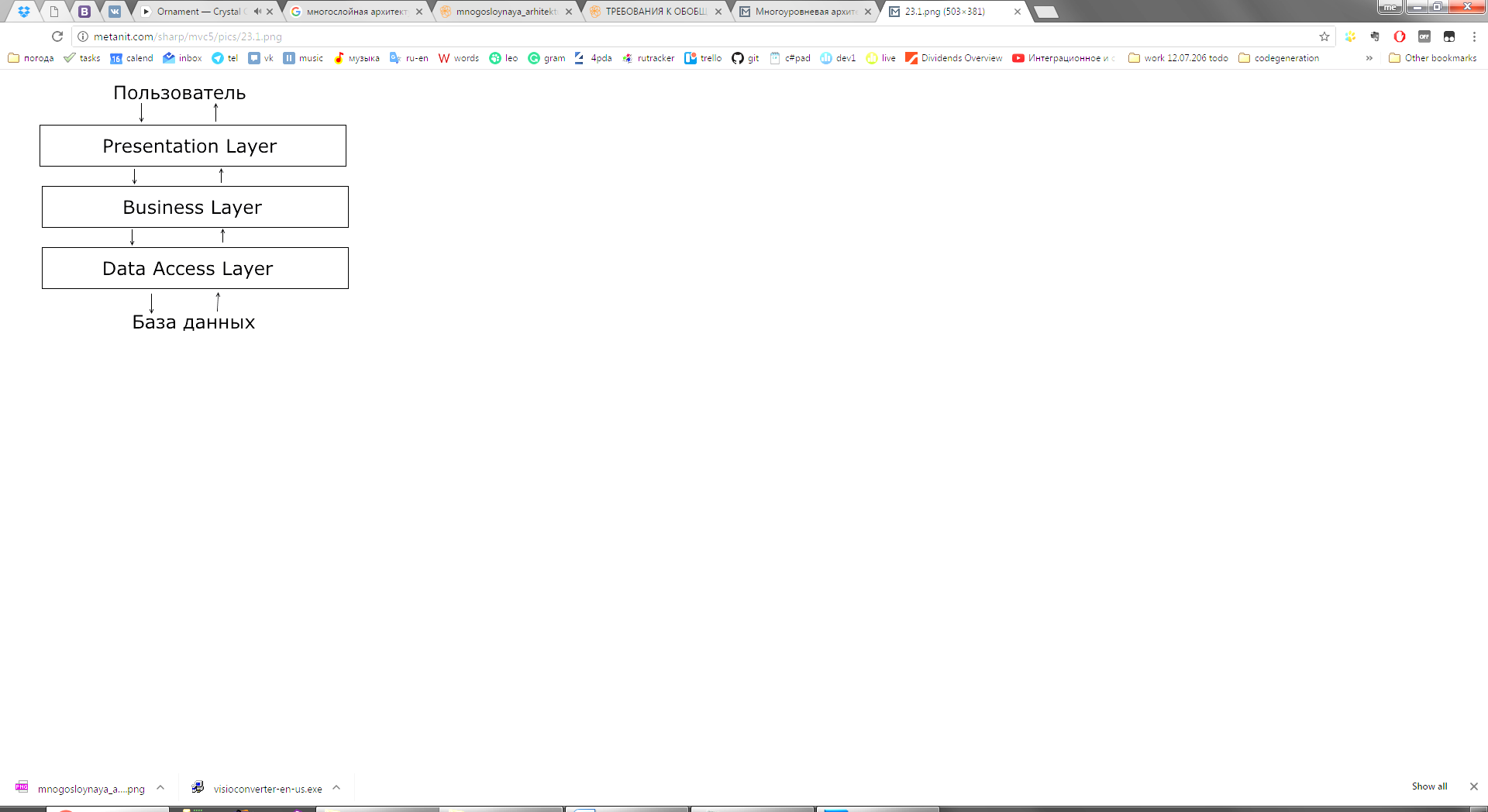
**Техническая спецификация проекта TrueMoney**

|  |  |  |  |
| --- | --- | --- | --- |
| **Дата** | **Версия** | **Описание** | **Автор** |
| 12.10.2016 | 1.0 | Создание документа | Лукьянов |
| 15.10.2016 | 1.1 | Редактирование | Черногребель, Артюх |
|  |  |  |  |

# 1 Архитектура приложения

## 1.2 Описание архитектурного шаблона

Приложение должно использовать многослойную архитектуру, которая обеспечивает группировку связанной функциональности приложения в разных слоях, выстраиваемых вертикально, поверх друг друга. Функциональность каждого слоя объединена общей ролью или ответственностью. Слои слабо связаны, и между ними осуществляется явный обмен данными. Правильное разделение приложения на слои помогает поддерживать строгое разделение функциональности, что в свою очередь, обеспечивает гибкость, а также удобство и простоту обслуживания. Общая схема приведена ниже.



