ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО

ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

**«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

ИНСТИТУТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ

И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

КАФЕДРА СИСТЕМНОГО АНАЛИЗА И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Специальность: информационная безопасность

ОТЧЕТ ПО ДИСЦИПЛИНЕ

«Проектирование защищенных информационных систем»:

**«Свободная фриланс-биржа труда с элементами социальной сети»**

**Работу выполнил:**

Рахимулин Р.Р, гр. 9-409

**Преподаватель:**

Андриянова А.А.

**Казань – 2015**

# Содержание

**1 Основополагающая идея системы**3

**2 Бизнес-логика и бизнес-процессы в системе3**

**3 Пользовательский режим4**

3.1 Панель регистрации5

3.2 Регистрация6

3.3 Обратная связь7

3.4 Фидбэк8

3.5 Аккаунт8

3.6 Заявки9

3.7 Обсуждения задачи12

3.8 Рабочая панель13

3.9 Зарегистрировавшиеся пользователи14

3.10 Панель «Мои заявки»15

3.11 Панели "Загрузить оплату" и "Пополнить баланс"16

3.12 Заявка на вывод средств17

3.13 Панель "Сообщить об ошибке"17

**4 Администрирование18**

4.1 Справочники18

4.2 Регистрирование оплаты. Биллинг20

4.2.1 Подтверждение оплаты - переводов21

4.2.2 Раздел "Подтверждение заявки на вывод средств"23

4.3 Блокировка24

**5. База данных25**

**Выводы27**

**Приложение 128**

1. **Основополагающая идея системы**

Предоставление безопасных сделок между заказчиками и исполнителями. Максимально быстро, максимально автоматизировано. Единственная слабая часть: проверка оператором переводов, совершенных на определенные счета системы. Т.к. это не банк, а система, использующая сторонние банковские системы При этом: полностью исключается возможность обмана. Как же это делается? Внутри системы - она защищает всех пользователей. Если же пользователи договариваются между собой и делают переводы вне системы - они не попадают под её правила. Никоим образом. Сеть защищенная, потому – open-source, т.е. доступна для свободного использования и корректирования кода

Репозиторий: https://github.com/GarageInc/hepldesk

1. **Бизнес-логика и бизнес-процессы в системе**
2. Единая внутренняя расчетная система. Вход - рубль, выход - рубль. Система сама проверяет и отслеживает переводы. Созданная заявка преобретает статус "Создана"
3. Заказчик размещает заказ.
4. Исполнитель получает заказ. Условия: назначено администрацией или же назначено самим заказчиком.
5. Исполнитель видит статус заявки: оплачено или нет? Если заявка не оплачена - он может решить(под собственную ответственность) или не решить задание. Если же он решил-выполнил задание, то загрузив его в систему он будет спокоен - пока заявка не получит статус "Оплачено" заказчик скачать его не сможет. Статус на оплаченность меняется оператором сайта при проверке всех исходных данных. Если он не решил задание - заказчик может поменять исполнителя.
6. Заказчик пополняет свой баланс. За счёт своего баланса - он может оплачивать свои заявки. Если задача не решена - он может отменить её. Если он(заказ) уже "оплачена", то она отменяется, деньги возвращаются на баланс заказчика. Он вправе их оставить на своём счету, либо же - вывести. Любая комиссия на вывод средств: 10%. Комиссия на ввод средств - 0%. Но исполнитель не получит денег на свой счёт до тех пор, пока не решит и не загрузит задание. Как только он загрузит решение и если заявка оплачена - заказчик может скачать задание и посмотреть его. Если в течение 3 дней от него не поступило фатальных претензий - деньги за заявку перечисляются на счет исполнителя.
7. Оператор администратор проверяет все поступившие переводы на счет от тех или иных Заказчиков и распределяет в их счет-баланс.

У заявки есть следующие статусы:

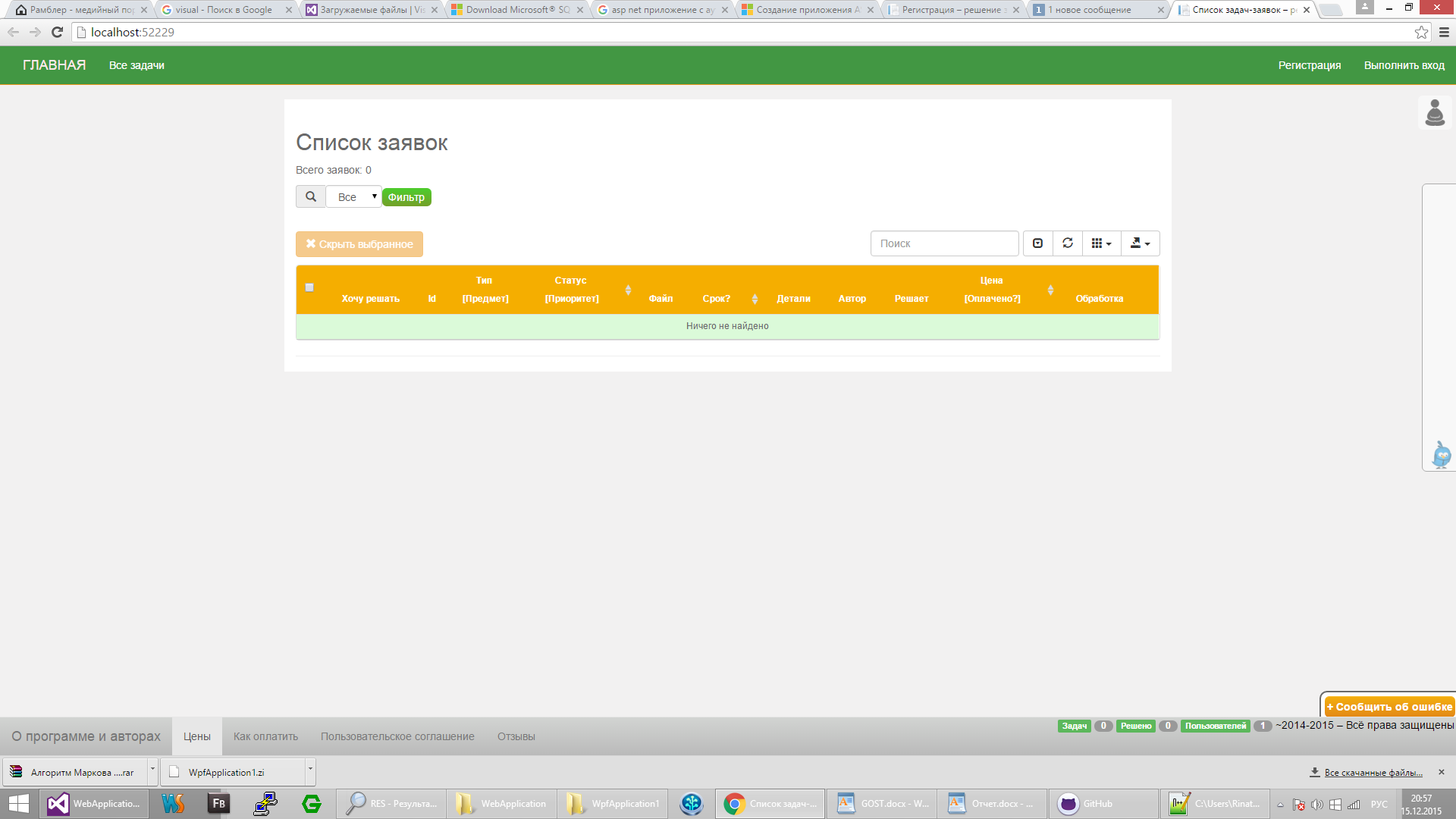
* Создана.
* Назначена.
* Оплачена.
* Отменена.
* Удалена.

1. У каждого пользователя есть своя карма - рейтинг.
2. Администратор-оператор следит за финансовой составляющей. Это:
3. отслеживание поступлений средств и их перераспределение по балансам
4. отслеживание переводов внутри системы от одного пользователя к другому(от заказчика к исполнителю)
5. отслеживание и контроль вывода средств со счетов исполнителей и заказчиков(10% комиссия)
6. Единый чат! Возможность общаться между собой заказчикам и исполнителям, с подключением Арбитража (администрации системы, КГБ, на вкус и цвет - человек, который может решить разногласия)
7. **Пользовательский режим**

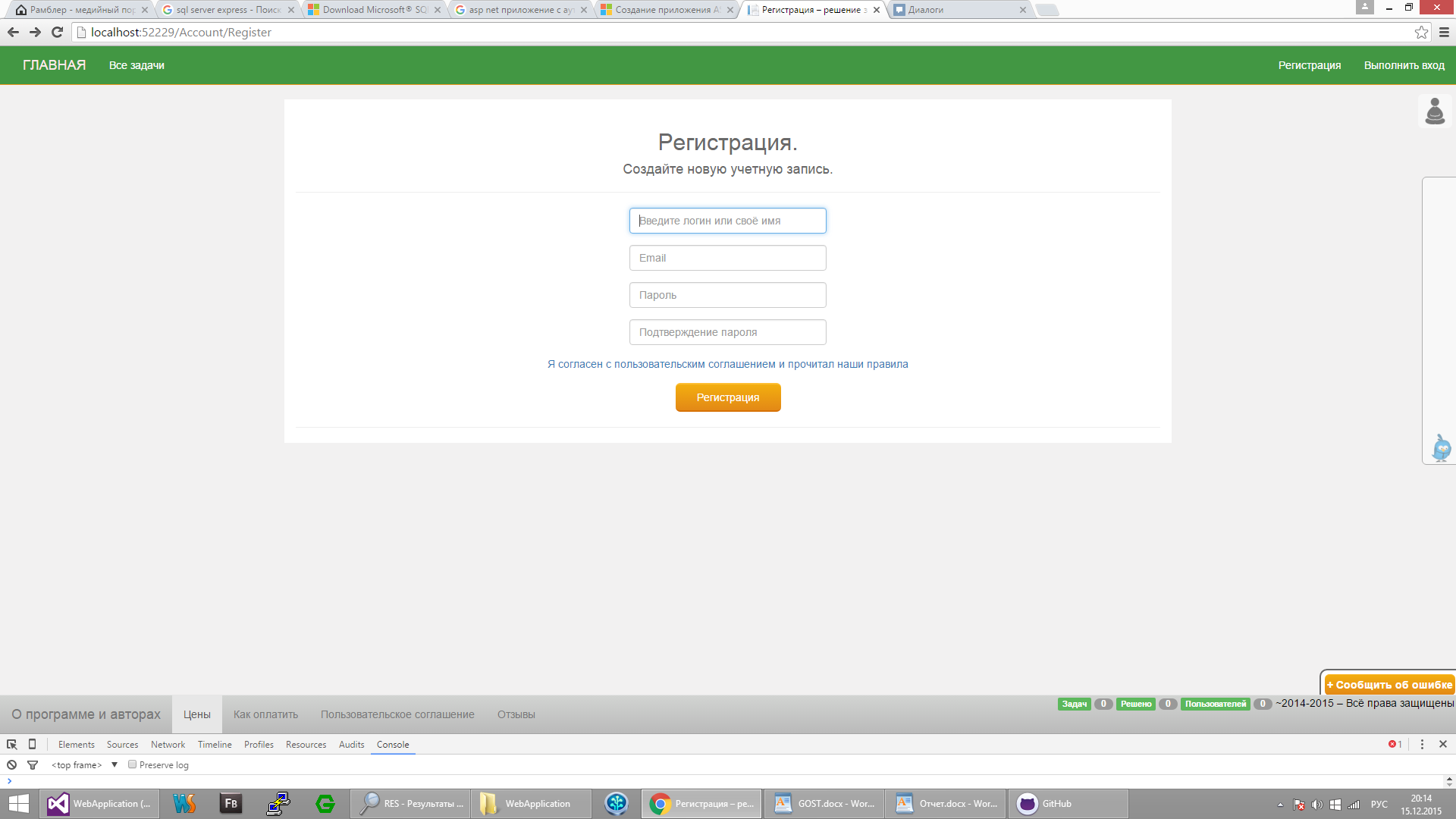
Пользователь, попав на сайт, ничего критичного не может сделать. Он - только может видеть панель задач, активные задачи, он видит, что сайт функционирует. Но чтобы принять участие в работе(получить задачи, решать их и т.п.) - ему нужно зарегистрироваться. И он попадает на панель регистрации.

Сейчас сайт находится в тестовом режиме, но он воспринимает три роли: Администратор, Модератор, Пользователь, с соответствующими логическими обязанностями и возможностями.

Нижний футер(footer) - это статистическая информация. Сколько задач загружено - сколько решено - сколько людей(пользователей) зарегистрировалось.



**3.1 Панель регистрации**



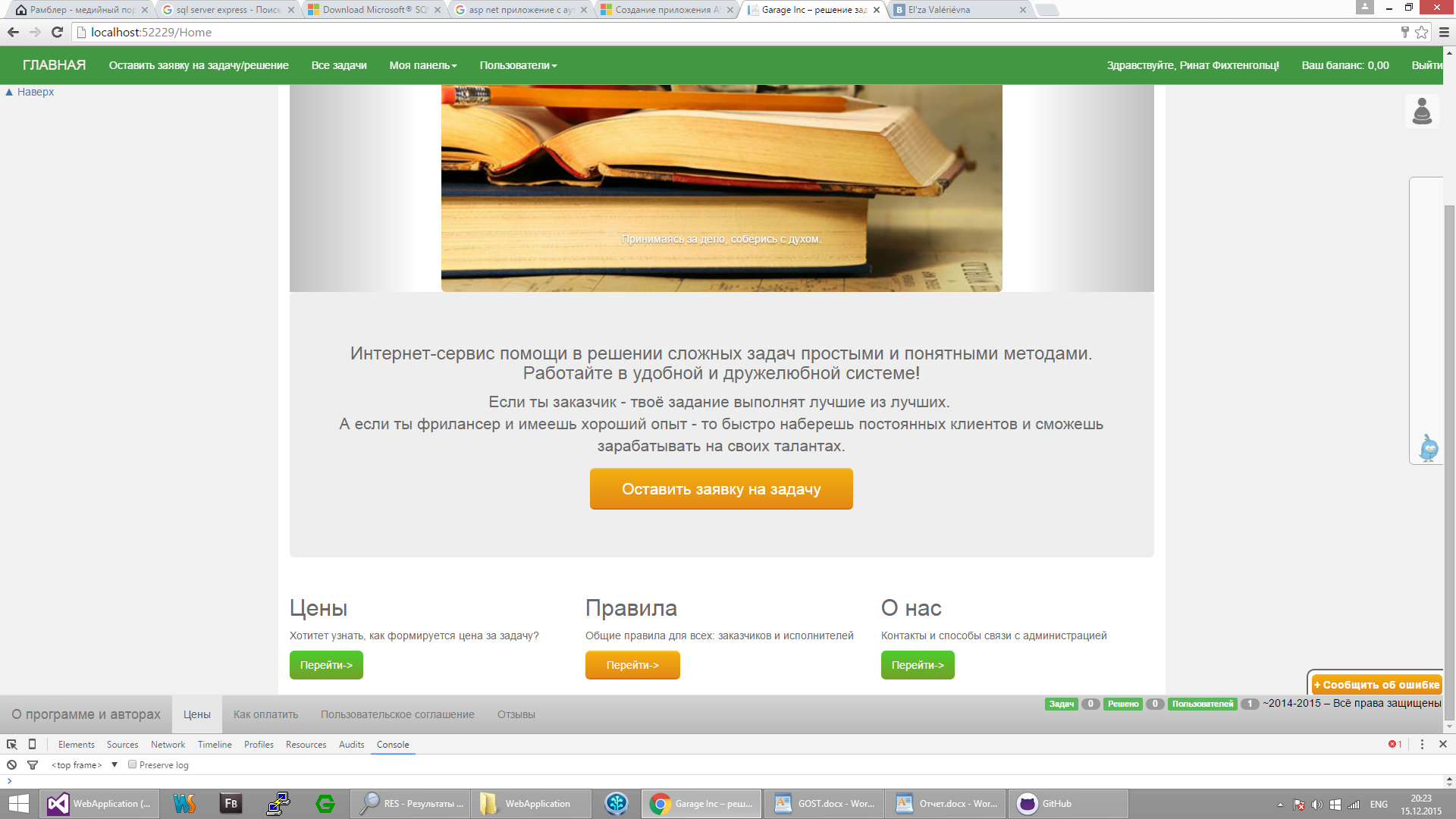
По умолчанию в тестовом режиме на сайте может зарегистрироваться кто угодно и ему предоставляются административные права. Режим ведь - тестовый. В "боевом" режиме такого, конечно же, нет. Никакого всевластия, строгое разграничение прав.

Разработчик в моём лице приветствует компонентный подход и эксперименты. Пару мегобайт и килотысяч строк кода многого не изменят, но позволят оперировать контентом так, как удобно.

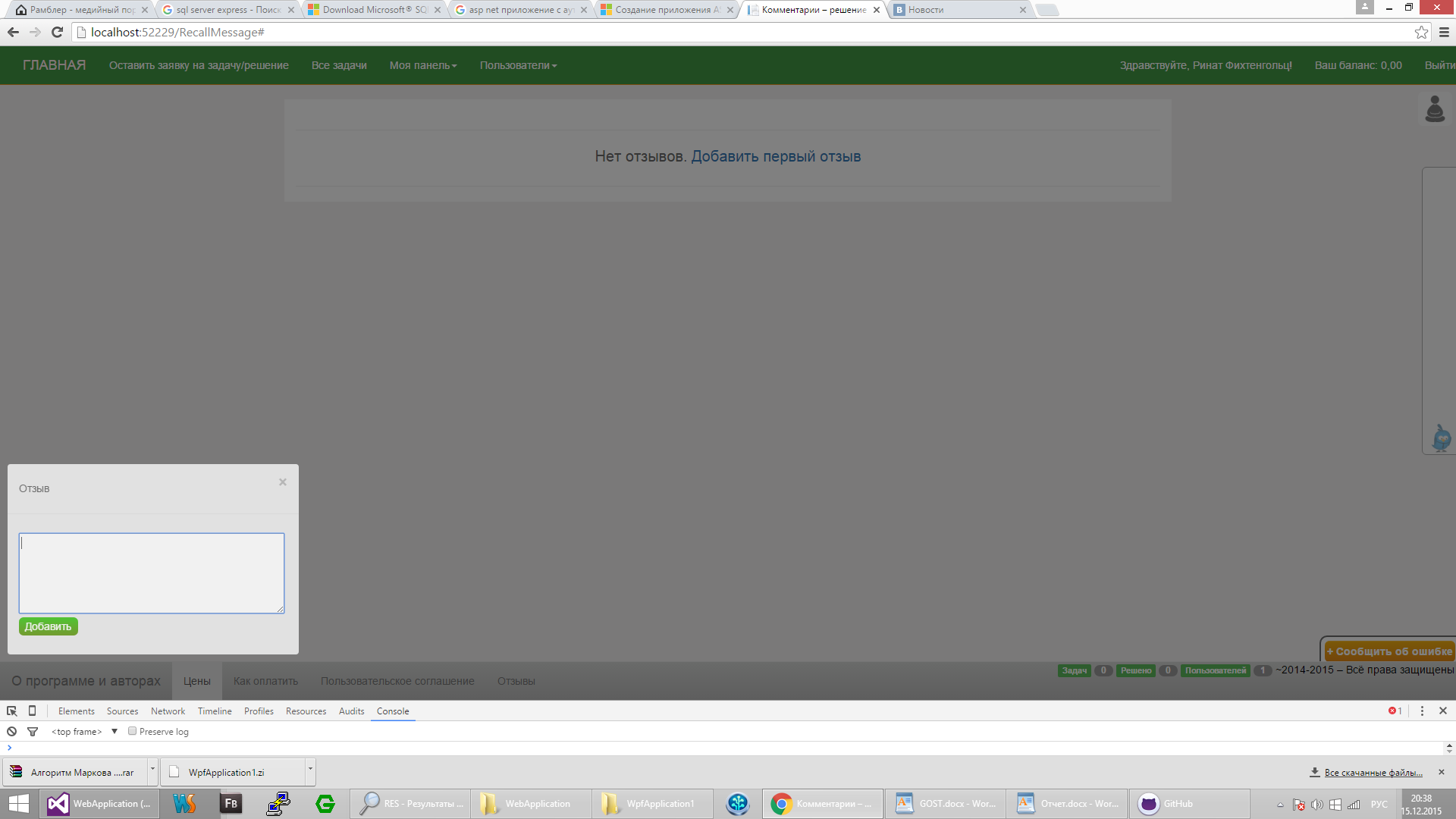
Итак, пользователь зарегистрировался. И он в режиме "бога". Ввод пароля и сам процесс регистрации - максимально простой, без всяких подтверждений. Это раздражает пользователей.

