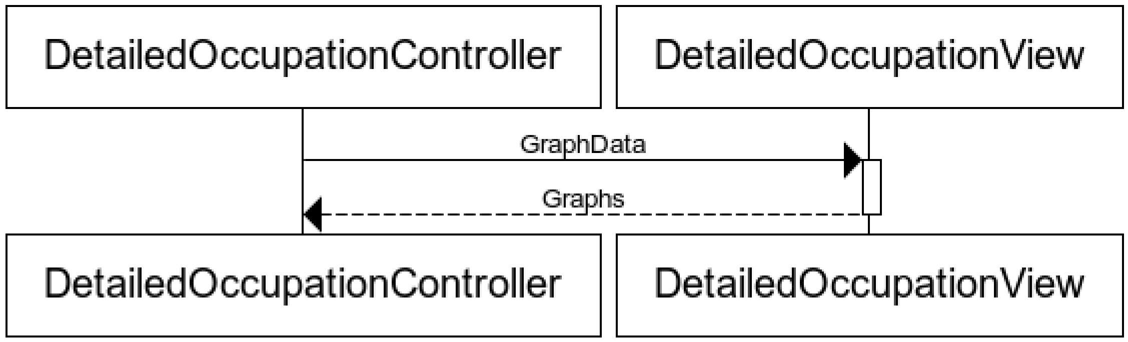
Adam Howard

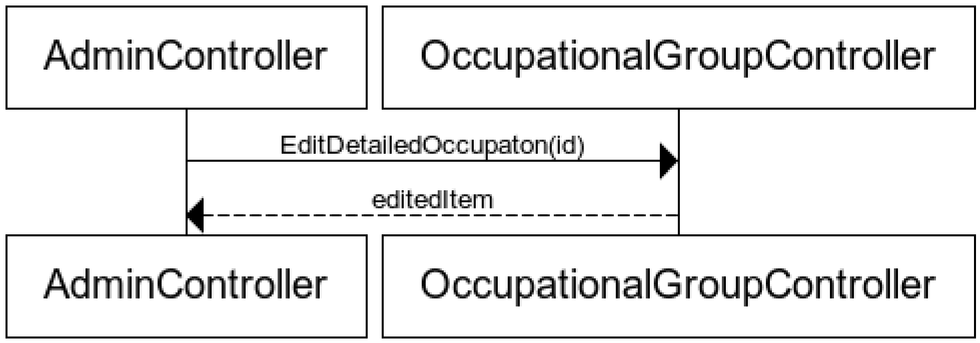
Design Document

Interaction Diagrams

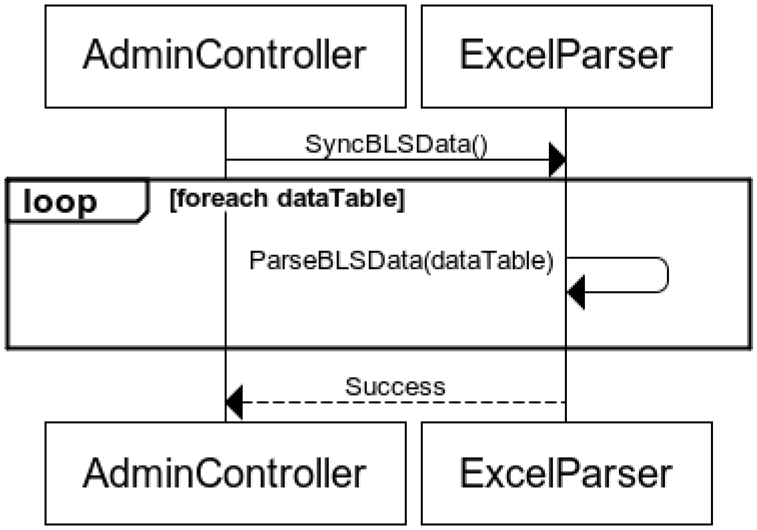
View Job Outlook Data Graphically



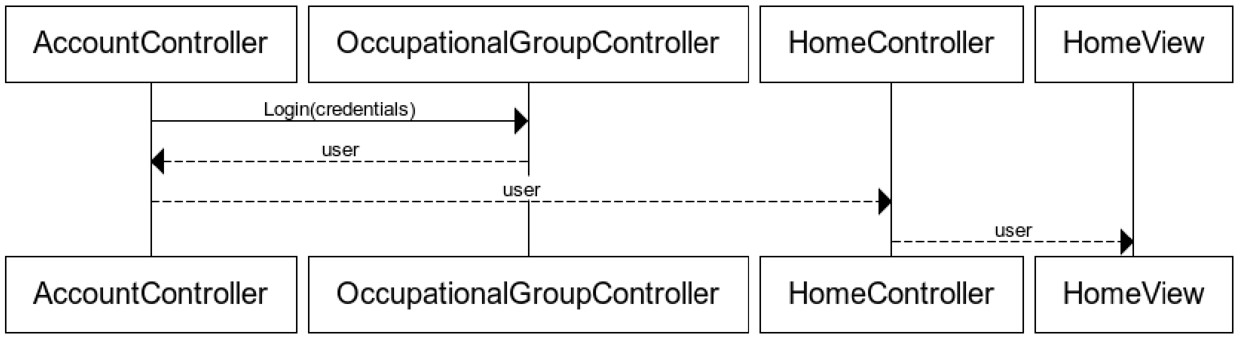
Manage Bureau of Labor Statistics Data


AdminController
OccupationalGroupController
EditDetailedOccupaton(id)
editedItem
AdminController
OccupationalGroupController

