# Scenario 1.

Users utilize an order list website to select an order number and begin working on it.  Once a user selects an order, they are required to make field value changes to it and resubmit the order for approval.

For this example, consider the following order data to begin with:

Order ID: 1234

First Name: Fred

Last Name: Jones

DOB: 1/1/2001

City: Chicago

State: Illinois

Problem:

•        User 1 selects order 1234 from the list to begin making modifications

•        User 2 selects the same order and also begins making modifications

•        User 1 then changes only the first name on the order from “Fred” to “Tom”

•        User 1 submits the modifications back into the system for approval by their supervisor

•        User 2 then changes only the DOB on the order from “1/1/2001” to “1/1/1999”

•        User 2 submits their changes to the order back into the system for approval by their supervisor

The resulting order data is as follows:

Order ID: 1234

First Name: Fred

Last Name: Jones

DOB: 1/1/1999

City: Chicago

State: Illinois

The supervisor rejected the modifications because the order contained the wrong first name.

Questions:

## 1)     What type of errors in the application would you look for to figure out why this error occurred?

Since neither user 1 or user 2 reported seeing an error in the website, this appears that it may be a symptom of two users saving the same record on a multi-user application without concurrency.

To confirm my theory, I would start by:

1. Looking at any available timestamp, history or log tables within the database to see if could reconstruct the exact order in which the data manipulation took place as well discover any errors the application recorded.
2. Check an application log file and / or the event log on the server. If no verifiable information was available, then I would
3. Conduct a user assisted test in dev to see if I could reproduce the behavior or discover additional errors.

Since this truly does sound like a design issue, rather than an actual bug I will assume for this exercise that I was able to reproduce the behavior.

## 2)     What could be done to resolve the issue?

There are a variety of different ways to address concurrency, and the method I would choose depends completely on how the application handles data interaction and the common use cases

The first step would be to have a conversation with the supervisor, and perhaps a peer if there is one, to determine how often two users usually edit the same order. For this exercise, let’s say that this is part of the normal use case, and happens often.

Since it can happen frequently, we probably don’t want to implement pessimistic concurrency ( lock the record ), because frequent locks can reduce performance, it requires and active database connection and the user experience wouldn’t be great. The would get to the point where they try to save a record and wouldn’t be able to.  We also know that this is a high priority issue, because data could be incorrect on multiple orders and may only be caught when a human recognizes that the data is incorrect.

The best method to correct the issue then, is to find a way to implement optimistic concurrency—basically check a value, rowversion or a timestamp on a record to make sure it hasn’t been changed since data was initially retrieved.

# Scenario 2:

To help combat the issue with the orders being overwritten, it is decided that an Order Queue project should be developed.  The following are the basic requirements for the project:

•        Each user can select their next order to work on from a queue of available orders to work on

•        Once an order is selected by one user, it cannot be selected by any other user

•        Once an order is selected, it no longer appears within the order queue

•        Once a user finishes making modifications to an order, they submit the order to their supervisor for approval.

•        Each user is able to see a listing of orders they specifically modified that are awaiting approval from their supervisor

Questions:

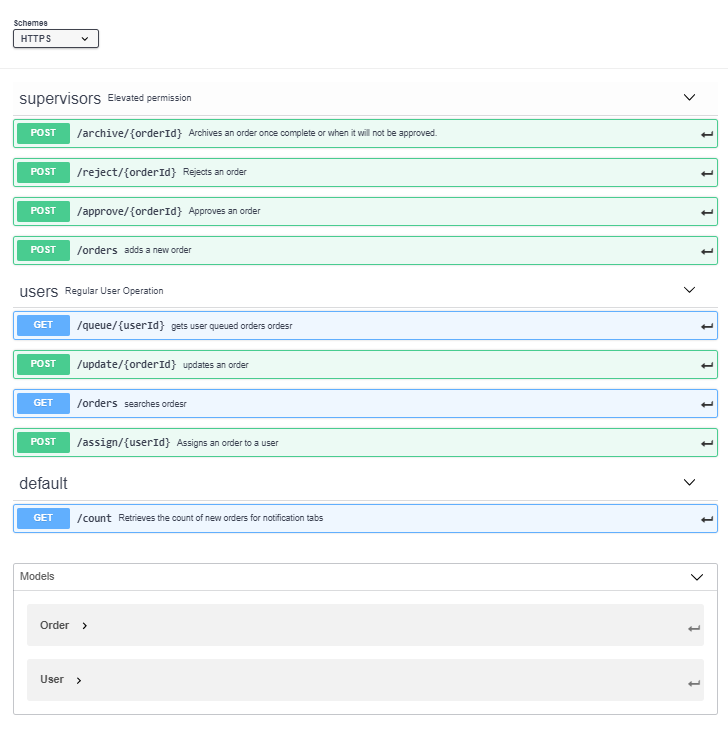
## 1)     Describe how a services based architecture could be used to implement the Order Queue

