
<td>YearMade</td>
</tr>
<tr>
<td>Password</td>
<td>Amount</td>
</tr>
<tr>
<td></td>
<td>Price</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stock</th>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>CustId</td>
<td>OrderID</td>
</tr>
<tr>
<td>CustFirstName</td>
<td>Brand</td>
</tr>
<tr>
<td>CustLastName</td>
<td>Model</td>
</tr>
<tr>
<td>ItemOrdered</td>
<td>OrderDate</td>
</tr>
</tbody>
</table>

This was the brainstorm phase for all of my use cases. I extracted the common nouns like customer, order, and stock and created use cases. I also extracted things from each of the nouns to form database tables and the foundation of the project.
The Domain classes that I decided to use were to identify certain aspects of a guitar, like brand name, model, name, and year made. Other tables used different domain classes, such as employee and customer first and last name to make sure that the correct person is identified. The order status table used classes like itemOrdered and Order date to correctly identify the date of the order. All tables also use Id as the primary key to accurately keep track of all of the data. All of the domain classes will be listed below here.

Employee

EmployeeId, FirstName, LastName, Username, Password

GuitarInventory

Id, Brand, Model, YearMade, Amount, Price

OrderStatus

OrderId, OrderDate, Brand, Model

PriorityList

CustId, CustFirstName, CustLastName, ItemOrdered
This ERD was a rough draft of how I was originally going to do this project. The shipments part of it was changed to Guitar Inventory because I had no way to look at guitar inventory. The List represents the priority list of people who ordered an item that was out of stock, which is related to the orders because that shows that a customer ordered an item out of stock. I also related orders to every entity because that is the centerpiece of keeping charge of stock.
This is the Domain Model class diagram with the tables for the database. I have the Employee class separate because it has no relation to anything and its sole purpose is for security. The other three tables are for keeping track of inventory and who has ordered something out of stock. The order has many to many with the Guitar Inventory because many guitars can have many orders at once. Priority List has one to many because one person could have one guitar or many guitars.
This is the state machine diagram that states that when an item goes to 10 or 3, it will do these actions. When the amount hits 10, a message should pop up saying, “only 10 left, hurry before it is too late.” When an Item hits 3, a message on the employee side should say, “only 3 left, place an order for this item.”
When an Item hits 3, a message on the employee side should say, “only 3 left, place an order for this item.” The bottom one is if the customer clicks buy it now and there is no more guitars. The system will pull up the priority list and the customer fills out his or her information.
This is the activity diagram stating all possibilities when a user clicks buy it now. The system will check if the item goes under 10 or less and act according to the programming. If it is 10 or less, pull up the only x left message. If it goes under 3, the program tells the employee side to order more stock. If there are none left, the Priority list page pulls up and the customer enters their information to be sent to the priority list table.
Use Cases

Using the event decomposition technique, a use case will be when an item gets to 3 left, the system will trigger an alert to send to the supplier to get more of the stock in as soon as possible. Another possible use case is when the item gets down to 10 because that is when the information system will put out a message saying there is 10 left. The second use case will be if the amount left gets under 3, that will be when the program puts a notice out to have the employees put out an order for said item. The final use case would be if a user put in an order on an out of stock item, the system will put the order on a priority list that says that person will get it as soon as it arrives.

| Customer | Order an item \[item\]  
| Supplier | Remove item  
| Puts in order to get item \[restocked\]  
| Gets notified if an item is 10 or less  

This is the use case table that puts together the use cases into one table. This tells us all of the possibilities and interactions between the customer and the supplier. All possibilities including what happens as the amount decreases and eventually hits 0. The supplier will interact on the last three cases as the customer directly influences the system by hitting buy it now.
This is the normalized Database schema with the tables for the database. I have the Employee class separate because it has no relation to anything and its sole purpose is for security. The other three tables are for keeping track of inventory and who has ordered something out of stock. The order has many to many with the Guitar Inventory because many guitars can have many orders at once. Priority List has one to many because one person could have one guitar or many guitars.
public class HomeController : Controller
{
    String ConnectionString;
    SqlConnection Connection;

