Criterion C: Development

Introduction

This program was made in the C# programming language using the .NET Core framework provided by Microsoft. Visual Studio was the chosen IDE. The program is a database which stores, retrieves, and displays musical chords for use by musicians. The program displays chords using the harmonics of its composite notes.

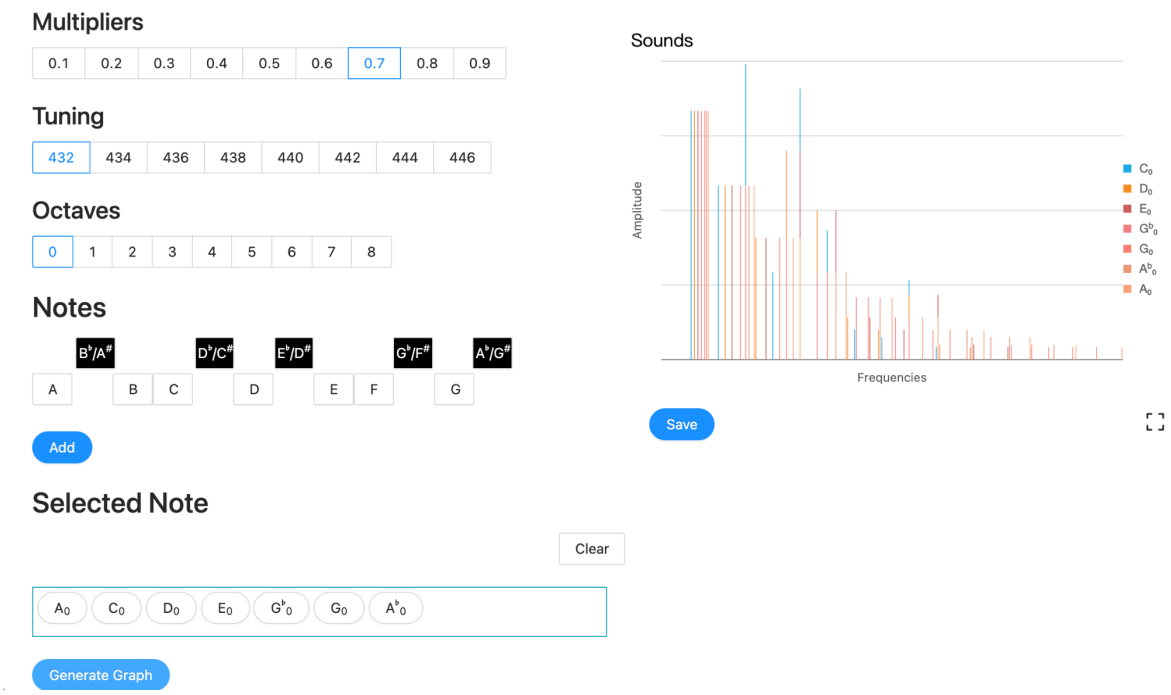
Summary of Programming

Techniques used

- Saving and opening data to and from file
- Encryption
- Error handling
- GUI using html
- Drawing Graph
- Variables
- Methods
- Encapsulation
- Inheritance

Graphical User Interface

Below is the Graphic User Interface of the application which allows users to provide inputs for musical notes in the form of chords. The chords data are analyzed and visually displayed as a graph.



