Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

К защите допустить:

Заведующая кафедрой ПОИТ

_____ Н. В. Лапицкая

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к дипломному проекту
на тему

ПРОГРАММНОЕ СРЕДСТВО ПРОВЕДЕНИЯ МНОГОПОЛЬЗОВАТЕЛЬСКИХ ВИКТОРИН С ИСПОЛЬЗОВАНИЕМ ПЛАТФОРМЫ .NET

БГУИР ДП 1-40 01 01 01 089 ПЗ

Студент Р. В. Мороз
Руководитель А. В. Хмелева
Консультанты:

от кафедры ПОИТ А. В. Хмелева
по экономической части А. А. Горюшкин

Нормоконтролёр С. В. Болтак

Рецензент

Минск 2022

РЕФЕРАТ

ПРОГРАММНОЕ СРЕДСТВО ПРОВЕДЕНИЯ МНОГОПОЛЬЗОВАТЕЛЬСКИХ ВИКТОРИН С ИСПОЛЬЗОВАНИЕМ ПЛАТФОРМЫ .NET: дипломный проект / Р. В. Мороз. – Минск, БГУИР, 2022, – п.з. – 100 с., чертежей (плакатов) – 6 л. формата А1.

Цель настоящего дипломного проекта состоит в разработке программной системы, предназначенной для эффективной автоматизации проведения многопользовательских викторин для участников игрового процесса: игроков и ведущего.

В процессе анализа предметной области были выделены основные аспекты процесса проведения викторин, которые в настоящее время практически не охвачены автоматизацией. Было проведено их исследование и моделирование. Кроме того, рассмотрены существующие средства, используемые желающими проверить свои знания в игровой форме (так называемые частичные аналоги). Выработаны функциональные и нефункциональные требования.

Была разработана архитектура программной системы, для каждой ее составной части было проведено разграничение реализуемых задач проектирование, уточнение используемых технологий и непосредственно разработка. Были выбраны наиболее современные средства разработки, широко применяемые в индустрии.

Полученные в ходе технико-экономического обоснования результаты о прибыли для разработчика, пользователя, уровень рентабельности, а также экономический эффект доказывают целесообразность разработки проекта.

Для проекта был проведен анализ оригинальности в системе «Антиплагиат». Процент оригинальности составляет 97.95%. Цитирования обозначены ссылками на публикации, указанные в «Списке использованных источников».

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ И СОКРАЩЕНИЯ

В настоящей пояснительной записке применяются следующие определения и сокращения.

Спецификация – документ, который желательно полно, точно и верифицируемо определяет требования, дизайн, поведение или другие характеристики компонента или системы, и, часто, инструкции для контроля выполнения этих требований.

Десктоп-приложение – полноценное приложение, работа которого не зависит от других приложений. Зачастую требует предварительной установки и (или) настройки.

Кроссплатформенность – способность программного обеспечения работать на нескольких аппартных платформах и (или) операционной системе.

Нативное программное средство – программное средство, специфичное для какой-либо конкретной платформы и (или) операционной системы.

Байт-код – стандартное промежуточное представление, в которое может быть переведена программа автоматическими средствами, для дальнейшей интерпретации другой программой.

Проприетарное программное обеспечение – программное обеспечение, являющееся частной собственностью авторов или правообладателей и не удовлетворяющее критериям свободного ПО: свобода использования программного обеспечения в любых целях, модификация и адаптация исходного кода программы под свои нужны, свобода дистрибуции, свобода улучшения исходного кода и публиция улучшений.

ПС – программное средство.

ПО – программное обеспечение.

БД – база данных.

XML – extensible markup language (расширяемый язык разметки).

XAML – extensible application markup language (расширяемый язык разметки приложений).

ЯП – язык программирования.

ООП – объектно-ориентированное программирование.

IL – intermidiate language (промежуточный язык).

HTML – hypertext markup language (язык разметки гипертекста).

CSS – cascade stylesheet (каскадная страница стилей).

P2P – peer-to-peer (от пользователя к пользователю)

LINQ – language-integrated query (запрос, интегрированный в язык програмирования).

JSON – javascript object notation (нотация объектов языка JavaScript).

CLR – common language runtime (общая среда выполнения).

API – application programming interface (программный интерфейс приложения).

UI – user interface (пользовательский интерфейс).

ТЭО – технико-экономическое обоснование.

ВВЕДЕНИЕ

Развлечения всегда были и будут частью жизни любого человека. Самой первой игрой настольной игрой считается Сенет – напоминающая современные шашки и появившаяся за четыре тысячи лет до нашей еры в Египте. По мере развития технологий каждой эре были свойственны свои знаковые игры и развлечения. Интересной особенностью являлось то, что в одну и ту же игру разные народы и культуры могли играть по разному, что порождало свои наборы правил, отличные от оригинала и способствовало дальнейшему развитию игр.

Человеку близок соревновательный аспект игр: в древние времена сильнейший был самым успешныи в обществе. Благодаря таким развлечениям человек может доказать себе и другим, что он достойный соперник и превосходит оппонентов без необходимости применения грубой силы. Множество настольных игр позволяют проверить такие качества, как: терпение, ум, стратегическое мышление, память, коммуникативность и так далее.

С развитием вычислительных машин появились и принципиально новые виды развлечений, позволяющие испытать себя в новых условиях без ущерба для здоровья. Например, шутеры, появившиеся в начале девяностых годов, которые проверяют скорость реакции и координации – навыки достаточно сложно проверяемые в настольных играх. В то время основным соперником человека был искусственный интеллект под управлением компьютера, что ограничивало сложность соревнования навыками программистов, писавших искусственный интеллект.

С началом повсеместного распространения интернета закономерным развитием индустрии развлечений стало появление игр, расчитанных на нескольких игроков. Интерес таких игр повышался многократно, по сравнению с играми, где был искусственный интеллект, так как человек по своей природе может быть достаточно непредсказуем, что повышает вариантивность игровых ситуаций, к которым игрок должен адаптироваться.

В настоящее время в мире существует множество жанров видеоигр, в которых участвует не два, не пять человек, а тридцать, шестьдесят или даже сотня человек одновременно. При этом общее количество игроков игре исчисляется сотнями тысяч. Большинство таких игр упирается на соревновательный аспект, например: сражение команда на команду или выявление лучшего среди всех. При этом порог входа в такие игры как правило не высок, но уровень знаний, необходимых для достижения лучших результов достаточно высокий, чтобы игроки оставались в игре на долгое время.

Целью настоящего дипломного проекта является разработка программного средства, которое позволяет участникам соревнования определить самого лучшего знатока предметной области в игровой форме на основе формата викторины. Причем в качестве вопросов могут выступать как текст и картинки, так и музыка с видео. В результате проведения викторины все игроки могут узнать свои и чужие результаты по количеству набранных баллов.

Программное средство должно помочь в решении следующих задач: проведение викторин с большим количеством игроков, посчет игровых очков для определения победителя, поддержка должного уровня разнообразия самого процесса, за счет вопросов разных видов и типов, помощь в проведении для игроков из разных точек земного шара, за счет использования интернет-соединения.

В пояснительной записке к дипломному проекту излагаются детали поэтапной разработки программного средства проведения многопользовательских викторин. В первом разделе приведены результаты анализа литературных источников по теме дипломного проекта, рассмотрены особенности существующих систем-аналого, выдвинуты требования к проектируемому программному средству. Во втором разделе приведено описание функциональности проектируемого программного средства, представлена спецификация функциональных требований. В третьем разделе приведены детали проектирования и конструирования программного средства. Результатом этапа конструирования является функционирующее программное средство. В четвертом разделе представлены доказательство того, что спроетированное программно средство работает в соответствии с выдвинутыми требованиями спецификации. В пятом разделе приведены сведения по развертыванию и запуску ПС, указаны требуемые аппаратные и программные средства. Обоснование целесообразности создания программного средства с технико-экономической точки зрения приведено в шестом разделе. Итоги проектирования, конструирования программного средста, а также соответствующие выводы приведены в заключении.

1 АНАЛИЗ ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ, ПРОТОТИПОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ

Конечный успех программного проекта во многом определяется до начала конструирования: на этапе подготовки, которая проводится с учетом всех особенностей проекта.

Первое предварительное условие, которое нужно выполнить перед конструированием, – ясное формулирование проблемы, которую система должна решать. Общая цель подготовки — снижение риска: адекватное планирование позволяет исключить главные аспекты риска на самых ранних стадиях работы, чтобы основную часть проекта можно было выполнить максимально эффективно.

Главный факторы риска в создании ПО — неудачная выработка требований. Требования подробно описывают, что должна делать программная система. Внимание к требованиям помогает свести к минимуму изменения системы после начала разработки [1].

Перед формулированием требований необходимо изучить ряд вопросов, которые напрямую влияют на все дальнейшие этапы разработки. В частности, необходимо рассмотреть вопросы выбора платформ, архитектуры. По результатам анализа можно будет составить техническое задание к проектируемому программному средству, которое станет основой для составления функциональных требований.

1.1 Аналитический обзор литературных источников

Далее приводится анализ сведений, которые влияют на формулирование требований, выбор архитектуры и дальнейшее проектирование и разработку программного средства.

1.1.1 Кроссплатформенность программного обеспечения

В настоящее время выбор платформы является серьезным фактором во время всех этапов разработки, от правильно выбранной платформы зависит набор возможных технологий, которые можно использовать при разработке программного продукта. Уже сейчас с каждым днем появляются все новые технологии, которые позволяют переиспользовать однажны написанный код на других платформах [2]. Примером таких технологий являются C# и Java. В их основе лежит так называемый IL – intermidiate language, своеобразный промежуточный язык, в который компилируется исходный

код программы [3]. В дальнейшем на целевую платформу устанавливается CLR - common language runtime или общая среда выполнения, которая читает код на промежуточном языке и представляет его в виде, понятном для целевой платформы [3]. Таким образом кроссплатформенность - это один из факторов, который необходимо учитывать при выборе техногии для реализации проекта.

1.1.2 Обзор целевых платформ

Несмотря на планируемое использование кроссплатформенных технологий, поддержка всех платформ может затребовать значительно больших средств и времени, чем есть в наличии для выполнения дипломного проектирования. Поэтому, необходимо осуществить выбор одной основной платформы с расчетом на продолжение разработки и реализацию проекта для других платформ. Рассмотрим достоинства и недостатки основных.

Настольное приложение – программное средство, которое запускается локально на компьютере пользователя. При его создании появляется возможность использования всех преимуществ аппаратного обеспечения, которым оснащен компьютер, например: прямой доступ к видеокарте, к устройствам периферии. Кроме того, появляется возможность взаимодействия с другими установленными приложениями, а также в подавляющем числе случаев возможность автономной работы. Тем не менее у настольных приложений есть и ряд недостатков. Когда пользователь работает удаленно, возникают проблемы, связанные с сетью, соединениями, сетевыми экранами. Один из самых больших недостатков заключается в огромной сложности развертывания приложения на десятки или сотни машин, их конфигурирования, периодического обновления [4].

Веб-приложения, в отличие от настольных, работают на удаленном аппаратном обеспечении и поставляются пользователю через браузер [5]. При их использовании разработчик избавляется от необходимости поддерживать установку большого числа зависимостей, он всегда может быть уверен, что все пользователи используют самую последнюю версию приложения. Вычислительные операции могут производиться на мощном сервере, а результаты вычислений поставляться пользователю – так называемая концепция «тонкого клиента» [6].

Несмотря на то, что ресурсоемкие вычисления производятся на сервере, задержки при передаче, особенно при нестабильном соединении, сама потребность в постоянном интернет соединении могут значительно снизить удобство пользования приложением. Кроме того, размер загружаемого при

каждом запуске кода и ресурсов может значительно увеличить траты пользователя, особенно если он использует дорогое мобильное подключение к интернету.

Для *мобильных приложений* актуальны ограничения платформ, на которых они запускаются, такие как меньшие размеры экранов, более медленные процессоры, ограниченное энергопотребление. Несмотря на это, мобильные устройства часто находятся рядом с пользователями, появляется возможность использования мгновенных оповещений [7]. Вместе с этим, большинство смартфонов оснащено модулями, такими как GPS, камера, NFC, что предоставляет разработчику новые возможности по их использованию. На основании рассмотренных характеристик различных платформ можно осуществить выбор одной из них, которая и станет целевой для разработки.

1.1.3 Обзор архитектурных стилей

Далее необходимо рассмотреть применяющиеся на практике архитектурные стили, провести их анализ и по результатам осуществить выбор архитектуры, которая затем будет применяться при проектировании программного средства.

Под разработкой *архитектуры* понимают специфицирование структуры всей системы: глобальную организацию и структуру управления, протоколы коммуникации, синхронизации и доступа к данным, распределение функциональности между компонентами системы, физическое размещение, состав системы, масштабируемость и производительность [8]. Набор принципов, используемых в архитектуре, формирует *архитектурный стиль*. Применение архитектурных стилей упрощают решение целого класса абстрактных проблем [8].

При проектировании архитектуры программной системы почти никогда не ограничиваются единственным архитектурным стилем, поскольку они могут предлагать решение каких-либо проблем в различных областях. В таблице 1.1 приведен вариант категоризации архитектурных стилей [9].

Сервис-ориентированная архитектура позволяет приложениям предоставлять некоторую функциональность с помощью набора слабосвязанных автономных сервисов; связь между сервисами обеспечивается с помощью заранее определенных контрактов. Данный стиль предоставляет следующие преимущества [9]:

– повторное использование сервисов снижает стоимость разработки;
– автономность и использование формальных контрактов способст-

Таблица 1.1 – Категоризация архитектурных стилей

Категория	Архитектурный стиль
Связь	SOA (Service-oriented architecture – архитектура, ориентированная на сервисы), Шина сообщений
Развертывание	Клиент-серверный, трехуровневый, N-уровневый
Предметная область	DDD (Domain-driven design – проблемно-ориентированное проектирование)
Структура	Компонентный, объектно-ориентированный, многоуровневый

вует слабой связанности и повышает уровень абстракции;

– сервисы могут использовать возможность автоматического обнаружения и определения интерфейса;

– сервисы и использующие их приложения могут быть развернуты на различных платформах.

Архитектура *шины сообщений* описывает принципы построения систем, которые используют обмен сообщениями как способ связи. Наиболее часто при реализации данной архитектуры используется модель маршрутизатора сообще ний или шаблон издатель-подписчик. Главные преимущества использования данного архитектурного стиля [9]:

– расширяемость, которая заключается в возможности добавлять и удалять приложения без влияния на другие;

– снижается сложность приложений, так как единственный интерфейс, который они должны поддерживать – интерфейс общей шины;

– гибкость, которая заключается в возможности подстраиваться под бизнес-требования или желания пользователей через изменения конфигурации или параметров маршрутизации сообщений;

– слабая связанность, поскольку единственное, чем связаны приложения – интерфейс общей шины;

– масштабируемость, которая заключается в возможности в случае необходимости присоединения к шине нескольких экземпляров одного и того же приложения.

Клиент-серверная архитектура описывает распределенную систему, которая включает независимые системы сервера, клиента и соединяющую их сеть. Иногда данную архитектуру называют двухзвенной. Из преимуществ выделяют следующие [8]:

– безопасность: все данные хранятся на сервере, обеспечивающем больший уровень безопасности, чем отдельные клиенты;

– централизованный доступ к данным, который предоставляет возможность более легкого управления, чем в других архитектурах;

– устойчивость и легкость поддержки: роль сервера могут выполнять несколько физических компьютеров, объединённых в сеть; благодаря этому клиент не замечает сбоев или замены отдельного серверного компьютера.

