Сопроводительная записка к тестовому заданию.

Решение представляет собой набор проектов в среде .Net Core 3.1 на стороне сервера и Angular 10 на стороне клиента, разделенный по типу используемых данных и логике, что позволяет реализовать слабую связанность и возможность легко разделить решение.

Проекты решения:

* DAL – Data Access Layer содержит в себе ORM (Entity Framework Core) для работы с БД. Также в этот проект вынесена сама база данных (Sqlite), миграции и средства первичного наполнения базы. Связь с другими проектами осуществляется слабосвязанно, посредством выделенного интерфейса. Загрузка зависимых данных реализована посредством lazy loading.
* DTO – Data Transfer Objects содержит с себе анемическую модель для взаимодействия с клиентским приложением, также частично отвечает за валидацию данных.
* Models – содержит в себе анемическую модель представляющую собой объектную модель базы данных, а также вспомогательные модели для бизнес логики. Связь с dto осуществляется с помощью маппинга (AutoMapper) в слое клиентского приложения.
* Services – проект содержащий в себе всю бизнес логику решения. Также в проект вынесена логика некоторых моделей данных посредством расширений. Взаимодействие с остальным частями решения осуществляется с помощью сервисов, внедренных через Dependency Injection.
* Tests – проект для тестирования бизнес логики (NUnit).
* WebClient – проект содержащий в себе клиентское приложение и Core Web Api предоставляющее rest сервис для передачи данных клиенту.

Для аутентификации была реализована система Jwt токенов и встроенных инструментов Core Identity, с использованием ролей для разделения прав доступа. Также для повышения безопасности пароли пользователя не пересылаются в прямом виде.

Была реализована особая система обработки исключений для уведомления пользователя, например, при неудавшейся аутентификации бросается специальное исключение, которое отлавливается глобальным фильтром исключений, преобразовывается в 400 ошибку (Bad Request), на стороне клиента эта ошибка обрабатывается перехватчиком и выводится пользователю в сообщении, подобным образом также отображаются ошибки валидации.

Решение задачи было реализовано посредством рекурсивного прохода дерева сотрудников. Берутся все сотрудники, у которых нет начальника и по ним и их подчиненным считается зарплата, таким образом расчет происходит для каждого сотрудника один раз. Если количество рассчитанных зарплат не совпадает с количеством работников, то значит присутствуют циклические зависимости и выбрасывается соответствующее исключение. Для расчета каждого типа сотрудника используется отдельный калькулятор, таким образом происходит разделение логики расчетов зарплаты. Калькулятор выбирается специальным методом и возвращает объект абстрактного базового класса, что позволяет абстрагироваться от конкретных реализаций. При расчете зарплат для типов Manager и Salesman используется метод, который рекурсивно проверяет на наличие циклических зависимостей и в случае обнаружения таких кидается соответствующее исключение. К слову, этот же метод должен использоваться при назначении подчиненного (данная часть не реализована в связи с своей тривиальностью). Расчет премии за каждый год работы реализован как extension метод над основным классов модели Employee (чтобы освободить модель от бизнес логики). Также есть дополнительная проверка даты расчета с датой трудоустройства. Возможно следовало бы добавить расчет зарплаты в зависимости от количества рабочих дней в текущем месяце.

Все ключевые алгоритмы расчета покрыты тестами.

На клиентской стороне есть только одна общедоступная страница, страница аутентификации. После удачной аутентификации для всех пользователей открывается страница с подробной информацией о текущем пользователе с рассчитанной зарплатой на текущий день и список его подчинённых, если таковые имеются. Для пользователей типа Manager и Salesman также доступна страница с расчетом зарплат и итоговым результатом для всех работников компании. Роли используются как для ограничения доступа к разным страницам, так и для отображения элементов в рамках одного контрола. При логине, логауте используется событийная модель для обновления данных с приветствием и меню.

Всего предопределено 7 пользователей по умолчанию, 2 типа Manager, 2 типа Salesman и 3 типа Employee, с прописанными аторизационными данными и ролями. Данные можно расширить, используя класс DataSeeder.

Есть пользователь superUser с правами Manager, доступ к которому предоставляется с любым паролем при компиляции под debug.

Сборка доступна в репозитории по адресу https://github.com/glavin-ma/Salary.git