    public HomeController()
    {
        ConnectionString = (@"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\Database1.mdf;Integrated Security=True");
        Connection = new SqlConnection(ConnectionString);
    }
    public ActionResult Index()
    {
        SqlCommand cmd = new SqlCommand("select * from GuitarInventory order by Brand, Model", Connection);
        Connection.Open();
        SqlDataAdapter adpt = new SqlDataAdapter(cmd);
        adpt.SelectCommand = cmd;
        DataSet ds = new DataSet("GuitarInventory");
        adpt.Fill(ds);
        Connection.Close();
        List<Models.GuitarInventory> GuitarList = new List<Models.GuitarInventory>();
        foreach (DataRow dr in ds.Tables[0].Rows)
        {
            GuitarList.Add(new Models.GuitarInventory
            {
                Id = Convert.ToInt32(dr["Id"]),
                Brand = Convert.ToString(dr["Brand"]),
                Model = Convert.ToString(dr["Model"]),
                YearMade = Convert.ToInt32(dr["YearMade"])
            });
        }
    }
}
Amount = Convert.ToInt32(dr["Amount"]),
Price = Convert.ToString(dr["Price"])
});
}

ViewData["GuitarList"] = GuitarList;
return View();

public ActionResult Purchase()
{
    try
    {
        int GuitarId = Convert.ToInt32(Request.Form["Id"]);
        SqlCommand command = new SqlCommand("select Amount from GuitarInventory where Id = @id", Connection);
        Connection.Open();
        command.Parameters.AddWithValue("@id", GuitarId);
        int Amount = (int32)command.ExecuteScalar();
        //System.Diagnostics.Debug.WriteLine(Amount);
        if (Amount > 0)
        {
            SqlCommand cmd = new SqlCommand(" update GuitarInventory set Amount = (Amount - 1) where Id = @id", Connection);
            cmd.Parameters.AddWithValue("@id", GuitarId);
            cmd.ExecuteNonQuery();
            return RedirectToAction("Index");
        }
        else
        {
            return RedirectToAction("OrderOutOfStock");
        }
    }
}
catch(Exception ex)
{
    return RedirectToAction("OrderOutOfStock");
}

public ActionResult About()
{
    ViewBag.Message = "Your application description page.";

    return View();
}

public ActionResult Contact()
{
    ViewBag.Message = "Your contact page.";
    ViewBag.Submitted = false;
    if (Request.Form.Count > 0)
    {
        String CustFirstName = Request.Form["CustFirstName"];  
        String CustLastName = Request.Form["CustLastName"];  
        String ItemOrdered = Request.Form["ItemOrdered"];  
        SqlCommand command = new SqlCommand("select top 1 CustId from PriorityList Order By CustId desc ", Connection);
        Connection.Open();
        int CustId = (Int32)command.ExecuteScalar() + 1;
This is the pseudocode for the website and how it is supposed to interact with the database. This code is what sends data to the Employee portal and updates it there as well as on the website.

This code also has queries to make sure that the right item is being modified when the buy it now button is clicked for a certain guitar. I also have a query to check if the amount for the guitar is at zero, the following If then statement determines that if the item is at zero, cue the OrderOutOfStock page, otherwise, the program continues as usual.

```csharp
SqlCommand cmd = new SqlCommand("insert into PriorityList(CustId, CustFirstName, CustLastName, ItemOrdered) values (@CustId, @CustFirstName, @CustLastName, @ItemOrdered)", Connection);

cmd.Parameters.AddWithValue("@CustId", CustId);

System.Diagnostics.Debug.WriteLine(CustId);

System.Diagnostics.Debug.WriteLine(CustFirstName);

System.Diagnostics.Debug.WriteLine(CustLastName);

System.Diagnostics.Debug.WriteLine(ItemOrdered);

cmd.ExecuteNonQuery();

ViewBag.Message = "Thank you for your interest, we will be in contact with you shortly.");

ViewBag.Submitted = true;
```