Многоуровневый архитектурный стиль заключается в группировании схожей функциональности приложения по уровням, которые выстроены в вертикальную структуру. Уровни связаны слабо, связь между ними осуществляется по явно установленным протоколам. Строгий вариант архитектуры предполагает, что компоненты какого-либо уровня могут взаимодействовать только с компонентами одного нижележащего уровня; ослабленный вариант разрешает взаимодействие с компонентами любого из нижележащих уровней. Использование данного архитектурного стиля предлагает следующие преимущества [9]:

– есть возможность осуществлять изменения на уровне абстракций;

– изолированность: изменения на каких-либо уровнях не влияют на другие, что снижает риск и минимизирует воздействие на всю систему;

– разделение функциональности помогает управлять зависимостями, что приводит к большей управляемости всего кода;

– независимые уровни предоставляют возможность повторного использования компонентов;

– строго-определенные интерфейсы способствуют повышению тестируемости компонентов.

Многозвенная архитектура предлагает схожее с многоуровневой архитектурой разбиение функциональности, отличие же заключается в предлагаемом размещении звеньев на физически обособленной машине [9]. Предлагается использовать данный стиль в случаях, когда компоненты одного звена могут проводить дорогие в ресурсном отношении вычисления, так что это может сказаться на других уровнях, или когда некоторую чувствительную информацию переносят со звеньев уровня представления на уровень бизнес-логики приложения. Далее приведены главные преимущества использования многозвенной архитектуры:

– удобство сопровождения, которое заключается в возможности внесения изменений и обновлений в некоторые компоненты при минимальном влиянии на всё приложение;

– масштабируемость, которая возникает из распределенного развер-

тывания;

– гибкость, которая появляется благодаря предыдущим двум пунктам;

– масштабируемость также приводит к повышению доступности приложения.

Проблемно-ориентированный архитектурный стиль основывается на предметной области, ее элементах, их поведении и связях между ними. Для применениях данного стиля необходимо иметь хорошее понимание предметной области или людей, которые бы смогли объяснить ее специфику разработчикам. Первое, что нужно сделать при принятии данного стиля – выработать единый язык, который бы знала вся команда разработчиков, который бы был избавлен от технических жаргонизмов и содержал только термины предметной области – только так можно избежать проблемы непонимания между участникам [10]. Преимущества, которые может предложить данный стиль:

– упрощение коммуникации между участниками процесса разработки благодаря выработке единого языка;

– модель предметной области обычно является расширяемой и гибкой при изменениях условий и бизнес-требований;

– хорошая тестируемость.

Компонентная архитектура основывается на декомпозиции системы в отдельные функциональные или логические компоненты, которые раскрывают другим компонентам только заранее определенные интерфейсы [9]. Преимущества данного подхода:

– легкость развертывания, которая заключается в обновлении компонентов без влияния на другие;

– использование компонентов сторонних разработчиков позволяет снизить расходы на разработку и поддержку;

– поддержка компонентами заранее определенных интерфейсов позволяет упростить разработку;

– возможность переиспользования компонентов также оказывает влияние на снижение стоимости.

Объектно-ориентированный архитектурный стиль выражается в разделении функциональности системы на множество автономных объектов, каждый из которых содержит некоторые данные и набор методов поведения, свойственных объекту. Обычной практикой является определение классов, соответствующих объектам предметной области. Применение данного стиля предоставляет следующие преимущества [9]:

– соотнесение классов программы и объектов реального мира делает программное средство более понятным;

– полиморфизм и принцип абстракции предоставляет возможность повторного их использования;

– инкапсуляция повышает тестируемость объектов;

– улучшение расширяемости, которое возникает благодаря тому, что изменения в представлении данных не оказывают влияния на внешний интерфейс объекта, что не ограничивает его способность взаимодействовать с другими объектами;

– высокая сцепленность объектов, которая достигается использованием разных объектов для разного набора действий.

Таким образом, применение рассмотренных архитектур на этапе проектирования окажет большое влияние на успешность всего процесса разработки; какие бы стили не были применены, можно быть уверенным в правильности выбора за счет того, что у всех из них есть определенные и зачастую разные преимущества.

1.2 Обзор существующих аналогов

Для решения задач управления процессом проведения викторин и их создания могут использоваться различные средства. Рассмотрим каждую из задач в отдельности.

1.2.1 Управление процессом создания викторин

Для организаторов проведения викторин важнейшим аспектом является необмходимость в предварительно составленных викторинах по желаемой тематике. Человеку известно тысячи различным областей знаний по самым разным предметам изучения и необходимо иметь возможность создать викторину по любой тематике, которая только может придти в голову, причем для повышения уровня вовлеченности в игровой процесс следует использовать нечто большее, чем просто текст.

Одним из безусловных лидеров в области проведения викторин является программное средство SIGame, разработанное Владимиром Хилем. Для проведения викторин их необходимо предварительно создать в программном средстве, поставляемым отдельно под названием SIQuester. Данное программное средство позволяет создавать пакеты вопросов для дальнейшего использования в SIGame. Интерфейс приложения выглядит как изображено на рисунке 1.1.

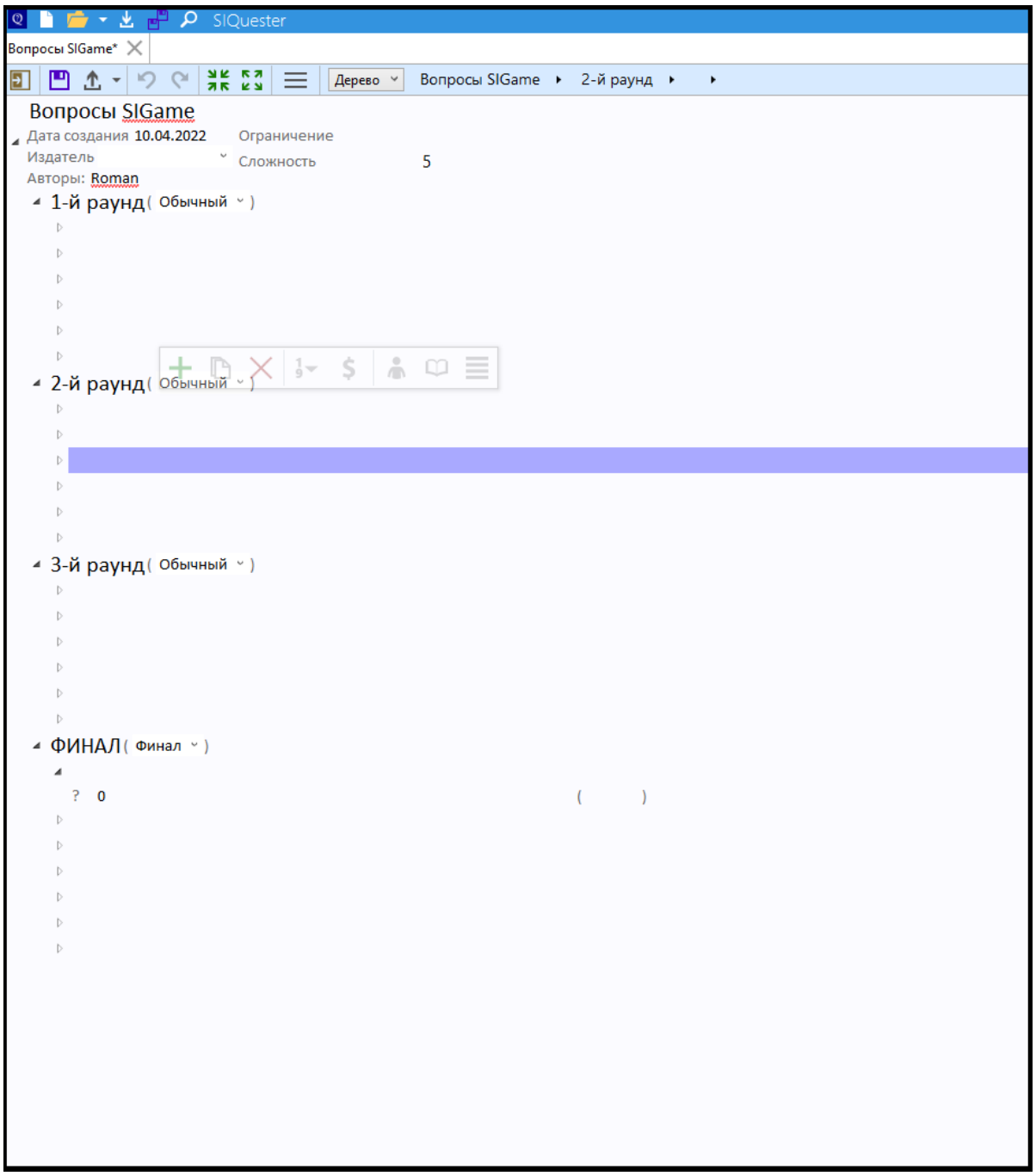

Рисунок 1.1 – Интерфейс приложения SIQuester

SIQuester поддерживает создание различных видов вопросов, как по содержанию, так и по формату разыгровки, среди поддерживаемых вопросов по содержанию есть:

– текст;
– изображение;
– фрагмент видео;
– звуковой фрагмент.

По формату разыгровки вопросов поддерживаются следующие вари-

анты:

Простой вопрос – отвечающий получает баллы в случае правильного ответа и теряет их в случае неверного ответа, это наболее частовстречающийся вопрос.

Вопрос со ставкой – перед тем как игроки узнают содержимое вопроса происходит аукцион за право отвечать на вопрос. Тот, кто поставит больше очков и получить право отвечать на вопрос при этом награда за правильный ответ или штраф за неправильный будет является финальным размером ставки на аукционе.

Вопрос с секретом – тема вопроса, а также его стоимость может не совпадать с той, которая указана в сетке вопросов. В этом и заключается секретность вопроса. Этот вопрос также можно передать любому из участников.

Вопрос без риска – в случае неправильного ответа участник, отвечающий на вопрос не теряет баллы, равные стоимости вопроса.

Примеры вопросов разных типов приведены на рисунке 1.2.

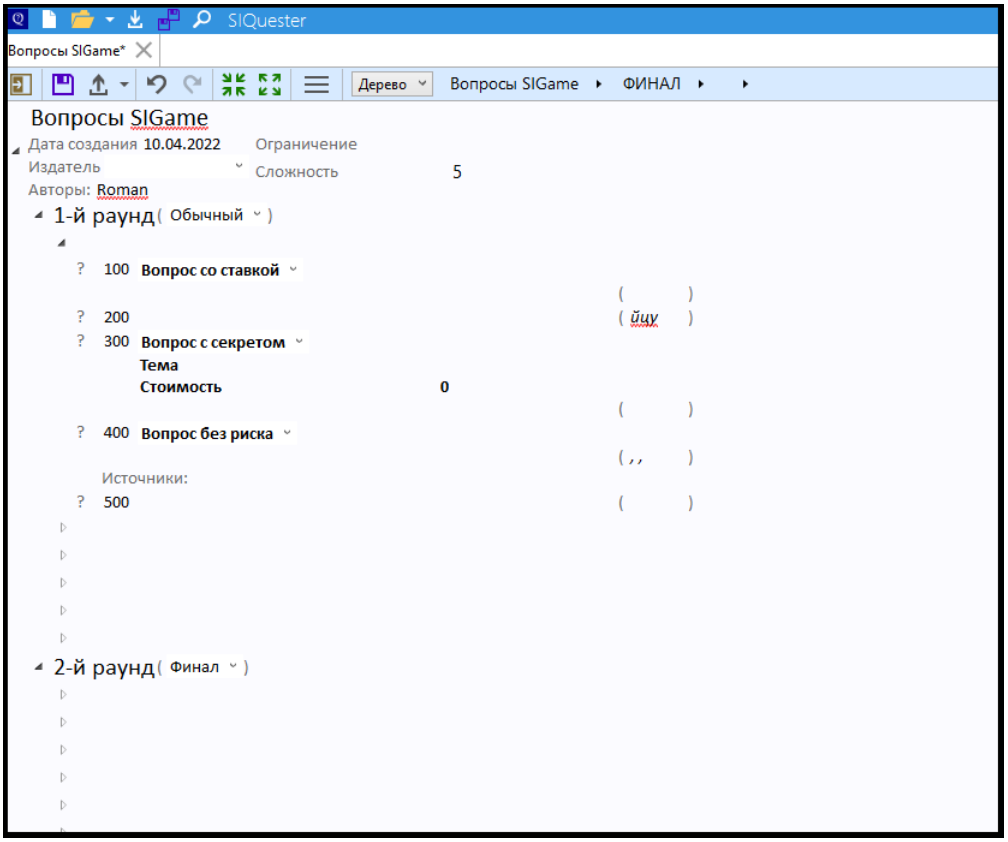

Рисунок 1.2 – Примеры вопросов в SIQuester

Анализ существующих аналогов по созданию вопросов показывает возможность наличия различных интерактивных видов вопросов для повы-

шения заинтересованности игроков в игровом процессе и повышения уровня эмоционального удовлетворения в результате игры, но в то же время такой подход усложняет игровой процесс для совсем неопытных игроков, поэтому в данном курсовом проекте будет использоваться единственный тип вопроса – простой.

1.2.2 Управление процессом идентификации игроков

Для идентификации в сети в программном средстве используется анонимный подход: не нужны никакие средства внешней идентификации пользователя, такие как электронная почта, номер телефона и так далее. Экран создания локального аккаунта для игры выглядит следующим образом (рисунок 1.3)

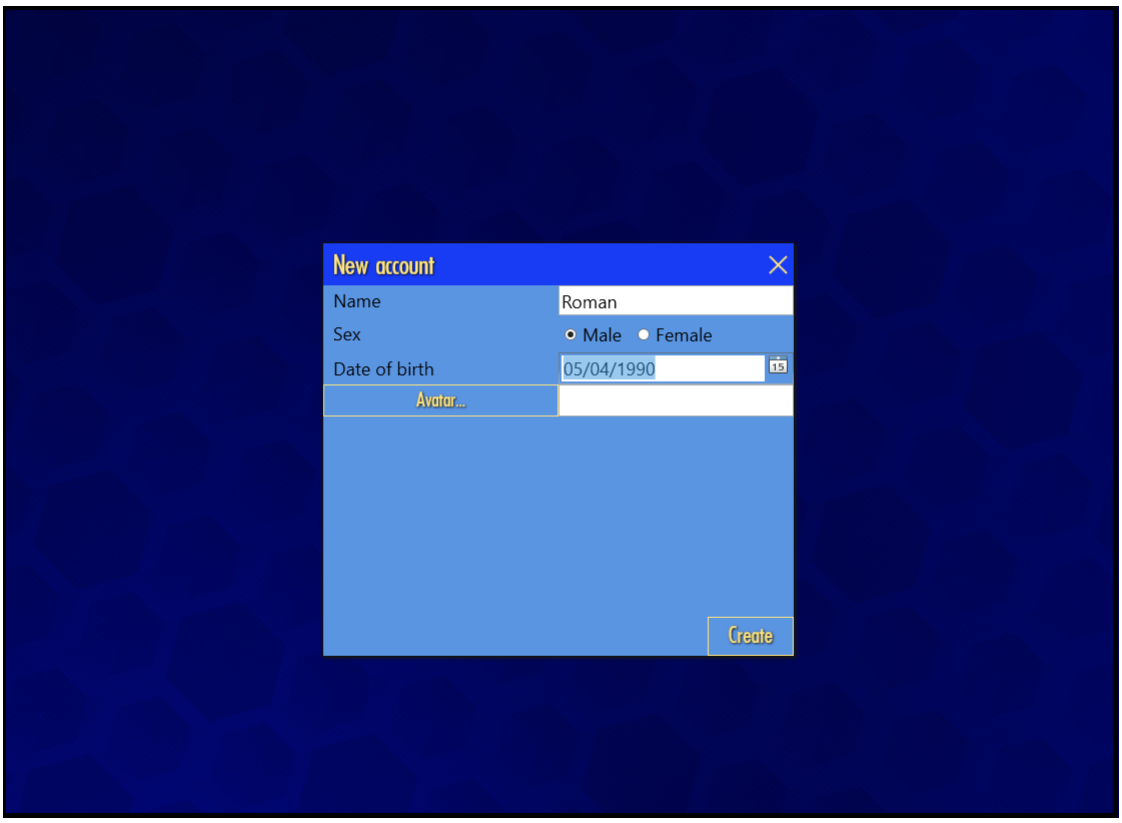

Рисунок 1.3 – Создание аккаунта в SIGame

Из этого можно сделать вывод, что аккаунты носят минимально функциональный характер и служит средством идентификации для людей, которые и так уже знают игроков в реальной жизни, поэтому злоумышленник не сможет украсть какие-либо конфиденциальные данные, так как приложение их нигде не хранит и не использует, а для проведения викторин они не требуются.

