Оглавление

[1. Важно 2](#_Toc502446737)

[2. Мотивация 4](#_Toc502446738)

[Недостатки Account Service 4](#_Toc502446739)

[Заметки 4](#_Toc502446740)

[3. Разработка 5](#_Toc502446741)

[Основные моменты 5](#_Toc502446742)

[Идея 5](#_Toc502446743)

[Архитектура 5](#_Toc502446744)

[4. Заключение 6](#_Toc502446745)

[Варианты улучшения проекта 6](#_Toc502446746)

# Важно

1. Всё изложенное в разделе [Мотивация](#_Мотивация) является **авторской точкой зрения** на сложившуюся ситуацию с Account Service проектом;
2. В файле Web.config введите свои данные (рисунок 1);

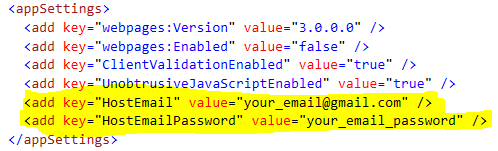
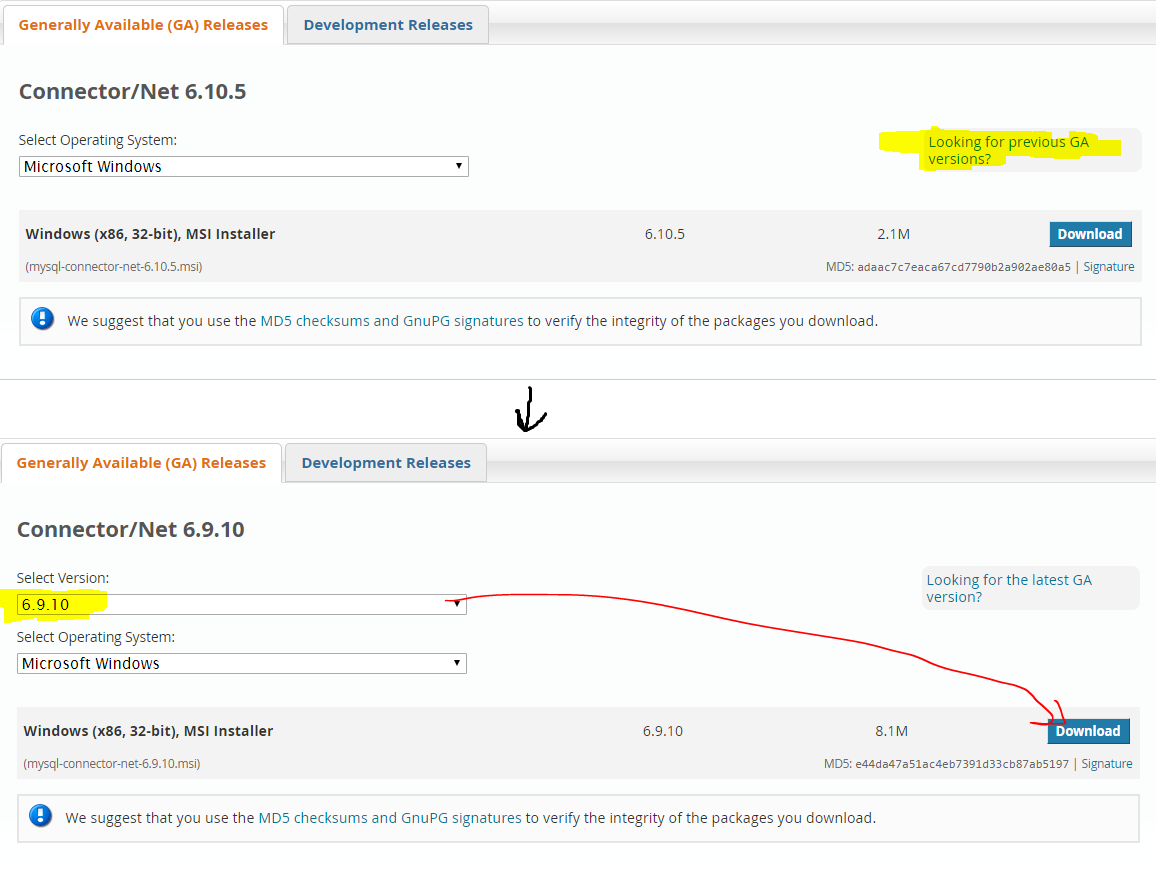


Рисунок 1 – Вырезка из Web.config

1. Возможна ошибка с gmail почтой. При регистрации может выпасть ошибка. В этом случае нужно проверить свою почту (на которую был зарегистрирован банковский аккаунт), затем в списке входящих сообщений будет оповещение о «ненадёжном приложении». В этом сообщении будет пункт настроек, в которых нужно разрешить приложению отправлять Вам письма;
2. Для нормальной работы проекта необходимо установить sql .Net connector 6.9.10.