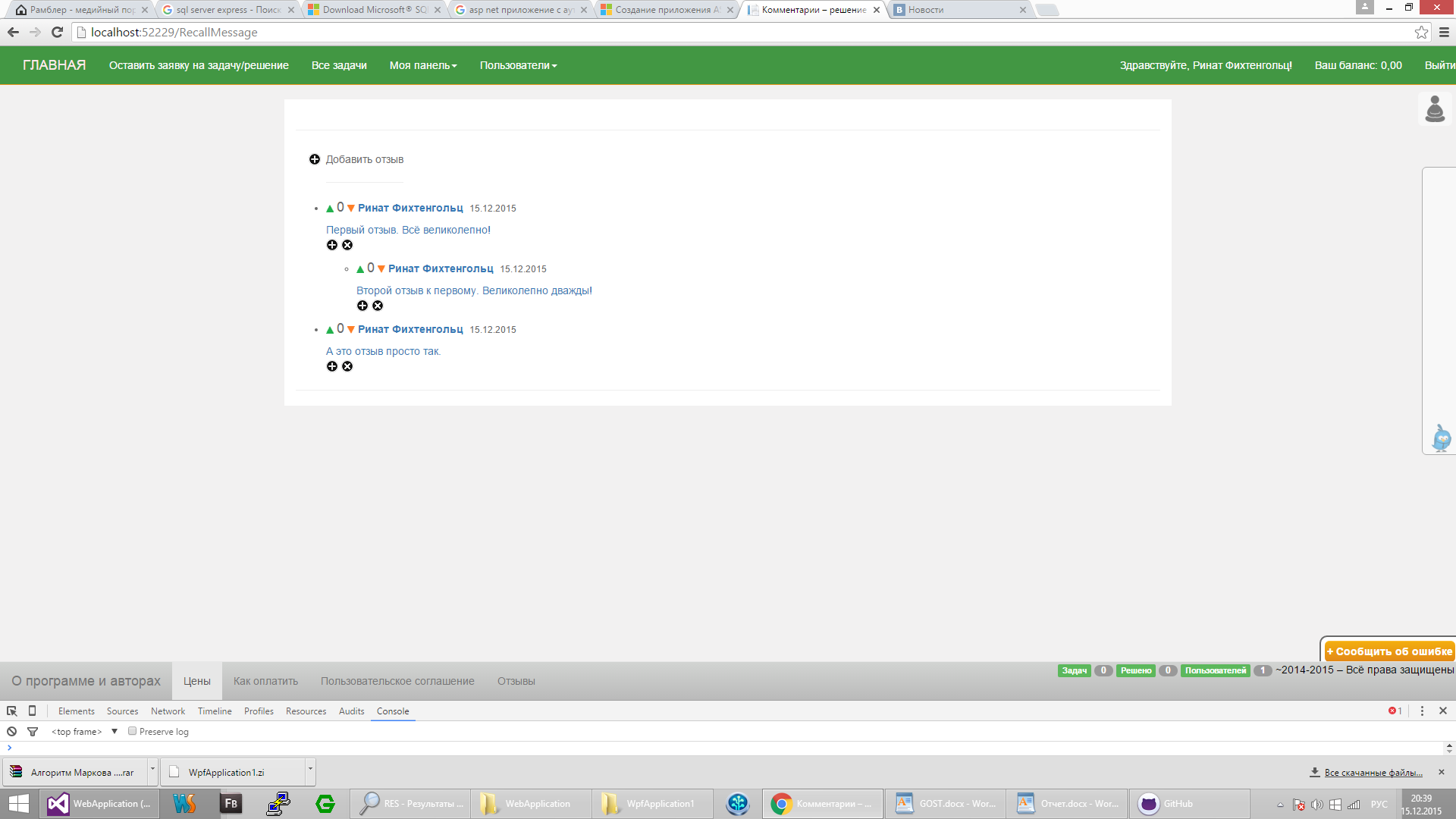
**3.2 Регистрация**

Итак, пользователь зарегистрировался. Теперь он может что-то делать. Что же именно?



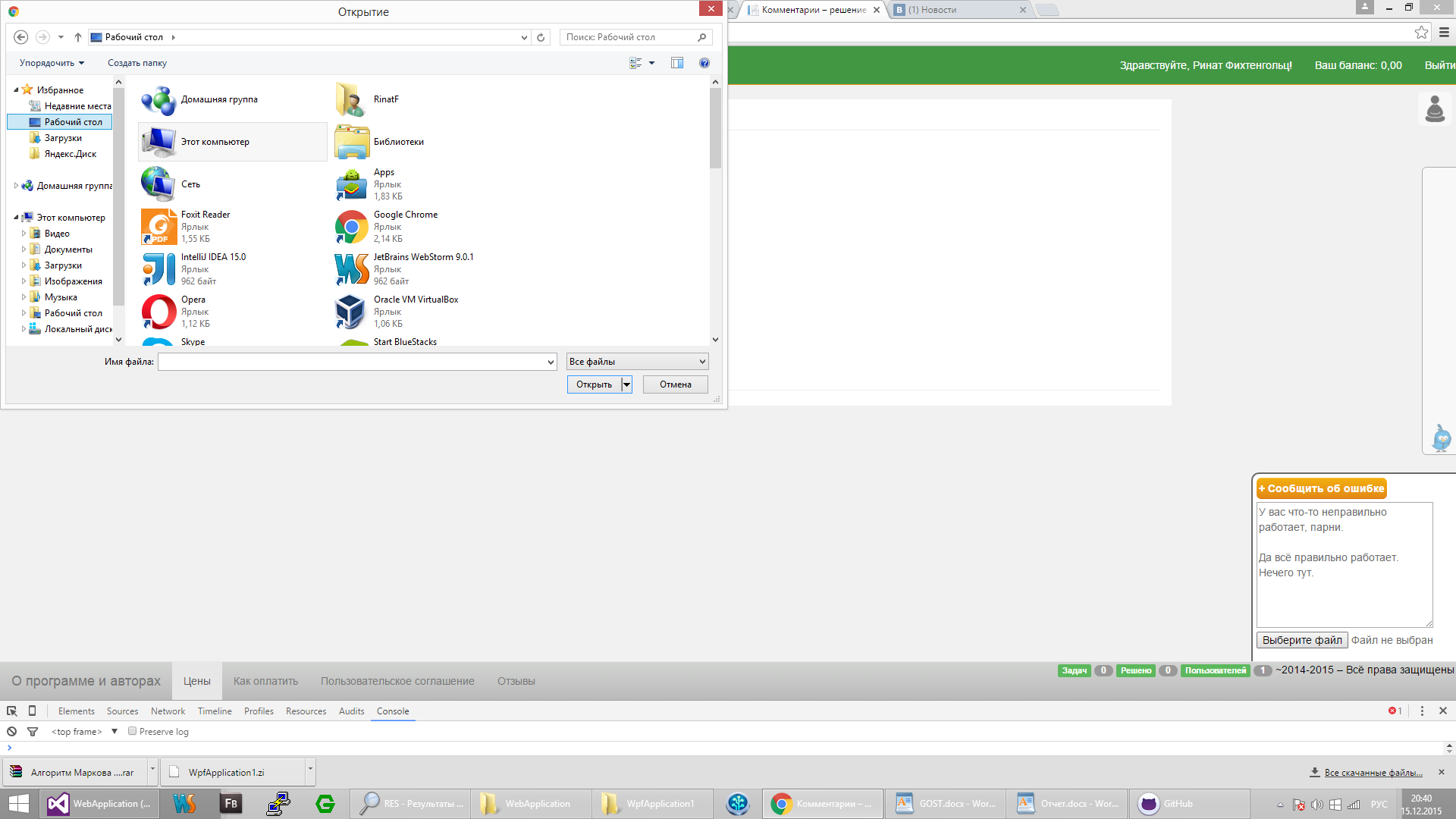
Внизу - информационная панель. Это "О программе и авторах", "Цены", "Как оплатить", "Пользовательское соглашение", "Отзывы" - т.е. полностью статичная информационная область. За исключением отзывов - я их сделал и в виде "дерева", любой пользователь может оставить свой отзыв. Удаление чьего-то отзыва автором - не вызывает удаление всех отзывов, закрепленных за ним.





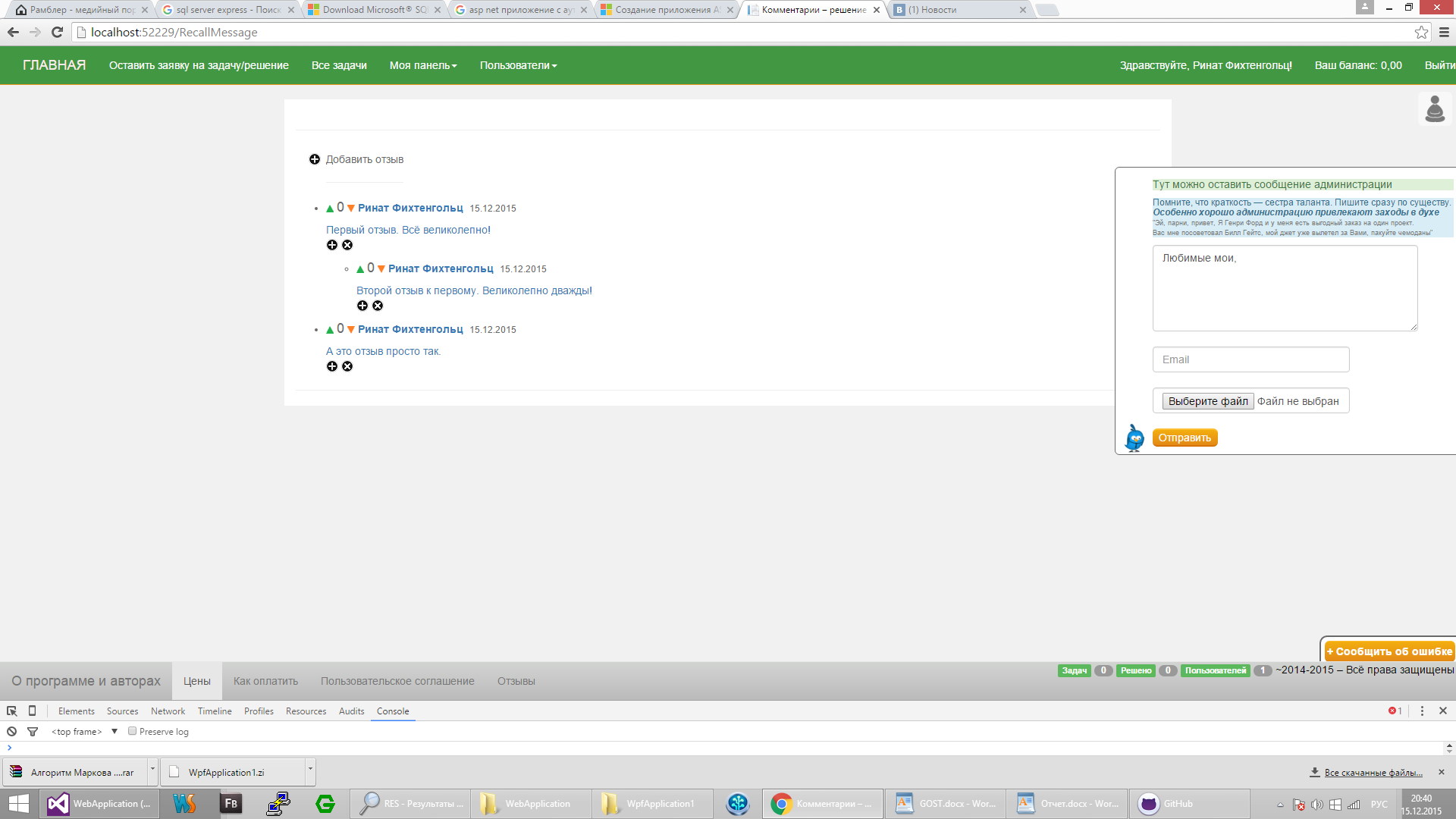
**3.3 Обратная связь**

Что же ещё может пользователь сделать? Если он вдруг нашёл баг в работе сайта - то может об этом сообщить. Справа внизу - кнопка "Сообщить об ошибке"



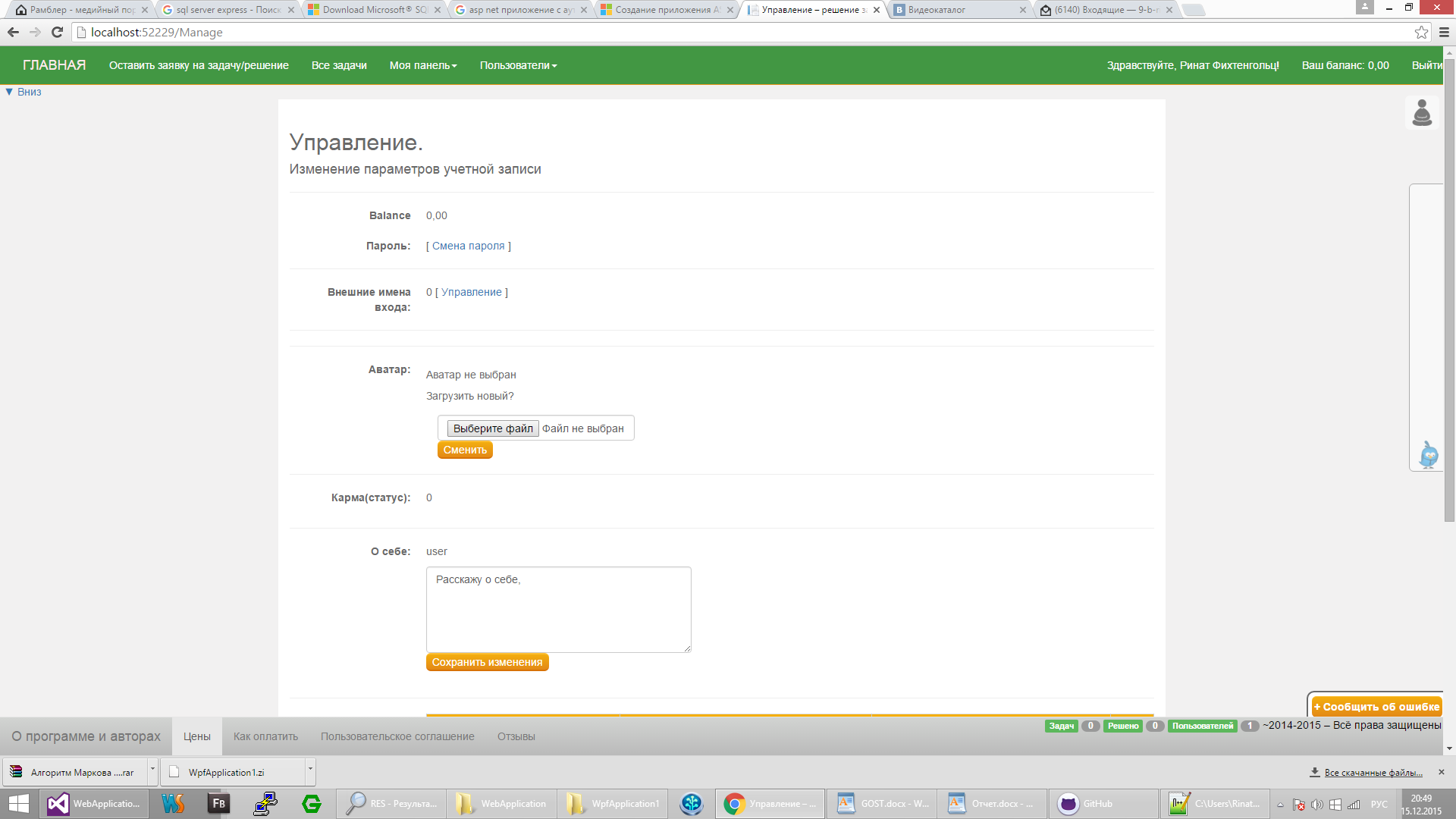
**3.4 Фидбэк**

Либо, если у него есть какая-то идея или предложение - может написать об этом администрации лично! Своеобразный фидбэк с помощью выдвижных панелей:



**3.5 Аккаунт**

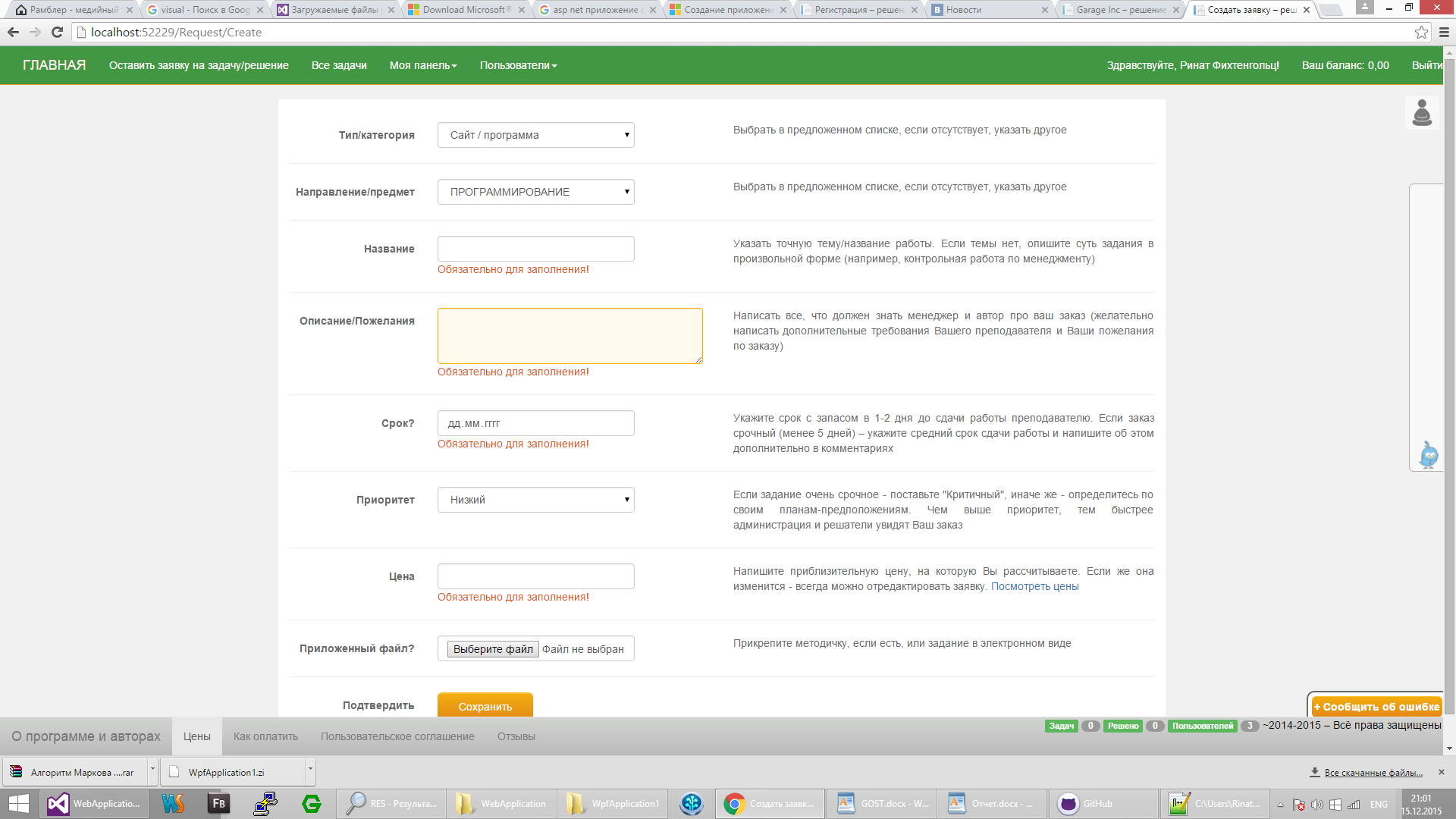
В верхнем правом углу - пользователь может зайти на свой аккаунт. Что он может там сделать?