Анализ показывает, что аккаунты в такого рода программных средствах используются лишь из необходимости минимальной идентификации в процессе игры и не служат для сбора информации, поэтому смысла реализовывать какого либо рода защиту нет.

1.2.3 Управление процессом создания игры

Перед тем как игроки смогут подключится к игре, необходимо, чтобы ведущий создал игровое лобби, пример реализации процесса создания лобби в SIGame изображен на рисунке 1.4.

Рисунок 1.4 – Создание игрового лобби в SIGame

Настройка лобби отвечает за несколько важных особенностей проведения викторины.

Название лобби позволяет игрокам, желающим присоединится к состязанию, найти игру в списке всех доступных игр, а зная название сделать это намного проще.

Пароль позволяет хосту защитить лобби от ограничить доступ нежелательным игрокам и оставить лишь тем, кто знает пароль.

Настройка вопросов позволяет выбрать заранее созданный в программе тестере набор вопросов, на которые предстоит отвечать игрокам.

Размер игрового лобби позволяет устанавливать лимит на максимальное количество активных игроков в сессии, что позволит избежать слишком большого количества игроков в случае создания игры без пароля.

Анализ показывает, что создание лобби это обязательная часть программного средства и от ее реализации зависит игровой опыт, который получат игроки в процессе проведения викторины. Минимальный набор настроек, необходимых для настройки лобби были перечислены выше и требуются для дальнейшей реализации игрового процесса.

1.2.4 Управление процессом подключения к игре

Игроки используют процесс подключения к игре, для того, чтобы попасть в одно игровое лобби с организатором викторины. Процесс подключения к игровому лобби выглядит следующим образом (рисунок 1.5).

Рисунок 1.5 – Подключение к игровому лобби в SIGame

Перед пользователем представлен список доступных игр для подключения, которые можно идентифицировать по названию игрового лобби и его организатору. В случае, если лобби имеет пароль, то при подключении появляется модальное окно, где его необходимо ввести. После подключения пользователь попадает в лобби и может приступать к игровому процессу.

Анализ показывает, что процесс подключения к игре является одним из важнейших во всем программном средстве. Без возможности подключится к игровому лобби смысл программного средства теряется.

1.2.5 Управление игровым процессом

Игровой процесс заключается в поочередном ответе на вопросы и зачислении очков, победителем считается тот, кто набрал больше всего очков к моменту, когда закончатся все вопросы. Фрагмент игрового процесса представлен на рисунке 1.6.

Рисунок 1.6 – Процесс игры в SIGame

Следует отметить, что хотя сам игровой процесс является довольно простым, при этом он также является достаточно увлекательным, чтобы на долго время задерживать игроков и оставаться увлеченным. Игроки отвечают на вопросы, в то время как ведущий контролирует правильность ответов и решает начислять очки, или нет. Здесь следует сделать оговорку, что предполагается, что игра проходит в устной форме, т. е. игроки общаются между собой посредством внешнего программного средства для голосовой связи, например: Discord, Skype и так далее, поэтому реализация средств коммуникации непосредственно в приложении не является необходимой.

Анализ показывает, что наибольшее внимание стоит уделить релизации логики игрового процесса, так как от него напрямую зависит удовлетворенность игроков процессом проведения викторин.

1.2.6 Обзор иных программных средств для проведения викторин

Рассмотренные процессы типичны для любого приложения с ориентацией на взаимодействие по сети, в то время как проведение викторин имеет свои особенности в разных реализациях.

Веб-аналоги позволяют избежать установки на персональный компьютер, но в то же время увеличивают нагрузку на сеть, что может быть не приемлемо для некоторых категорий пользователей. К тому же в случае ограничения доступа к сайту или отказа в обслуживании больше воспользоваться программным средством не получится.

Реализации для мобильных телефонов имеют более скромный функционал по сравнению с версией для настольного компьютера, это обусловлено тем, что небольшой экран и нестабильное интернет-соединение не могут обеспечить достаточный уровень комфорта при игре. К тому же общение с игроками затруднено тем, что использование средств связи на мобильном телефоне параллельно с проведением викторины не является удобным, поэтому популярность такие реализации не снискали.

После анализа типичных реализаций был сделан вывод, что наилучшим решением для разработки программного средства проведения многопользовательских викторин является настольное приложение.

1.3 Требования к проектируемому программному средству

По результатам изучения предметной области, анализа литературных источников и обзора существующих систем-аналогов сформулируем требования к проектируемому программному средству.

1.3.1 Назначение проекта

Назначением проекта является разработка программного средства, позволяющего автоматизировать процессы проведения викторин на произвольные темы.

1.3.2 Основные функции

Программное средство должно поддерживать следующие основные функции:

– создание и редактирование пакетов с вопросами, которые будут использоваться во время проведения викторин;

– поддержка нескольких форматов вопросов (текст, звук, видео и т. д.);

– поддержка нескольких видов вопросов (обычный, с секретом, со ставкой и т. д.);

– анонимная авторизация с помощью локально созданного аккаунта;

– создание игрового лобби, как организатор викторины;

– настройка игрового лобби для проведения викторины (выбор пакета вопросов, пароль и т. д.);

– присоединение к игровому лобби в качестве игрока;

– управление игровым процессом для ведущего;

– автоматическое определение победителя после ответов на все вопросы.

1.3.3 Требования к входным данным

Входные данные для программного средства должны представлять из себя вводимый с клавиатуры текст или выбор доступных опций на пользовательском интерфейсе. Должны быть реализованы проверки вводимых данных на корректность с отображением информации об ошибка в случае их некорректности.

1.3.4 Требования к выходным данным

Выходные данные программного средства должны быть представлены посредством отображения информации с помощью различных элементов пользовательского интерфейса.

1.3.5 Требования к временным характеристикам

Производительность программно-аппаратного комплекса должна обеспечивать следующие временные характеристики: время реакции на запрос пользователя не должно превышать одной секунды при минимальной скорости соединения 1 МБит/с. Допускается невыполнение данного требования в случае, когда невозможно обеспечить заявленную пропускную способность интернет-канала по внешним причинам и не зависящим от пользователя обстоятельствам.

1.3.6 Требования к надежности

Надежное функционирование программы должно быть обеспечено выполнением следующих организационно-технических мероприятий:

– организация бесперебойного питания;

– выполнение требований ГОСТ 31078-2002 «Защита информации. Испытания программных средств на наличие компьютерных вирусов»;

– выполнение рекомендаций Министерства труда и социальной защиты РБ, изложенных в Постановлении от 23 марта 2011 г. «Об утверждении норм времени на работы по обслуживанию персональных электронно-вычислительных машин, организационной техники и офисного оборудования»;

– необходимым уровнем квалификации пользователей.

Время восстановления после отказа, вызванного сбоем электропитания технических средств (иными внешними факторами), нефатальным сбоем операционной системы, не должно превышать времени, необходимого на перезагрузку операционной системы и запуск программы, при условии соблюдения условий эксплуатации технических и программных средств. Время восстановления после отказа, вызванного неисправностью технических средств, фатальным сбоем операционной системы, не должно превышать времени, требуемого на устранение неисправностей технических средств и переустановки программных средств.

Отказы программы возможны вследствие некорректных действий пользователя при взаимодействии с операционной системой. Во избежание возникновения отказов программы по указанной выше причине следует обеспечить работу конечного пользователя без предоставления ему административных привилегий.

1.3.7 Требования к аппаратному обеспечению серверной части

ЭВМ, на которой должна функционировать серверная часть программного средства, должна обладать следующими минимальными характеристиками:

– процессор Intel Core i7 с тактовой частотой 4 ГГц;

– жесткий диск объемом 100 Гб;

– оперативная память 16 Гб;

– сетевая карта Ethernet 100 МБит/с.

Также для функционирования серверной части требуется Docker, который является кроссплатформенным программные средством, вследствие чего вопрос о целевой операционной системе не рассматривается. Кроме того, процедуры установки и настройки данного программного средства выходят за рамки данного проекта и также не рассматриваются.

1.3.8 Требования к аппаратному обеспечению клиентской части

Клиентская часть программного средства должна функционировать на ЭВМ со следующими минимальными характеристиками:

– процессор Intel Core i7 с тактовой частотой 4 ГГц;
– оперативная память 4 Гб;
– сетевая карта Ethernet 10/100 МБит/с.

Для корректной работы программного средства необходимы следующие программные компоненты:

– операционная система Windows 10;
– .NET Framework 4.8;
– visual c++ redistributable.

1.3.9 Выбор технологий программирования

Язык программирования, на котором будет реализована система, имеет большое значение, так как он будет использоваться с начала конструирования программы и до самого конца. Исследования показали, что выбор языка программирования несколькими способами влияет на производительность труда программистов и качество создаваемого ими кода. Если язык хорошо знаком программистам, они работают более производительно. Данные, полученные при помощи модели оценки Cocomo II, показывают, что программисты, использующие язык, с которым они работали три года или более, примерно на 30% более продуктивны, чем программисты, обладающие аналогичным опытом, но для которых язык является новым [11]. В более раннем исследовании, проведенном в IBM, было обнаружено, что программисты, обладающие богатым опытом использования языка программирования, были более чем втрое производительнее программистов, имеющих минимальный опыт [12].

Язык программирования C#, указанный в задании на дипломное проектирование, является кроссплатформенным языком программирования и является частью платформы .NET. Данный ЯП представляет собой самостоятельный язык, который имеет C-образный синтаксис и семантику управляющих конструкций. Так как язык является строго типизированным, то основные ошибки можно отловить еще на этапе компиляции проекта. Свою популярность он снискал за небольшое количество легаси-кода и большое количество синтаксического "сахара который позволяет компактно описывать сложные конструкции. Еще одной сильной стороной является технология LINQ, которая позволяет работать с любыми коллекциями, будь то база данных или локальный массив одинаково, причем в стандарте уже реализованы все основные операции работы с данными, такие как добавление, обновление, удаление, фильтрация и так далее. Язык компилируется в IL и в дальнейшем выполняется в CLR, что обеспечивает кроссплатформен-

ность и поддержку принципа write once – run anywhere (напиши один раз – запуская везде). На данном языке разработано множество библиотек для работы в самых разных областях, например: настольные приложения, веб-приложения, машинное обучение и так далее. Таким образом программист, знающий C#, может писать под что угодно, лишь за исключением систем, для которых объем памяти строго ограничен, к таким относятся встраиваемые системы.

Среди достоинств платформы .NET можно выделить следующие:

– обеспечение согласованной объектно-ориентированной среды программирования для локального сохранения и выполнения объектного кода, для локального выполнения кода, распределенного в Интернете, либо для удаленного выполнения;

– обеспечение среды выполнения кода, минимизирующей конфликты при развертывании программного обеспечения и управлении версиями;

– обеспечение среды выполнения кода, гарантирующей безопасное выполнение кода, включая код, созданный неизвестным или не полностью доверенным сторонним изготовителем;

– обеспечение среды выполнения кода, исключающей проблемы с производительностью сред выполнения сценариев или интерпретируемого кода;

– обеспечение единых принципов работы разработчиков для разных типов приложений, таких как приложения Windows и веб-приложения;

– разработка взаимодействия на основе промышленных стандартов, которое обеспечит интеграцию кода платформы .NET с любым другим кодом.

Исходя из достоинств данного языка программирования, можно сделать вывод, что он наиболее подходящий для решения проблем, схожих с поднимаемыми в данной пояснительной записке. Именно поэтому C# и выбран как основной язык программирования в задании к текущему дипломному проекту. Он является простым, современным, объектно-ориентированным, обеспечивающим безопасность типов языком программирования. Он соответствует международному стандарту Европейской ассоциации производителей компьютеров — стандарт ECMA-334, а также стандарту Международной организации по стандартизации (International Standards Organization, ISO) и Международной электротехнической комиссии — стандарт ISO/IEC 23270. Компилятор Microsoft C# для .NET согласуется с обоими этими стандартами [13].

Передовой средой программирования, которая поддерживает C# явля-

ется Microsoft Visual Studio, которая входит в линейку продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментальных средств. Она включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью рефакторинга кода. Встроенный отладчик может работать как отладчик уровня исходного кода, так и как отладчик машинного уровня. Visual Studio позволяет создавать и подключать сторонние дополнения для расширения функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий исходного кода, добавление новых наборов инструментов или инструментов для прочих аспектов процесса разработки программного обеспечения. Именно поэтому она и выбрана в качестве основной среды программирования.

Для разработки сервиса-координатора, который будет работать на сервере будет также использоваться C# и технология сокетов с написанием собственного протокола передачи данных. Данный протокол является необходимым по той причине, что клиентам и серверу необходимо понимать формат пакетов, приходящих друг от друга. Разрабатываемый сервер представляет из себя приложение-сервис, которое работает постоянно и отвечает на запросы клиентов, при этом оно хранит некое промежуточное состояние работы, которое может использовать при обработке запросов от клиентов. В случае разработки проектируемого программного средства веб-сервис будет хранить список активных игровых лобби, которые будут запрашивать клиенты при попытке подключится к организатору викторины, а также игровые состояния для каждого лобби.

Сформулированные требования позволят осуществить успешное проектирование и разработку программного средства.

2 АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ

2.1 Функциональная модель программного средства

Функциональная модель программного средства представлена в виде схемы алгоритма игрового процесса и диаграммы вариантов использования. Алгоритм игрового процесса показывает как будет идти игровой процесс в зависимости от действий игроков. Варианты использования отражают функциональность системы в ответ на внешние воздействия с точки зрения получения значимого результата для пользователей.

2.1.1 Анализ игрового процесса

Перед началом проектирования необходимо проанализировать игровой процесс и определить его цикл. Целесообразным будет проведение анализа как с точки зрения игрока, так и с точки зрения организатора. Результат анализа представлен в виде схем алгоритмов на рисунках 2.1 для игрока и 2.2 для организатора. Схема ограничивается лишь непосредственно игровым циклом и не включает в себя процесс авторизации и установки соединения с игровым лобби.

Одной из особенностей данных схем является цикличность, с которой проходит игровой процесс.

Цикл для игрока начинается с того, что игрок получает право на выбор вопроса из списка, после сделанного выбора все игроки видят содержимое вопроса и тот, кто первый сможет дать правильный ответ на вопрос и получит сумму баллов на свой игровой счет. В случае неправильного ответа на вопрос баллы вычитаются из игрового счета. Здесь стоит обратить внимание на то, что игрок, выбравший вопрос не получает никакого игрового преимущества по сравнению с другими игроками. Поэтому в игре все равны и факт того, что кто-то получил право выбирать вопрос первым не имеет большого значения.

Цикл для организатора викторины начинается с того, что игрок выбирает вопрос, далее организатор ждет, пока кто-нибудь из игроков решится ответить на вопрос. Если желания отвечать на вопрос ни у кого нет, то вопрос пропускается. В случает, когда игрок изъявляет желание дать ответ, то организатор ждет ответа и проверяет, является ли он верным. Если игрок дал правильный ответ, то ему на игровой баланс поступают очки, равные стоимости вопроса. В случае неправильного ответа игрок теряет очки со своего баланса, равные стоимости вопроса.