The GUI is designed using HTML code with embedded C# code.

```
<div>
    <AuthorizeView>
        <Authorized>
            <Spin Spinning="@MusicVM.Loading" Size="large">
                <Row>
                    <Col Span="12">
                    <Row>
                        <Col Span="24"><Multipliers
@ref="@MusicVM.CurrentMultipliers"
@bind-Multiplier="@MusicVM.SelectedMultiplier" /></Col>
                    </Row>
                    <Row>
                        <Col Span="24"> </Col>
                    </Row>
                    <Row>
                        <Col Span="24"><Tuning
@ref="@MusicVM.CurrentTuning" @bind-Tune="@MusicVM.SelectedTune" /></Col>
                    </Row>
                    <Row>
                        <Col Span="24"> </Col>
                    </Row>
                    <Row>
                        <Col Span="24"><Octaves
@ref="@MusicVM.CurrentOctaves" @bind-Octave="@MusicVM.SelectedOctave"
/></Col>
                    </Row>
                    <Row>
```

```
                        <Col Span="24"> </Col>
                    </Row>
                    <Row>
                        <Col Span="24"><Notes
@bind-Note="@MusicVM.SelectedNote" OnNoteChanged="@MusicVM.OnNoteChanged"
/></Col>
                    </Row>
                    <Row>
                        <Col Span="24"> </Col>
                    </Row>
                    <Row>
                        <Col Span="24"><SelectedNote
@bind-SelectedList="@MusicVM.SelectedNotes" OnClear="@MusicVM.OnClear"
/></Col>
                    </Row>
                    </Col>
                    <Col Span="12"><ChartOfNote
@ref="@MusicVM.CurrentChart" @bind-visibleZoom="@MusicVM.VisibleChartZoom"
OnSave="@MusicVM.OnSave" @bind-MyNote="@MusicVM.MyNoteName" /></Col>
                </Row>
                <br />
                <Button Type="primary" Shape="round"
OnClick="@MusicVM.OnGenerateGraph">Generate Graph</Button>
            </Spin>
        </Authorized>
    </AuthorizeView>
</div>


<RadioGroup @bind-Value="@multipliers" OnChange="@OnMultiplierChanged"
TValue="double">
    <Radio RadioButton Value="@(0.1)">0.1</Radio>
    <Radio RadioButton Value="@(0.2)">0.2</Radio>
    <Radio RadioButton Value="@(0.3)">0.3</Radio>
    <Radio RadioButton Value="@(0.4)">0.4</Radio>
    <Radio RadioButton Value="@(0.5)">0.5</Radio>
    <Radio RadioButton Value="@(0.6)">0.6</Radio>
    <Radio RadioButton Value="@(0.7)">0.7</Radio>
    <Radio RadioButton Value="@(0.8)">0.8</Radio>
    <Radio RadioButton Value="@(0.9)">0.9</Radio>
</RadioGroup>

<RadioGroup @bind-Value="@tuning" OnChange="@OnTuningChanged" TValue="int">
    <Radio RadioButton Value="@(432)">432</Radio>
    <Radio RadioButton Value="@(434)">434</Radio>
    <Radio RadioButton Value="@(436)">436</Radio>
    <Radio RadioButton Value="@(438)">438</Radio>
    <Radio RadioButton Value="@(440)">440</Radio>
    <Radio RadioButton Value="@(442)">442</Radio>
    <Radio RadioButton Value="@(444)">444</Radio>
    <Radio RadioButton Value="@(446)">446</Radio>
</RadioGroup>
```

```
<RadioGroup @bind-Value="@octaves" OnChange="@OnOctaveChanged"
TValue="int">
    <Radio RadioButton Value="@(0)">0</Radio>
    <Radio RadioButton Value="@(1)">1</Radio>
    <Radio RadioButton Value="@(2)">2</Radio>
    <Radio RadioButton Value="@(3)">3</Radio>
    <Radio RadioButton Value="@(4)">4</Radio>
    <Radio RadioButton Value="@(5)">5</Radio>
    <Radio RadioButton Value="@(6)">6</Radio>
    <Radio RadioButton Value="@(7)">7</Radio>
    <Radio RadioButton Value="@(8)">8</Radio>
</RadioGroup>

<Row class="mx-1 my-1">
    <div Class="col-1"></div>
    <Button Class="col-1 sharp" OnClick="@(e=> this.OnSelectedNote(new
CustomEvent<MouseEventArgs,
string>(e,"Bb")))">B<sup>♭</sup>/A<sup>#</sup></Button>
    <div Class="col-1"></div>
    <div Class="col-1"></div>
    <Button Class="col-1 sharp" OnClick="@(e=> this.OnSelectedNote(new
CustomEvent<MouseEventArgs,
string>(e,"Db")))">D<sup>♭</sup>/C<sup>#</sup></Button>
    <div Class="col-1"></div>
    <Button Class="col-1 sharp" OnClick="@(e=> this.OnSelectedNote(new
CustomEvent<MouseEventArgs,
string>(e,"Eb")))">E<sup>♭</sup>/D<sup>#</sup></Button>
    <div Class="col-1"></div>
    <div Class="col-1"></div>
    <Button Class="col-1 sharp" OnClick="@(e=> this.OnSelectedNote(new
CustomEvent<MouseEventArgs,
string>(e,"Gb")))">G<sup>♭</sup>/F<sup>#</sup></Button>
    <div Class="col-1"></div>
    <Button Class="col-1 sharp" OnClick="@(e=> this.OnSelectedNote(new
CustomEvent<MouseEventArgs,
string>(e,"Ab")))">A<sup>♭</sup>/G<sup>#</sup></Button>
</Row>
<Row class="my-1">
    <Button Class="col-1" OnClick="@(e=> this.OnSelectedNote(new
CustomEvent<MouseEventArgs, string>(e,"A")))">A</Button>
    <div Class="col-1"></div>
    <Button Class="col-1" OnClick="@(e=> this.OnSelectedNote(new
CustomEvent<MouseEventArgs, string>(e,"B")))">B</Button>
    <Button Class="col-1" OnClick="@(e=> this.OnSelectedNote(new
CustomEvent<MouseEventArgs, string>(e,"C")))">C</Button>
    <div Class="col-1"></div>
    <Button Class="col-1" OnClick="@(e=> this.OnSelectedNote(new
CustomEvent<MouseEventArgs, string>(e,"D")))">D</Button>
    <div Class="col-1"></div>
    <Button Class="col-1" OnClick="@(e=> this.OnSelectedNote(new
CustomEvent<MouseEventArgs, string>(e,"E")))">E</Button>
```

```
    <Button Class="col-1" OnClick="@(e=> this.OnSelectedNote(new
CustomEvent<MouseEventArgs, string>(e,"F")))">F</Button>
    <div Class="col-1"></div>
    <Button Class="col-1" OnClick="@(e=> this.OnSelectedNote(new
CustomEvent<MouseEventArgs, string>(e,"G")))">G</Button>
    <div Class="col-1"></div>
</Row>

<h3>Selected Note</h3>
<Row>
    <Col Span="22" />
    <Col Span="2">
    <Button OnClick="Clear">Clear</Button>
    </Col>
</Row>
```