**Presentation layer** (уровень представления): это тот уровень, с которым непосредственно взаимодействует пользователь. Этот уровень включает компоненты пользовательского интерфейса, механизм получения ввода от пользователя. На данном уровне расположены представления и все те компоненты, который составляют пользовательский интерфейс (стили, статичные страницы html, javascript), а также модели представлений, контроллеры, объекты контекста запроса.

**Business layer** (уровень бизнес-логики): содержит набор сервисов, которые отвечают за обработку полученных от уровня представлений данных, реализует всю необходимую логику приложения, все вычисления, взаимодействует с базой данных и передает уровню представления результат обработки.

**Data Access layer** (уровень доступа к данным): хранит модели, описывающие используемые сущности, также здесь размещаются специфичные классы для работы с разными технологиями доступа к данным, например, класс контекста данных Entity Framework. Здесь также хранятся репозитории, через которые уровень бизнес-логики взаимодействует с базой данных.

Крайние уровни не могут взаимодействовать между собой, то есть уровень представления (контроллеры) не могут напрямую обращаться к базе данных и даже к уровню доступа к данным, а только через уровень бизнес-логики.

Уровень доступа к данным не зависит от других уровней, уровень бизнес-логики зависит от уровня доступа к данным, а уровень представления - от уровня бизнес-логики.

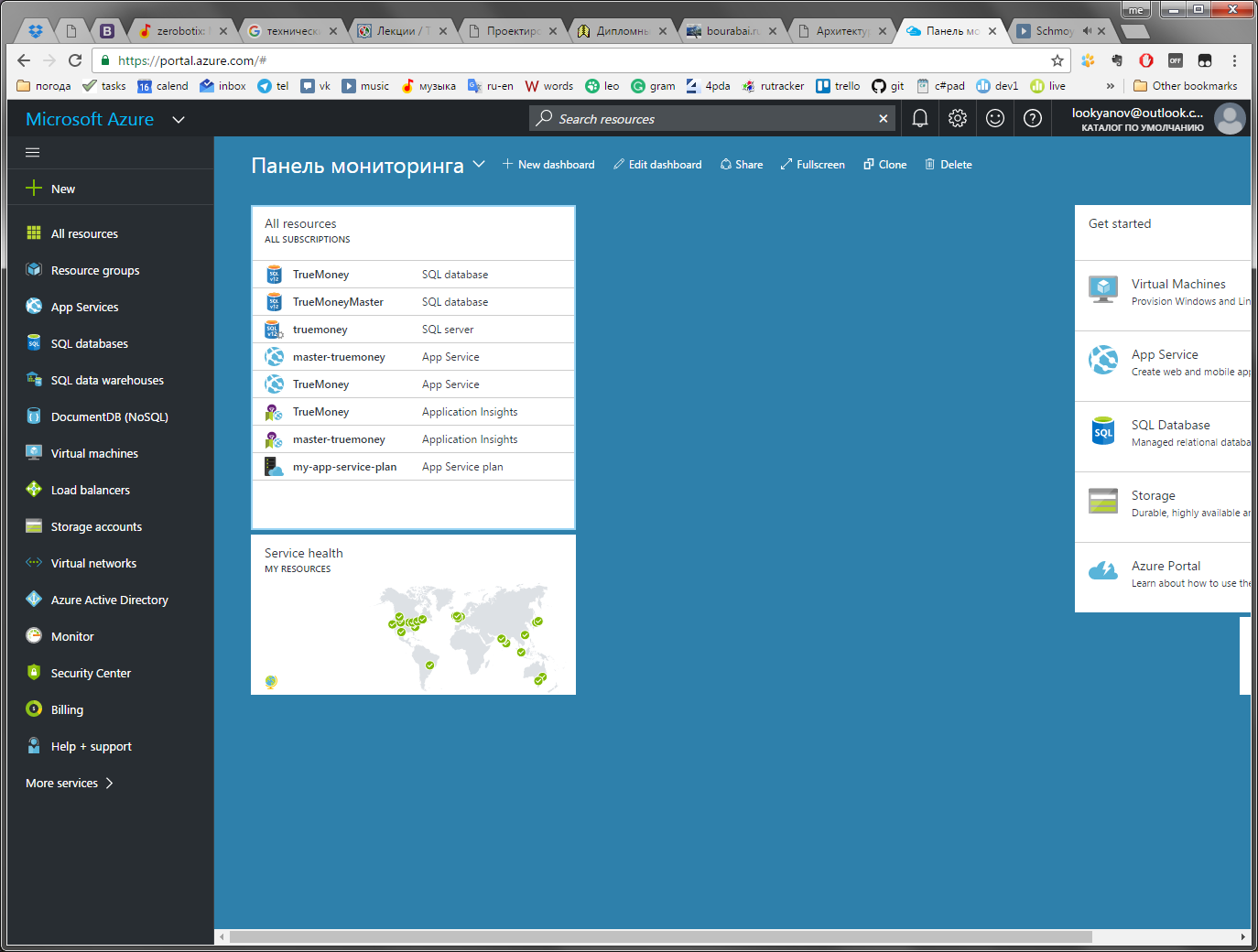
Компоненты должны быть слабосвязанными (loose coupling), поэтому необходимо использовать механизм внедрения зависимостей (dependency injection).