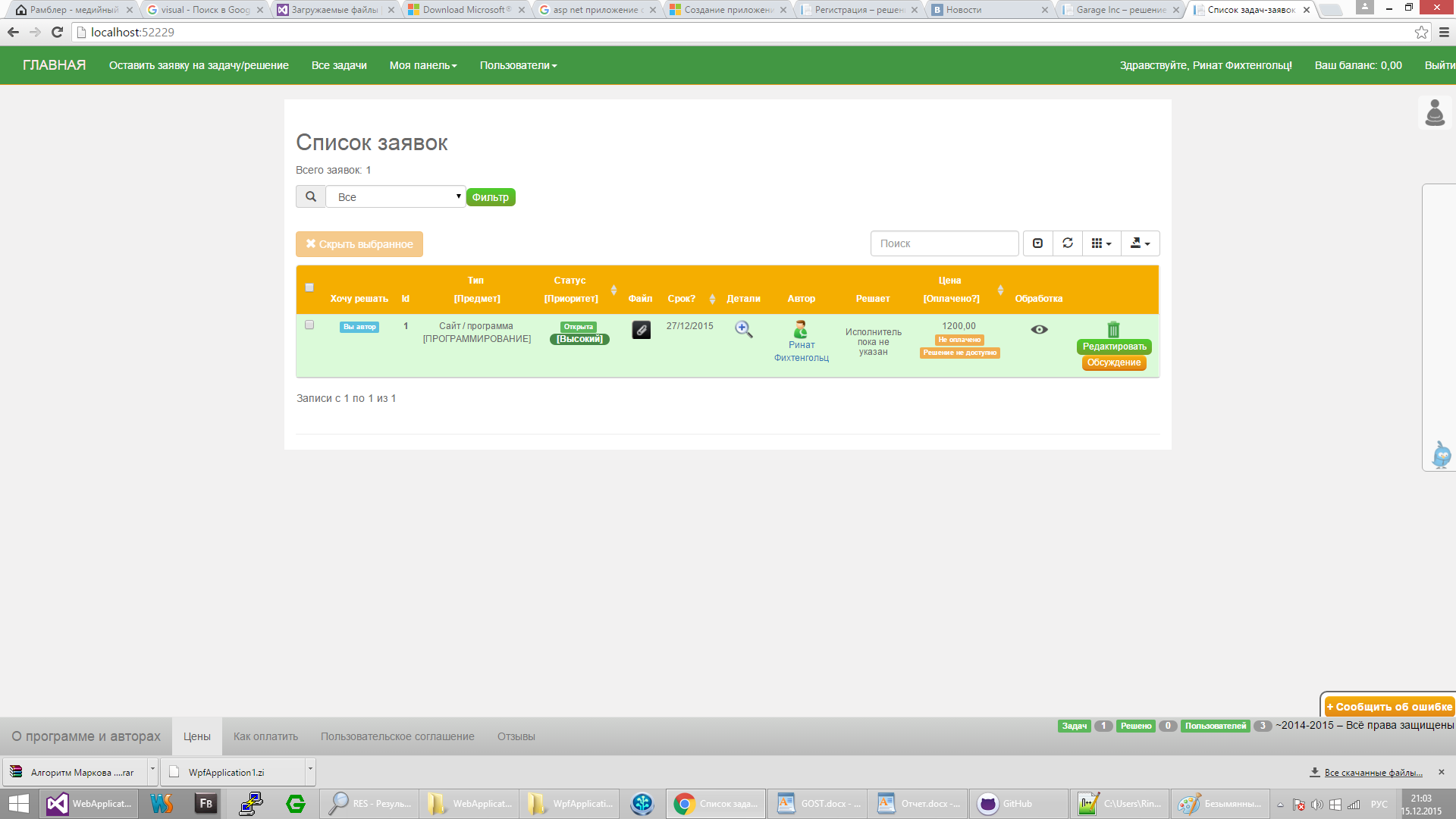
Прежде всего:

1. Посмотреть свой баланс.
2. Сменить пароль
3. Добавить внешние имена входа(соц.сети)
4. Изменить аватар(пиктограммочка напротив его записей)
5. Добавить небольшое своё резюме - информацию о себе.
6. Добавить реквизиты - Тип и номер счета-кошелька.
7. А так же - добавить контакты(как с ним можно связаться, в случае близкого дедлайна, например).
   1. **Заявки**

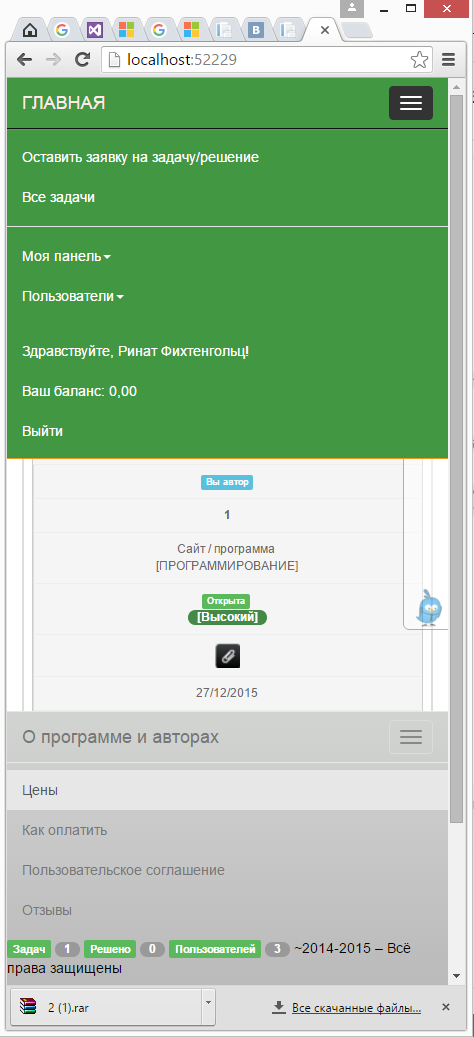
И главное! Для чего создавался сайт, его идея? Прежде всего, это возможность оставить задачу на решение!



После введения параметров своего задания - пользователь нажимает сохранить и видит своё задание в списке актуальных задач:



Отступление от темы: система также адаптирована под разные размеры экранов:

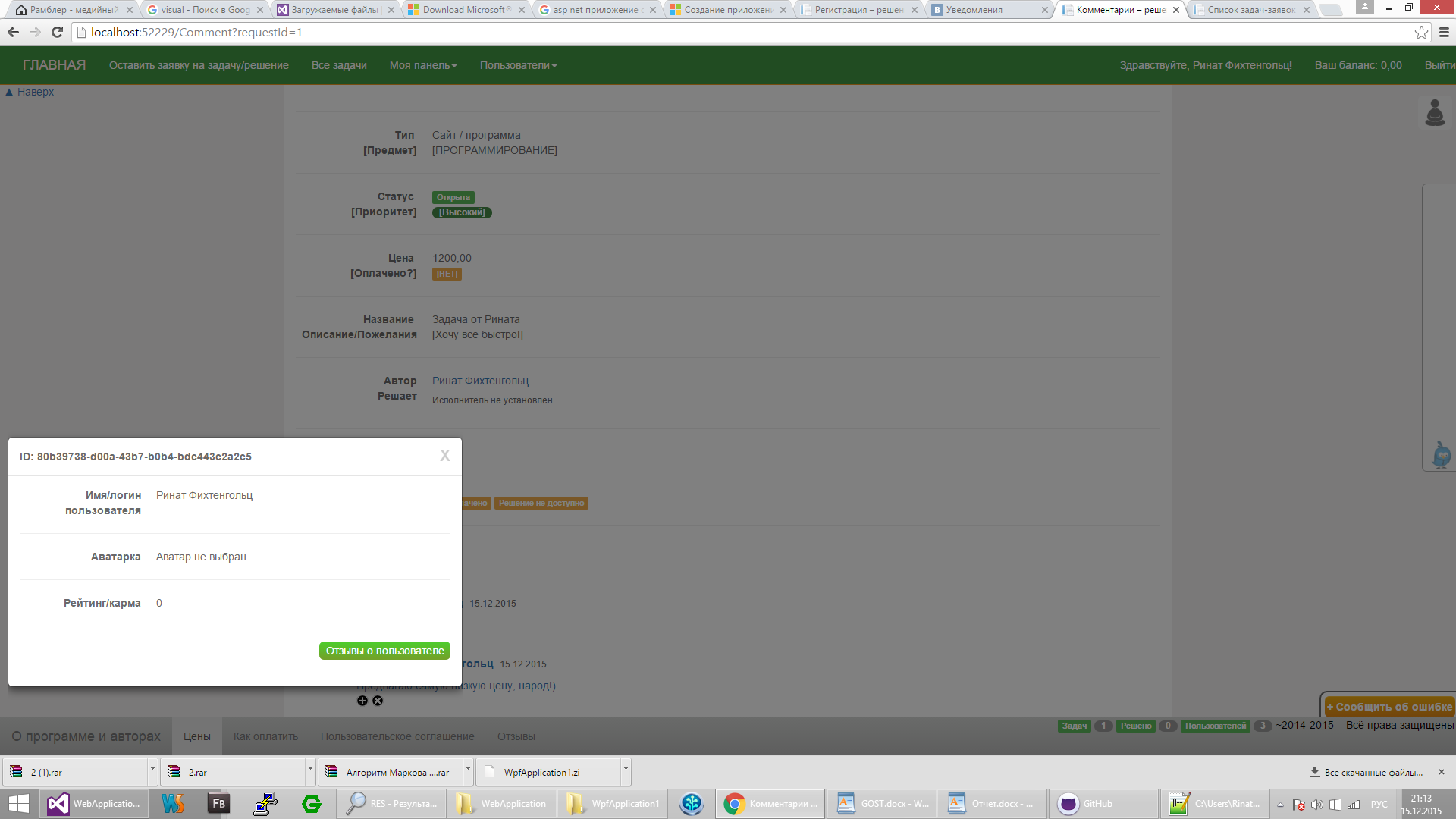


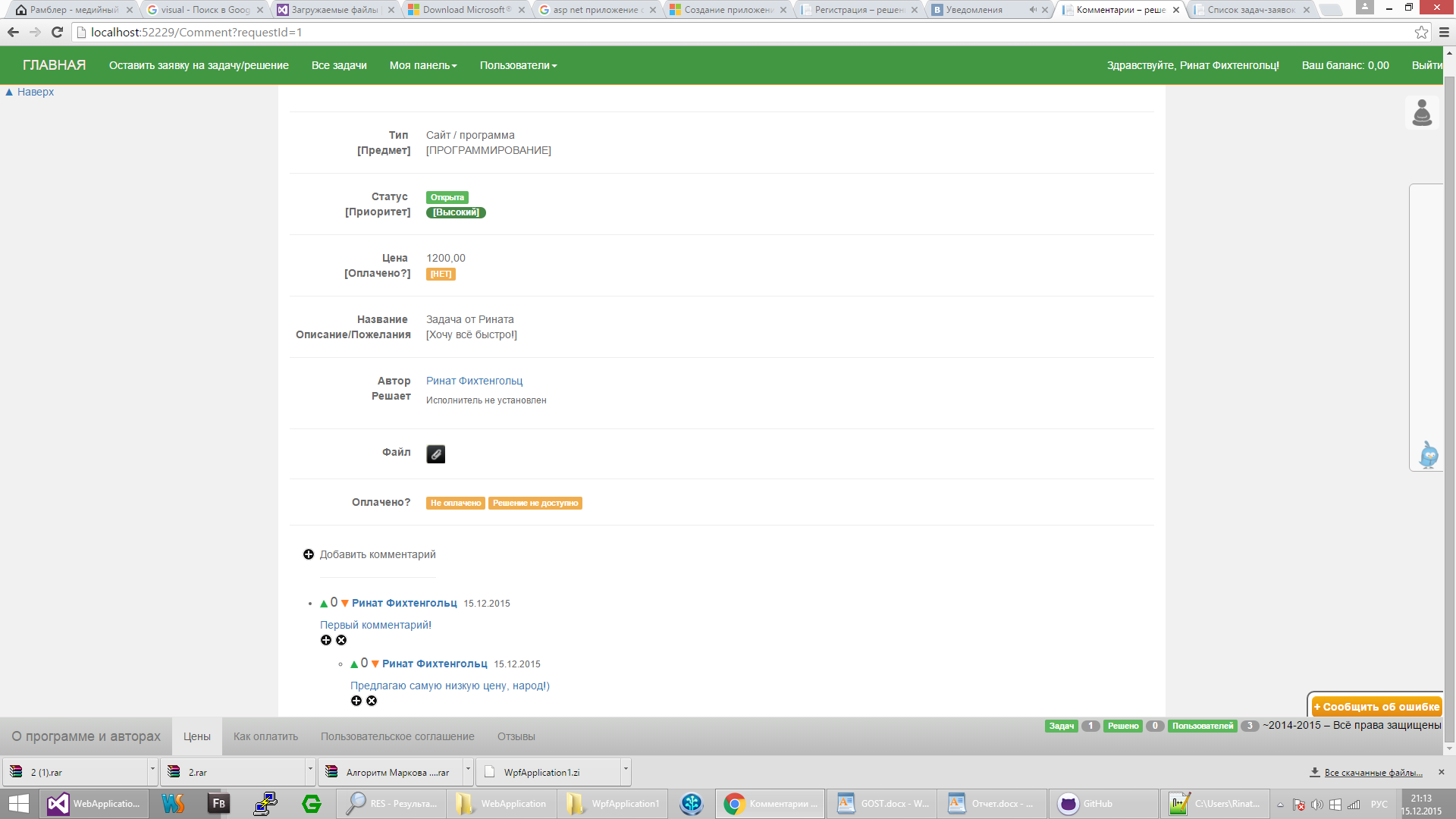
Что может тут увидеть пользователь(на экране всех задач):

Прежде всего:

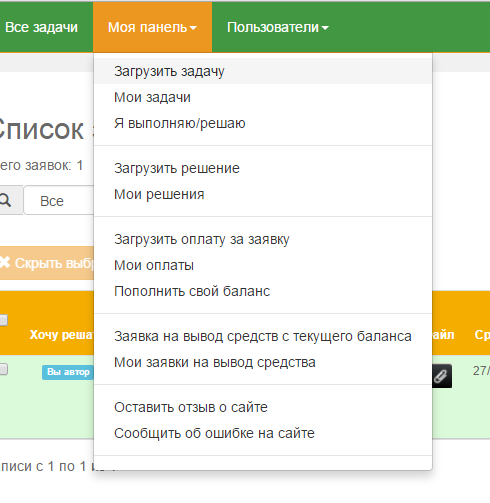
1. Таблицу актуальных задач.
2. Фильтр этих актуальных задач.
3. Возможность выбора любой задачи на решения(он нажимает "Хочу решать", его отклик - получает автор задачи и сам может назначить решателя, если это не сделает администрация)
4. Видит приоритет этот задачи
5. Может скачать задание
6. Видит срок последней сдачи.
7. Может узнать информацию по автору либо решателю задачи
8. Узнает финансовое состояние(стоимость и оплаченность).
9. Как автор - он может удалить задачу(в реальности, задача не удаляется, а просто помечается флагом как скрытая), редактировать заявку(если вдруг цена-описание изменились) или же - может перейти к обсуждению задачи. Что такое обсуждение? Прежде всего его цель - возможность исполнителей договориться по цене с заказчиком. При это заказчик может видет рейтинг исполнителей, их выполненные задачи и, самое важное, отзывы по исполнителям от других заказчиков! Система полностью прозрачна и чиста для обеих сторон. Нет никаких раздражающих скрытых блоков.

**3.7 Обсуждения задачи**





* 1. **Рабочая панель**

У пользователя есть также своя рабочая панель в виде меню:  
  


Там он может:

1. Загрузить задачу. Логику рассмотрели только что.

2. Увидеть свои задачи. То есть те задачи, которые он загрузил.

3. Увидеть задачи, которые он взялся решать. Т.е. те задачи, которые ему назначил автор или администратор и за которые он взялся.

Так же он может:

4. Загрузить решение к той задаче, которую взялся решать.

5. Увидеть те решения, которые он загрузил ранее.

Если же он отправил заявку на задачу - ему нужно её оплатить. Как он может это сделать?

6. Панель "Загрузить оплату за заявку" - позволяет оплатить.

7. "Мои оплаты" - позволяет увидеть все оплаты.

8. Пополнить свой баланс - он может пополнить свой баланс.