Variables

Global and private variables are used throughout the program. Public variables are accessible by all other classes, whereas private variables have to be retrieved or changed using access methods: set, get.

```
Global variables

public const string TokenKey = "asdf##@~#@3!@#$%^&*(ksfdsanjklskjf32";

public const string HashKey = "TP@ssw0rd";

public const string Issuer = "MusicDatabase API";

public const string Audience = "Music Database";


Private variables

    public class User
    {
        public string UserName { get; set; }

        public string FirstName { get; set; }

        public string LastName { get; set; }

        public string Password { get; set; }
    }

    public class SavedNote
```

```
    {
        public string UserName { get; set; }

        public double Multiplier { get; set; }

        public int Tune { get; set; }

        public string NoteName { get; set; }

        public List<string> Notes { get; set; }

        public DateTime CreatedAt { get; set; }
    }
```

Local variables are used to keep track of values used in local processes, like counting the number of music notes, plotting charts.

```
Local variable example

    private string NoteName; //to store name of the music notes

    private bool visibleSave; //to manipulate save status

    private PlotData[] data1; // prepare data to plot chart

    private List<PlotChart> plots; //array of chart data
```

Methods

The main methods used in this program are saving and retrieving data from files, token allocation and identification, and encryption.

This method is used to store the data input from the user in the json CSV format. The stored data type is a Chord which is associated with specific users, this means that different users will only be able to access their data. The data is retrieved in this manner according to the token code below.

```
//Retrieving/Saving data to file in json format

public async Task CreateSavedNotesFile(SaveNotesRequest request, string
userName, string path)
        {
            SavedNotesContext savedNotesDto = new SavedNotesContext();
            savedNotesDto.SavedNotes.Add(new SavedNote
            {
                UserName = userName,
                Multiplier = request.Multiplier,
                Tune = request.Tune,
                NoteName = request.NoteName,
                Notes = request.Notes,
                CreatedAt = DateTime.UtcNow
            });

            await File.WriteAllTextAsync(path,
JsonConvert.SerializeObject(savedNotesDto, Formatting.Indented));
        }

public async Task UpdateSavedNotesFile(SaveNotesRequest request, string
userName, string path, SavedNotesContext context)
        {
            context.SavedNotes.Add(new SavedNote
            {
                UserName = userName,
                Multiplier = request.Multiplier,
                Tune = request.Tune,
                NoteName = request.NoteName,
                Notes = request.Notes,
                CreatedAt = DateTime.UtcNow
            });

            await File.WriteAllTextAsync(path,
JsonConvert.SerializeObject(context, Formatting.Indented));
        }
```

