***Please do not include your name or any information that could identify you, as doing so will result in disqualification.***

# Requirements

**R1.** Company XYZ needs an online invoicing system to manage the **invoices** of its **employees**.

**R2**. The company's invoicing system is managed by designated **invoice managers** who are also **employees** of the company.

**R3**. An employee is identified by their **first** and **last** **name** as well as with a unique **employee identifier**.

**R4**. Every employee has a **contract** and each contract has a **validity period** specified as a date range. The contract is not valid outside of the specified validity period.

**R5**. Every employee's **daily pay rate** is defined in their contract. Once their contract ends, an employee is given a new contract with a new validity period. An employee cannot have more than one valid contract at a time – validity periods of two contracts for the same employee should not overlap.

**R6**. Every employee submits **invoices** in order to get paid. The invoice specifies the **invoicing period** as a date range, the **number of days** worked during the period, as well as the **total amount** invoiced**. For simplicity, let's assume that every employee only works full days and not fractions.**

**R7**. The system accepts only invoices from employees with a contract which is valid for the entire invoicing period. I.e. if the invoicing period specified in the invoice is from March 5 until March 27, the employee must have a contract with a start date of at least March 5 and an end date on or after March 27.

**R8**. The system should also check whether the number of days invoiced fits within the invoice period. If the invoice specifies 30 days worked while the invoice period is from March 5 until March 27, this should fail the validation test. **For simplicity, let’s assume that all days in the invoice period are billable at the same rate including weekends and holidays. No special treatment for weekends or holidays is necessary.**

**R9**. The system must also validate the **total amount invoiced** against the **daily rate** specified in the employee's contract matching the **invoice period**. If the daily rate in the contract is USD 100 and the number of days invoiced is 10, the system should not accept an invoice for an amount other than USD 1000.

**R10**. An employee can submit only one **invoice** in any billing period. Before accepting an invoice, the system must check that no other invoice was already submitted by that employee, overlapping fully or partially with the invoicing period of this invoice.

**IMPORTANT** As a general rule, you are not required to cater for all possible scenarios and complications a real-life invoicing system may need to handle. Make sure that your code validates only against the rules specifically described above and don't worry about edge cases and other possible real-life scenarios which you may think of that are not explicitly mentioned above. This test is not about billing, but about software development and architecture.

# DELIVERABLES

## 1. Database design – 30 points

Please design the database diagram for the invoicing system. Make sure to include all relevant entities with their relationships, properties, to account for all the requirements listed above. You can either draw the diagram or present in the form of DDL. SQL syntax doesn’t need to be exact, and you can write it for any database engine you choose (SQL Server, MySQL, etc.)

## 2. Data seeding scripts – 20 points

A script written in either SQL or C# (in case of code first approach) to insert seeding data in your database. It only needs to insert a few records of sample data in each of the necessary tables according to your design, so that the web application will work. You must make sure the referential integrity of your data is preserved.

## 3. Web application – 50 points

Please create a .Net MVC web application, to allow XYZ's invoice manager to view and enter invoices in the system and for employees to view all their submitted invoices. Depending on the user's role, the system has 2 simple use cases:

**USE CASE 1 – invoice manager entering invoices (user role of invoice manager)**

* A user logs in and is authorised in the role of **invoice manager**. This is an administrative role.
* He/she has access to all data in the system. He/she can see a list of all invoices in the system.
* The invoice manager has received an invoice from an employee and has to enter it into the system.
* The invoice must be linked to the employee who submitted it.
* The system must perform data validation according to the rules described in the Requirements section above.

**USE CASE 2 – employee viewing a list of their submitted invoices (user role of employee)**

* An employee logs in and is authorized as an employee. This is the default role.
* He/she can see a list of all their invoices and no other data.
* An employee must not be able to enter invoices unless they are authorised as an invoice manager.

You only need to build the screens related to the handling of invoices. Those are to allow viewing the list of invoices and entering and validating the invoice data. **To summarize, you should only build the following screens**:

* A login screen to allow the application to determine between the 2 user roles described above.
* A screen listing the invoices in the system according to the role of the logged-in user.
* A screen showing the details for the invoice being created or modified.

You don't need to build screens to manage supporting data such as employees, application users, contracts, etc. Those, and other needed supporting entities can be seeded in the database tables through scripts.

You don’t need to build the user management and authentication of the application. You can mock-up a simple user login screen to allow your application to distinguish between the two user roles – the **invoice manager** who views and enters invoices and the **employee** who can see the list of his/her own invoices.

# Considerations

You are free to choose the architecture and the components of the application. You may choose to use MVC with only the Razor view engine or combine it with a JavaScript framework of your choice.

You are free to choose whether to use code first or database first approach. It is also up to you whether to use an ORM or another way to implement the data layer.

As with any programming project, in this case there is no single correct solution. The test aims to assess your programming skills and experience with Microsoft .Net, C# and the related web technologies.

You may be tempted to write all the code in a single source file, but please don't. This application should be structured with the same principles in mind as a larger one.

If you don't have enough time to complete the entire exam, please do turn in the work you manage to produce. We are not looking for a complete billing system. We will look at the way you organize your code, your programming style, the way you model the database and how you architect your solution. If you are unable to fully write all features and methods, just provide method stubs and scaffolding without the actual implementations. As mentioned above, it is not critical if the solution does not compile. You will be graded on what you have submitted.

Below are some important considerations that will affect the final score:

* Overall architecture. Although simple, the application must adhere to sound design principles. It must have clearly separated data, business and presentation logic.
* Your programming style and how you structure your code. Whether the code is readable and clear to understand.
* Careful choice of implementation of cross cutting concerns such as user authorisation and data validation logic.
* The data model must fit the requirements.

----------------------------------------------------------------- END OF EXAM -----------------------------------------------------------------