## 1.2 Основные модули приложения

1. TrueMoney.Infrastructure – данный проект содержит общие классы, к которым имеют доступ все остальные проекты. Сюда включаются константы, перечисления, классы-сущности.
2. TrueMoney.Data – проект, отвечающий за работу с базой данных
3. TrueMoney.Services – проект, который содержит в себе всю бизнес-логику приложения.
4. TrueMoney.Web – в этом проекте полностью реализуется слой презентации. Также, этот проект является точкой входа в приложение и содержит в себе все необходимые инициализации.
5. TrueMoney.DependencyInjection – вспомогательный проект, отвечающий за регистрацию зависимостей в приложении
6. TrueMoney.Mapping – вспомогательный проект, помогающий конвертировать сущности одного слоя в сущности другого слоя
7. Bank – проект-заглушка, представляющий из себя сторонний банковский сервис, позволяющий производить оплату

## 1.3 Облачный хостинг

Разработанное приложение должно использовать ресурсы облачного хостинга, в частности Azure. Примерный список необходимых ресурсов виден на скриншоте ниже.



# 2 Архитектура базы данных

1. SQL сервер должен находится в облаке Azure.
2. На сервере должно быть 2 базы данных: TrueMoney (для активной разработки) и TrueMoneyMaster (продакшен). Обе базы должны иметь идентичную структуру, которая генерируется приложением автоматически.
3. Все данные в базе должны быть в третьей нормальной форме, однако допускаются исключения, сделанные для увеличения производительности.
4. Данные в базе должны быть целостными – то есть полными и не противоречивыми, для чего должны быть настроены необходимые связи (foreign keys)
5. База данных для разработки должна быть доступна из любого места извне, к базе данных продакшена доступ должен фильтроваться по IP (whitelist)
6. Для доступа к базе должна использоваться SQL-аутентификация
7. Пароли пользователей в базе данных должны храниться в зашифрованном виде
8. Каждая таблица в базе должна содержать колонку с уникальным идентификатором типа int, который должен являться первичным ключом (за исключением таблиц, которые обеспечивают связь «многие-ко-многим»)
9. Взаимодействие пользователя с БД не должно зависеть от особенностей её физической организации

# 3 Требования к разработке

1. Проектируемые классы должны соответствовать следующим принципам SOLID:
   * Single responsibility principle(S) - говорит о том, что на каждый класс должна быть возложена только одна определенная обязанность.
   * Open/closed principle(O) - говорит о том, что программные сущности должны быть открыты для расширения, но закрыты для изменений.
   * Liskov substitution principle(L) - принцип подстановки Барбары Лисков, который говорит, что функция, использующая базовый тип, должна иметь возможность использовать подтипы базового типа, не зная об этом.
   * Interface segregation princilpe(I) - говорит о том, что лучше иметь множество специализированых интерфейсов, чем один универсальный.
   * Dependency inversion principle(D) - говорит о том, что зависимости в системе должны строиться на основе абстракций. Модули верхнего уровня не должны зависеть от модулей нижнего уровня. И те, и те должны зависеть от абстракций. В свою очередь, абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций.
2. В случае непредусмотренных ошибок всегда должны генерироваться исключения.
3. Уровни приложения должны допускать независимое тестирование.
4. Названия интерфейсов, классов, методов и переменных должны отражать то, за что они отвечают. Допускается использовать только названия на английском языке.
5. В коде не должно быть «магических чисел», в случае необходимости значение выносится в отдельный класс с константами.
6. Производительность методов должна быть достаточной для того, чтобы на скорости 1Мбит/сек загрузка любой страницы и обработка любого запроса не превышала 10 сек.
7. Разработка должна происходить с использованием системы контроля версий Git
   * Ветка master содержит «чистовой» вариант приложения, протестированный и рабочий. Код из этой ветки автоматически собирается и публикуется (deploy) в интернете, под этот сайт используется отдельная бд.
   * Ветка dev содержит самую актуальную версию приложения, используется всеми разработчиками как основная.