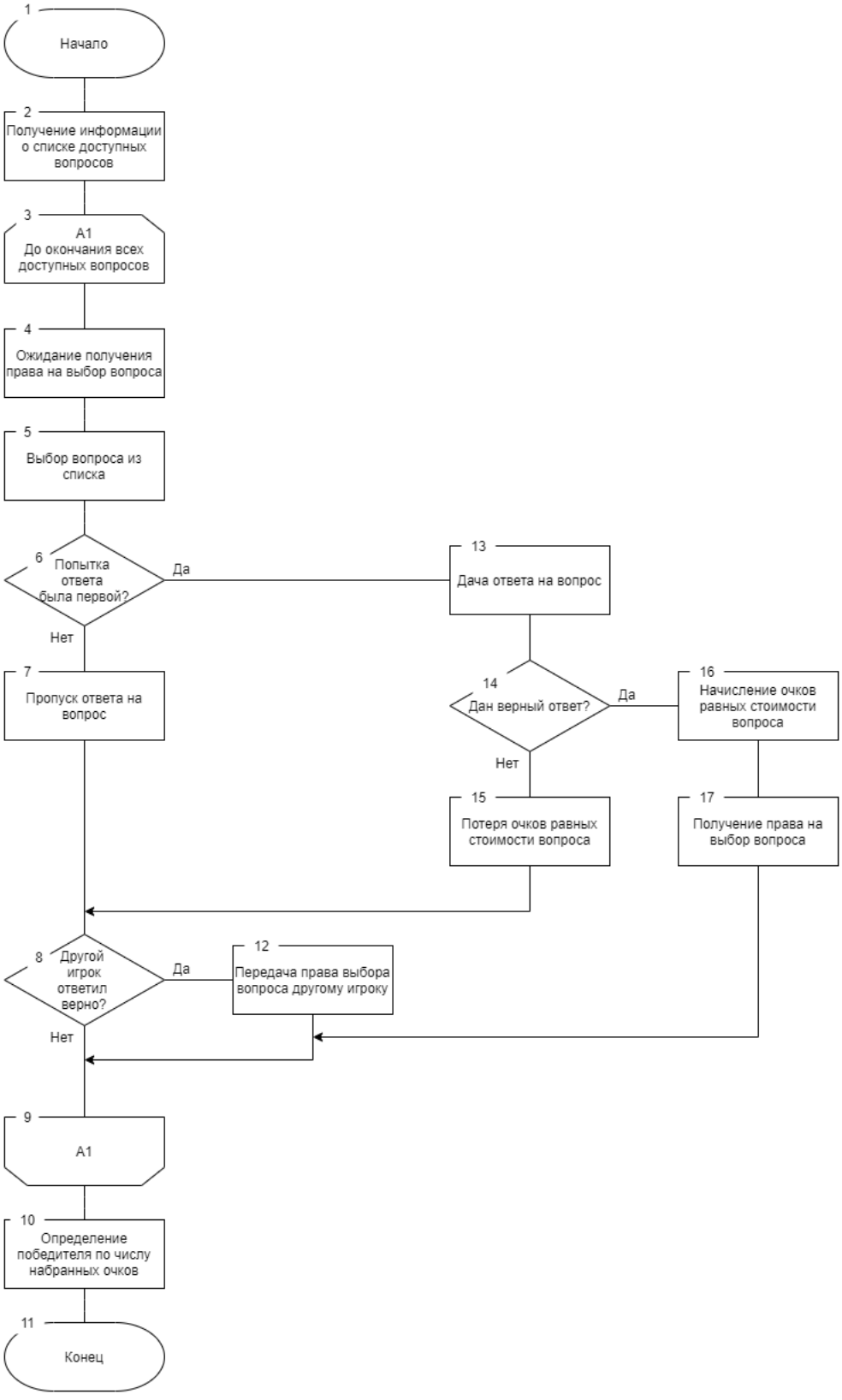

Рисунок 2.1 – Схема алгоритма игрового процесса с точки зрения игрока

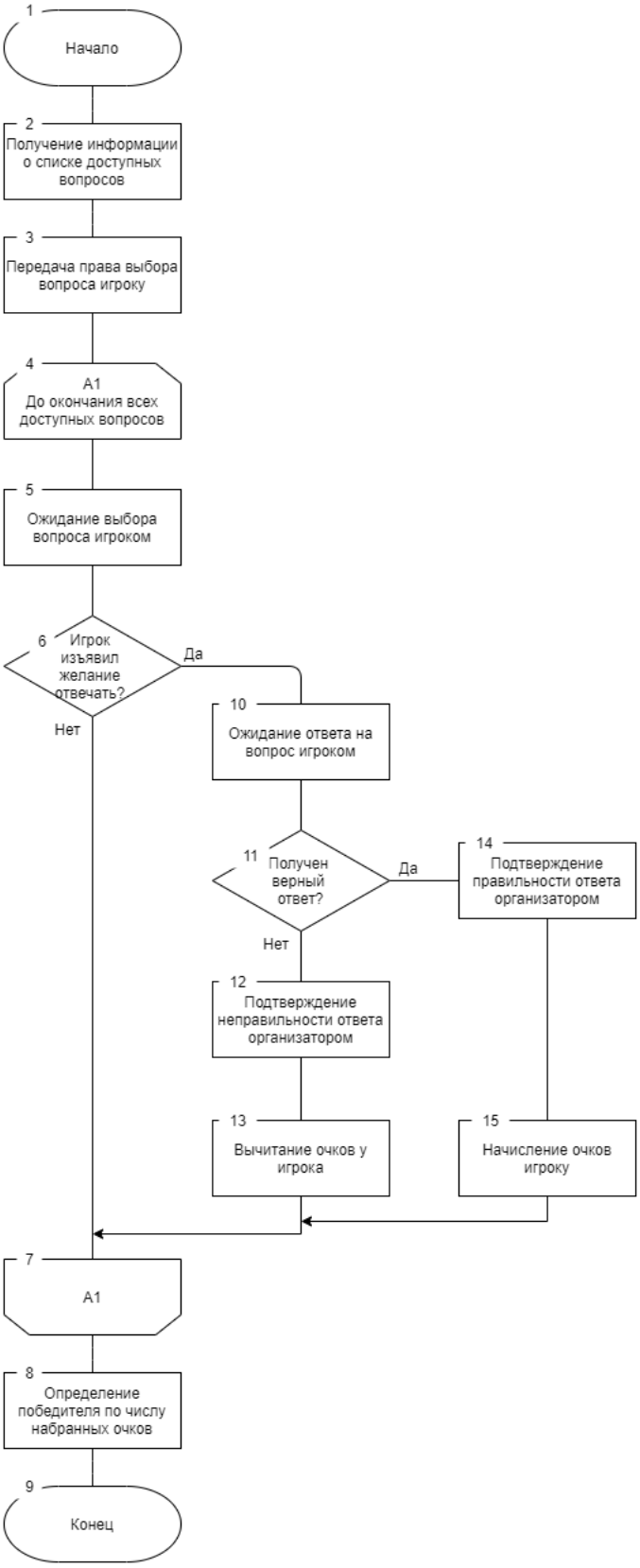

Рисунок 2.2 – Схема алгоритма игрового процесса с точки зрения организатора

Таким образом на схемах алгоритмов преведены типичные варианты автоматизации процесса проведения викторин, как с точки зрения игрока, так и с точки зрения организатора. Программное средство, разработка которой ведется в данном дипломном проекте значительно упростит процесс организации и проведения викторин для всех желающих.

2.1.2 Варианты использования программного средства

По результатам анализа предметной области и существующих аналогов можно сделать вывод, что проектируемое программное средство должно поддерживать ряд функций для уменьшения так называемых накладных расходов на организацию и проведение викторин, ключевыми из которых являются следующие:

– *Система ролей*. Данная система позволяет разделять функционал разных пользователей в рамках одной игровой сессии.

– *Список доступных игровых лобби*. Загрузка списка игровых лобби с использованием внешнего API сервиса-координатора иего отображение в виде последовательного списка позволит игрокам быстрой найти нужного организатора среди всех остальных.

– *Игровые пакеты с вопросами*. Возможность создания и использования игровых пакетов облегчает процесс организации проведения викторин, а также позволяет делится понравившимися пакетами с другими игроками.

– *Результаты викторины*. Автоматический подсчет очков здорово сэкономит время на рутине, которая никому не интересна.

– *Типы вопросов*. Различные типы вопросов повышают уровень вовлечения игроков в игровой процесс: просмотр коротких роликов и прослушивание музыки намного приятнее, чем чтение текста.

– *Контроль организатора за игровым процессом*. Организатор следит за соблюдением дисциплины и проверяет правильность ответов игроков.

– *Анонимная авторизация*. Данный вид авторизации не требует никаких личных данных, но при этом позволяет различать игроков между собой.

Диаграмма вариантов использования игрового клиента, разработанная с использованием нотации UML, представлена на рисунке 2.3.

Рассмотрим подробно представленные на рисунке прецеденты.

Анонимная авторизация – функция, которая доступна для роли «Гость» (пользователь, не авторизованный в системе). В качестве данных для авторизации используется псевдоним и аватар, при этом проверок на

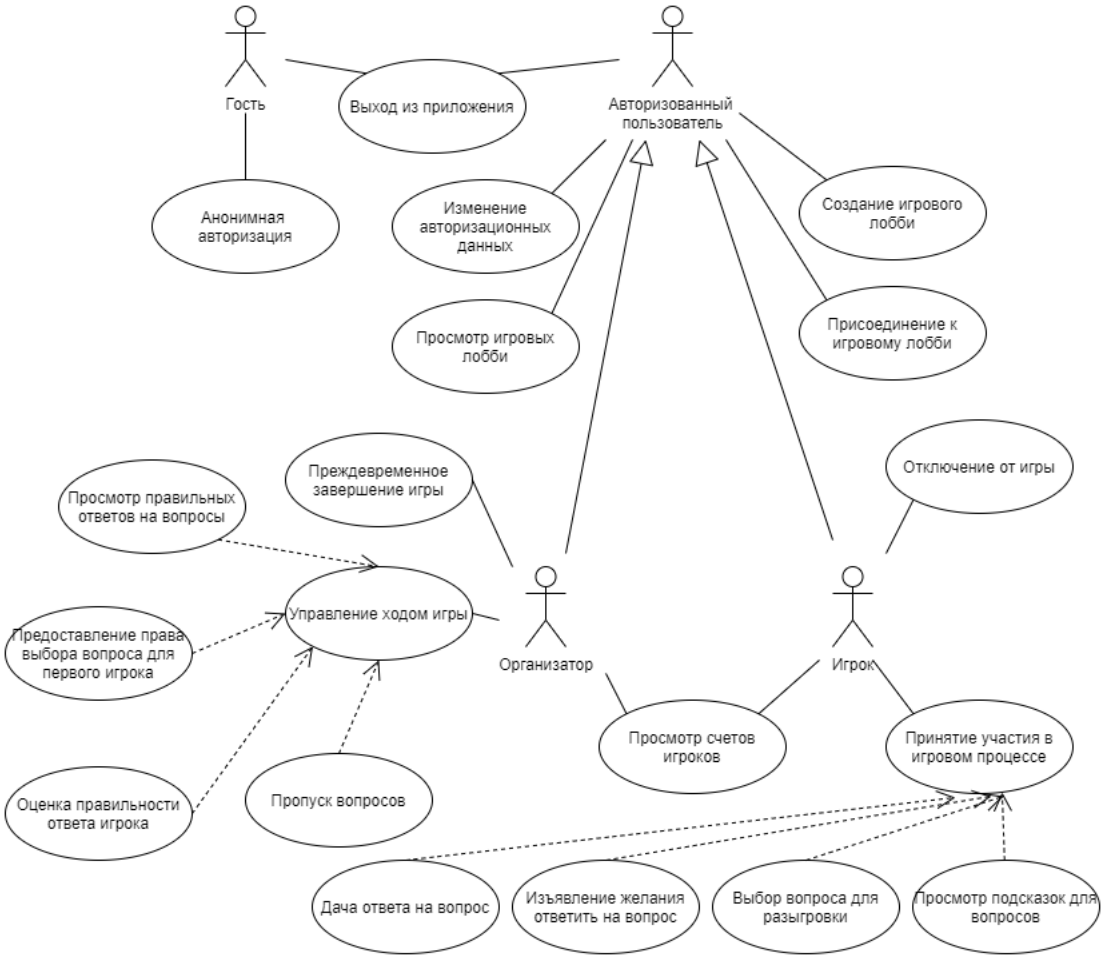

Рисунок 2.3 – Диаграмма вариантов использования игрового клиента

уникальность не производится, так как в ином случае нарушается анонимность процесса.

Неавторизованный пользователь, как и авторизованный может в любой момент выйти из приложения.

После авторизации пользователь получает доступ к *изменению авторизационных данных*, *просмотру активных игровых лобби*, *созданию игрового лобби*, *присоединению к игровому лобби*. Измение авторизационных данных позволит пользователю изменить свой псевдоним или аватар, если ему не нравится его текущий. Просмотр активных игровых лобби позволяет всем авторизованным пользователям найти себе лобби для игры по сети. Перед тем, как лобби появится в списке, организатор должен его создать. После присоединения к лобби пользователь получает роль игрока. Создание игрового лобби необходимо для того, чтобы другие игроки могли присоединится к викторине. После создания лобби пользователь получает роль организатора. Для создания лобби необходимо выбрать один из заранее созданных пакетов вопросов.

Основная функциональность системы предусматривается для непосредственных участников игрового процесса: игроков и организатора.

Одной из самых главных функций, которую использует организатор, является *управление игровым процессом*. Сюда входят такие функции как *предоставление права выбора вопроса для первого игрока* и *оценка правильности ответа игрока*. Так как игра имеет пошаговый характер, то необходимо выбрать того, кто из игроков должен выбрать первый вопрос. Эта одна из задач организатора. В ходе игрового процесса игроки дают ответы на самые разные вопросы, ни один человек не может знать ответы на абсолютно все, поэтому на плечи организатора ложится задача по определению правильности ответа игрока. От его решений зависит исход всей викторины. В качестве помощи оргизатору приложение отображает *правильный ответ* на текущий вопрос, что облегчает процесс оценки правильности ответов игроков. Таким образом организатор является ключевой ролью в обеспечении должного уровня качества проведения викторины и уровня эмоций, получаемых в игровом процессе. Вспомогательным инструментом организатора является возможность *пропустить вопрос* если никто из игроков не знает ответа, в таком случае очки не вычитаются.

Самой главной функцией, которую использует игрок, является участие в игровом процессе. В него входят такие подфункции, как *выбор вопроса для разыгровки*, *изъявление желания дать ответ на вопрос*, *дача ответа на вопрос*, *передача вопроса с секретом другому игроку*, *создание ставки в вопросе со ставкой*. Выбор вопроса для разыгровки прямым образом влияет на сложность самого вопроса: как правило вопросы с большей наградой имеют повышенную сложность. Для того, чтобы дать понять организатору, что игрок хочет ответить на вопрос, можно воспользоваться функцией изъявляния желания. Если игрок это сделал раньше других, то он получает право ответа на вопрос. Здесь стоит обратить внимание, что игрок может изъявить желание, но позже не дать ответа из-за волнения или осознания того, что ответа он не знает. В таком случае отказ от ответа приравнивается к неправильному ответу с соответствующими вычетами из игрового баланса. Все игроки видят *небольшую подсказку* к ответу на вопрос, которая позволяет убрать некоторые сомнения, в случае если составитель викторины не слишком ясно составил вопросы.

Диаграмма вариантов использования программного средства по созданию пакетов вопросов, представлена на рисунке 2.4.

