**Анализ требований**

1. Базовые требования
2. Требования к работе
   1. с системой баг трекинга
   2. с системой контроля версий
   3. к написанию кода
3. Механизмы безопасности системы
4. Ядро системы
5. Механизмы доступа к системе
6. Подсистема Api для взаимодействия с другими системами/подсистемами
7. Подсистема пользовательского интерфейса
   1. браузер
8. Развертывание системы
   1. в develop режиме
   2. в production режиме

**Базовые требования**

1. Основной стек разработки
   1. C# 6.0
   2. T-SQL
   3. HTML5
   4. CSS3.0
   5. JS ES-2015
   6. Entity Framework 6.0
   7. ASP .NET MVC 4.0
   8. Bootstrap4
   9. JQuery 3.1.1
2. Система виртуализации рабочего места: VMware® Workstation v. 14.0.0 build-6661328

P. S. применение системы виртуализации желательно но не обязательно, все зависит от ресурсов компьютера программиста

1. Базовая операционная система для разработки Windows 10 Pro x64 v. 1703
2. Базовая операционная система для хостинга: Windows Server 2012 x64
3. Основные типы серверов
   1. Сервер приложений:
      1. Develop: IISExpress v10.0
      2. Production: IIS v. 8.5
   2. Сервер баз данных: MS SQL 2016 R0
4. Базовая среда разработки: MS VS 2017 Enterprise
5. Требуемые расширения среды разработки
   1. JetBrains ReSharper Ultimate 2016.3.2 Eng
   2. RazorGenerator
6. Текущий базовый адрес production версии системы:
7. В качестве системы контроля версий используется система Git
8. В качестве системы баг трекинга используется система Redmine
9. В качестве системы Continue Integration рекомендую использовать TeamCity т. к. у него сравнительно легко настраивается работа с MS Build, основным механизмом сборки решения
10. Отслеживание истории коммитов git легко интегрируется в систему багтреккинга

**Требования к работе**

**с системой баг трекинга**

Каждая задача, связанная с системой заводиться в систему багтреккинга!!!

P. S. Для данного правила нет исключений!!!

**с системой контроля версий**

1. Основная система контроля версий – Git
2. Основное средство работы с системой контроля версий – программа Git Extensions
3. Все есть ветка ☺, т. е. на любую!!! внешнюю задачу делаться локальное ответвление от основной ветки разработки, после решения текущей задачи, идет слияние с основной веткой разработки
4. Название ветки кода, формируется по принципу “task-{id}” где id –номер задачи заведенной в системе багтрекенга

При коммите в коментариях ,обязательно сперва указывается task-id ветки по щаблону “#task-id”, после уже идет описание коммита

**по написанию кода**

Т. к. мы разрабатываем много пользовательскую систему, то обязательна поддержка многопоточности и асинхронности для снижения нагрузки на сервер, соответственно учитывая нововведения С# 6.0для любых методов любых контролеров системы обязательна!!! поддержка и применение пары async/await

Т. е. если кратко, синхронных методов контроллера не должно существовать!!!

**Механизмы безопасности системы**

Взаимодействие ядра системы с внешним миром обязано происходить по защищенному каналу связи. В современном мире для данных целей применяют протокол https в связке с SSL сертификатом, подписанным корневым центром сертификации, к примеру, центром сертификации компании [VerySign](https://ru.wikipedia.org/wiki/Verisign). На этапе разработки можно использовать самоподписанный сертификат, сгенерированный через программу OpenSSL

**Ядро системы**

Ядром системы является решение ModulesMVC, самописная модульная система на ASP .NET 4.0 + C# 6.0 которая на данный момент состоит из 8 модулей

1. Libraries.Core.Backend – библиотека содержащая базовые классы ядра системы
2. Libraries.Core.Frontend – библиотека содержащая фронтэнд модули на JavaScript и/или Css
3. Modules.Core – не изменяемые или слабо изменяемые части системы на подобии модуля авторизации
4. Themes.Core – библиотека для модулей - тем основанных на Bootstrap4
5. Modules.Site – основные используемые и часто изменяющиеся модули системы
6. Modules.WebApi – модуль для WebApi системы
7. Dal – библиотека доступа к БД на основе миграций Entity Framework 6.0
8. Frontend.Web – центральное ядро которое подгружает и связывает все остальные компоненты