This could be accomplished by splitting the whole application into separate client and services applications. The Server could be a web API using .NET Core with EntityFramework. For the client, we could use Angular or React to create a single page application, or work with the standard Bootstrap libraries and build out the view in a multi-page application.

The Client would make calls to the server application and provide and interface for analyzing and manipulating data.

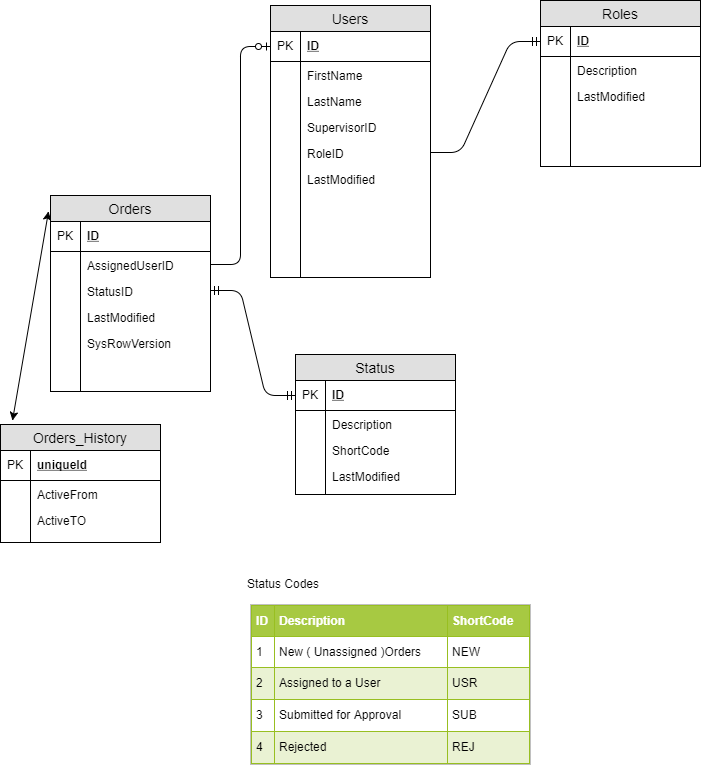
### What endpoints are needed? (include naming examples):

Here is an example of possible endpoints created in swagger:



## 2)     Describe potential database structures and relations that would be needed to accomplish the Order Queue

Below is a minimalistic diagram of the data structures we would need. It’s assumed that there would be a lot more needed for the full system, but these would be needed to accomplish the requirements. One thing, the Orders\_History table would not required, but having history tables on these types of entities has proven very useful to me in the past.



## 3)     Are there any other solutions besides a queueing service that could solve the core issue?

In a Word, Yes.

**Premise and Assumptions:**

In a perfect world, we would be able to implement concurrency in EntityFramework by adding a rowversion column and configuring EF. Then, we could handle DBUpdateConcurrencyExceptions on update or delete actions to display the updated information to the second user in the scenario. Finally, we give the user an option of how to proceed. This, however, could be a very large and sweeping change in an application that isn’t already configured this way. It’s probably not our best option, since this could take some time to correctly implement across the system, and this issue could have wide spread effects on data integrity.

I do not believe I would have recommended a full rewrite, or a separate supplementary application. Below, I will make a lot of assumptions, but this is how I would propose dealing with Optimistic Concurrency from Scenario 1.

How does the application interact with the database? Is it using EntityFramework, calling Stored Procs from code to Get / Update datasets, or \*cringe\* building SQL Queries in line and sending them to the database? Was a wrapper class created to handle data interaction?

For this exercise, let’s assume we are dealing with an older WebForms Application that has a custom data access class written to handle connecting to the Database, retrieve DataSets and update data via table adapter methods.