This is the method which generates a token for the user when they log into the program. The program uses this token to verify users' requests for information as well as identify their identity.

```
Token allocation and verification
public string GenerateToken(string userName)
        {
            var tokenHandler = new JwtSecurityTokenHandler();
            var key = Encoding.ASCII.GetBytes(CommonConst.TokenKey);
            var tokenDescriptor = new SecurityTokenDescriptor
            {
                Subject = new ClaimsIdentity(new[]
                    {
                       new Claim(ClaimTypes.Name,userName)
                    }),
                Expires = DateTime.Now.AddDays(7),
                Issuer = CommonConst.Issuer,
                Audience = CommonConst.Audience,
                SigningCredentials = new SigningCredentials(new
SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)
            };

            var JWTToken = tokenHandler.CreateToken(tokenDescriptor);
            var token = tokenHandler.WriteToken(JWTToken);
            return token;
        }
```

Inheritance

Inheritance was used to reduce repeated code in the program. Below is an example of inheritance.

```
public class NotesController : BaseApiController
    {
        private readonly INotesService NotesService;

        public NotesController(INotesService notesService)
        {
            this.NotesService = notesService;
        }

        [HttpPost]
        [Route("save")]
        public async Task<IActionResult> SaveNotes([FromBody]
SaveNotesRequest request)
        {
            var response = await NotesService.SaveNotes(request,
this.User.Identity.Name);

            switch (response.Code)
            {
                case HttpStatusCode.Created:
                    return StatusCode((int)HttpStatusCode.Created);
```

```
                case HttpStatusCode.BadRequest:
                    return BadRequest();
                default:
                    return
StatusCode((int)HttpStatusCode.InternalServerError);
            }
        }

        [HttpGet]
        [Route("load")]
        public async Task<IActionResult> GetNotes()
        {
            var response = await
NotesService.GetNotes(this.User.Identity.Name);

            switch (response.Code)
            {
                case HttpStatusCode.OK:
                    return Ok(response);
                case HttpStatusCode.BadRequest:
                    return BadRequest();
                default:
                    return
StatusCode((int)HttpStatusCode.InternalServerError);
            }
        }
    }
```

Encryption

These methods are responsible for the encryption used in this program by way of a hash. The only use of encryption are the passwords of the users. Most of the code references imported libraries.

```
public static byte[] GetComputeHash(this string input, eHashType hashType,
Encoding encoding)
        {
            try
            {
                return GetHash(input, hashType, encoding);
            }
            catch
            {
                return null;
            }
        }

        private static byte[] GetHash(string input, eHashType hash,
Encoding encoding)
        {
```

```
            byte[] inputBytes = encoding.GetBytes(input);

            switch (hash)
            {
                case eHashType.MD5:
                    return MD5.Create().ComputeHash(inputBytes);

                case eHashType.SHA1:
                    return new SHA1Managed().ComputeHash(inputBytes);

                case eHashType.SHA256:
                    return new SHA256Managed().ComputeHash(inputBytes);

                case eHashType.SHA384:
                    return new SHA384Managed().ComputeHash(inputBytes);

                case eHashType.SHA512:
                    return new SHA512Managed().ComputeHash(inputBytes);
                default:
                    return inputBytes;
            }
        }

        private static KeyedHashAlgorithm GetKeyHashAlgorithm(eKeyHashType
hashType, byte[] keyByte)
        {
            switch (hashType)
            {
                case eKeyHashType.HMACMD5:
                    return new HMACMD5(keyByte);

                case eKeyHashType.HMACSHA1:
                    return new HMACSHA1(keyByte);

                case eKeyHashType.HMACSHA256:
                    return new HMACSHA256(keyByte);

                case eKeyHashType.HMACSHA384:
                    return new HMACSHA384(keyByte);

                case eKeyHashType.HMACSHA512:
                    return new HMACSHA512(keyByte);

                default:
                    return new HMACSHA1(keyByte);
            }
        }

        public static byte[] GetComputeHMAC(string data, string secreteKey,
eKeyHashType hashType, Encoding encoding)
        {
            if (encoding == null)
```

```
                encoding = Encoding.UTF8;
            return GetKeyHashAlgorithm(hashType,
encoding.GetBytes(secreteKey)).ComputeHash(encoding.GetBytes(data));
        }
```