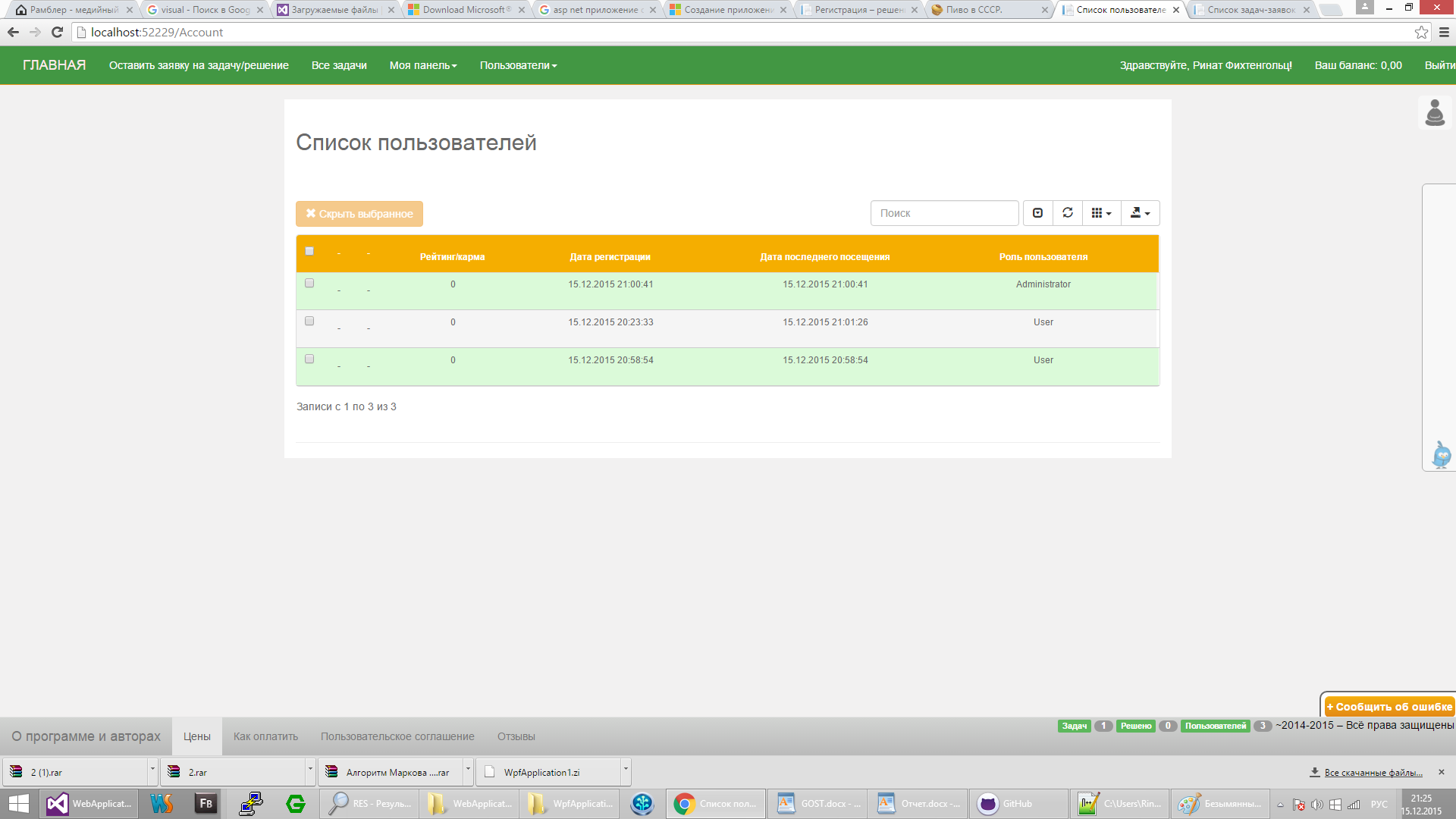
9. Если же вдруг пользователь решил, что прилично заработал и надо бы погулять на заработанные деньги, он может отправить заявку на вывод средств с текущего баланса. Обработка происходит настолько быстро оператором, насколько это возможно.

10. Так же он может увидеть свои заявки на вывод средств(которые отправил ранее).

А так же - в виде панелей тот функционал, который он видит и по бокам: возможность оставить отзыв и возможность сообщить об ошибке на сайте.

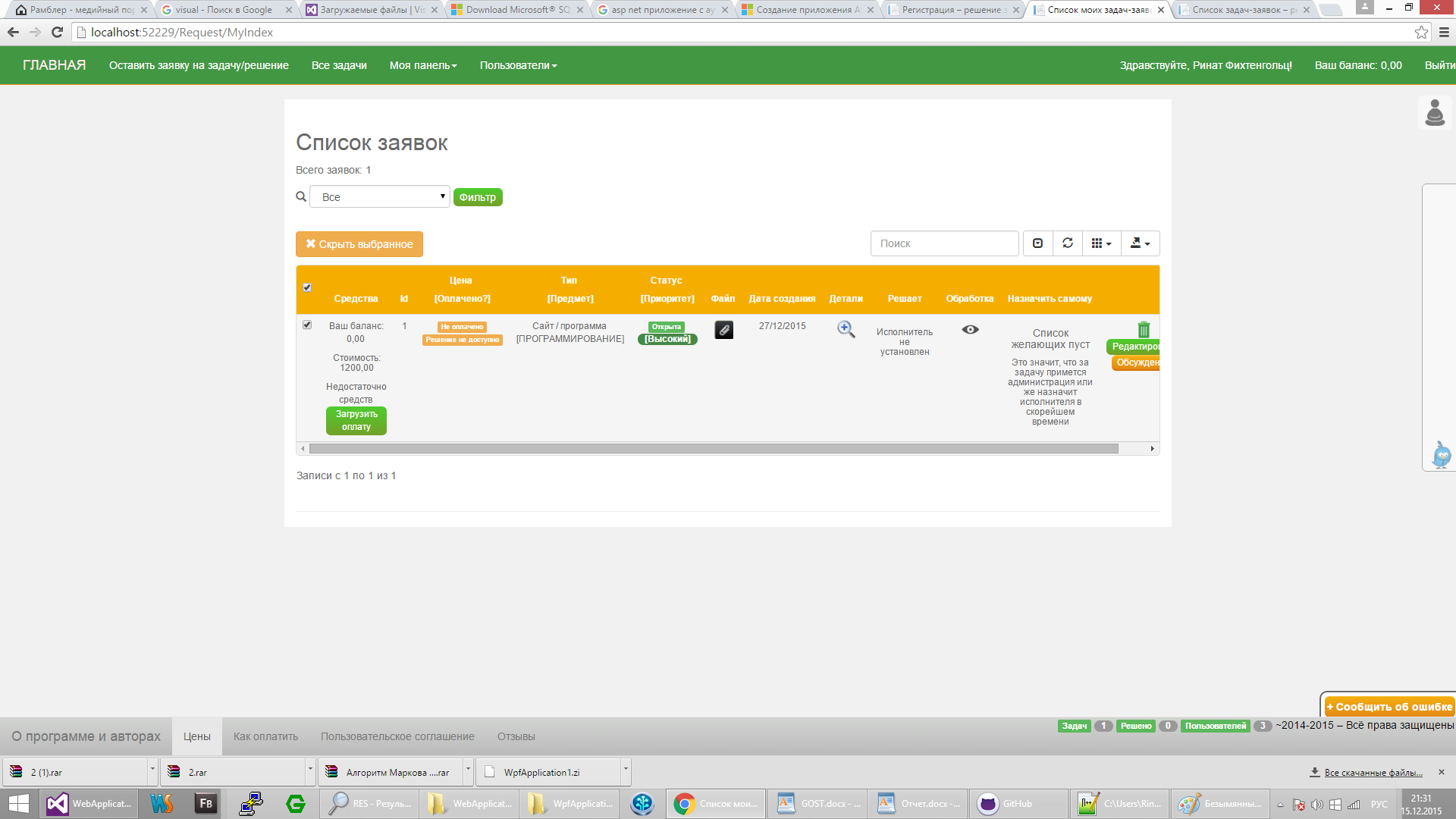
* 1. **Зарегистрировавшиеся пользователи**

Так же обычный пользователь может увидеть список всех пользователей, но, увы, только в виде статистики: для администрации возможен доступ по логину-паролю.



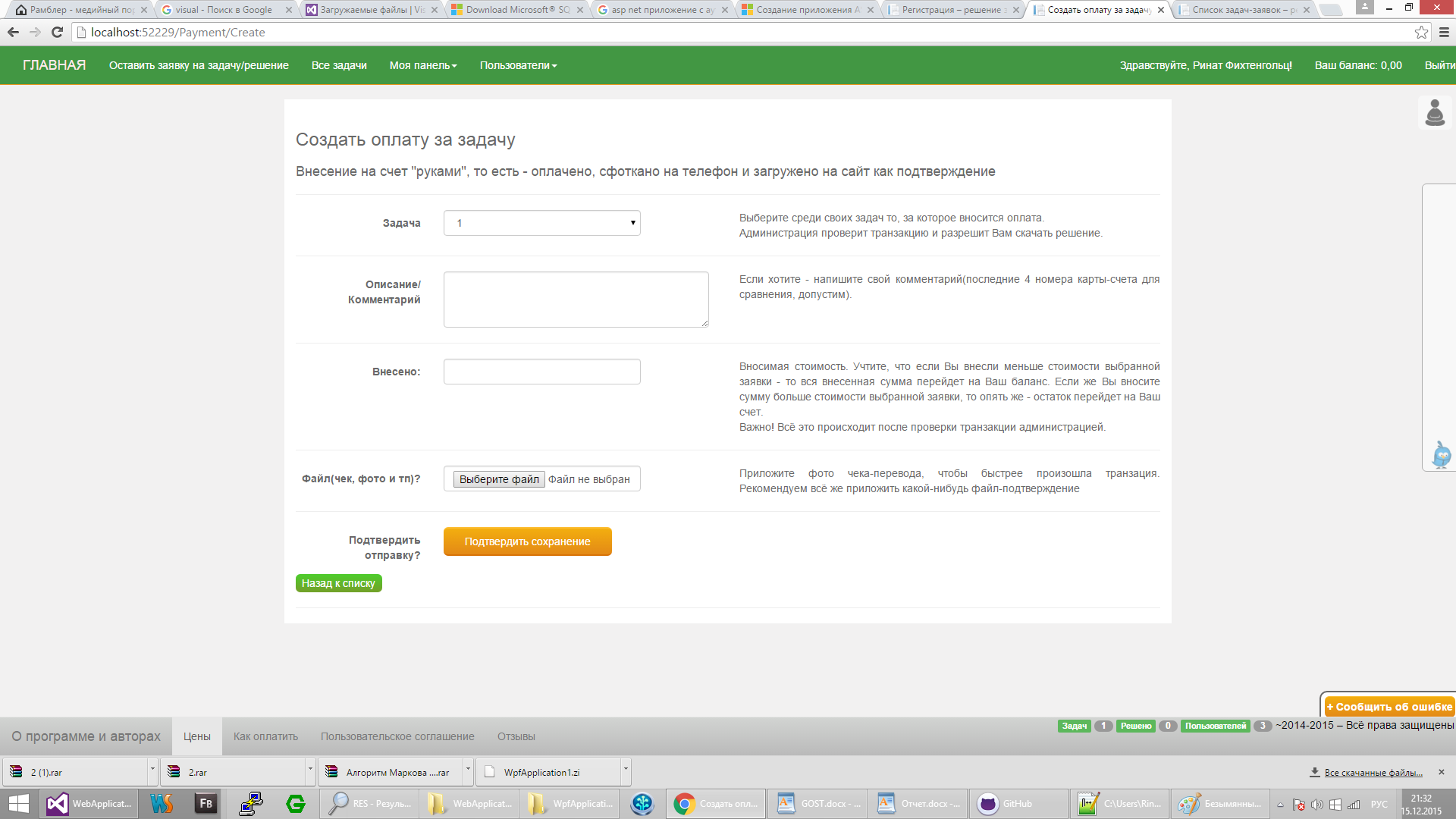
Посмотрим подробнее, что может делать пользователь как заказчик и как исполнитель.

**3.10 Панель "Мои заявки"**



Тут пользователь видит свои заявки, которые он загрузил.

a. Если заявка не оплачена - он может оплатить.

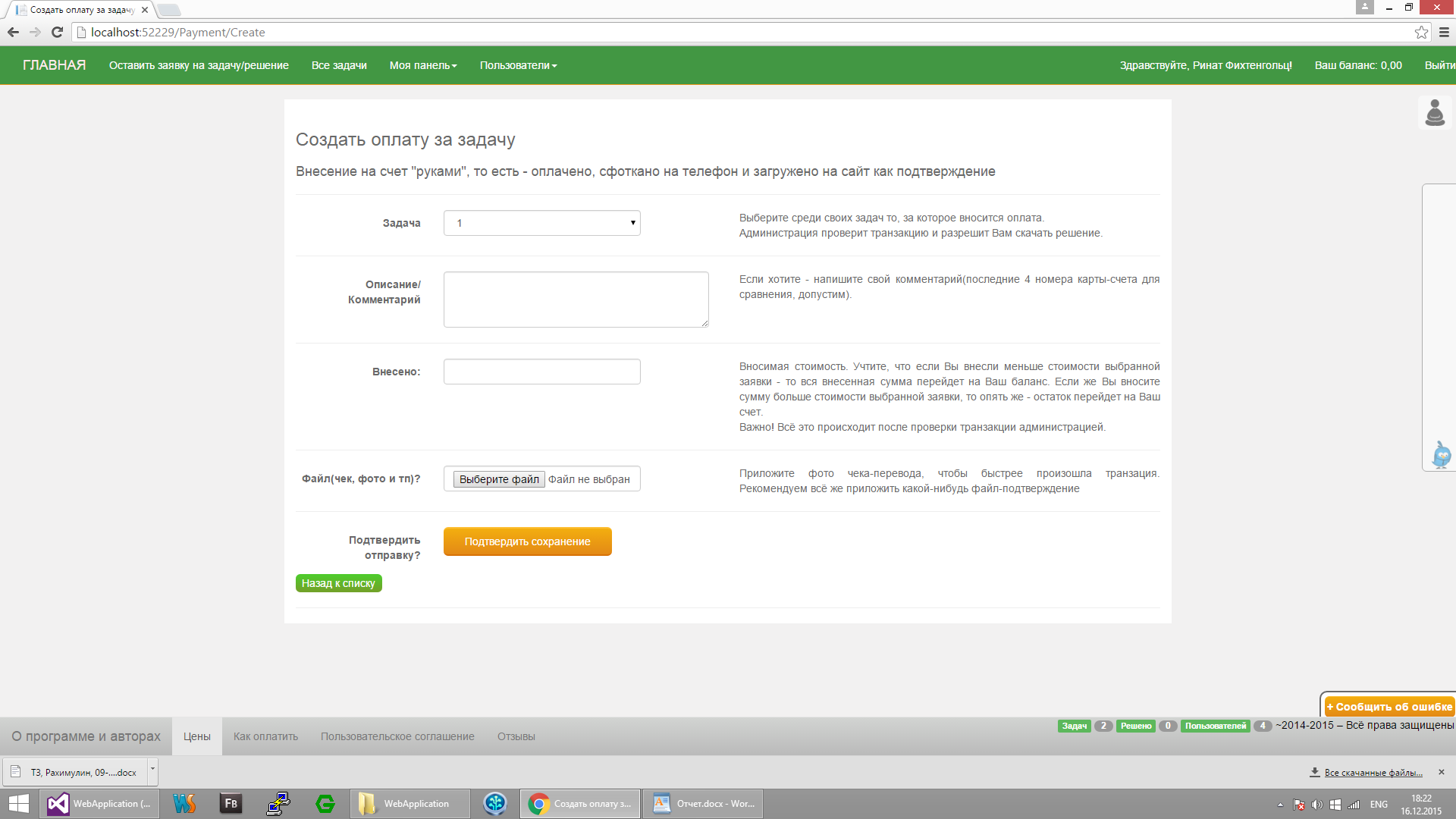


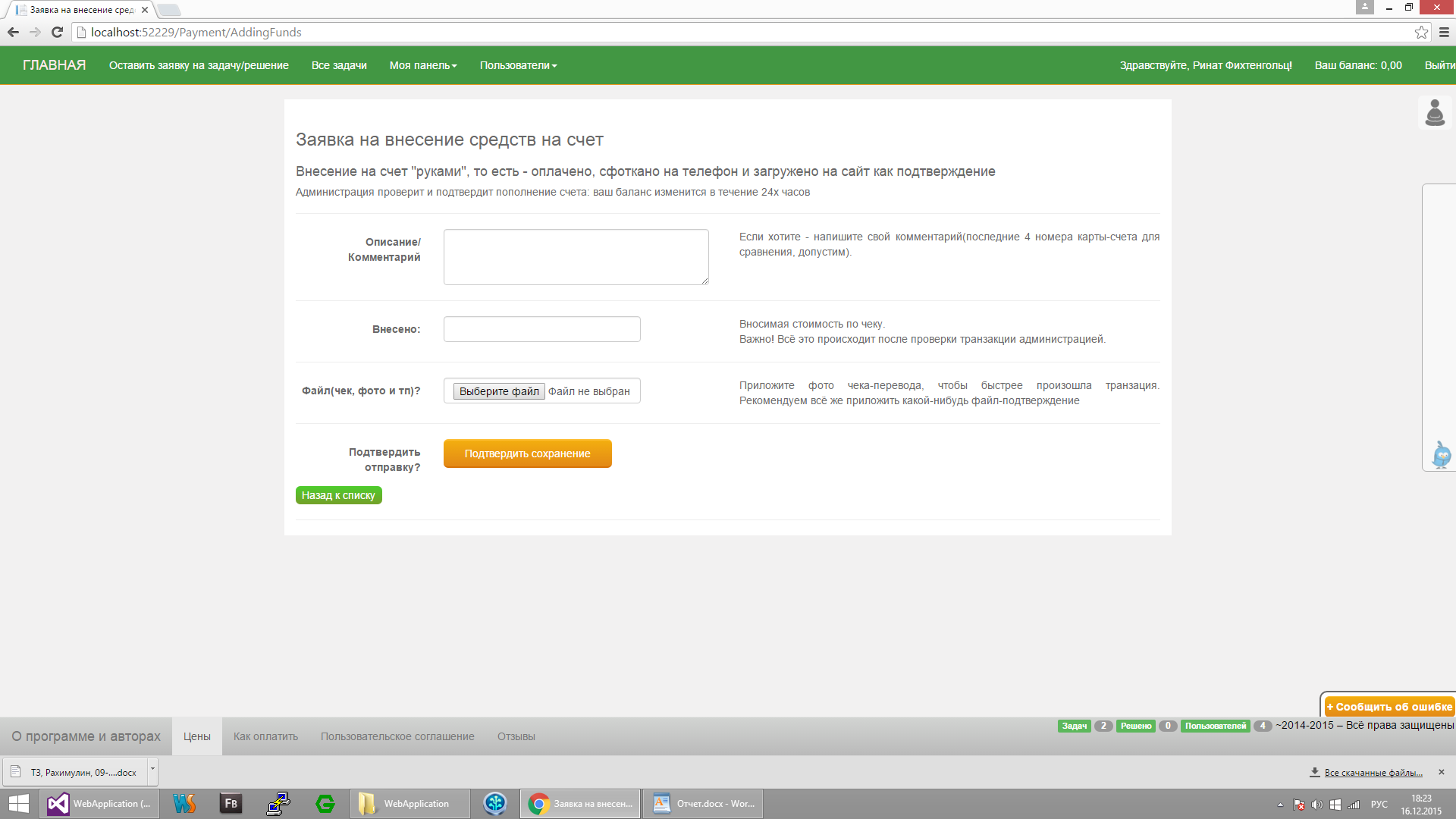
б. Пока заявка не оплачена - даже если решение уже кем-то загружено - он не может скачать. Это - элемент системы защиты решателя, что предусмотрено и в серверной части.