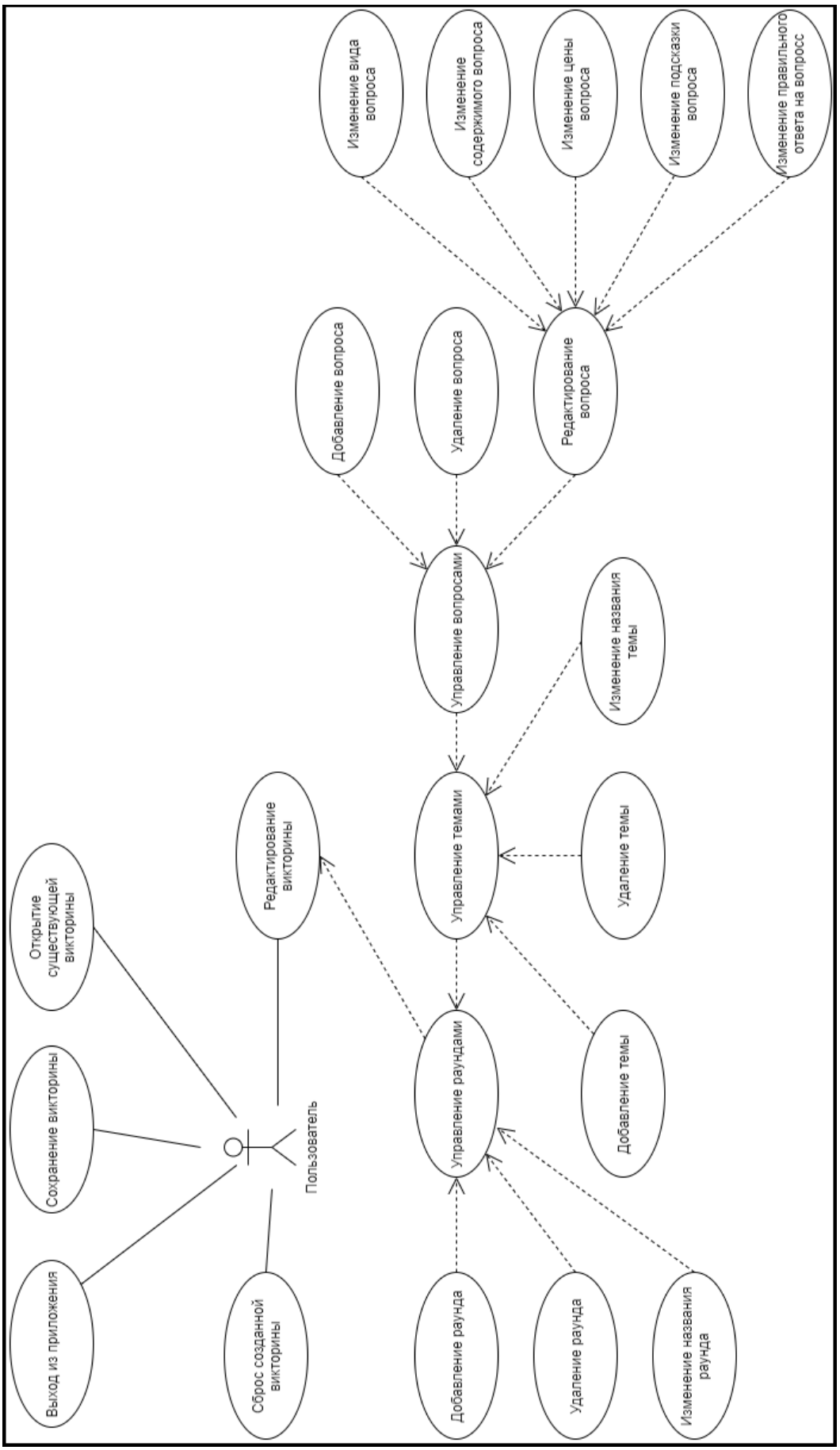

Рисунок 2.4 – Диаграмма вариантов использования ПС создания пакетов с вопросами

Для выполнения поставленной задачи проектируются следующие функции:

– *редактирование викторины*, что позволяет создавать комплексный пакет вопросов, содержащий множество раундов и тем;

– *сохранение викторины*, что позволяет сохранить результат редактирования на диск с целью дальнейшей доработки или для игры на клиенте;

– *открытие существующей викторины*, что означает возможность открыть ранее сохраненный результат редактирования викторины для дальнейших изменений.

Создание пакетов вопросов, является неотъемлемой частью работы программного средства: созданные пакеты в дальнейшем интерпретируются клиентом и позволяют участникам принять участие в игровом опыте. Достоинством такого подхода является возможность делиться викторинами путем передачи файла.

2.2 Разработка спецификации функциональных требований

С учетом требований, определенных в подразделе 1.3, необходимо детализировать функциональные требования для проектируемого программного средства.

2.2.1 Функция анонимной авторизации

Функция авторизации должна быть реализована с учетом следующих требований:

а) процесс авторизации инициируется пользователем системы (на рисунке 2.3 представлен в виде роли «Гость»);

б) функция реализуется собственными средствами без обращения к внешним поставщикам;

в) для авторизации пользователь обязан предоставить псевдоним и выбрать аватар;

г) уникальность псевдонима и аватара не проверяется;

д) должна быть предусмотрена возможность смены псевдонима и аватара после авторизации.

2.2.2 Система ролей

При реализации системы ролей следует учесть требования:

а) должны быть реализованы следующие роли:

1) игрок;
2) организатор.

б) организатор должен иметь возможность управлять игровым процессом;

в) для получения роли организатора, пользователь должен создать игровое лобби с выбранным пакетом вопросов;

г) игрок должен иметь возможность непосредственно принимать участие в игровом процессе;

д) для получения роли игрока, авторизованный пользователь должен присоединится к игровому лобби;

е) должна быть реализована возможность смены роли путем отключения от лобби и создания/присоединения к новому.

2.2.3 Функция просмотра списка игровых лобби

Просмотр списка игровых лобби входит в число основных функций разрабатываемого приложения. При реализации данной функции необходимо учесть следующие требования:

а) необходимо обеспечить отображение списка лобби в виде последовательного списка, отсортированного в порядке убывания дат;

б) список отображает все лобби, к которым можно подключится, т. е. которые еще активны;

в) в списке для каждого лобби должна быть отображена следующая информация:

1) название лобби;
2) псевдоним организатора;
3) количество игроков в лобби;
4) максимальное количество игроков;
5) защита паролем.

г) список должен автоматически подгружаться используя API сервиса-координатора.

2.2.4 Функция создания игрового лобби

Функция создания игрового лобби должна быть реализована с учетом следующих требований:

а) для создания лобби необходимо выбрать заранее созданный пакет вопросов;

б) в настройках можно задать следующие параметры:

1) название лобби;
2) пароль;
3) максимальное количество игроков.

2.2.5 Функция управления игровым процессом

Функция управления игровым процессом является ключевой. При ее реализации должны быть учтены следующие требования:

а) только организатор может управлять игровым процессом;

б) при проверке правильности ответа, организатор должен видеть сноску с правильным ответом от составителя вопроса;

в) в случае правильного ответа игроку начисляются очки;

г) в случае неправильного ответа игроку снимаются очки;

д) в случае преждевременного завершения игры победитель не определяется.

2.2.6 Функция принятия участия в игровом процессе

Функция принятия участия в игровом процессе также является ключевой. При ее реализации должны быть учтены следующие требования:

а) участие может принимать только игрок;

б) можно выбирать вопрос только в том случае, если у игрока есть право выбора;

в) игрок имеет право как изъявить желание ответить на вопрос, так и не изъявлять его;

г) время ответа на вопрос контролируется организатором;

д) во время ответа на вопрос игроки должны видеть подсказку к ответу;

2.2.7 Функция редактирования викторин

Следующая группа требований относится к деятельности пользователя, создающего набор вопросов:

а) каждый вопрос имеет следующий базовый набор данных:

1) содержимое и тип содержимого вопроса;
2) цена вопроса;
3) правильный ответ для организатора;
4) подсказка к ответу для игроков.

б) должна быть возможность изменять общее количество раундов в викторине;

в) должна быть возможность изменять количество тем в каждом раунде;

г) должна быть возможность изменять количество вопросов в каждой теме;

д) должна быть возможность открытия/сохранения викторин.

3 ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

3.1 Разработка архитектуры программного средства

Как было указано в пункте 1.3.9, на основании анализа вариантов проектирования приложения для различных платформ, было принято решение выбрать основной для разработки настольное приложение с небольшим веб-сервисом координатором.

Одним из вариантов архитектуры для приложений данного типа является двухзвенная клиент-серверная архитектура представленная на рисунке 3.1. В проектируемом приложении задачами сервера будет являтся координация клиентов и помощь в установке интернет соединения на основе протокола TCP, что позволит достичь максимально возможного уровня надежности данных, хоть и с недостатками в виде сниженной скорости передачи данных. Задачей сервера будет возврат запрашиваемых данных по запросу клиентов, отправка команд игрокам, а также хранение небольшой локальной таблицы со списком игровых лобби. По этой причине необходимость в базе данных отпадает – так как большого объема данных не будет и ценность хранимых данных невысока. В дополнение стоит сказать, что доступ к локальному хранилищу быстрее, чем к бд. Выбранная в пункте 1.3.9 программная платформа позволяет нам это реализовать.

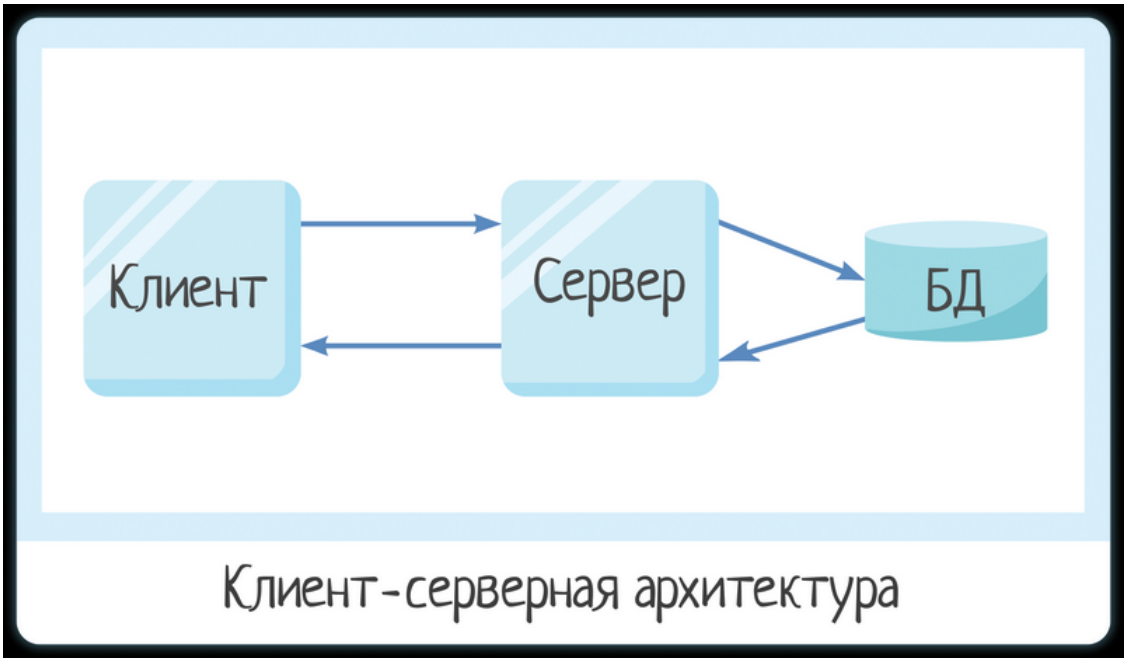

Рисунок 3.1 – Типичная двухзвенная архитектура

В свою очередь задача клиентской части при взаимодействии с серве-

ром состоит в том, чтобы при создании игрового лобби передать на сервер информацию о лобби, а также о сети, в которой находится машина-хост. Это нужно для того, чтобы другой клиент, который хочет подключится к игре мог воспользоваться координатором и принимать участие в игровом процессе. В данном случае сервер является своеобразным валидатором игровых действий и хранит всегда валидные состояния, что повышает надежность разрабатываемого программного обеспечения, а также позволяет без лишних усилий реализовать некоторый функционал, например, подключение посреди игры. Общение между клиентами будет проходить посредством комманд, которые будут выполяться на сервере и рассылатся клиентам для локального выполнения.

В итоге архитектура, по которой будут работать клиенты и сервер-координатор выглядит следущим образом (рис. 3.2). В центре находится сервер-координатор, все остальные компьютеры – клиенты.

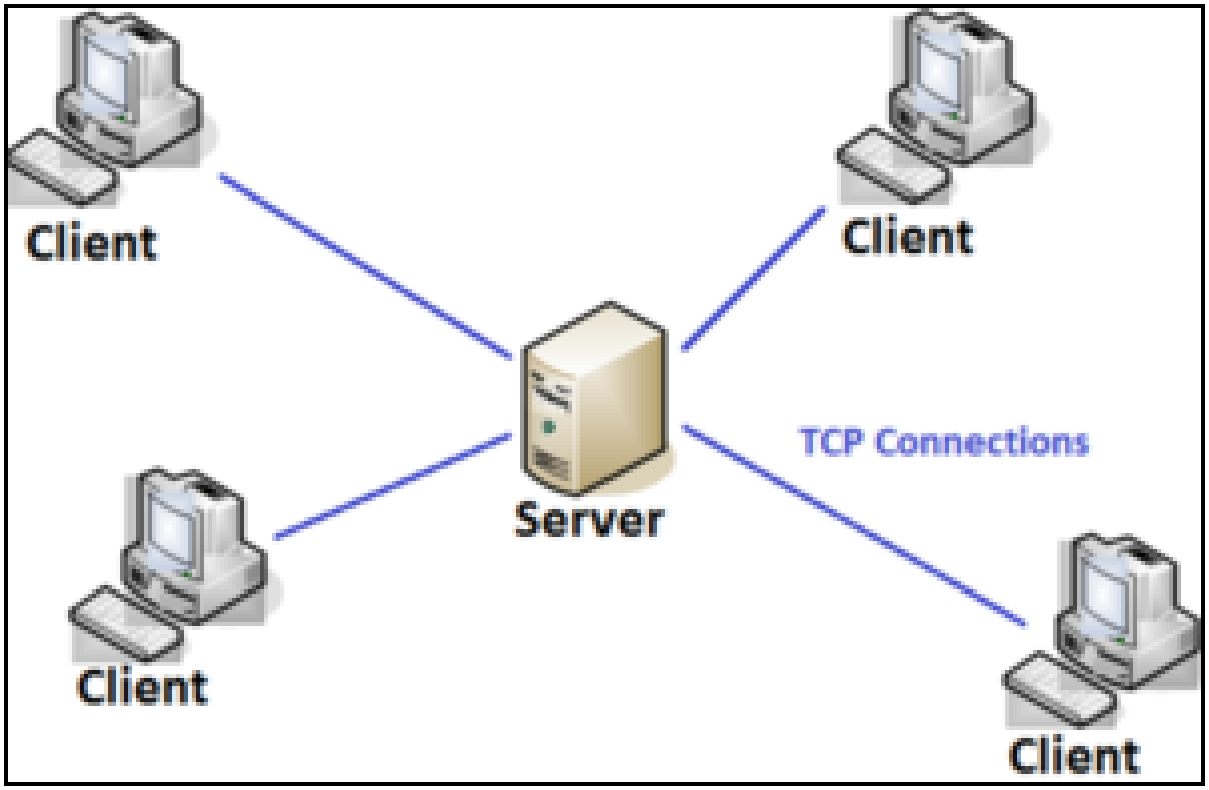

Рисунок 3.2 – Архитектура клиент-сервер

3.2 Проектирование и разработка программного средства создания пакетов вопросов

Перед тем как проводить викторины, сначала необходимо создать пакет с вопросами, для этой задачи и существует программное средство со-

здания пакетов вопросов. Предполагается, что им может пользоваться кто угодно, а созданную викторину можно распространить по сети Интернет, чтобы другие игроки также смогли попробовать ее провести.

