Основные принципы разработки решений

на платформе ASP.NET MVC 5

1. По возможности следуй принципам SOLID.
2. Используй Inversion of Control и Dependency Injection (I и D из SOLID соответственно), благо платформа MVC предоставляет для этого все возможности и необходимые точки подключения собственных компонентов.
3. Используй для объединения компонентов приложения один из известных контейнеров внедрения зависимостей (информацию по производительности контейнеров можно посмотреть [вот тут](https://github.com/danielpalme/IocPerformance)).
4. Контроллер должен быть максимально, до предела тупым. Как валенок. Получил данные запроса от юзера (чаще всего – как экземпляр некой модельки через параметр метода с атрибутом [FromBody]), пробросил эту модельку в сервис, получил ответ от сервиса и отдал его либо в виде вьюхи на рендер, либо в виде json-а, либо в виде файла.
5. Старайся следовать принципу «1 контроллер – 1 сервис», причём сервис инжектируется через конструктор. Вообще, это напрямую следует из п. 4. Самое плохое, что можно сделать, и за что следует убивать Ржавой Секирой Ужаса – это создание экземпляра контекста к БД прямо в контроллере в виде поля и выдёргивание данных в методе контроллера. Это всегда путь боли и страданий, потому что:
   * систему становится сложнее тестировать
   * ты не используешь внедрение зависимостей
   * если потребуется вместо БД использовать в качестве источника данных сторонний сервис – придётся переписать большой кусок кода (контроллер, интерфейс сервиса, реализация сервиса для взаимодействия с БД, реализация сервиса для взаимодействия с внешним сервисом) => лучше сразу писать правильно.
6. Используй асинхронный ввод / вывод, и вообще предпочитай везде использовать асинхронные операции (при обращении к БД, при обращении к внешним сервисам по HTTP и т.д.). Так ты можешь значительно повысить производительность даже на слабых серверах.
7. Явным образом раздели механизм обработки данных внутри приложения на три (иногда четыре) слоя:
   * слой доступа к данным (классы-сущности EF – без внутренней логики!!!; контекст EF)
   * абстракция над слоем доступа к данным - репозитории, Unit of Work – опционально, чаще всего реализацию этого слоя считают оверинжинирингом (но иногда её реализация оправдана – тут каждый случай индивидуален)
   * слой сервисов (модели для получения / передачи данных клиенту; собственно сами сервисы)
   * слой контроллеров

При этом преобразование между классами-моделями из слоя сервисов и классами-сущностями из слоя доступа к данным должно выполняться строго внутри слоя сервисов; контроллер должен быть тупым (см. п. 4).

1. Никакой логики внутри классов-сущностей и классов-моделей быть не должно. Вся бизнес-логика приложения должна быть сосредоточена внутри слоя сервисов. Максимум, что может происходить вне этого слоя – валидация получаемых от клиента моделей на соответствие заданным через атрибуты [валидации] критериям.

Для этого используется собственный обработчик http-запроса, определяемый внутри приложения, который отрабатывает после механизма авторизации, но до передачи выполнения в соответствующий метод контроллера.

1. В случае разработки приложения и проектирования БД с нуля пользуйся подходом Code First и миграциями уровня EF (никаких GUI-мастеров! они от лукавого!) для возможности быстрого отката базы к предыдущему состоянию.
2. Выполняй связывание классов-сущностей с соответствующими таблицами в БД ***вручную***. Так ты предоставишь более полный контроль над происходящим в коде не только для себя, но и для людей, которые будут дорабатывать проект после тебя.
3. Если руководство ещё не поставило задачу, то придумай её себе самостоятельно и сразу же начинай кодить – то есть, старайся иметь постоянную практику по изученному материалу. Идеально будет забегать немного вперёд, чтобы потом при появлении нового взгляда на старые вещи аккуратно изменить уже существующий функционал на реализованный более оптимальным способом.
4. Заведи репозиторий на GitHub или Bitbucket и активно пили там код.