Import Bureau of Labor Statistics Data



Login Using Third-Party Authentication



Class Diagram

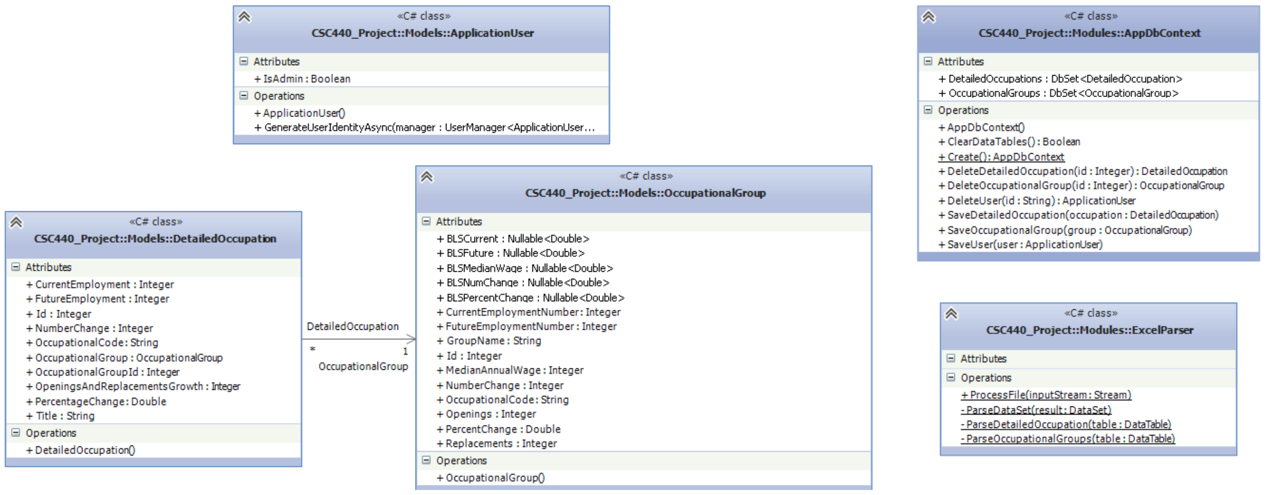

«C# class»
CSC440_Project::Models::ApplicationUser
Attributes
+ IsAdmin : Boolean
Operations
+ ApplicationUser()
+ GenerateUserIdentityAsync(manager : UserManager<ApplicationUser…
«C# class»
CSC440_Project::Modules::AppDbContext
Attributes
+ DetailedOccupations : DbSet<DetailedOccupation>
+ OccupationalGroups : DbSet<OccupationalGroup>
Operations
+ AppDbContext()
+ ClearDataTables() : Boolean
+ Create() : AppDbContext
+ DeleteDetailedOccupation(id : Integer) : DetailedOccupation
+ DeleteOccupationalGroup(id : Integer) : OccupationalGroup
+ DeleteUser(id : String) : ApplicationUser
+ SaveDetailedOccupation(occupation : DetailedOccupation)
+ SaveOccupationalGroup(group : OccupationalGroup)
+ SaveUser(user : ApplicationUser)
«C# class»
CSC440_Project::Models::DetailedOccupation
Attributes
+ CurrentEmployment : Integer
+ FutureEmployment : Integer
+ Id : Integer
+ NumberChange : Integer
+ OccupationalCode : String
+ OccupationalGroup : OccupationalGroup
+ OccupationalGroupId : Integer
+ OpeningsAndReplacementsGrowth : Integer
+ PercentageChange : Double
+ Title : String
Operations
+ DetailedOccupation()
DetailedOccupation
*
1
OccupationalGroup
«C# class»
CSC440_Project::Models::OccupationalGroup
Attributes
+ BLSCurrent : Nullable<Double>
+ BLSFuture : Nullable<Double>
+ BLSMedianWage : Nullable<Double>
+ BLSNumChange : Nullable<Double>
+ BLSPercentChange : Nullable<Double>
+ CurrentEmploymentNumber : Integer
+ FutureEmploymentNumber : Integer
+ GroupName : String
+ Id : Integer
+ MedianAnnualWage : Integer
+ NumberChange : Integer
+ OccupationalCode : String
+ Openings : Integer
+ PercentChange : Double
+ Replacements : Integer
Operations
+ OccupationalGroup()
«C# class»
CSC440_Project::Modules::ExcelParser
Attributes
Operations
+ ProcessFile(inputStream : Stream)
- ParseDataSet(result : DataSet)
- ParseDetailedOccupation(table : DataTable)
- ParseOccupationalGroups(table : DataTable)

Diagram Explanation

The class diagram above implements several GRASP patterns. These patterns are described below, along with the associated reasoning for implementing each pattern.