Сама по себе викторина делится на раунды, раунды в свою очередь делятся на темы, а каждая тема содержит перечень вопросов, на которые нужно отвечать. Проектируемое программное средство должно уметь создавать пакеты с вопросами с помощью графического пользовательского интерфейса, а также иметь возможность редактировать уже существующие пакеты. Рассмотрим решение каждой из задач подробнее и выясним наиболее предпочтительные подходы к разработке.

3.2.1 Проектирование типа данных викторины

Проектирование будет проводится снизу вверх, поэтому типы данных идут от самого низкоуровнего к самому высокоуровневому.

Программное средство должно поддерживать простой тип вопросов, суть которого заключается в том, что игрок, первый пожелавший дать ответ на него и отвечает, в случае правильного ответа очки начисляются на счет, в случае неправильного – отнимаются. В данной версии программного средства закладываются архитектурные основы для дальнейшего расширения по необходимости.

Для реализации этого типа данных можно использовать перечислимый тип (enumerable), поддержка которого есть в C#. Его главным достоинством является то, что программист может дать осмысленные имена числовым константам и избавится от "магических"чисел в коде [1], что облегчает дальнейшее понимание и разработку программного обеспечения. Но в данном программном средстве лучше воспользоваться объектно-ориентированным подходом и использовать интерфейс с конкретными реализациями. Как это сделать будет описано позже.

Помимо типов вопросов, сам вид содержимого вопросов может также отличаться. Должны поддерживаться следующие виды вопросов:

– текст, представляющий из себя одно или несколько предложений;
– изображение, которое отображается в течении нескольких секунд;
– звук, играющий от начала и до конца;
– видеоролик, играющий от начала и до конца.

Для реализации данных видов вопросов стоит использовать массив байт, который представляет из себя абстрактное содержимое вопроса, таким образом можно представить абсолютно любой тип содержимого, для которого в будущем будет необходимо реализовать обработчики. Также для

высокоуровневой отметки типа данных можно использовать перечислимый тип описанный выше, таким образом обработчики смогут корректно преобразовать массив байт в нужный тип контента. Доступ к контенту в данной версии программного средства будет единственный и будет является встроенным, т. е. локальной копией, находящейся на компьютере. Задаток для интернет-контента будет заложен в архитектуре.

Описав примитивы, являющиеся частью вопроса можно перейти к проектированию самого вопроса. Любой вопрос состоит из следующих полей:

– тип вопроса;
– тип содержимого вопроса;
– тип доступа к содержимому вопроса;
– подсказка к ответу на вопрос;
– стоимость ответа на вопрос, которая будет прибавлятся или отниматься;
– содержимое вопроса, представленное массивом байт;
– сноска для организатора с правильным ответом на вопрос.

Для всех вопросов будет использоваться класс Question, содержащий в себе все поля необходимые для работы. В дальнейшем для различия типов вопросов в C# предусмотрена концепция под названием pattern-matching [14], позволяющая передавать управление в разные ветви программы, в зависимости от типа данных, который скрывается за интерфейсом либо от значения некоего поля, такой подход позволяет использовать более удобный синтаксис, а также повышает расширяемость программы за счет того, что в будущем различные типы могут существенно различаться, а управляющие конструкции уже менять не придется.

Категории вопросов отражают какую-то общую тему, по которой были составлены вопросы. Категория должна содержать следующие поля:

– список вопросов;
– название категории.

Здесь стоит отметить, что сам список вопросов тоже должен является абстракцией, для этого в C# предусмотрено несколько интерфейсов, например IList. Абстракция позволяет не опираться на конкретные реализации, что в свою очередь уменьшает связанность, повышает сопровождаемость и уменьшает объем кода, который необходимо написать при расширении.

По аналогии с категориями проектируются и раунды, из которых собственно и состоит викторина. Раунд должен содержать следующие поля:

– список категорий;

– название категории.

Здесь нужно обратить внимание на то, что абстракции не применяются для полей раунда, так как подразумевается что будет существовать только одна имплементация, поэтому необходимость в дополнительном уровне абстракций отпадает сама собой. Это соответствует одному из принципов програмированния под названием KISS – keep it simple, stupid.

Сама викторина будет агрегировать в себе уже разработанные типы данных и состоять только лишь из списка раундов. Разработанный тип данных и будет результатом создания в программе по созданию пакетов вопросов и в дальнейшем использоваться для проведения викторин в приложении-клиенте.

3.2.2 Проектирование функции редактирования викторин

Так как программное средство представляет из себя CRU (create, read, update) приложение, то для упрощения разработки стоит использовать технологию WPF от компании Microsoft. Ключевой особенностью данной технологии является уникальный подход к работе с данными. Этот подход называется MVVM – model, view, viewmodel и основывается на механизме привязки к данным [15]. Рассмотрим сам механизм и отдельные уровни подробнее.

Уровень view представляет из себя набор представлений, которые пользователь видит в качестве пользовательского интерфейса приложения [15]. Он выглядит как набор кнопок, полей для ввода и прочих интерактивных элементов. На этом уровне задаются формат ввода данных, сами интерактивные элементы, валидации, а также стили. Каждое представление имеет свой формат файла с расширением .xaml. Этот файл состоит из двух подфайлов, разделяющих функции: файл разметки, содержащий описание того, как отображать элементы, а также файл логики, содержащий минимальную логику по обработке информации с элементов интерфейса. Именно на этом уровне используется механизм привязки к данных (data binding) модели представления.

Модель представления (viewmodel) – это промежуточный уровень между бизнес-логикой и самим представлением, его задача заключается в получении данных от слоя model и передачу оного в слой view [15]. Для этого все модели представления реализуют интерфейс INotifyPropertyChanged, который содержит одно единственное поле-событие: PropertyChanged. Когда происходит связывание представления и модели представления, то представление подписывается на событие PropertyChanged и автоматически пе-

рерисовывает элементы управления, которые были связаны с обновляемыми данными. В свою очередь модель представления после обновления данных уведомляет представление о том, что данные действительно изменились, для этого происходит вызов Invoke в событии PropertyChanged.

Привязка к данным – подход, который позволяет четко разделить уровень представлений и моделей представлений, для настройки его правильной работы необходимо выполнить следующие действия:

а) создать модель представления, которая реализует интерфейс INotifyPropertyChanged;

б) добавить требуемые свойства в класс, к которым будет осуществляться привязка;

в) установить контекст данных (datacontext) представления на экземпляр класса модели представления;

г) в файле с разметкой представления использовать binding синтаксис для привязки к свойствам модели представления.

При таком подходе все изменения на пользовательском интерфейсе будут автоматически отражаться в хранимых типах данных и наоборот. Это избавляет программиста от большого количества однообразного кода, который не выполняет ничего полезного, а только лишь создает/обновляет/удаляет простые данные.

Уровень модели представляет из себя весь набор бизнес-логики, которая необходима приложению для выполнения поставленных задач [15]. Сюда входит все, что не является простым манипулированием данными. Методы из этого уровня вызываются из модели представления, а аргументы для вызова в свою очередь беруться из представления. Таким образом уровни приложения связаны достаточно свободно, что облегчает изменение и расширение, но в то же время обеспечивает достаточный уровень удобства при разработке.

3.2.3 Проектирование функций сохранения и открытия викторины

После того, как была спроектирована викторина, ее необходимо представить в однообразном виде, который будет понятен как программному средству для создания пакетов вопросов, так и игровому клиенту. Для решения поставленной задачи можно прибегнуть к концепту сериализации [16]. Суть сериализации заключается в следующем: любой объект можно однозначно представить в некоем формате, например, для передачи по сети, а затем из этого формата получить объект обратно. Процесс перевода объекта в некий формат называется сериализацией, а процесс перевода из

формата в объект называется десериализацией.

В настоящее время популярны несколько форматов сериализации:

– JSON – JavaScript object notation;
– XML – Extensible markup language;
– бинарное представление (массив байт).

Рассмотрим каждый из форматов подробнее:

JSON – является одним из самым популярных форматов сериализации, применяется во многих областях разработки: настольные приложения, веб-приложения и даже базы данных. Простота, читаемость и относительно небольшой размер сериализованных объектом обеспечили ему достаточно прочные позиции в списке популярных форматов сереализации.

XML – является вторым по популярности после JSON, имея все преимущества первого он все же проигрывает в читаемости за счет дублирования тегов, а также сериализованные объекты имеют больший размер за счет того, что сами теги могут попрой занимать больше места чем данные, которые в них хранятся.

Бинарный формат – является самым простым и компактным из приведенных. Главным недостатком является абсолютная нечитаемость без посторонних инструментов, благодаря чему он особенно не популярен в веб-разработке. Тем не менее, для разработки настольных приложений он более чем годится. Минимальный размер сериализации, возможность сериализовать циклические зависимости и простота использования являются серьезными преимуществами при разработке сетевых приложений, передача данных которых должны происходить как можно реже и как можно меньшими объемами.

Для сериализации пакетов с вопросами будет использоваться бинарный формат, так как сериализация музыки и видео обусловлена значительными трудностями при использовании иных форматов. Кроме того, подразумевается, что пакет будет передаваться по сети другим игрокам, поэтому данный способ является наиболее предпочтительным среди аналогов.

3.3 Проектирование и разработка сервиса-координатора

3.3.1 Этапы разработки программного средства

Архитектурный стиль разделения функциональности программной системы на уровни поднимает сразу несколько проблем. Их решение в типичном случае заключается в выполнении следующих этапов [9]:

а) определение стратегии разбиения на уровни;

б) определение уровней;

в) распределение функциональности по уровням и компонентам;

г) уточнение количества уровней: при необходимости некоторые из них объединяются;

д) установление правил взаимодействия уровней;

е) идентификация функциональности, которая используется на всех уровнях (cross cutting concerns);

ж) определение интерфейсов уровней;

з) выбор стратегии развертывания;

и) выбор конкретных протоколов взаимодействия.

К данному этапу разработки дипломного проекта выполнены шаги с 3.3.1а по 3.3.1д. Шаг 3.3.1е вследствие использования различных программных средств, применяемых на уровнях, будет выполняться позднее – на соответствующих уровнях.

Таким образом, следующий этап – это определение интерфейсов уровней. Предполагается, что инициатором любых действий будет являться клиентская часть программного средства. Следовательно, единственный интерфейс, который следует определить – это интерфейс сервиса-координатора части.

Как правило веб-сервисами пользуется большое количество людей одновременно, поэтому при проектировании необходимо учесть данный фактор и реализовать автономность и независимость частей проектируемой системы. Именно по этой причине был выбран сервис-ориентированный архитектурный стиль.

3.3.2 Программный интерфейс сервиса-координатора

Предварительным условием для проектирования серверной части приложения является определение его интерфейса в высокоуровневых терминах предметной области. Таким образом, далее приведены методы его API, которые основаны на функциональных требованиях, определенных в подразделе 2.2:

- метод создания игрового лобби;
- метод подключения к игровому лобби;
- метод отключения от игрового лобби;
- метод выполнения игровых комманд;
- метод получения списка игровых лобби.

На рисунке 3.3 приведена схема алгоритма обработки запросов перечисленных методов программного интерфейса серверной частью приложе-

ния. Далее приведены архитектурные решения, которые были применены при реализации данной части программной системы.

Следует отметить, что доступ к методам веб-сервиса может получить любой желающий, использующий протокол TCP и знающий конечную точку для подключения, в том числе разработчик альтернативного клинта для игры, например, для мобильных телефонов. Поэтому разработанный подход является максимально гибким с точки зрения дальнейшего расширения программного средства, а также поддержки кроссплатформенности.

3.3.3 Выбор протоколов коммуникации

В пункте 3.3.1 приведены этапы, в соответствии с которыми рекомендуется производить проектирование программного системы [9]. Этап 3.3.1з будет рассмотрен позднее вследствие неоднородности и отсутствия схожести в применяющихся технологий между уровнями. Таким образом, следующий актуальный этап – 3.3.1и: выбор конкретных протоколов взаимодействия.

Можно выделить два основных протокола, которые (с необходимыми дополнениями в каждом конкретном случае) применяются в информационных системах, содержащих веб-сервисы [9]: TCP (Transmission control protocol – протокол управления передачей данных) и UDP (User datagram protocol – протокол пользовательский дейтаграм). Они оба используются для передачи данных от клиента серверу и обратно.

Основное отличие между данными протоколами состоит в способе передачи данных. UDP является протоколом передачи данных без установки соединения, в таком случае данные могуть не достичь адресата. Этот протокол является более быстрым по сравнению с TCP, в то время как надежность значительно меньше. Этот протокол используется в приложениях, где точность данных не слишком важна или, где объем пересылаемых данных за единицу времени достаточно большой.

Протокол TCP является более надежным по сравнению с UDP, перед отправкой пакетов необходимо установить соединение, качество которого проверяется через систему трех рукопожатий. Такой подход гарантирует высокую целостность данных при более низкой скорости их передачи. Скорость теряется в процессе пересылки служебных данных, отвечающих за целостность, в то время как полезные данные ждут своей очереди.

В данном приложении используется TCP, так как для выполнения команд необходима высокая точность передаваемых данных, а сами данные пересылаются не очень часто.

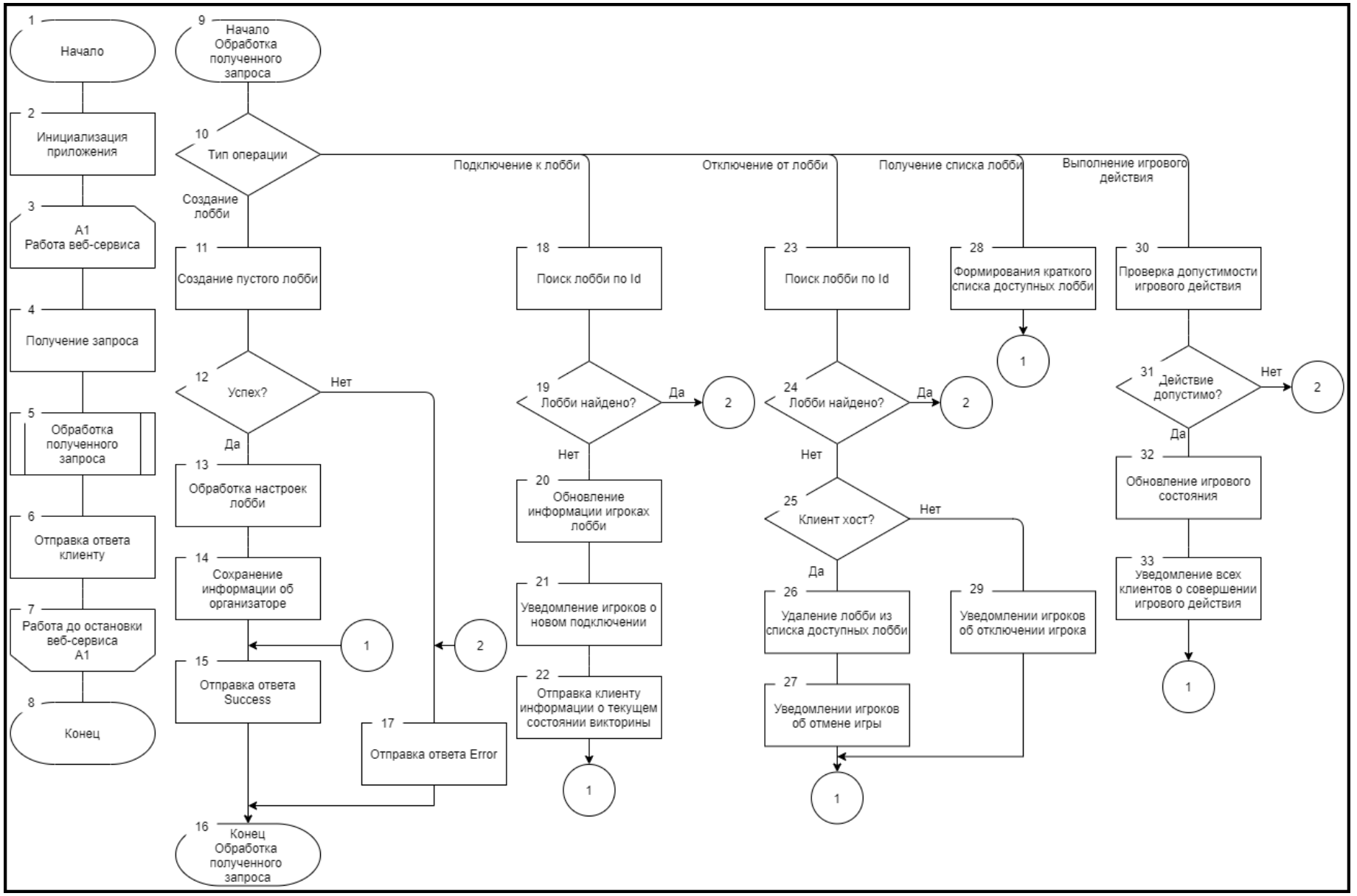

Рисунок 3.3 – Схема программы серверной части программного средства

3.4 Проектирование и разработка клиентской части программного средства

Задачи клиентской части приложения включают отображение пользовательского интерфейса и обработку действий пользователя. Типичными этапами его проектирования являются следующие [9, с. 78]:

– Идентификация типа клиентской части приложения, которое удовлетворяет установленным требованиям. Осуществление данного этапа осуществлено в подразделе 3.1.