Также он видит список желающих и если среди них тот, в ком он уверен - то может сам назначит исполнителя своей задачи. И тут же - он видит свой баланс, может перейти к обсуждениям задачи.

**3.11 Панели "Загрузить оплату" и "Пополнить баланс"**

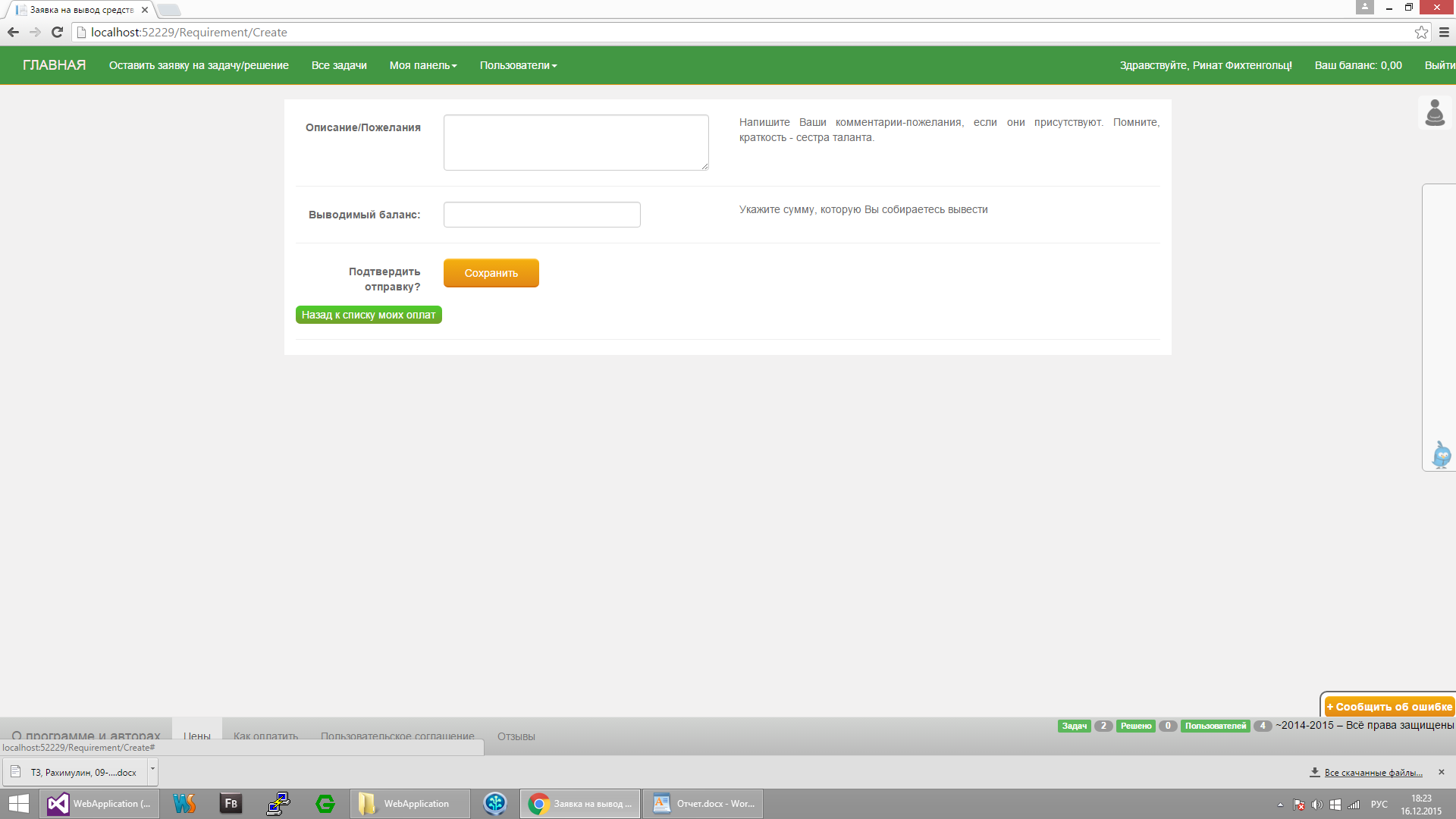
Они позволяют пополнить баланс пользователя либо оплатить одну из загруженных задач





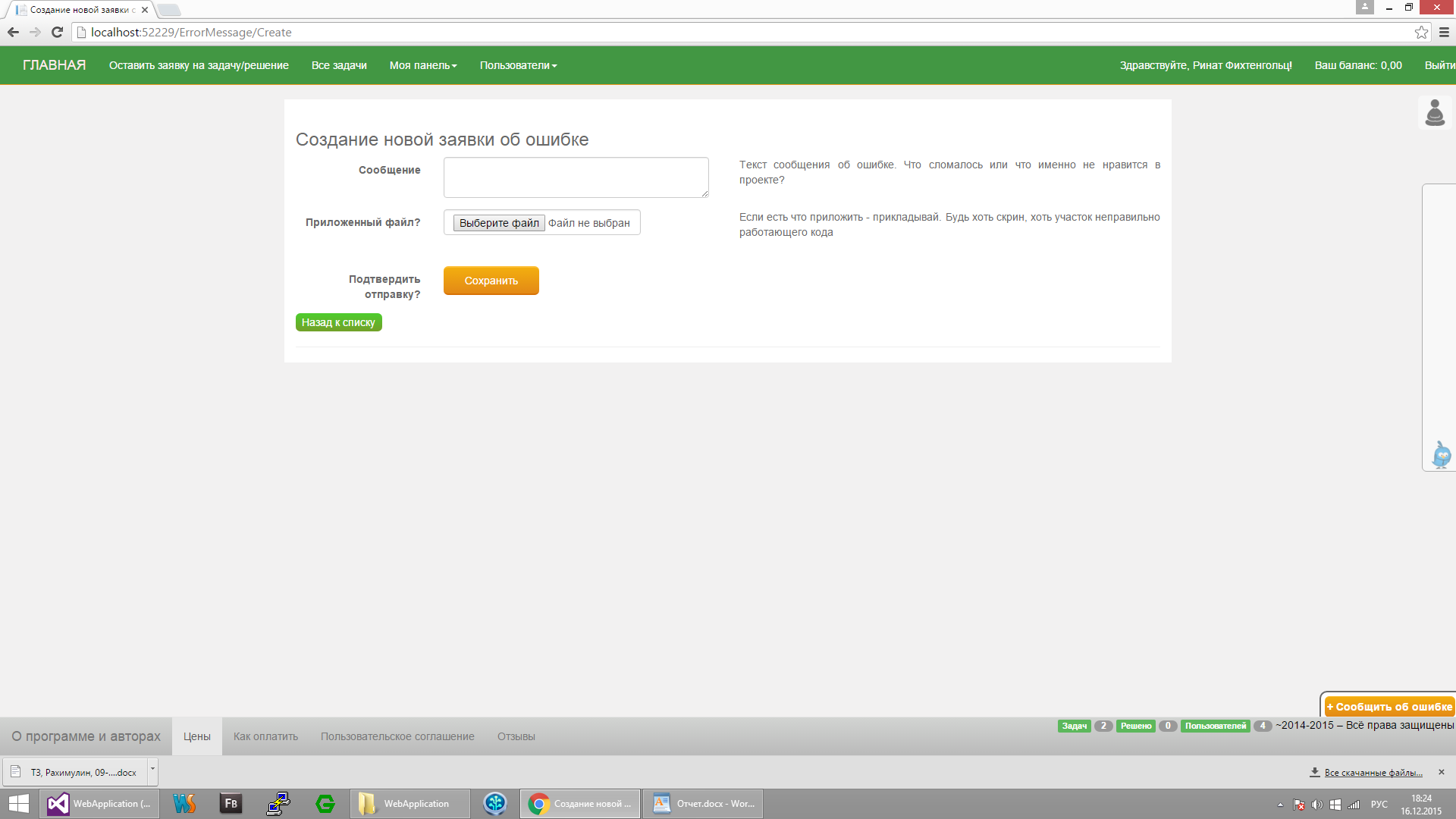
**3.12 Заявка на вывод средств**

Данная панель позволяет пользователю отправить заявку на вывод средств.



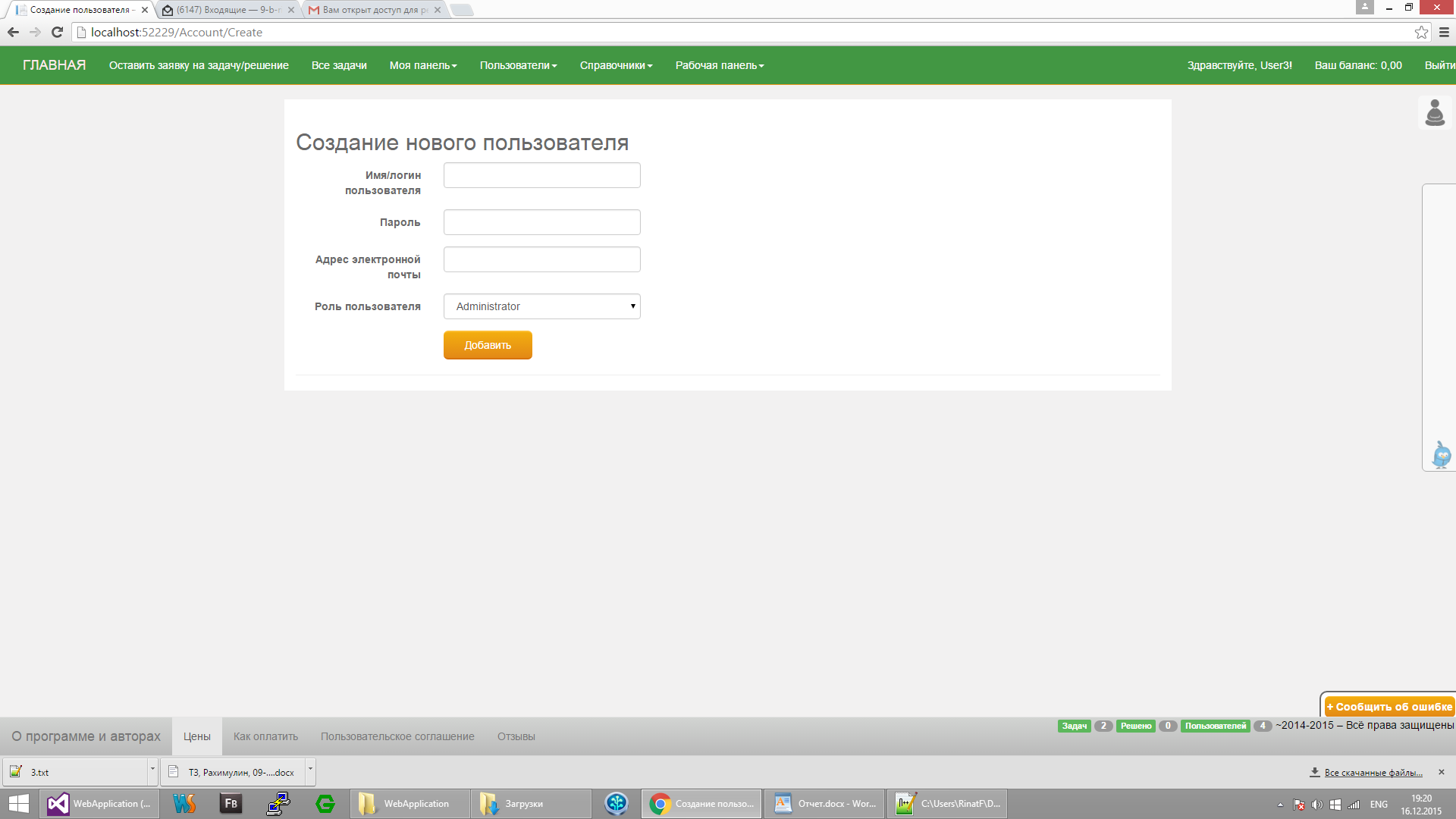
**3.13. Панель "Сообщить об ошибке"**

Позволяет в случае чего оперативно решать возникающие вопросы в виде обратной связи с администрацией.



**4 Администрирование**

Что может делать администратоор? Администратор может создать пользователя и редактировать пользоваетелей(Изменять их пароли, назначать роли и тп.

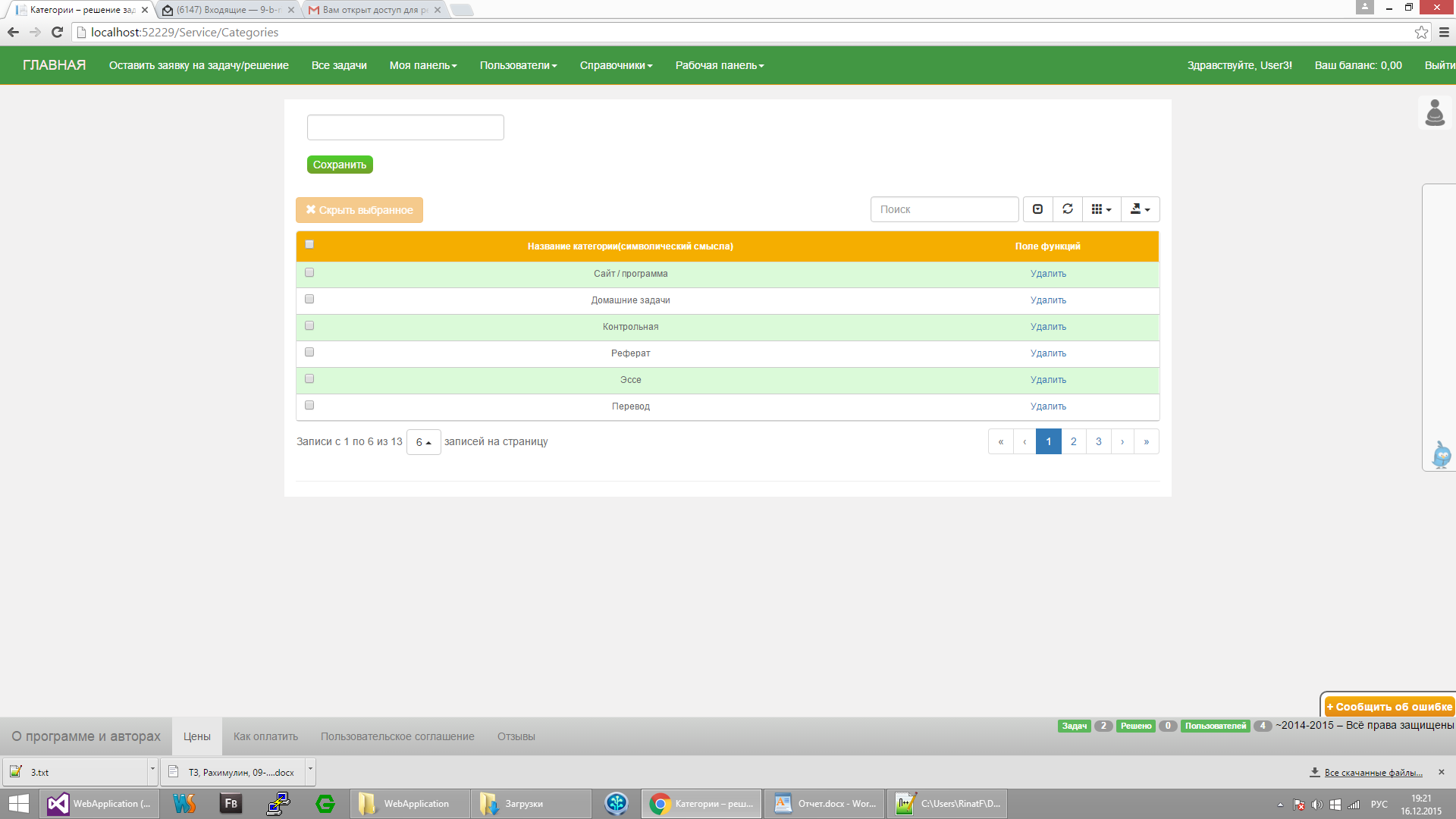


**4.1 Справочники**

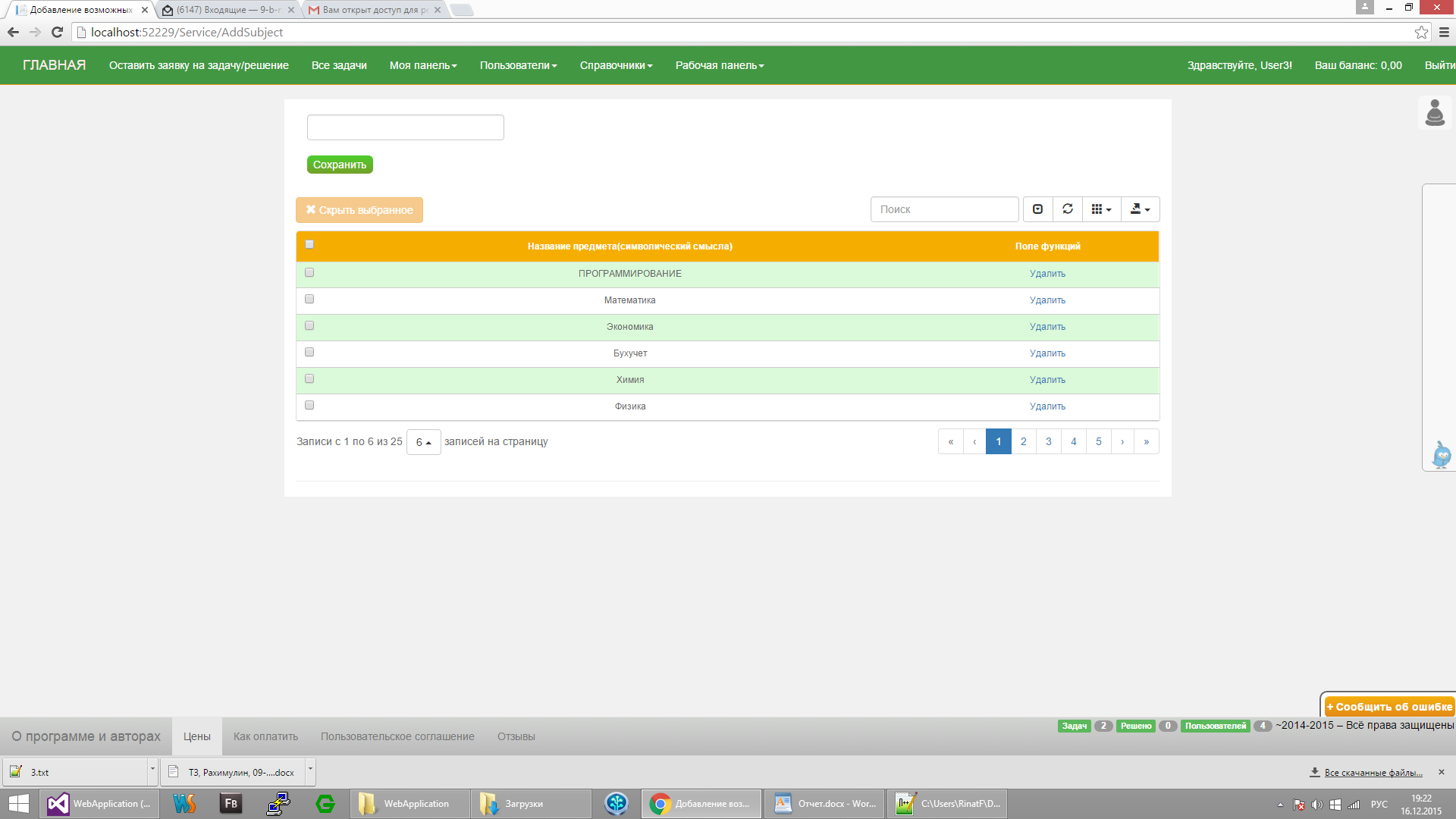
База сайта построена на справочниках, которые создаются в базе данных при самом первом запуске системы.