return View();
This is the user interface for the customers online guitar website. I made it as simple as possible to make sure it interacted with the database. When the user clicks buy it now, the 20 on whatever product they clicked went down to 19, then 18, and so on so forth. This change was also reflected on the employee side in the guitar inventory page as it would show the change there as well.
<table>
<thead>
<tr>
<th>Guitar Model</th>
<th>Price</th>
<th>Buy it Now</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dean FlyingV</td>
<td>$699.99</td>
<td>Only 20 left, hurry before it's too late!</td>
</tr>
<tr>
<td>Epiphone Casino</td>
<td>$649.99</td>
<td>Only 20 left, hurry before it's too late!</td>
</tr>
<tr>
<td>Fender Mustang</td>
<td>$449.99</td>
<td>Only 20 left, hurry before it's too late!</td>
</tr>
<tr>
<td>Fender Strat</td>
<td>$499.99</td>
<td>Only 20 left, hurry before it's too late!</td>
</tr>
<tr>
<td>Fender Telecaster</td>
<td>$549.99</td>
<td>Only 20 left, hurry before it's too late!</td>
</tr>
<tr>
<td>Gibson Les Paul</td>
<td>$799.99</td>
<td>Only 20 left, hurry before it's too late!</td>
</tr>
<tr>
<td>Gibson SG</td>
<td>$599.99</td>
<td>Only 20 left, hurry before it's too late!</td>
</tr>
<tr>
<td>Ibanez RG</td>
<td>$299.99</td>
<td>Only 20 left, hurry before it's too late!</td>
</tr>
<tr>
<td>Jackson JS</td>
<td>$399.99</td>
<td>Only 20 left, hurry before it's too late!</td>
</tr>
<tr>
<td>Schecter Synyster</td>
<td>$849.99</td>
<td>Only 20 left, hurry before it's too late!</td>
</tr>
</tbody>
</table>
This is the security used for the employee side. This makes sure that the user logging in is in the database before continuing. If someone tries to enter in their username and password and it is not in the employee table, they will get denied access.