**Механизмы доступа к системе**

В текущий момент времени основным методом аутентификации и авторизации на сайтах является метод OAuth2 или его более продвинутая версия OpenID, благодаря подобному решению аутентификации и авторизации можно легко текущую систему интегрировать с такими вещами как социальные сети и/или аккаунты Google что в современном мире является не маловажным преимуществом

Методы, связанные с Basic Authorization не приемлемы по соображениям безопасности, тут можно подробно прочитать о недостатках Basic Authorization: <http://samag.ru/archive/article/1631>

В текущей системе реализован механизм аутентификации и авторизации на основе ролей и пользователей для гибкой настройки доступа к разделам системы

**Подсистема Api для взаимодействия с другими системами/подсистемами**

1. Данная подсистема использует единый механизм аутентификации и авторизации системы
2. Все серверные действия должны проходить через данную под систему как единую точку входа
3. Единой точкой доступа к WebApi является путь <https://servername/api>
4. Т. к. фактически все запросы к WebApi изменяют состояние системы, то соответственно, по сути, являются POST запросами, содержимое запросов передается в виде JSON объекта превращенного строку в кодировке UTF-8, таким образом, повышается безопасность доступа к системе и организуется единый механизм доступа к WebApi. Поток POST запросов, по сути, выступает просто в качестве канала связи, а не механизма влияния на систему, основную суть будут нести JSON объекты, переданные на необходимые URI

P. S. небольшим количеством GET запросов на выборку данных можно пренебречь с учетом стандартизации и унификации механизма доступа к WebApi

P. S. S. PUT/DELETE запросы не корректно поддерживаются современными браузерами и серверами и требуют дополнительной работы по их реализации соответственно реализовывать их не рационально

1. Текущее описание требований к методам WebApi можно найти в документе:

“ Платежная система — Общий бриф.pdf ”, “WebApi платежной системы”

**Подсистема пользовательского интерфейса тонкого клиента**

**Браузер**

Данное приложение ориентированно на браузеры IE, Chrome и Firefox последних версий

Страницы сайта нужно разрабатывать с учетом нескольких факторов

1. Разработка ведется на HTML5+CSS3+JS ES 2015
2. Текущие технологии для разработки клиентской стороны Bootstrap 4+ модули JQuery 3.1.1
3. Работа с ядром системы ведется через только Ajax.post механизм
4. Страницы должны быть адаптивными под различный класс устройств, поэтому в текущий момент основной сущностью клиентской страницы есть адаптивная таблица на основе Bootstrap состоящая из строк, которые в свою очередь состоят из столбцов
5. Строк в таблице неограниченное количество и они могут растягиваться по высоте
6. Столбцов в строке может быть максимум 12, основные css классы столбцов – md\*
7. Текущее описание требований к пользовательскому интерфейсу сайта можно найти в документе: “[Тестовое задание по ASP.NET MVC](https://docs.google.com/document/d/1hdILzxYk3xLnLorbdvzrCWEF7UqtHGeSP5pfMf2bmK8)”

**Развертывание системы**

**в develop режиме**

1. Полностью разворачивается рабочее место программиста
2. Через Git скачивается текущая версия кода системы
3. Создается персонализированная ветка разработки для программиста
4. Создается персонализированная конфигурация окружения для программиста
5. В качестве основного сервера разработки применяется IIS сервер, рекомендуется использовать встроенный Visual Studio 2017 IIS Express v.10

**в production режиме**

1. TeamCity при обнаружении изменений запускает “чистую” сборку решения в песочнице
2. После корректной сборки должны проходить Unit тесты программистов
3. После корректных Unit тестов должны пройти тесты QA Automation
4. После корректно завершенных QA Automation тестов в песочнице делается “чистый” deploy на production сервер