Прежде всего, это:

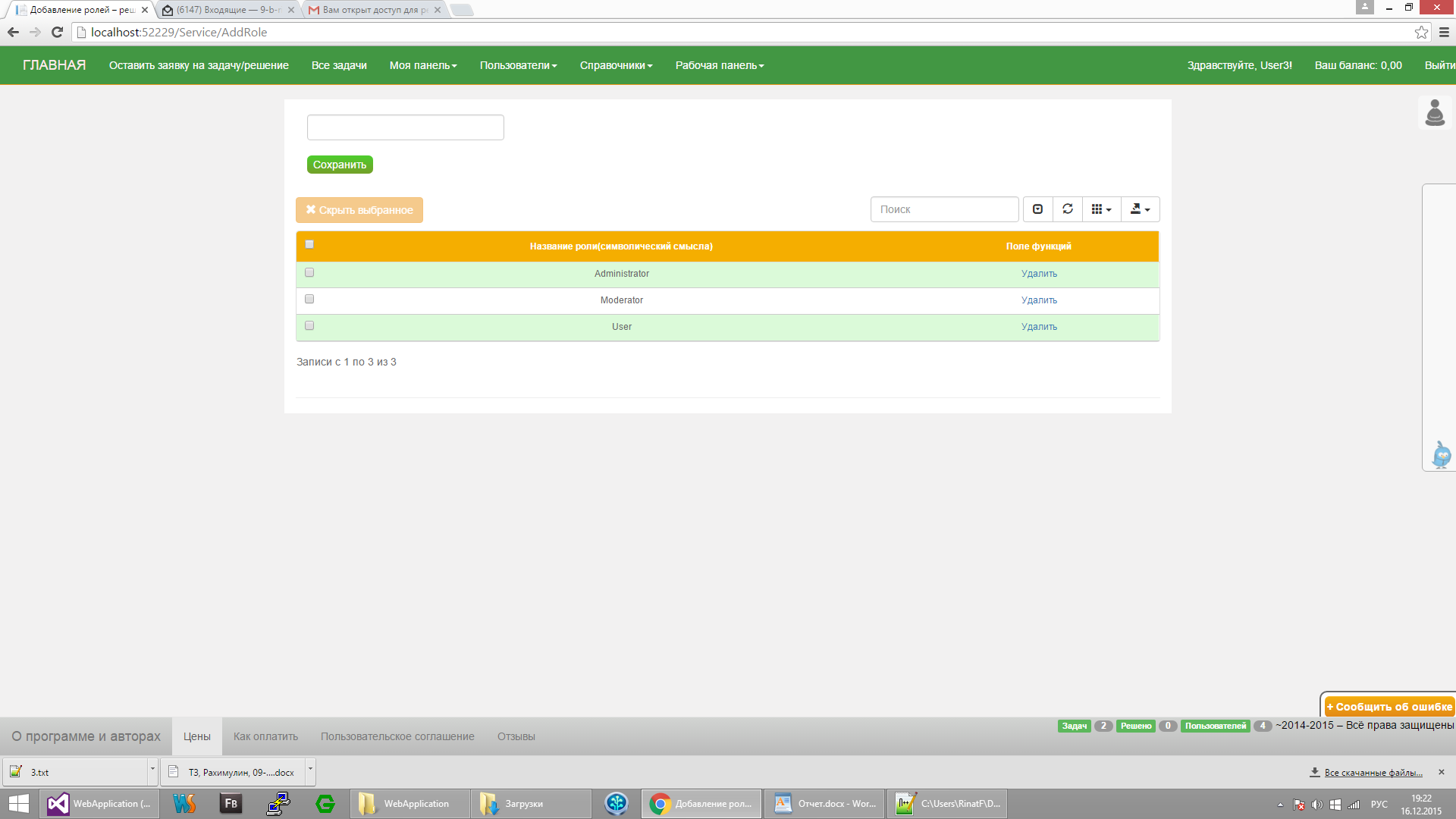
а) справочник типов задач



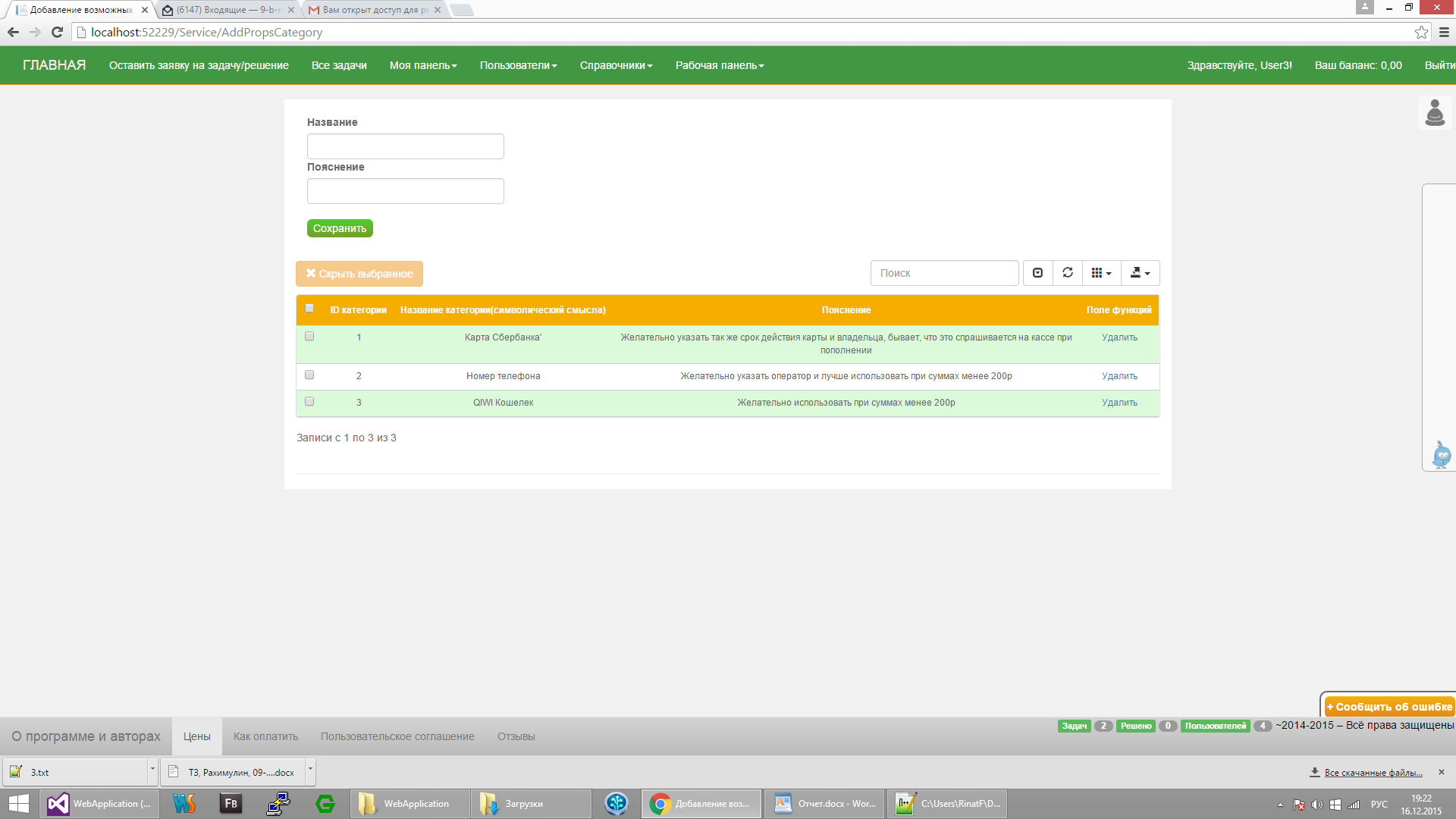
б) справочник направлений



в) справочник ролей

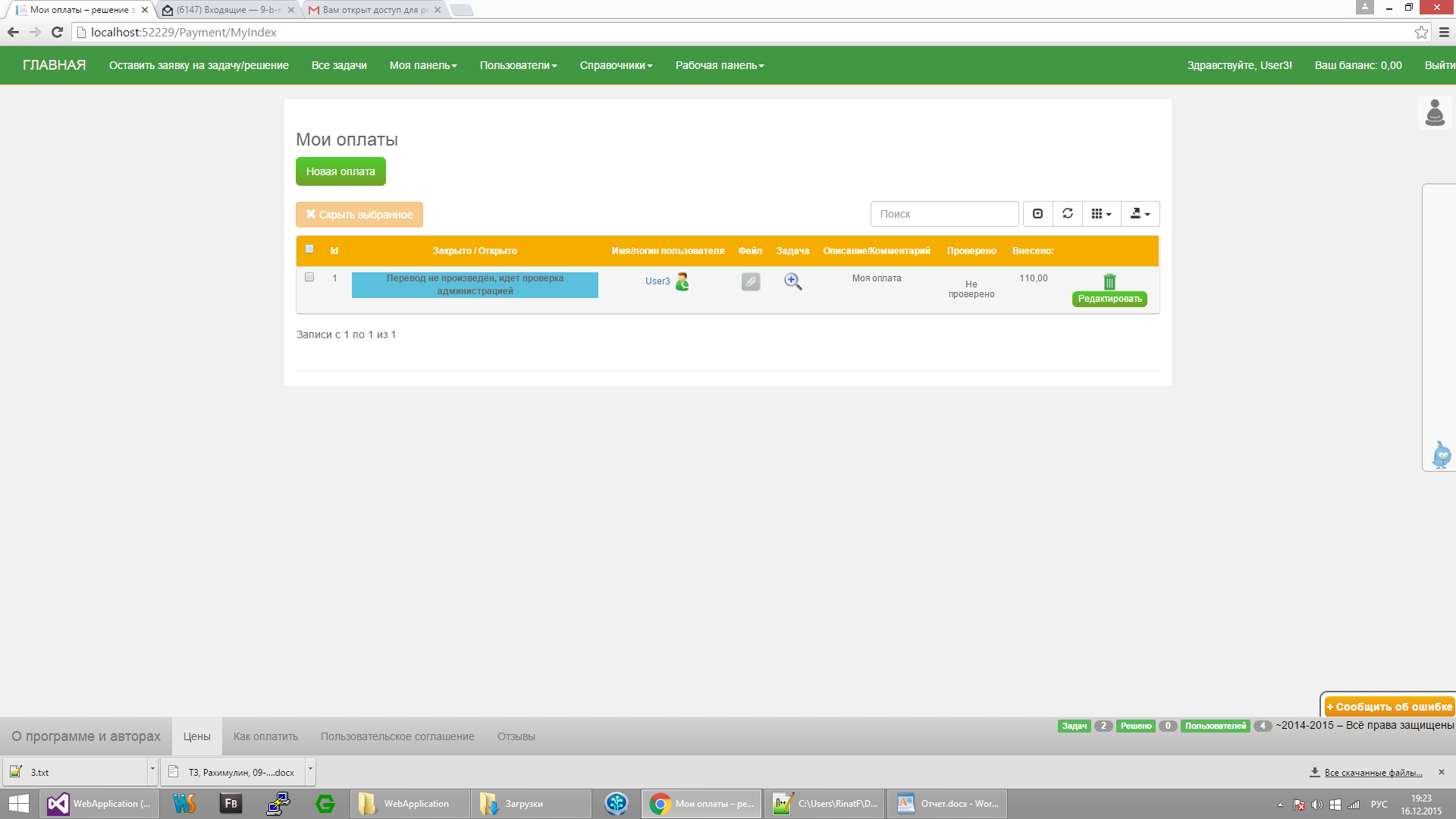


г) справочник реквизитов



**4.2 Регистрирование оплаты. Биллинг**

Когда же пользователь производит оплату, он видит, что проверка оплаты ещё не произведена:



Как же проверяет администрация оплату?

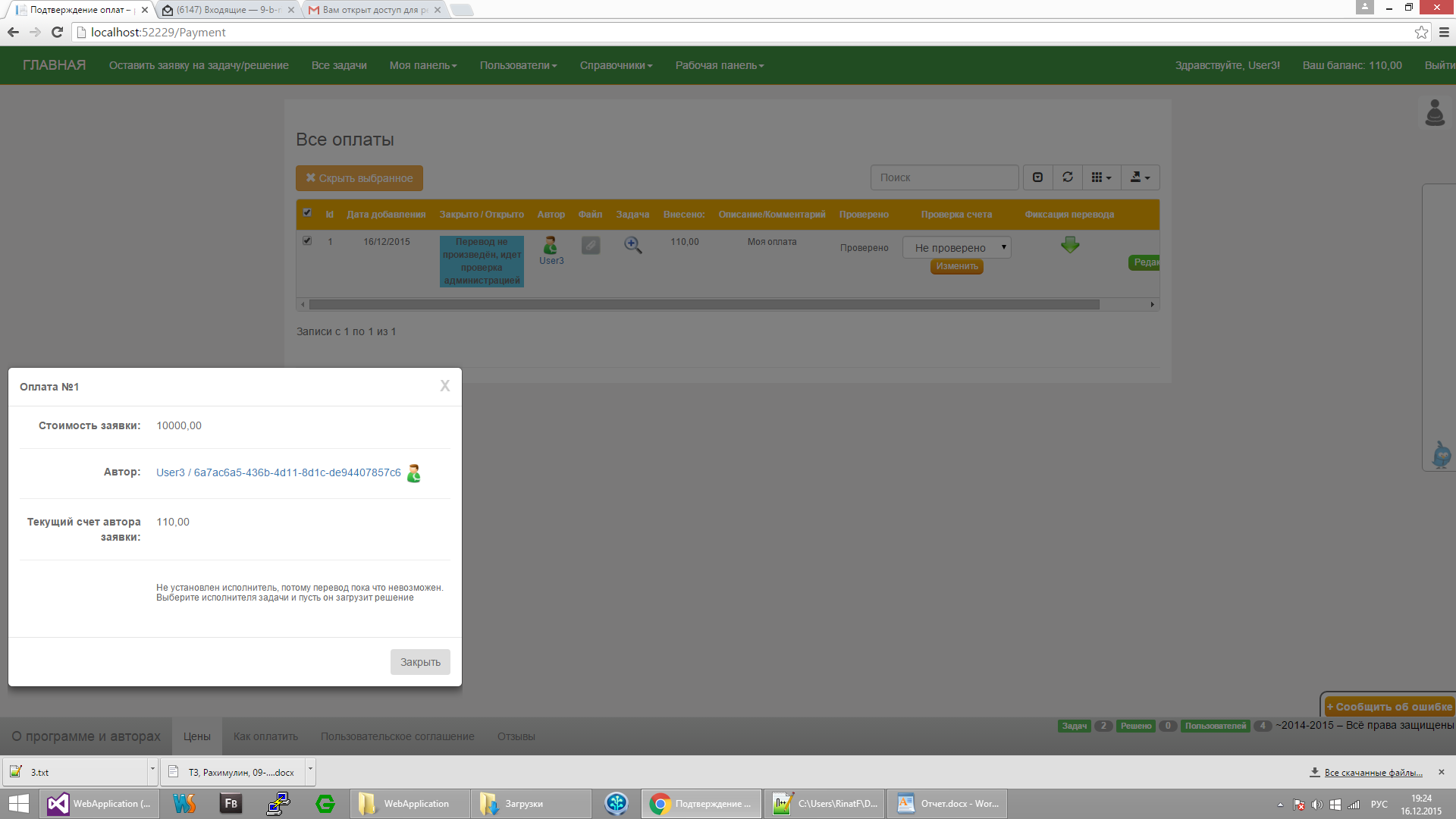
Проанализируем рабочую панель, которая содержит несколько управляющих разделов.

* + 1. **Подтверждение оплаты - переводов.**

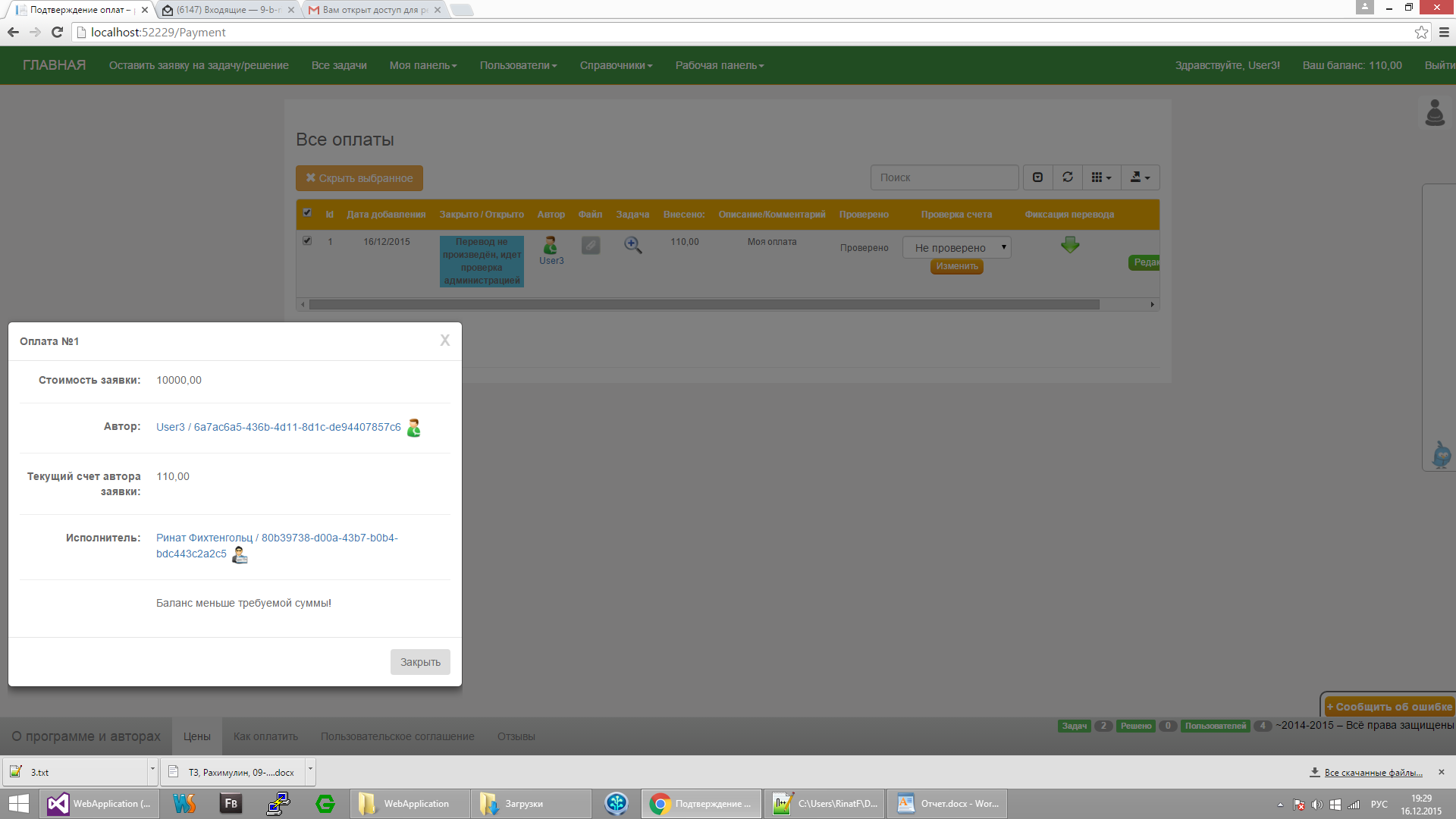
Делится на 2 стадии:

а) проверка администрацией.