Однако можно осуществить запуск проекта без sql .Net connector 6.9.10. В этом случае работоспособность проекта не гарантирована. Это требуется для работы провайдера ролей.

<https://dev.mysql.com/downloads/connector/net/> (рисунок 2);

  
Рисунок 2 – sql connector

1. Чтобы отключить кастомные ошибки, нужно удалить customErrors, httpErrors и

protected void Application\_Error(object sender, EventArgs e) (рисунок 3);



Рисунок 3 – Элементы обработки ошибок

1. В приложении предусмотрена роль администратора. По умолчанию нельзя стать администратором, осуществив регистрацию на сайте. Необходимо вручную создать эту учётную запись в базе данных (рисунок 4).

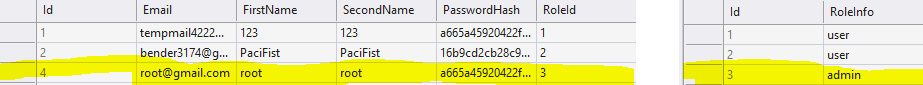


Рисунок 4 – Данные администратора

# Мотивация

## Недостатки Account Service

1. В проекте Account Service для добавления регистрации/авторизации нужно либо делать прямую связь PL уровня с DAL уровнем + писать отдельный репозиторий для работы с пользователями (так сделал я), либо добавлять в класс Account на уровне BLL.Interface необходимые для пользователей поля (минимально email и password). Это повлечёт изменения во всех остальных уровнях;
2. При добавлении этих полей в класс Account он становится слишком сложным в плане хранения данных (полностью хранит данные об аккаунте и его владельце). И тут уже просится разделение этого класса на Account и AccountOwner/BankUser, каждый из которых имеет доступ к данным другого;
3. Не менее важно следующее: работа с аккаунтами должна происходить только через Account Service, но класс Account, расположенный в BLL.Interface, имеет свои методы DepositMoney и WithdrawMoney, которые использует только Account Service. В итоге класс Account предоставляет API, которое PL уровень не использует и это просто нагружает его лишними возможностями;
4. Методы DepositMoney и WithdrawMoney в классе Account, по сути, используются только для того, чтобы посчитать бонусные балы при проведении операций с аккаунтом. С таким же успехом можно было сделать все поля класса Account public и в Account Service прибавлять/вычитать «средства» с аккаунта, дополнительно, прибавляя/вычитая бонусные балы;
5. Этот пункт очень спорный, но всё же. Если представить себя клиентом Account Service (допустим, мы разработчик PL), то по всей логике вещей я должен получить набор DLL библиотек и использовать API Account Service’a для разработки уровня презентации проекта. В данной реализации для открытия аккаунта нужен AccountIdService, генерирующий уникальные идентификаторы аккаунтов. С авторской точки зрения разработчик PL должен знать только то, что у него есть Account Service, и он всё сделает сам: откроет акканут, будет им управлять и т.д., и разработчику нужно только вызывать его методы;
6. Самый важный пункт в этом списке. В данной реализации Account Service’a аккаунты управляют пользователями. Всё отталкивается от того, что есть Account, он в центре реализации приложения, этакая доменная сущность всего проекта. **Владелец аккаунта стоит на втором месте, и Account его использует только при авторизации. Факт в том, что владелец есть у аккаунта (аккаунт владеет/контролирует/управляет владельцем, от аккаунта всё отталкивается), а не аккаунты есть у владельца (владелец должен управлять своими аккаунтами, а не наоборот). Такая небольшая инверсия контроля :)**

## Заметки

Так как данный проект является финальным, значит, нет строгой спецификации по его разработке. Я реализовал своё видение Account Service (с блэкджеком).