Error Handling

There are many errors which could happen in any of the many functions within this program. Below is an example of how the program may catch such an error and throw an exception.

```
//Error handling examples

protected TResponse OnException<TResponse>(Exception exception, string
fuction) where TResponse : BaseResponse, new()
        {
            this.ExecuteLog($"{fuction} Error : {exception}",
LogLevel.Error);

            return new TResponse();
        }


//Function specific error handling

public async Task<IActionResult> SaveNotes([FromBody] SaveNotesRequest
request)
        {
            RequestLog(request);

            var response = await NotesService.SaveNotes(request,
this.User.Identity.Name);

            ResponseLog(response);

            switch (response.Code)
            {
                case HttpStatusCode.Created:
                    return StatusCode((int)HttpStatusCode.Created);
                case HttpStatusCode.BadRequest:
                    return BadRequest();
                default:
                    return
                    StatusCode((int)HttpStatusCode.InternalServerError);
            }
        }
```

GUI

Drawing the Graph

Below is the HTML and C# code responsible for displaying Chords as loaded from the json file.

```
//ChartOfNote.razor

@code{
    private string NoteName;

    private bool visibleSave;

    private PlotData[] data1;

    [Parameter]
    public string visibleZoom { get; set; }

    [Parameter]
    public EventCallback<string> visibleZoomChanged { get; set; }

    [Parameter]
    public string MyNote { get; set; }

    [Parameter]
    public EventCallback<string> MyNoteChanged { get; set; }

    [Parameter]
    public EventCallback<MouseEventArgs> OnSave { get; set; }

    private StackedColumn<PlotData> stackedColumn;

    private StackedColumn<PlotData> stackedColumn2;

    private StackedColumnConfig config1 = new StackedColumnConfig
    {
        Title = new Title
        {
            Visible = true,
            Text = "Sounds"
        },
        ForceFit = true,
        Padding = "auto",
        XField = "frequency",
        YField = "wavelength",
        YAxis = new ValueAxis
        {
            Visible = true,
            Label = new BaseAxisLabel { Visible = false }
        },
```

```
        Label = new ColumnViewConfigLabel
        {
            Visible = false
        },
        XAxis = new CatAxis
        {
            Visible = true,
            Label = new BaseAxisLabel { Visible = false }
        },
        Color = new[] { "#1ca9e6", "#f88c24", "#cd5c5c", "#f08080",
"#fa8072", "#e9967a", "#ffa07a", "#dfff00", "#ffbf00", "#de3163",
"#9fe2bf", "#40e0d0" },
        Meta = new
        {
            Frequency = new
            {
                Alias = "Frequencies"
            },
            Wavelength = new
            {
                Alias = "Amplitude"
            }
        },
        Legend = new Legend { Position = "right" },
        StackField = "legend"
    };

    protected override void OnInitialized()
    {
        visibleZoomChanged.InvokeAsync("collapse");
    }

    public void GenerateGraph(List<PlotData> data)
    {
        data1 = data.OrderBy(d => d.Frequency).ToArray();

        this.stackedColumn.ChangeData(data1);
        if (this.stackedColumn2 != null)
        {
            this.stackedColumn2.ChangeData(data1);
        }

        visibleZoomChanged.InvokeAsync("");
    }

    public void Clear()
    {
        data1 = new PlotData[] { };
        this.stackedColumn.ChangeData(data1);

        if (this.stackedColumn2 != null)
        {
```

```
            this.stackedColumn2.ChangeData(data1);
        }

        visibleZoomChanged.InvokeAsync("");
    }

    private async Task Zoom()
    {
        RenderFragment content =@<div><StackedColumn @ref="stackedColumn2"
Data="data1" Config="config1" /></div>;

        var options = new ConfirmOptions
        {
            Width = 1360,
            Content = content,
            OkCancel = false,
            OkText = "Close",
            MaskClosable = false,
            OnOk = e => { Console.WriteLine("OnOk"); return
Task.CompletedTask; },
            OnCancel = e => { Console.WriteLine("OnCancel"); return
Task.CompletedTask; },
        };

        var confirmRef = await _modalService.CreateConfirmAsync(options);
    }

    private void Save(MouseEventArgs e)
    {
        visibleSave = true;
    }

    private void HandleOk(MouseEventArgs e)
    {
        this.MyNoteChanged.InvokeAsync(this.NoteName);
        this.OnSave.InvokeAsync(e);
        visibleSave = false;
    }

    private void HandleCancel(MouseEventArgs e)
    {
        visibleSave = false;
    }
}
```