K
After you log in and get into the employee side. You will the Order status page pop up, and on the bottom of that are two buttons. From there, you can click on the two buttons down on the bottom that say “Priority List” and “Guitar Inventory”. The Guitar Inventory button takes you to the Guitar Inventory Page and the Priority List takes you to the Priority List. When you go to either of those pages, the format is the same and you can add things to the Guitar Inventory or the Priority List. At the bottom of the two pages, there are also two buttons at the bottom that say “Priority List” and “Order Status” for the Guitar Inventory Page and “Order Status” and “Guitar Inventory” on the Priority List page. The Order Status, Guitar Inventory, and Priority List pages are functionally identical. You can move around to each different page and edit the tables on all of these pages.
Interfaces at design time
```csharp
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Design3_Alex_Didonato
{
    public partial class Employee : Form
    {
        public employee()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            try
            {
                string Conn = @"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\Database1.mdf;Integrated Security=True";
                SqlConnection con = new SqlConnection(Conn);
                SqlCommand cmd = new SqlCommand("select * from Employee where Username = @Username and Password = @Password", con);
                cmd.Parameters.AddWithValue("@Username", txtUsername.Text);
                cmd.Parameters.AddWithValue("@Password", txtPassword.Text);
                con.Open();
                SqlDataAdapter adapt = new SqlDataAdapter(cmd);
                DataSet ds = new DataSet();
                adapt.Fill(ds);
                con.Close();
                int count = ds.Tables[0].Rows.Count;
                if (count == 1)
                {
                    OrderStatus os = new OrderStatus();
                    os.Show();
                    this.Hide();
                }
                else
                {
                    MessageBox.Show("Invalid username or password, please try again");
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
    }
}
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Design_3_Alex_DiDonato
{
    public partial class GuitarInventory : Form
    {
        public GuitarInventory()
        {
            InitializeComponent();
        }

        private void btnPriorityList_Click(object sender, EventArgs e)
        {
            PriorityList pl = new PriorityList();
            pl.Show();
            this.Hide();
        }

        private void btnOrderStatus_Click(object sender, EventArgs e)
        {
            OrderStatus os = new OrderStatus();
            os.Show();
            this.Hide();
        }

        private void guitarInventoryBindingNavigatorSaveItem_Click(object sender, EventArgs e)
        {
            this.Validate();
            this.guitarInventoryBindingSource.EndEdit();
            this.tableAdapterManager.UpdateAll(this.databaseDataSet2);
        }

        private void GuitarInventory_Load(object sender, EventArgs e)
        {
            // TODO: This line of code loads data into the 'databaseDataSet2.GuitarInventory' table. You can move, or remove it, as needed.
            this.guitarInventoryTableAdapter.Fill(this.databaseDataSet2.GuitarInventory);
        }

        private void guitarInventoryBindingNavigatorSaveItem_Click(object sender, EventArgs e)
        {
            this.Validate();
            this.guitarInventoryBindingSource.EndEdit();
            this.tableAdapterManager.UpdateAll(this.databaseDataSet2);
        }
    }
}
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Design_J_Alex_DiDonate
{
    public partial class OrderStatus : Form
    {
        public OrderStatus()
        {
            InitializeComponent();
        }

        private void orderStatusBindingNavigatorSaveItem_Click(object sender, EventArgs e)
        {
            this.Validate();
            this.orderStatusBindingSource.EndEdit();
            this.tableAdapterManager.UpdateAll(this(orderStatusDataSet);
        }

        private void Form_Load(object sender, EventArgs e)
        {
            // TODO: This line of code loads data into the 'orderStatusDataSet.OrderStatus' table. You can move, or remove it, as needed.
            this.orderStatusTableAdapter.Fill(this.orderStatusDataSet.OrderStatus);
        }

        private void btnPriorityList_Click(object sender, EventArgs e)
        {
            PriorityList pl = new PriorityList();
            pl.Show();
            this.Hide();