Всё выше перечисленное и ещё некоторые небольшие факты привели меня к выводу, что Account Service достиг своей конечной точки. Я решил полностью переписать Account Service, руководствуясь принципом «Не нагружай пользователя ненужными ему возможностями».

# Разработка

## Основные моменты

### Идея

В моей интерпретации Account Service превращается в Bank Service. Идея следующая. Есть 4 главные сущности: а именно: Account, BankUser(AccountOwner), BankService, Account Service. BankService управляет пользователями (регистрация в банке, получение информации о пользователе и т.д.), Account Service управляет работой с банковскими аккаунтами (OpenAccount, DepositMoney, CloseAccount и т.д.), Account и BankUser являются главными доменными сущностями. У аккаунта есть ссылка на своего владельца, а у владельца есть коллекция всех своих аккаунтов.

### Архитектура

Стандартный stairway pattern с небольшими изменениями (рисунок 5). Суть изменений: как и всегда, есть PL уровень, который выступает в качестве клиента для бизнес-логики; он имеет доступ только к BLL.Interface.

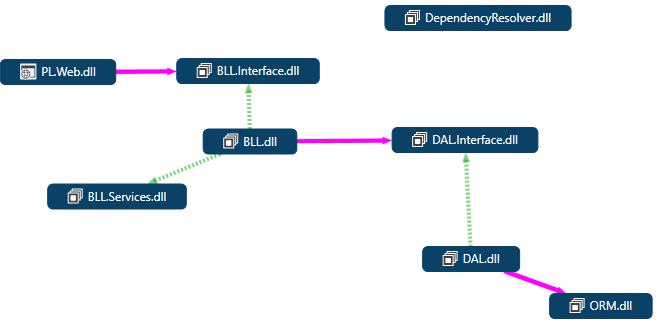


Рисунок 5 – Общая схема архитектуры приложения

На рисунке 6 показана обобщённая структура приложения. PL.Web выступает в качестве клиента для бизнес-логики, BLL.Interface является неким переходником (заглушкой, proxy), которой поставляет сущности и API для работы с банком. BLL.Interface не должен поставлять API, которое пользователь не будет использовать. Поэтому он предоставляет только IBankService, Account, AccountType, BankUser. Обратите внимание на то, что метод IBankService.OpenAccount не принимает в качестве параметра ссылку на AccountIdService, так как я считаю, что это не нужное API для клиента. Клиент должен знать, что есть банк, и он может в нём открыть аккаунт, а реализация этой задачи(открытие аккаунта) не относится к компетенции клиента[[1]](#footnote-1).

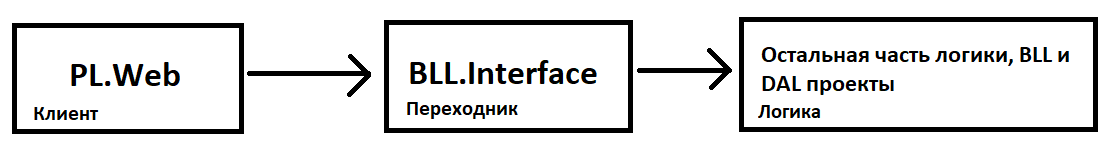


Рисунок 6 – Обобщённая структура приложения

# Заключение

В результате проделанной работы получился проект, открытый к будущим расширениям не только на уровне представления, но и на уровне логики.

## Варианты улучшения проекта

1. Добавление «живого поиска» для домашней страницы пользователя;
2. Добавление пагинации для домашней страницы;
3. Добавление возможности сортировки данных по ряду критериев;
4. Добавление станицы просмотра данных пользователя для авторизованного пользователя;
5. Добавление возможности изменения данных пользователя;
6. Добавление функции «Забыли пароль?»;
7. Добавление возможности сменить e-mail пользователя;
8. Добавление возможности «Добавить аккаунт» как в gmail
9. Добавление возможности авторизации через vk/facebook/twitter.

Проект Bank Service можно скачать по ссылке: <https://drive.google.com/drive/u/0/folders/10YulxMcS4bSaz8M9rEiBBwTzsFhbha_r>

Проект Account Service можно скачать по ссылке: <https://drive.google.com/drive/u/0/folders/1OcfdyZJjSTw9YTomruP1d59HoP0zwIL1>

1. Если кратко, то это и есть основная идея архитектуры приложения. [↑](#footnote-ref-1)