– Выбор технологии пользовательского интерфейса. Осуществляется на основе анализа требуемой для реализации функциональности.

– Проектирование UI. Хорошей практикой является реализация модульности, а также принципа разделения ответственности компонентов.

– Определение стратегии и протоколов обмена информации между уровнями. Поскольку рассматриваемый уровень является самым верхним, а его протоколы связи с серверной частью приложения были рассмотрены в пункте 3.3.3, то данный вопрос считается решенным и рассматриваться не будет.

Ранее был проведен анализ и осуществлен выбор целевой платформы для реализации клиентской части приложения, был выбран основной язык программирования. Однако, существует огромное число специализированных технологий и фреймворков по созданию пользовательских интерфейсов. Представляется целесообразным провести уточнение средств разработки.

3.4.1 Уточнение выбора технологий программирования

Язык программирования C#является языком общего назначения. Он является строготипизированным, то есть компилятор проверяет соответствие типов при работе с данными. Код, написанный на C#может быть скомпилирован компилятором MSVC от компании Microsoft. Для того, чтобы программная среда Visual Studio поддерживала программы на C#необходимо воспользоваться пакетным менеджером Visual Studio Installer, в котором нужно выбрать пакет инструментов для разработки под платформу .NET. После установки инструментов в среде можно создать проект, с целевой платформой .NET, где и будут использоваться установленные ранее инструменты. Все работает по умолчанию и программисту не нужно тратить свое время на дополнительную настройку.

При разработке настольных приложений на платформе .NET есть два

основных подхода: WinForms и WPF. Рассмотрим каждый из них подробнее.

WinForms - первоначальный подход к высокоуровневому проектированию пользовательского интерфейса на платформе .NET. До их появления, единственным вариантом для программиста было использование WinApi, которое является крайне не удобным с точки зрения разработки комплексных решения для пользовательского интерфейса. Главным минусом WinApi является то, что он не является кроссплатформенным, так как доступен только на платформе с операционной системой Windows, что и следует из названия. Для решения этой проблемы инженеры компании Microsoft и придумали слой абстракций для разработки пользовательского интерфейса, который получил название WinForms. Ключевой особенностью является то, что разработанный пользовательский интерфейс является кроссплатформенным, то есть может работать, например, под операционной системой Linux. Разработка интерфейса происходит с помощью перетягивания элементов управления на окно дизайнера интерфейса. Такой подход позволяет быстро реализовать самые сложные интерфейсы за счет того, что больше не нужно тратить время на поиск нужных функций WinApi, к тому же его не нужно переписывать в случае, если приложение разрабатывается под несколько платформ. Не смотря на удобство в использовании, разработанный с помощью WinForms интерфейс имеет ряд недостатков, которые сложно устранить штатными средствами, а именно:

– Отсутствие глубокой кастомизации внешнего вида элементов управления. Программист может выбрать один из нескольких уже готовых пресетов, а в случае если нужного пресета нет, то необходимо прибегать к WinApi, из-за чего теряется кроссплатформенность.

– В подавляющем числе случаев интерфейс не будет является адаптивным по отношению к размерам окна. Это снижает удобство использования для конечных пользователей.

– Использование обработчиков событий является устаревшим подходом. Такой подход сильно связывает пользовательский интерфейс и классы-контроллеры, что ухудшает расширяемость и снижает продуктивность разработки.

Несмотря на все недостатки, WinForms были популярны достаточно долгое время, благодаря низкому порогу вхождения и простоте проектирования. Инженеры Microsoft проанализировали недостатки WinForms и сделали новый пакет инструментов, устраняющий все недостатки, упомянутые выше. Он получил название WPF – Windows Presentation Foundation.

WPF представляет из себя дизайнер, который поддерживает привяз-

ки к данным, разработку адаптивного интерфейса, а также глубокую систему стилей, со своими видами наследования, а также обработчиками в контексте пользовательского интерфейса, которые называются триггерами. Рассмотрим основные возможности платформы WPF.

Представления, как обособленные сущности, являются полностью независимыми от какого-либо контекста и могут обрабатывать действия пользователя никого не уведомляя. Для этих целей используются триггеры – специальные процедуры, которые срабатывают в заданном программистом случае и чаще всего выполняют примитивные задачи. Например скрытие списка, если он пуст. Триггер должен проверять содержимое элемента управления (ItemsSource) и если количество (Count) элементов равно нулю, то видимомость элемента управления (Visibility) устанавливается в false. Это лишь один из многих примеров использования триггеров, с их помощью можно автоматизировать множество задач без необходимости написания кода обработчиков.

При разработке формата представлений инженеры вдохновлялись HTML и CSS, типичное представление содержит информацию о расположении элементов управления в формате XAML. XAML очень похож на HTML, его главной особенностью является то, что он расширяемый, то есть программист может писать свои расширения разметки, которые можно будет использовать в разработке. Одной из сильнейших сторон WPF по сравнению с WinForms является система стилей. Она похожа на CSS из мира веб-разработки. Каждый элемент управления имеет свой набор атрибутов, таких как видимость, цвет фона, шрифт, размер шрифта и так далее. Все это можно оформить в один стиль, дать ему имя, которое будет является ключом и всем похожим элементам управления присваивать ключ стиля. Таким образом все одинаковые элементы управления будут иметь одинаковые стили, которые в случае чего можно будет поменять только в одном месте. Стоит также отметить, что все стили поддерживают механизм наследования, то есть при желании разработчик может добавить новый атрибут к стилю, при этом сохранив все значения атрибутов базового стиля, что особенно удобно когда нужно внести минимальные правки, но создавать стиль с нуля слишком накладно.

Еще одним существенным отличием WPF от WinForms является наличие шаблонных элементов управления и поддержка автогенерации интерфейса. В WinForms для генерации нужно было использовать code-behind самой формы, что увеличивало уровень связности, а также приводило к большому количеству шаблонного кода, который не решал никакой бизнес-

задачи. В WPF эту проблему решили, позволив создавать шаблоны пользовательского интерфейса, причем шаблоны могут отличаться в зависимости от текущего типа данных, для которого генерируется интерфейс. Например, у программиста есть три класса, которые реализуют один и тот же интерфейс и хранятся в списке как интерфейс класса, генератор UI может определять какой тип содержит каждый элемент из списка и создавать требуемый формат для заполнения полей. Для этого используется концепт pattern-matching [14].

Про привязку данных уже шла речь в разделе 3.2.2, поэтому здесь пойдет речь про команды, как механизм реакции на действия пользователя. Одним из главных недостатков WinForms являлся механизм событий, который жестко связывал конкретный элемент управления и код обработчика. Команды в свою очередь призваны предоставить требуемый уровень абстракций, позволяющий отвязать элемент управления от обработчика. Команды работают на основе привязки к данным и не требует никакой информации о том, кто их вызывает, поэтому разрабатываемое программное средство получает дополнительный уровень гибкости в случае измений требований.

Подводя итоги выбора технологии, перечислим причины, по которым был выбран WPF:

– четкое разделение представлений и механизмов обработки данных;
– поддежка комплексных стилей;
– поддержка триггеров, как локализованных обработчиков событий;
– поддержка команд, как абстрактных обработчиков событий;
– простота разработки адаптивного пользовательского интерфейса;
– механизм привязки к данным.

Таким образом, на основании проведенного уточнения технологий для использования выбираем WPF.

3.4.2 Проектирование клиентской части приложения

Далее рассмотрим вопрос взаимодействия пользователя с клиентской частью приложения. Известно, что удобство пользования программным средством может во многом определять успешность проекта в целом [1, с. 44].

Схема работы клиентской части программной системы представлена на рисунке 3.4. Среди ее особенностей можно выделить в высокой степени соответствие диаграмме прецедентов, которая была составлена и рассмотрена в пункте 2.1.2.

Таким образом, составленная схема клиентской части программного

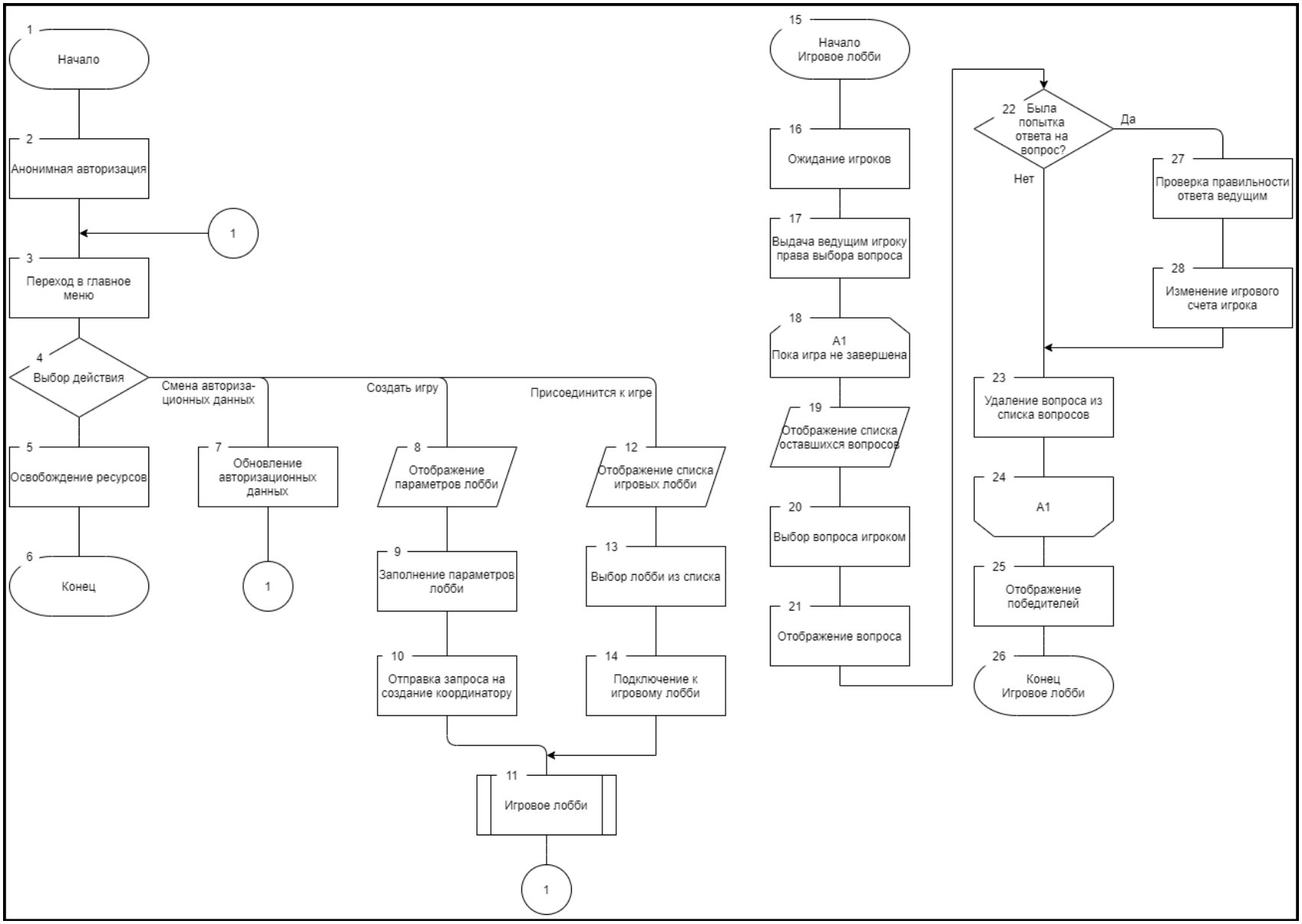

Рисунок 3.4 – Схема программы клиентской части программного средства

средства будет использована при разработке непосредственно самого приложения.

3.4.3 Конструирование клиентской части приложения

Наконец, по завершению этапов проектирования и подготовки можно приступить к собственно созданию исходных кодов приложения. Ключевые фрагменты кода, описание которых приведено в данном пункте, приведены в приложении.

Для начала необходимо создать проект типа WPF, с целевой платформой .NET Core [3]. Структура проекта будет иметь следующий вид:

а) папка views, содержащая описания UI;

б) папка viewmodels, содержащая модели представления, для привязки на UI;

в) папка models, содержащая бизнес-логику по обработке данных с UI;

Вся разделяемая логика вынесена в отдельный проект, туда относятся бизнес-сущности и общие методы обработки. Среди команд реализуется RelayCommand<T>, представляющая из себя универсальную команду, способную работать с параметрами произвольного типа, благодаря шаблонному типу T. Реализуется два варианта команды: с параметром и без, каждый вариант применяется по ситуации и избавляет от необходимости писать неэффективный код, в случае, если параметр не нужен.

Также реализуются различные расширения разметки, позволяющие более эффективно разрабатывать пользовательский интерфейс и связанную с ним однобразную логику. Такой подход исключает необходимость дублирования комманд или логики в случае использования одного и того же элемента управления.

Для упрощения процесса оповещения представлений об изменениях в моделях представлений используется сторонняя библиотека Fody.PropertyChanged, ее суть заключается в том, чтобы автоматически добавлять вызов события PropertyChanged для всех публичных свойств, если модель представления реализует интерфейс INotifyPropertyChanged. Ее использование уменьшает объем кода, необходимы для описания свойств примерно на 80%, что значительно ускоряет процесс разработки. Для того, чтобы разработчикам не нужно было везде подключать эту библиотеку было принято решение о вынесении ее в отдельный базовый класс ViewmodelBase.

Таким образом, были разработаны исходные коды клиентской части приложения.

3.5 Развертывание программного средства

После выявления, а также завершения проектирования всех компонентов программного средства появляется вопрос о планировании развертывания всей системы. Необходимо составить описание требуемых аппаратно-программных комплексов, которые понадобятся для работы приложения. Для этих целей и составляется диаграмма развертывания.

Данная диаграмма представлена на рисунке 3.5. Она отражает следующие особенности развертывания:

– На узле конечного устройства в качестве среды выполнения перечислен список операционных систем, поддерживаемых клиентским приложением и программой для создания пакетов вопросов.

– Предполагается, что пользовательское конечное устройство значительно удалено от серверов программной системы, доступ осуществляется через сеть Интернет, что и упрощенно показано на диаграмме.

Таким образом, после развертывания программного средства пользователи могут уже использовать клиент приложения для организации и проведения викторин.