        }

        private void btnGuitarInventory_Click(object sender, EventArgs e)
        {
            GuitarInventory gl = new GuitarInventory();
            gl.Show();
            this.Hide();
        }
    }
}
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Design_3_Alex_Dodonato
{
    class Program
    {
        static void Main(string[] args)
        {
            Application.Run(new MainForm());
        }
    }
}

public partial class PriorityList : Form
{
    public PriorityList()
    {
        InitializeComponent();
    }

data private void btnOrderStatus_Click(object sender, EventArgs e)
{
    OrderStatus os = new OrderStatus();
    os.Show();
    this.Hide();
}

data private void prioritylistBindingNavigatorSaveItem_Click(object sender, EventArgs e)
{
    this.Validate();
    this.prioritylistBindingSource.EndEdit();
    this.prioritylistTableAdapter.Update(this.prioritylistDataSet.PriorityList);
}

data private void PriorityList_Load(object sender, EventArgs e)
{
    // TODO: This line of code loads data into the 'prioritylistDataSet.PriorityList' table. You can move, or remove it, as needed.
    this.prioritylistTableAdapter.Fill(this.prioritylistDataSet.PriorityList);
}

data private void btnGuitarInventory_Click(object sender, EventArgs e)
{
    GuitarInventory gi = new GuitarInventory();
    gi.Show();
    this.Hide();
}
.grid-item {
    background-color: rgba(255, 255, 255, 0.8);
    border: 1px solid rgba(0, 0, 0, 0.8);
    padding: 20px;
    width: 33.33%;
    font-size: 30px;
    text-align: center;
}
</style>
</head>
<body>

<header>
    <h1 style="color:navy;"> Alex DiDonato's Guitar Store </h1>
</header>
<div class="grid-container">
    @{
        var GuitarList = ViewBag.GuitarList;
    }
    @foreach (var Guitar in GuitarList) {
        <div class="grid-item">
            <h2>@Guitar.Brand @Guitar.Model</h2>
            <p>@Guitar.Price</p>
            <form action="/Home/Purchase" method="post">
                <button type="submit" value="Buy"> Buy it Now </button>
                <input type="hidden" name="id" value="@Guitar.Id" />
            </form>
            <h4>Only @Guitar.Amount left, hurry before it's too late!</h4>
        </div>
    }
</div>
</body>
</html>
Database Documentation
### dbo.OrderStatus.sql

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
<th>Allow Nulls</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>OrderId</td>
<td>int</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OrderDate</td>
<td>varchar(50)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brand</td>
<td>varchar(50)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Model</td>
<td>varchar(50)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### dbo.SuppliesInventory.sql

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
<th>Allow Nulls</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>int</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brand</td>
<td>nchar(10)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Model</td>
<td>nchar(10)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>YearMade</td>
<td>int</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Amount</td>
<td>int</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Price</td>
<td>varchar(50)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### dbo.Employee.sql

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
<th>Allow Nulls</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employeeld</td>
<td>int</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FirstName</td>
<td>varchar(50)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LastName</td>
<td>varchar(50)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Username</td>
<td>varchar(50)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Password</td>
<td>varchar(50)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Employee Data

<table>
<thead>
<tr>
<th>Employeeld</th>
<th>FirstName</th>
<th>LastName</th>
<th>Username</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alex</td>
<td>DiDonato</td>
<td>D3monDrum3r</td>
<td>D00m3d</td>
</tr>
<tr>
<td>2</td>
<td>Zarreen</td>
<td>Farooci</td>
<td>ZFarooci</td>
<td>HelloWorld</td>
</tr>
<tr>
<td>3</td>
<td>John</td>
<td>Lennon</td>
<td>Lenny</td>
<td>Help!</td>
</tr>
<tr>
<td>4</td>
<td>George</td>
<td>Harrison</td>
<td>Hairy</td>
<td>Something</td>
</tr>
<tr>
<td>5</td>
<td>Billy</td>
<td>Bob</td>
<td>BillyBob</td>
<td>BillyBob</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>
### Table 1: Guitar Orders

<table>
<thead>
<tr>
<th>OrderId</th>
<th>OrderDate</th>
<th>Brand</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3-31-2021</td>