б) подтверждение перевода(дабы уменьшить случайности)



Причем перевод невозможен, если денег на счету меньше, чем внесено. То есть – везде реализовано предупреждение таких мелких возможных ошибок, способных вылиться в финансовые потери.

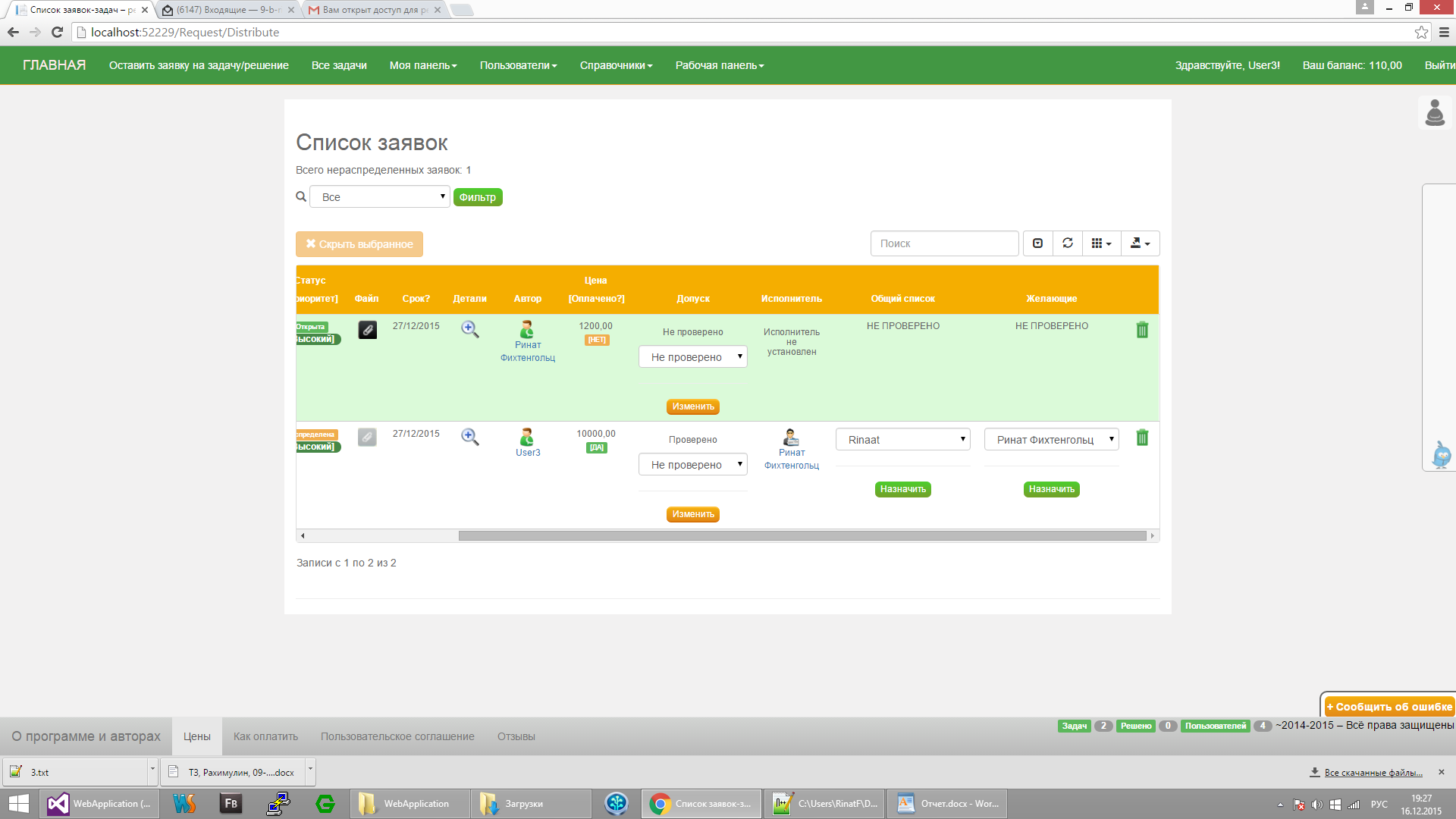


Если же исполнитель не установлен - перевод на счёт исполнителя не возможен. Как же установить исполнителя задачи?

Если этого до сих пор не сделал автор задачи, то это делает администратор через раздел "Распределить задачи". Это тоже 2 этапа безопасного выбора:

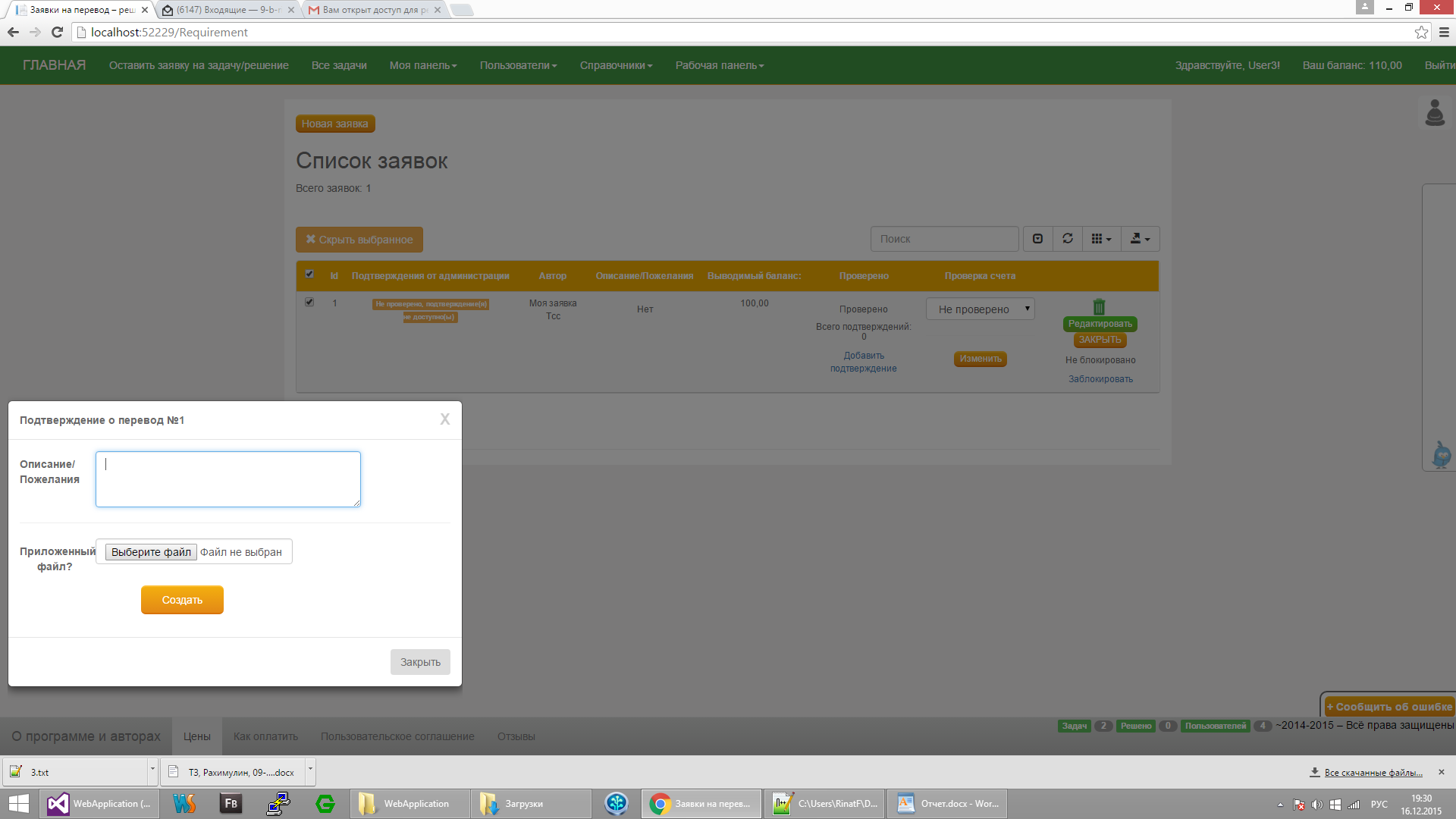
а) проверка задания администрацией.

б) появление возможности выбор из общего списка или из списка желающих того исполнителя, который может выполнить задание

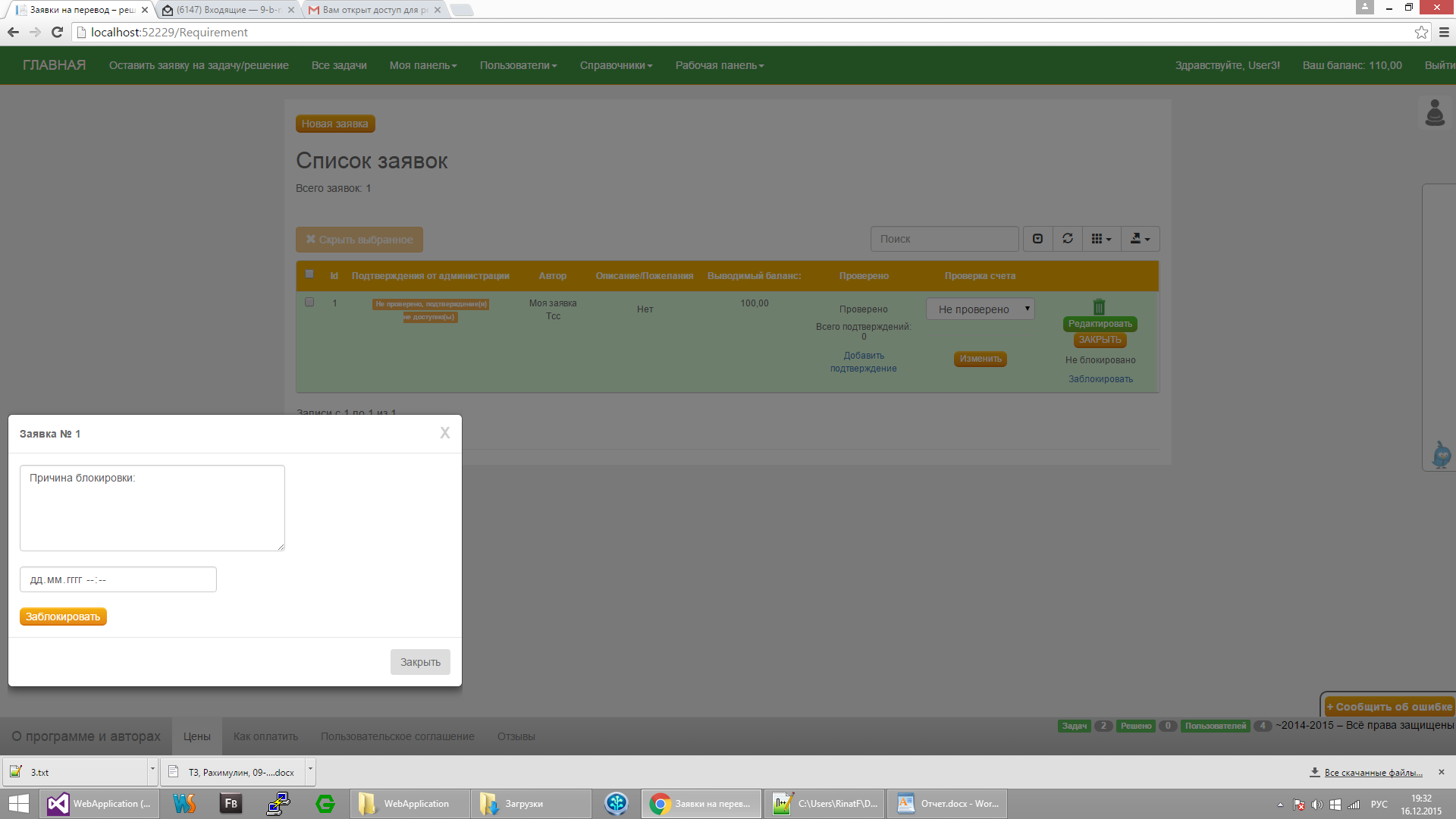


* + 1. **Раздел "Подтверждение заявки на вывод средств"**

Позволяет администрации в режиме реального времени проводить вывод средств на указанный счет заявителя.

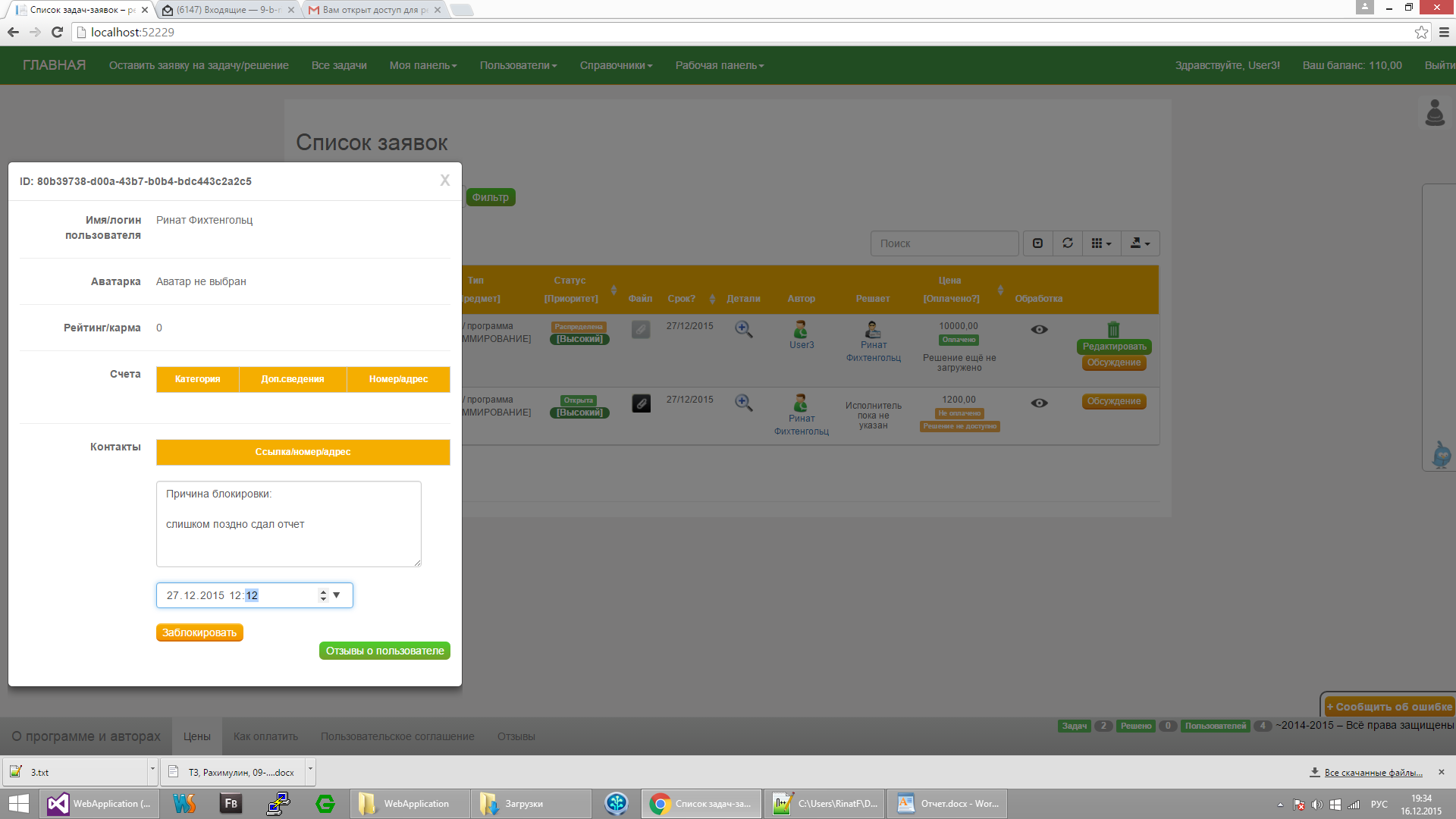


А так же позволяет прикрепить к подтверждению документ(фото чека-скрин, выписку и тп.). Если же вывести нельзя - то заявка блокируется до определенного срока. Мало ли, нужно проверить честность намерений на вывод крупной суммы.