The first pattern that was used in the design of the application was the Information Expert pattern. This is used in the Job Outlook Portal by allowing classes to handle the data that they are most interested in. For example, the OccupationalGroup and DetailedOccupation groups both are concerned primarily with managing and creating instances of their own class. All of the maintenance that goes into an OccupationalGroup or DetailedOccupation object are done by the class itself. This allows for straightforward updating and maintenance of code.

Secondly, the Creator pattern is used in the Job Outlook Portal. As shown in the Class Diagram, all of the classes are responsible for creating instances of themselves. It is the sole responsibility of the class to create instances of itself and because of this it is always known where classes are instantiated and initialized.

Thirdly, the Controller pattern is used heavily in the Job Outlook portal. Because the application uses the .NET Model View Controller pattern, using controllers is straightforward and encouraged. The controller pattern allows for each model class to have an associated controller classes. Along with several views for each model, this allows for straightforward coding, adequate separation of concerns, and increased ease of unit testing. By allowing one class to maintain responsibility of business logic, the centralization of code is greatly increased, and unit testing can be performed far more easily than with traditional web applications.

The sequence diagrams were concerned with similar principles in mind. Each transaction with the system is atomic and as simple as logically possible. This means that if errors occur in the application, their root cause can be determined fairly quickly, as most methods rely on the single-responsibility guideline, and never handle more than a very specific task. Most methods and actions in the Job Outlook Portal operate in a similar fashion.

In future iterations, it would be wise to introduce interfaces and various other design patterns to help with unit testing and expanding the project. For example, the application data context could use the Singleton pattern to ensure only one data context existed for the lifetime of the current application execution. Also using a method like the Factory pattern would help increase the use of polymorphism in the application.