Also, I’m assuming that Primary Keys in each tables are called ID, in a similar fashion to EntityFramework Standards

**User Story:**

Users will edit records exactly how they did before:

1. User clicks edit on the Order, and controls appear with the data from the database.
2. Users manipulate the data
3. Users Click Submit to save changes
   1. If the record has changed since the user click edit in step 1:
      1. The Edit Form is displayed again, but two sets of controls are there.
      2. 1 set displays the modified data from Step 2.
      3. The other set displays the current values in the database
      4. Differences are highlighted.
      5. The user can select which fields they want to save, and submit changes or, they can cancel.
4. When a user modify the order, their user id is saved on the Order, and this is displayed on the main order screen to prevent unintentional editing of another user’s work.

**Design**

A new stored procedure will dynamically create a simple query to return the LastModifiedDate of a given table and ID. In our application, we will evaluate this date when attempting to submit changes to see if it is different from the LastModifiedDate we retrieved from the database when beginning the edit.

If it is different, we will notify the user and load a revised Edit view, that will show a side by side comparison of the dataset the user tried to submit, next to the current values with differences highlighted.

At this point, the user will be able to select the changes they want to keep, and submit, or cancel.

This design has several pros and cons. First, SQL Server won’t be able to optimize the stored proc that’s being executed, and we may lose some performance over just parameterizing the query in C#. This will be mitigated, because we will always be selecting a single value using an index in our where clause. Secondly, I’m assuming for this scenario that the standard is to use ID as the PK. If any table did not use ID as a primary key, has a concatenated key or does not have a key, the proc as coded will not work.

However, I do believe this design’s benefits outweigh the negatives. With minimal changes to the existing application, risk it mitigated on breaking the app somewhere else. It is scalable and it can be applied to other tables with minimal effort as time allows. It can implemented fast, which reduces cost and increases user confidence and approval.

**Work Items**

**Database Changes:** ( See Attached SQL Script )

* Add Timestamp column
* New Stored Proc: GetLastModifiedDate
  + Parameters
    - Table Name
    - ID
  + Returns
    - LastModifiedDate of ID ( record ) in Table
  + Description
    - This Stored Proc generates a sql statement and uses sp\_executesql to return the LastModifiedDate from the given ID and Table name. See Attachement
    - 

**Code Changes**

**In Data Access Class**

**See attachment for example and in line documentaion**

****

* New Property: HasConcurrencyIssue ( boolean )
* New Method: HasRecordChanged
  + Parameters
    - ID ( int64 )
    - tableName ( string )
    - OriginalLastModDate ( DateTime )
      * This is the date that was retrieved from the HTTPGET Edit Action
  + Returns
    - Bool
  + Called by SaveData()
  + Logic
    - The method will return true if the Current LastModified Date for the specified record does not match the OriginalLastModDate.
* Edit or Create New Polymorphic Method: SaveData
  + Parameters
    - I’m assuming the the current SaveData method already has something like these parameters:
      * queryOrProc ( string )
      * params ( SqlParamater Array )
    - New Parameter:
      * CheckConcurrency ( boolean, optional – default = false )
      * In order to preserve exist functionality, we can use Polymorphism, and create a different function to get a DataSet with a different signature ( like passing in the LastModifiedDate retrieved from the Database
  + Returns
    - Boolean ( successful flag )
  + Additional Logic
    - IF CheckConcurrency == true
      * Call HasRecordChanged for the record In question
        + IF HasRecordChanged == true

Return

**In Edit Page**

* In Method attempting to save:
  + If DataAccess Object has concurrency error
    - Redirect to new Edit Page, and pass in two sets of data:

         1 set to represent the modified data in the database

         1 set to represent the changes made by the user.

  Flag fields that are different with a css classes

**New Edit Page**

* + Base off existing edit page
  + Add controls to select any combination of fields from either dataset to pass as intended values.
  + Submit button. ( this will call the SaveData method with CheckConcurancy flag set to false

**Recap:**

There are many other more elegant ways to implement concurrency, but we have to look at the cost, risk and benefit of large changes to an existing legacy application.

With legacy applications, there are often small variations or breaks in seemingly unconnected areas of code when you change one line of code. It’s something I refer to as the spider web affect. Touch a spider web on the one line that attaches to its framework, and you can visibly see the web quiver in its entirety. The older the system, the more tendency it has to have solutions bolted on with little to no documentation. Sweeping changes can bring about drastic affects if they are rushed.

With that being said, I think something like this could be completed in a week or so by one developer. Designing a larger scale system for queues, while having its benefits would take more than one developer a few weeks a least. Additionally, with the data already in place we could create new pages or filters to view the grid by user, in effect created the “user queue”.