<td>Epiphone</td>
<td>Casino</td>
</tr>
<tr>
<td>2</td>
<td>3-31-2021</td>
<td>Fender</td>
<td>Strat</td>
</tr>
<tr>
<td>3</td>
<td>3-31-2021</td>
<td>Gibson</td>
<td>SG</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>

### Table 2: Customer Information

<table>
<thead>
<tr>
<th>CustId</th>
<th>CustFirstName</th>
<th>CustLastName</th>
<th>ItemOrdered</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alex</td>
<td>DiDonato</td>
<td>Casino</td>
</tr>
<tr>
<td>2</td>
<td>Zareen</td>
<td>Farooqi</td>
<td>Strat</td>
</tr>
<tr>
<td>3</td>
<td>Billy</td>
<td>Bob</td>
<td>SG</td>
</tr>
<tr>
<td>4</td>
<td>Alex</td>
<td>DiDonato</td>
<td>Casino</td>
</tr>
<tr>
<td>5</td>
<td>Alex</td>
<td>DiDonato</td>
<td>FlyingV</td>
</tr>
<tr>
<td>6</td>
<td>Alex</td>
<td>DiDonato</td>
<td>Casino</td>
</tr>
<tr>
<td>7</td>
<td>Alex</td>
<td>DiDonato</td>
<td>Casino</td>
</tr>
<tr>
<td>8</td>
<td>Alex</td>
<td>DiDonato</td>
<td>Casino</td>
</tr>
<tr>
<td>9</td>
<td>Alex</td>
<td>DiDonato</td>
<td>Casino</td>
</tr>
<tr>
<td>10</td>
<td>Alex</td>
<td>DiDonato</td>
<td>Casino</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>

### Table 3: Guitar Inventory

<table>
<thead>
<tr>
<th>Id</th>
<th>Brand</th>
<th>Model</th>
<th>YearMade</th>
<th>Amount</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Epiphone</td>
<td>Casino</td>
<td>1965</td>
<td>20</td>
<td>$649.99</td>
</tr>
<tr>
<td>2</td>
<td>Gibson</td>
<td>Les Paul</td>
<td>0</td>
<td>20</td>
<td>$799.99</td>
</tr>
<tr>
<td>3</td>
<td>Fender</td>
<td>Strat</td>
<td>1967</td>
<td>20</td>
<td>$499.99</td>
</tr>
<tr>
<td>4</td>
<td>Fender</td>
<td>Telecaster</td>
<td>1984</td>
<td>20</td>
<td>$549.99</td>
</tr>
<tr>
<td>5</td>
<td>Gibson</td>
<td>SG</td>
<td>1978</td>
<td>20</td>
<td>$599.99</td>
</tr>
<tr>
<td>6</td>
<td>Schecter</td>
<td>Synyster</td>
<td>2007</td>
<td>20</td>
<td>$849.99</td>
</tr>
<tr>
<td>7</td>
<td>Ibanez</td>
<td>RG</td>
<td>1987</td>
<td>20</td>
<td>$299.99</td>
</tr>
<tr>
<td>8</td>
<td>Jackson</td>
<td>JS</td>
<td>1985</td>
<td>20</td>
<td>$399.99</td>
</tr>
<tr>
<td>9</td>
<td>Fender</td>
<td>Mustang</td>
<td>1991</td>
<td>20</td>
<td>$449.99</td>
</tr>
<tr>
<td>10</td>
<td>Desi</td>
<td>FlyingV</td>
<td>1995</td>
<td>20</td>
<td>$699.99</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>
Invalid data

I tested putting in a string for the amount even though it says it must be an integer. When I put that in here, it would not let me move textboxes because the information in the textbox is not an integer and it would not move unless someone corrected it.
Valid data

I tested this as valid data and this is how an entry should be entered into a database. I did this as main testing to see if the program would run okay.
Interfaces at runtime
# Alex DiDonato's Guitar Store

<table>
<thead>
<tr>
<th>Guitar Model</th>
<th>Price</th>
<th>Buy it Now</th>
<th>Quantity Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dean FlyingV</td>
<td>$699.99</td>
<td>[link]</td>
<td>Only 20 left, hurry before it's too late!</td>
</tr>
<tr>
<td>Epiphone Casino</td>
<td>$649.99</td>
<td>[link]</td>
<td>Only 20 left, hurry before it's too late!</td>
</tr>
<tr>
<td>Fender Mustang</td>
<td>$449.99</td>
<td>[link]</td>
<td>Only 20 left, hurry before it's too late!</td>
</tr>
<tr>
<td>Fender Strat</td>
<td>$499.99</td>
<td>[link]</td>
<td>Only 20 left, hurry before it's too late!</td>
</tr>
<tr>
<td>Fender Telecaster</td>
<td>$549.99</td>
<td>[link]</td>
<td>Only 20 left, hurry before it's too late!</td>
</tr>
<tr>
<td>Gibson Les Paul</td>
<td>$799.99</td>
<td>[link]</td>
<td>Only 20 left, hurry before it's too late!</td>
</tr>
<tr>
<td>Gibson SG</td>
<td>$599.99</td>
<td>[link]</td>
<td>Only 20 left, hurry before it's too late!</td>
</tr>
<tr>
<td>Ibanez RG</td>
<td>$299.99</td>
<td>[link]</td>
<td>Only 20 left, hurry before it's too late!</td>
</tr>
<tr>
<td>Jackson JS</td>
<td>$399.99</td>
<td>[link]</td>
<td>Only 20 left, hurry before it's too late!</td>
</tr>
<tr>
<td>Schecter Synyster</td>
<td>$849.99</td>
<td>[link]</td>
<td>Only 20 left, hurry before it's too late!</td>
</tr>
</tbody>
</table>

---

**OrderOutOfStock.**

Place your information here and you will be put on a Priority List.

First name: [ ]

Last name: [ ]

Item Ordered: [ ]

Submit
Output

Employee Login

Username: adfsdf
Password: *****

Invalid user name or password, please try again
Error messages


This exception was originally thrown at this call stack:
[External Code]
Design_3_Alex_DiDonato.Program.Main() in Program.cs

View Details | Copy Details | Start Live Share session...

Exception Settings
  □ Break when this exception type is thrown
  Except when thrown from:
    □ Design 3 Alex DiDonato.exe
Open Exception Settings | Edit Conditions

System.Data.NonNullAllowedException: 'Column 'CustFirstName' does not allow nulls.'

This exception was originally thrown at this call stack:
[External Code]
Design_3_Alex_DiDonato.PriorityList.priorityListBindingNavigatorSave
[External Code]

View Details | Copy Details | Start Live Share session...

Exception Settings
  □ Break when this exception type is thrown
  Except when thrown from:
    □ Design 3 Alex DiDonato.exe
Open Exception Settings | Edit Conditions

System.Data.NonNullAllowedException: 'Column 'OrderDate' does not allow nulls.'

This exception was originally thrown at this call stack:
[External Code]
Design_3_Alex_DiDonato.OrderStatus.orderStatusBindingNavigatorSave
[External Code]

View Details | Copy Details | Start Live Share session...

Exception Settings
  □ Break when this exception type is thrown
  Except when thrown from:
    □ Design 3 Alex DiDonato.exe
Open Exception Settings | Edit Conditions
The following exception occurred in the DataGridView:

System.Data.NonNullAllowedException: Column 'Model' does not allow nulls.
  at System.Data.DataColumn.CheckNullable(DataRow row)
  at System.Data.DataTable.RaiseRowChanging(DataRowChangeEventArgs args, DataRow eRow, DataRowAction eAction, Boolean fireEvent)
  at System.Data.DataTable.SetNewRecordWorker(DataRow row, Int32 proposedRecord, DataRowAction action, Boolean isInMerge, Boolean suppressEnsurePropertyChanged, Int32 position, Boolean fireEvent, Exception& deferredException)
  at System.Data.DataTable.InsertRow(DataRow row, Int64 proposedID, Int32 pos, Boolean fireEvent)
  at System.Data.DataView.FinishAddNew(Boolean success)
  at System.Data.DataRowView.EndEdit()
  at System.Windows.Forms.DataGridViewDataGridViewDataConnection.OnRowEnter(DataGridViewCellEventArgs e)

To replace this default dialog please handle the DataError event.
Conclusion

A. I was able to bring most of my vision to reality but some corners had to be cut in order to make some deadlines. I was able to bring the main concepts that I wanted but tiny details had to be cut. Making the employee portal and the website were fine, the main issue I have is integrating the database to the website.

B. The problem that I did not anticipate was the scope of the project as it should have been bigger. I only had the employee side because I was trying to cut corners because I thought it was too big and it was too late to do anything about it.

C. I learned how to connect a database to a website and how to apply the concepts I learned in Systems analysis into this project. This also teaches me time management as I had to balance this project, other classes, and life events that I did not expect to happen.

D. Considering this is only version 1 of my project, there is still a lot more I could have done. For example, I could actually add a login form for the customers and make it more like a shopping site as well. Other minor things that I would worry about are bug fixes and changes to make the program run better.
Users Manual

This manual will show you everything you need to know about using this application, whether an employee or a customer. To use the employee side, enter your username and password, and then hit the log in button. This takes you to the Order Status page where you can see all of the orders that have been placed. From there, you can click on the two buttons down on the bottom that say “Priority List” and “Guitar Inventory”. The Guitar Inventory button takes you to the Guitar Inventory Page and the Priority List takes you to the Priority List. When you go to either of those pages, the format is the same and you can add things to the Guitar Inventory or the Priority List. At the bottom of the two pages, there are also two buttons at the bottom that say “Priority List” and “Order Status” for the Guitar Inventory Page and “Order Status” and “Guitar Inventory” on the Priority List page. All of these buttons function the same, you click on a certain button and they take you to the page that is listed on the button.

Customer Side

For the Customer, They will go to the shop and see 10 guitars that have a buy it now button on all of them. When you click buy it now on any of the guitars, the number below it will go down one as that shows it was purchased. When the amount hits 0 and you click buy it now, it will take you to the OrderOutOfStock page. This is where you will insert your first name, last name, and item you tried to order. When you click submit, a page that says you will be in contact with us shows up, and the order you submitted gets sent to the PriorityList table in the database.