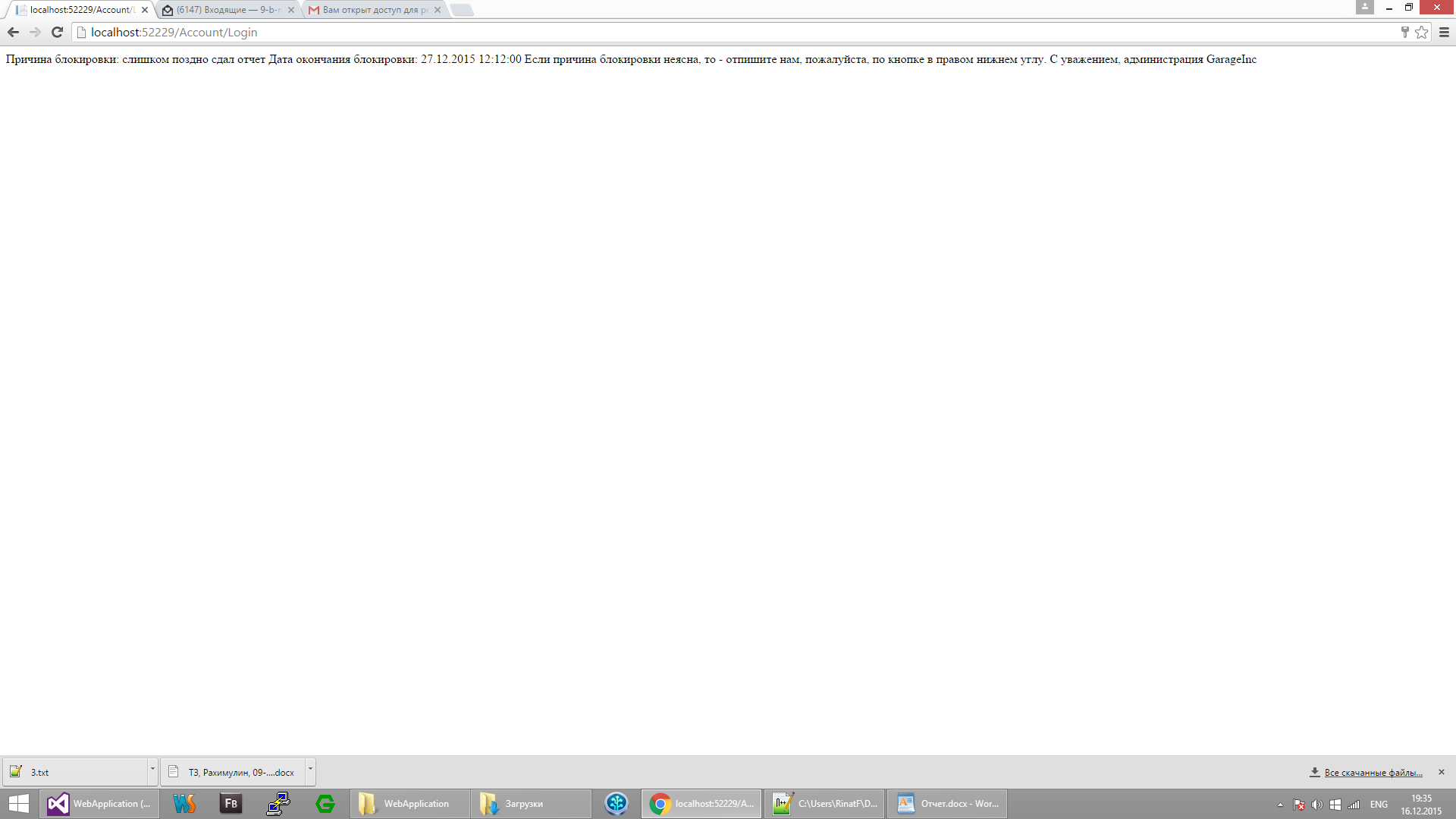


* 1. **Блокировка.**

Администрация также может заблокировать любого пользователя за несоблюдение правил сайта на неопределенный промежуток времени:



И заблокированный пользователь - потеряет доступ к сайту. Например, он получит в ответ на любое действие такое окно



**5. База данных**

Содержит следующие таблицы:

- Category

- Comment

- Contact

- Document

- ErrorMessages

- IdentityModels

- Lifecycle

- Payment

- Props

- PropsCategory

- RecallMessages

- Request

- RequestSolution

- Requirement

- RequirementConfirmation

- Subject

Пример миграции см. в приложении 1

**Выводы**

Система дет возможность для пользователей легко и быстро загружать свои задачи. И тем самым позволяет ожидать от системы быстрого их решения путём поиска фрилансеров. Она так же позволяет проводить "безопасные сделки",т.е. сделки, которые безопасны для обеих сторон: заказчик не боится, что его оставят с невыполненным заданием и без денег, а исполнитель не переживает насчет оплаты его труда(нет безвозмездного решения. Система дет возможность свободно общаться и исполнителям и решателям. Многих очень часто нервируется отсутствие явно цены и сравнения с другими предложениями - в этой системе данный аспект переработан в сторону полной открытости.

Планируется, что система фриланс-биржи станет несколько более модульной, пройдет редизайн, станет более близкой к социальной сети и будет интегрирована с диалогами сети vk.com. После тестовых проверок в течение нескольких месяцев планируется вывести её в широкое использование.

**Приложение 1**

CreateTable(

"dbo.Categories",

c => new

{

Id = c.Int(nullable: false, identity: true),

Name = c.String(nullable: false, maxLength: 50),

CreateDateTime = c.DateTime(nullable: false),

IsDeleted = c.Boolean(nullable: false),

DateOfDeleting = c.DateTime(),

})

.PrimaryKey(t => t.Id);

CreateTable(

"dbo.Comments",

c => new

{

Id = c.Int(nullable: false, identity: true),

AuthorId = c.String(maxLength: 128),

Karma = c.Int(nullable: false),

Text = c.String(),

ParentId = c.Int(),

RequestId = c.Int(),

CreateDateTime = c.DateTime(nullable: false),

IsDeleted = c.Boolean(nullable: false),

DateOfDeleting = c.DateTime(),

ApplicationUser\_Id = c.String(maxLength: 128),

ApplicationUser\_Id1 = c.String(maxLength: 128),

ApplicationUser\_Id2 = c.String(maxLength: 128),

})

.PrimaryKey(t => t.Id)

.ForeignKey("dbo.AspNetUsers", t => t.ApplicationUser\_Id)

.ForeignKey("dbo.AspNetUsers", t => t.ApplicationUser\_Id1)

.ForeignKey("dbo.Requests", t => t.RequestId)

.ForeignKey("dbo.AspNetUsers", t => t.ApplicationUser\_Id2)

.ForeignKey("dbo.AspNetUsers", t => t.AuthorId)

.Index(t => t.AuthorId)

.Index(t => t.RequestId)

.Index(t => t.ApplicationUser\_Id)

.Index(t => t.ApplicationUser\_Id1)

.Index(t => t.ApplicationUser\_Id2);

CreateTable(

"dbo.AspNetUsers",

c => new

{

Id = c.String(nullable: false, maxLength: 128),

Name = c.String(nullable: false),

Password = c.String(nullable: false, maxLength: 100),

Karma = c.Int(nullable: false),

LastVisition = c.DateTime(nullable: false),

RegistrationDate = c.DateTime(nullable: false),

UserInfo = c.String(),

IsBlocked = c.Boolean(nullable: false),

BlockForDate = c.DateTime(nullable: false),

BlockReason = c.String(),

DateOfBlocking = c.DateTime(nullable: false),

Balance = c.Decimal(nullable: false, precision: 18, scale: 2),

Email = c.String(maxLength: 256),

EmailConfirmed = c.Boolean(nullable: false),

PasswordHash = c.String(),

SecurityStamp = c.String(),

PhoneNumber = c.String(),

PhoneNumberConfirmed = c.Boolean(nullable: false),

TwoFactorEnabled = c.Boolean(nullable: false),

LockoutEndDateUtc = c.DateTime(),

LockoutEnabled = c.Boolean(nullable: false),

AccessFailedCount = c.Int(nullable: false),

UserName = c.String(nullable: false, maxLength: 256),

Request\_Id = c.Int(),

})

.PrimaryKey(t => t.Id)

.ForeignKey("dbo.Requests", t => t.Request\_Id)

.Index(t => t.UserName, unique: true, name: "UserNameIndex")

.Index(t => t.Request\_Id);

CreateTable(

"dbo.Documents",

c => new

{

Id = c.Int(nullable: false, identity: true),

Url = c.String(),

Size = c.Int(nullable: false),

Type = c.String(),

CreateDateTime = c.DateTime(nullable: false),

IsDeleted = c.Boolean(nullable: false),

DateOfDeleting = c.DateTime(),

ApplicationUser\_Id = c.String(maxLength: 128),

})

.PrimaryKey(t => t.Id)

.ForeignKey("dbo.AspNetUsers", t => t.ApplicationUser\_Id)

.Index(t => t.ApplicationUser\_Id);

CreateTable(

"dbo.AspNetUserClaims",

c => new

{

Id = c.Int(nullable: false, identity: true),

UserId = c.String(nullable: false, maxLength: 128),

ClaimType = c.String(),

ClaimValue = c.String(),

})

.PrimaryKey(t => t.Id)

.ForeignKey("dbo.AspNetUsers", t => t.UserId, cascadeDelete: true)

.Index(t => t.UserId);

CreateTable(

"dbo.Contacts",

c => new

{

Id = c.Int(nullable: false, identity: true),

ContactAdress = c.String(nullable: false, maxLength: 200),

AuthorId = c.String(maxLength: 128),

CreateDateTime = c.DateTime(nullable: false),

IsDeleted = c.Boolean(nullable: false),

DateOfDeleting = c.DateTime(),

})

.PrimaryKey(t => t.Id)

.ForeignKey("dbo.AspNetUsers", t => t.AuthorId)

.Index(t => t.AuthorId);

CreateTable(

"dbo.RecallMessages",

c => new

{

Id = c.Int(nullable: false, identity: true),

Text = c.String(nullable: false),

AuthorId = c.String(maxLength: 128),

Karma = c.Int(nullable: false),

ParentId = c.Int(),

AboutSite = c.Boolean(nullable: false),

UserId = c.String(maxLength: 128),

CreateDateTime = c.DateTime(nullable: false),

IsDeleted = c.Boolean(nullable: false),

DateOfDeleting = c.DateTime(),

ApplicationUser\_Id = c.String(maxLength: 128),

ApplicationUser\_Id1 = c.String(maxLength: 128),

ApplicationUser\_Id2 = c.String(maxLength: 128),

})

.PrimaryKey(t => t.Id)

.ForeignKey("dbo.AspNetUsers", t => t.AuthorId)

.ForeignKey("dbo.AspNetUsers", t => t.UserId)

.ForeignKey("dbo.AspNetUsers", t => t.ApplicationUser\_Id)

.ForeignKey("dbo.AspNetUsers", t => t.ApplicationUser\_Id1)

.ForeignKey("dbo.AspNetUsers", t => t.ApplicationUser\_Id2)

.Index(t => t.AuthorId)

.Index(t => t.UserId)

.Index(t => t.ApplicationUser\_Id)

.Index(t => t.ApplicationUser\_Id1)

.Index(t => t.ApplicationUser\_Id2);

CreateTable(

"dbo.ErrorMessages",

c => new

{

Id = c.Int(nullable: false, identity: true),

Text = c.String(nullable: false),

AuthorId = c.String(maxLength: 128),

ErrorStatus = c.Int(nullable: false),

ForAdministration = c.Boolean(nullable: false),

Email = c.String(),

CreateDateTime = c.DateTime(nullable: false),

IsDeleted = c.Boolean(nullable: false),

DateOfDeleting = c.DateTime(),

Document\_Id = c.Int(),

})

.PrimaryKey(t => t.Id)

.ForeignKey("dbo.Documents", t => t.Document\_Id)

.ForeignKey("dbo.AspNetUsers", t => t.AuthorId)

.Index(t => t.AuthorId)

.Index(t => t.Document\_Id);

CreateTable(

"dbo.AspNetUserLogins",

c => new

{

LoginProvider = c.String(nullable: false, maxLength: 128),

ProviderKey = c.String(nullable: false, maxLength: 128),

UserId = c.String(nullable: false, maxLength: 128),

})

.PrimaryKey(t => new { t.LoginProvider, t.ProviderKey, t.UserId })

.ForeignKey("dbo.AspNetUsers", t => t.UserId, cascadeDelete: true)

.Index(t => t.UserId);

CreateTable(

"dbo.Payments",

c => new

{

Id = c.Int(nullable: false, identity: true),

RequestSolutionId = c.Int(),

RequestId = c.Int(),

Description = c.String(nullable: false),

DocumentId = c.Int(),

Checked = c.Boolean(nullable: false),

Closed = c.Boolean(nullable: false),

AddingFunds = c.Boolean(nullable: false),

AuthorId = c.String(maxLength: 128),

Price = c.Decimal(nullable: false, precision: 18, scale: 2),

CreateDateTime = c.DateTime(nullable: false),

IsDeleted = c.Boolean(nullable: false),

DateOfDeleting = c.DateTime(),

})

.PrimaryKey(t => t.Id)

.ForeignKey("dbo.Documents", t => t.DocumentId)

.ForeignKey("dbo.Requests", t => t.RequestId)

.ForeignKey("dbo.RequestSolutions", t => t.RequestSolutionId)

.ForeignKey("dbo.AspNetUsers", t => t.AuthorId)

.Index(t => t.RequestSolutionId)

.Index(t => t.RequestId)

.Index(t => t.DocumentId)

.Index(t => t.AuthorId);

CreateTable(

"dbo.Requests",

c => new

{

Id = c.Int(nullable: false, identity: true),

Name = c.String(nullable: false, maxLength: 50),

Description = c.String(nullable: false, maxLength: 200),

DocumentId = c.Int(),

AuthorId = c.String(maxLength: 128),

ExecutorId = c.String(maxLength: 128),

Deadline = c.DateTime(nullable: false),

Status = c.Int(nullable: false),

Priority = c.Int(nullable: false),

CategoryId = c.Int(),

SubjectId = c.Int(),

LifecycleId = c.Int(),

Price = c.Decimal(nullable: false, precision: 18, scale: 2),

IsPaid = c.Boolean(nullable: false),

Checked = c.Boolean(nullable: false),

CanDownload = c.Boolean(nullable: false),

IsOnline = c.Boolean(nullable: false),

CreateDateTime = c.DateTime(nullable: false),

IsDeleted = c.Boolean(nullable: false),

DateOfDeleting = c.DateTime(),

ApplicationUser\_Id = c.String(maxLength: 128),

})

.PrimaryKey(t => t.Id)

.ForeignKey("dbo.AspNetUsers", t => t.AuthorId)

.ForeignKey("dbo.Categories", t => t.CategoryId)

.ForeignKey("dbo.Documents", t => t.DocumentId)

.ForeignKey("dbo.AspNetUsers", t => t.ExecutorId)

.ForeignKey("dbo.Lifecycles", t => t.LifecycleId)

.ForeignKey("dbo.Subjects", t => t.SubjectId)

.ForeignKey("dbo.AspNetUsers", t => t.ApplicationUser\_Id)

.Index(t => t.DocumentId)

.Index(t => t.AuthorId)

.Index(t => t.ExecutorId)

.Index(t => t.CategoryId)

.Index(t => t.SubjectId)

.Index(t => t.LifecycleId)

.Index(t => t.ApplicationUser\_Id);

CreateTable(

"dbo.Lifecycles",

c => new

{

Id = c.Int(nullable: false, identity: true),

Opened = c.DateTime(nullable: false),

Distributed = c.DateTime(),

Proccesing = c.DateTime(),

Checking = c.DateTime(),

Closed = c.DateTime(),

CreateDateTime = c.DateTime(nullable: false),

IsDeleted = c.Boolean(nullable: false),

DateOfDeleting = c.DateTime(),

})

.PrimaryKey(t => t.Id);

CreateTable(

"dbo.RequestSolutions",

c => new

{

Id = c.Int(nullable: false, identity: true),

Name = c.String(maxLength: 50),

RequestId = c.Int(),

DocumentId = c.Int(),

AuthorId = c.String(maxLength: 128),

Description = c.String(),

CreateDateTime = c.DateTime(nullable: false),

IsDeleted = c.Boolean(nullable: false),

DateOfDeleting = c.DateTime(),

})

.PrimaryKey(t => t.Id)

.ForeignKey("dbo.Documents", t => t.DocumentId)

.ForeignKey("dbo.Requests", t => t.RequestId)

.ForeignKey("dbo.AspNetUsers", t => t.AuthorId)

.Index(t => t.RequestId)

.Index(t => t.DocumentId)

.Index(t => t.AuthorId);

CreateTable(

"dbo.Subjects",

c => new

{

Id = c.Int(nullable: false, identity: true),

Name = c.String(nullable: false, maxLength: 50),

CreateDateTime = c.DateTime(nullable: false),

IsDeleted = c.Boolean(nullable: false),

DateOfDeleting = c.DateTime(),

})

.PrimaryKey(t => t.Id);

CreateTable(

"dbo.Props",

c => new

{

Id = c.Int(nullable: false, identity: true),

PropsCategoryId = c.Int(),

Number = c.String(nullable: false, maxLength: 200),

AuthorId = c.String(maxLength: 128),

CreateDateTime = c.DateTime(nullable: false),

IsDeleted = c.Boolean(nullable: false),

DateOfDeleting = c.DateTime(),

})

.PrimaryKey(t => t.Id)

.ForeignKey("dbo.PropsCategories", t => t.PropsCategoryId)

.ForeignKey("dbo.AspNetUsers", t => t.AuthorId)

.Index(t => t.PropsCategoryId)

.Index(t => t.AuthorId);

CreateTable(

"dbo.PropsCategories",

c => new

{

Id = c.Int(nullable: false, identity: true),

Name = c.String(nullable: false),

Info = c.String(nullable: false, maxLength: 200),

CreateDateTime = c.DateTime(nullable: false),

IsDeleted = c.Boolean(nullable: false),

DateOfDeleting = c.DateTime(),

})

.PrimaryKey(t => t.Id);

CreateTable(

"dbo.RequirementConfirmations",

c => new

{

Id = c.Int(nullable: false, identity: true),

Description = c.String(nullable: false, maxLength: 200),

DocumentId = c.Int(),

AuthorId = c.String(maxLength: 128),

RequirementId = c.Int(),

CreateDateTime = c.DateTime(nullable: false),

IsDeleted = c.Boolean(nullable: false),

DateOfDeleting = c.DateTime(),

})

.PrimaryKey(t => t.Id)

.ForeignKey("dbo.Documents", t => t.DocumentId)

.ForeignKey("dbo.Requirements", t => t.RequirementId)

.ForeignKey("dbo.AspNetUsers", t => t.AuthorId)

.Index(t => t.DocumentId)

.Index(t => t.AuthorId)

.Index(t => t.RequirementId);

CreateTable(

"dbo.Requirements",

c => new

{

Id = c.Int(nullable: false, identity: true),

Description = c.String(nullable: false, maxLength: 200),

AuthorId = c.String(maxLength: 128),

Checked = c.Boolean(nullable: false),

CanDownload = c.Boolean(nullable: false),

Closed = c.Boolean(nullable: false),

IsBlocked = c.Boolean(nullable: false),

BlockForDate = c.DateTime(nullable: false),

BlockReason = c.String(),

DateOfBlocking = c.DateTime(nullable: false),

Price = c.Decimal(nullable: false, precision: 18, scale: 2),

CreateDateTime = c.DateTime(nullable: false),

IsDeleted = c.Boolean(nullable: false),

DateOfDeleting = c.DateTime(),

})

.PrimaryKey(t => t.Id)

.ForeignKey("dbo.AspNetUsers", t => t.AuthorId)

.Index(t => t.AuthorId);

CreateTable(

"dbo.AspNetUserRoles",

c => new

{

UserId = c.String(nullable: false, maxLength: 128),

RoleId = c.String(nullable: false, maxLength: 128),

})

.PrimaryKey(t => new { t.UserId, t.RoleId })

.ForeignKey("dbo.AspNetUsers", t => t.UserId, cascadeDelete: true)

.ForeignKey("dbo.AspNetRoles", t => t.RoleId, cascadeDelete: true)

.Index(t => t.UserId)

.Index(t => t.RoleId);

CreateTable(

"dbo.AspNetRoles",

c => new

{

Id = c.String(nullable: false, maxLength: 128),

Name = c.String(nullable: false, maxLength: 256),

})

.PrimaryKey(t => t.Id)

.Index(t => t.Name, unique: true, name: "RoleNameIndex");