Saved Chords List

This code is responsible for displaying all the saved chords of the user upon their request.

MyNote.razor

```
<Table TItem="GetNotes" DataSource="@Notes" Context="note"
SelectedRowsChanged="OnSelectedRow" PageSize="5">
    <Selection Key="@note.NoteName" Type="radio" />
    <Column @bind-Field="note.NoteName" Sortable />
    <Column @bind-Field="note.Notes">
        @foreach (var item in note.Notes)
        {
            if (item?.Length == 2)
            {
                <Button
Shape="round">@item.ElementAtOrDefault(0)<sub>@item.ElementAtOrDefault(1)</
sub></Button>
            }

            if (item?.Length == 3)
            {
                <Button
Shape="round">@item.ElementAtOrDefault(0)<sup>♭</sup><sub>@item.ElementAtO
rDefault(2)</sub></Button>
            }
        }
    </Column>
    <Column @bind-Field="note.Multiplier" Sortable/>
    <Column @bind-Field="note.CreatedAt" Sortable />
</Table>
@code {

    [Parameter]
    public List<GetNotes> Notes { get; set; }

    [Parameter]
    public EventCallback<List<GetNotes>> NotesChanged { get; set; }

    [Parameter]
    public IEnumerable<GetNotes> SelectedRows { get; set; }

    [Parameter]
    public EventCallback<IEnumerable<GetNotes>> SelectedRowsChanged { get;
set; }

    private void OnSelectedRow(IEnumerable<GetNotes> selectedRow)
    {
        this.SelectedRowsChanged.InvokeAsync(selectedRow);
    }
}
```

Graph Comparisons

Here is the code which displays a list of chords which can be selected to be compared.

```
MyNoteCompare.razor

<Table TItem="GetNotes" DataSource="@Notes" Context="note"
SelectedRowsChanged="OnSelectedRow" PageSize="5">
    <Selection Key="@note.NoteName" />
    <Column @bind-Field="note.NoteName" Sortable />
    <Column @bind-Field="note.Notes">
        @foreach (var item in note.Notes)
        {
            if (item?.Length == 2)
            {
                <Button
Shape="round">@item.ElementAtOrDefault(0)<sub>@item.ElementAtOrDefault(1)</
sub></Button>
            }

            if (item?.Length == 3)
            {
                <Button
Shape="round">@item.ElementAtOrDefault(0)<sup>♭</sup><sub>@item.ElementAtO
rDefault(2)</sub></Button>
            }
        }
    </Column>
    <Column @bind-Field="note.Multiplier" Sortable />
    <Column @bind-Field="note.CreatedAt" Sortable />
</Table>
@code {

    [Parameter]
    public List<GetNotes> Notes { get; set; }

    [Parameter]
    public EventCallback<List<GetNotes>> NotesChanged { get; set; }

    [Parameter]
    public IEnumerable<GetNotes> SelectedRows { get; set; }

    [Parameter]
    public EventCallback<IEnumerable<GetNotes>> SelectedRowsChanged { get;
set; }

    private void OnSelectedRow(IEnumerable<GetNotes> selectedRow)
    {
        this.SelectedRowsChanged.InvokeAsync(selectedRow);
    }
}
```

(The full source code can be found in the appendix)