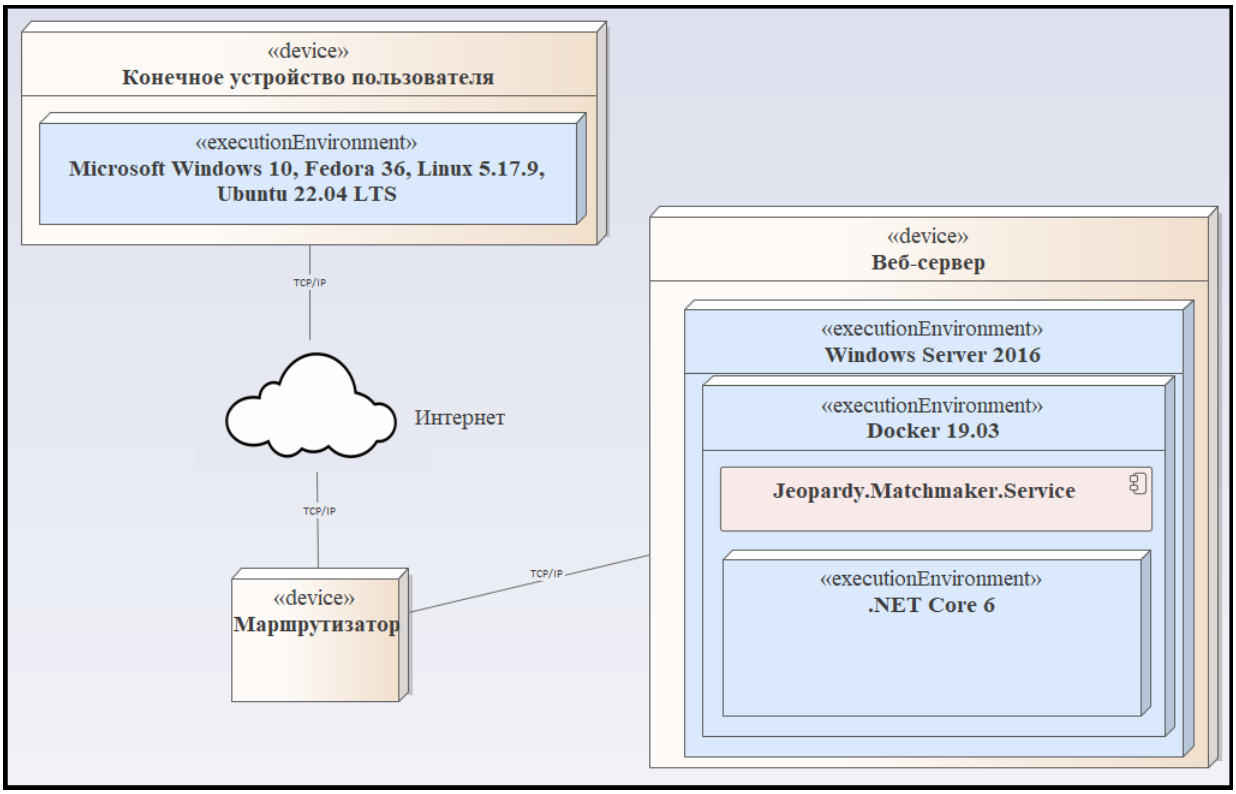

Рисунок 3.5 – Диаграмма развертывания ПС

4 ТЕСТИРОВАНИЕ И ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРОГРАММНОГО СРЕДСТВА

Тестирование программного обеспечения – процесс анализа программного средства и сопутствующей документации с целью выявления дефектов и повышения качества продукта [17]. Вот уже несколько десятков лет его стабильно включают в планы разработки как одна из основных работ, причем выполняемая практически на всех этапах проектов. Важность своевременного выявления дефектов подчеркивается выявленной эмпирически зависимостью между временем допущения ошибки и стоимостью ее исправления: график данной функции круто возрастает.

Тестирование можно классифицировать по очень большому количеству признаков. Основные виды классификации включают следующие [17]:

а) по запуску кода на исполнение:

1) статическое тестирование – без запуска программного средства;

2) динамическое тестирование – с запуском;

б) по степени автоматизации:

1) ручное тестирование – тестовые случаи выполняет человек;

2) автоматизированное тестирование – тестовые случаи частично или полностью выполняет специальное инструментальное средство;

в) по принципам работы с приложением:

1) позитивное тестирование – все действия с приложением выполняются строго в соответствии с требованиями без недопустимых действий или некорректных данных;

2) негативное тестирование – проверяется способность приложения продолжать работу в критических ситуациях недопустимых действий или данных;

В данном разделе проведем динамическое ручное тестирование. В таблице 4.1 приведен список тестовых случаев, относящихся к позитивному тестированию, в таблице 4.2 – к негативному.

Таблица 4.1 – Тестовые случаи позитивного тестирования

Модуль (экран)	Описание тестового случая	Ожидаемые результаты	Тестовый случай пройден?
1	2	3	4
Авторизация	1. **Авторизация**. Предусловие: не иметь авторизации. 1) Ввести имя пользователя. 2) Нажать кнопку «Use credentials».	1) Кнопка использования данных становится активной. 2) Отображается главное меню. Над главным меню отображается выбранный аватар и имя пользователя.	Да
Главное меню	2. **Создание лобби**. Предусловие: необходимо быть авторизованным. 1) В главном меню выбрать пункт «Host game». 2) Задать имя лобби и выбрать набор вопросов для викторины. 3) Нажать кнопку «Create lobby».	1) Отображается страница создания лобби. 2) Кнопка «Create lobby» становится активной. 3) Отображается игровое лобби, где пользователь является ведущим.	Да

Продолжение таблицы 4.1

1	2	3	4
Главное меню	3. **Подключение к лобби без пароля**. Предусловие: необходимо быть авторизованным. 1) В главном меню выбрать пункт «Join game». 2) Выбрать лобби без пароля из списка доступных. 3) Нажать кнопку «Join».	1) Отображается страница подключения к лобби. 2) Кнопка «Join lobby» становится активной. 3) Отображается игровое лобби, где пользователь является игроком.	Да
Главное меню	4. **Подключение к лобби с паролем**. Предусловие: необходимо быть авторизованным. 1) В главном меню выбрать пункт «Join game». 2) Выбрать лобби для подключения. 3) Нажать кнопку «Join». 4) Ввести верный пароль. 5) Нажать кнопку «Continue».	1) Отображается страница подключения к лобби. 2) Кнопка «Join lobby» становиться активной. 3) Отображается окно ввода пароля. 4) Кнопка «Continue» становится активной. 5) Отображается игровое лобби, где пользователь является игроком.	Да

Продолжение таблицы 4.1

1	2	3	4
Главное меню	5. **Смена авторизационных данных**. Предусловие: необходимо быть авторизованным. 1) В главном меню выбрать пункт «Change credentials». 2) Ввести имя пользователя. 3) Нажать кнопку «Use credentials».	1) Отображается страница авторизации. 2) Кнопка использования данных становится активной. 3) Отображается главное меню. Над главным меню отображается выбранный аватар и имя пользователя.	Да
Игровое лобби	6. **Начало игры**. Предусловие: необходимо быть хостом лобби. 1) Ждать подключения всех игроков. 2) Нажать кнопку «Start game».	1) Отображается страница игрового лобби со списком подключенных игроков. 2) Отображается список вопросов первого раунда. 3) Хост получает право выбора игрока для первого хода.	Да
Игровое лобби	7. **Выбор игрока для первого хода**. Предусловие: необходимо начать игру и быть хостом лобби. 1) Нажать на аватар любого из подключенных игроков.	1) Выбранный игрок получает право выбора вопроса.	Да

Продолжение таблицы 4.1

1	2	3	4
Игровое лобби	8. **Выбор вопроса**. Предусловие: иметь право выбора вопроса. 1) Выбрать неразыгранный вопрос из списка вопросов текущего раунда.	1) Отображается содержимое вопроса. 2) У игроков появляется кнопка «Answer question»	Да
Игровое лобби	9. **Ответ на вопрос игроком**. Предусловие: в данный момент отображается содержимое вопроса. 1) Нажать кнопку «Answer question». 2) Дать ответ организатору в устной форме. 3) Дождаться оценки ответа организатором.	1) У организатора появляются кнопки оценки правильности ответа. 2) Изменение игрового баланса в соответствии с правильностью ответа.	Да
Игровое лобби	10. **Положительная оценка правильности ответа организатором**. Предусловие: игрок дает ответ на вопрос. 1) Нажать кнопку «Correct answer».	1) Игроку начисляются очки, равные стоимости вопроса. 2) Изменение игрового баланса в соответствии с правильностью ответа.	Да
Игровое лобби	11. **Отображение победителей**. Предусловие: остался один неразыгранный вопрос. 1) Разыграть последний вопрос.	1) Отображается надпись Quiz is over. 2) Над игроками победителями отображается надпись Winner.	Да

Таблица 4.2 – Тестовые случаи негативного тестирования

Модуль (экран)	Описание тестового случая	Ожидаемые результаты	Тестовый случай пройден?
1	2	3	4
Авторизация	1. **Авторизация без данных**. Предусловие: необходимо находится на странице авторизации. 1) Ничего не вводить в поле имени пользователя.	1) Кнопка использования данных неактивна. 2) Используется аватар по умолчанию.	Да
Создание лобби	2. **Создание лобби с неправильными данными**. Предусловие: необходимо находится на странице создания лобби. 1) Ничего не вводить в поле названия лобби. 2) Не выбирать пакет вопросов.	1) Кнопка создания лобби неактивна. 2) Около поля ввода названия лобби отображается валидационная ошибка "lobby name is empty". 3) В поле пути к пакету вопросов отображется "quiz pack is not set"	Да

Продолжение таблицы 4.2

1	2	3	4
Подключение к лобби	3. **Подключение к лобби с неправильным паролем**. Предусловие: необходимо находится на странице подключения лобби. 1) В главном меню выбрать пункт «Join game». 2) Выбрать лобби для подключения из списка доступных. 3) Нажать кнопку «Join». 4) Ввести неверный пароль. 5) Нажать кнопку «Continue».	1) Отображается страница подключения к лобби. 2) Кнопка «Join lobby» становится активной. 3) Отображается окно для ввода пароля. 4) Кнопка «Continue» становится активной. 5) Отображется валидационная ошибка "password is invalid".	Да

5 МЕТОДИКА ИСПОЛЬЗОВАНИЯ ПРОГРАММНОГО СРЕДСТВА

В данном разделе приведены основные сведения по работе с программным средством.

Приложения данного дипломного проекта (а точнее, его клиентская часть) не требует настройки на конечных устройствах пользователя. Достаточно лишь полной установки, так как оно представляет из себя настольное приложение. Как и было заявлено в требованиях, для корректной работы программного средства требуется операционная система Windows 10, а также пакет следующих системных библиотек:

– .NET Framework 4.8;

– .NET Core 6.

Далее расмотренны основные функции, предоставляемые приложениями.

Приложение-тестер необходимо для создания новых и редактирования существующих пакетов вопросов. Интерфейс приложения имеет следующий вид (рисунок 5.1):

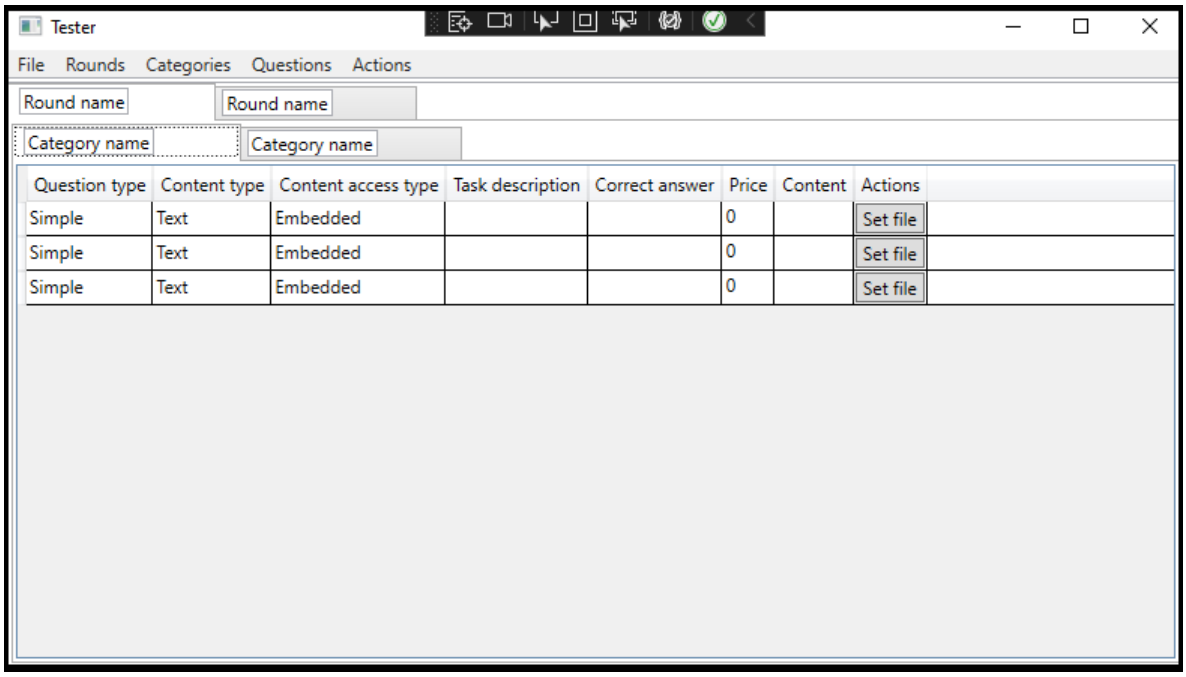

Рисунок 5.1 – Интерфейс приложения-тестера

Для того, чтобы создать новый пакет вопросов необходимо использовать сочетание клавиш CTRL+N,F либо их аналог на панели меню. Для открытия уже существующего пакета можно воспользоваться сочетанием CTRL+O. Для сохранения созданного или редактированного пакета можно

воспользоваться сочетанием CTRL+S.

Для того, чтобы добавить новый раунд в викторину можно воспользоваться сочетанием клавиш CTRL+N,R. Список раундов отображается сверху, каждый раунд обладает своим именем. Рекомендуется давать осмысленные имена, хотя ограничений на это нет. Для удаления уже добавленного раунда можно воспользоваться сочетанием CTRL+D,R. Следует отметить, что вместе с раундом будут удалены все категории, связанные с ним.

Для добавления новой категории в раунд можно воспользоваться сочетанием клавиш CTRL+N,C. Аналогично с раундами каждая категория имеет свое имя, рекомендуется называть категории в соответствии с тематикой вопросов, находящихся в ней. Для удаления категории можно воспользоваться сочетанием CTRL+D,C. Следует отметить, что вместе с категорией будут удалены все вопросы, относящиеся к удаляемой категории, поэтому следует соблюдать осторожность.

Для добавления нового вопроса можно использовать сочетание клавиш CTRL+N,Q, а для удаления CTRL+D,R соответственно. Редактирование вопросов происходит непосредственно в результате редактирования соответствующей строки в таблице.

При сохранении пакета запускаются валидации введенных данных, в случае возникновения ошибок информацию можно получить в соответствующем окне (рисунок 5.2). Все пакеты с вопросами сохраняются на диск в формате .qpck. В дальнейшем эти файлы можно использовать в клиенте при создании викторин.

Рисунок 5.2 – Пример валидационных ошибок тестера

Приложение-клиент является основным для использования и служит для непосредственно проведения многопользовательских викторин. Экран авторизации выглядит следующим образом (рисунок 5.3).

Здесь можно задать идентичность, по которой знакомые смогут опознать игрока. Из настраиваемых параметров имя пользователя и аватар, при этом если пользователь по какой-либо причине не имеет желания устанавливать в качестве аватара собственное изображение, то будет использоваться изображение по умолчанию.

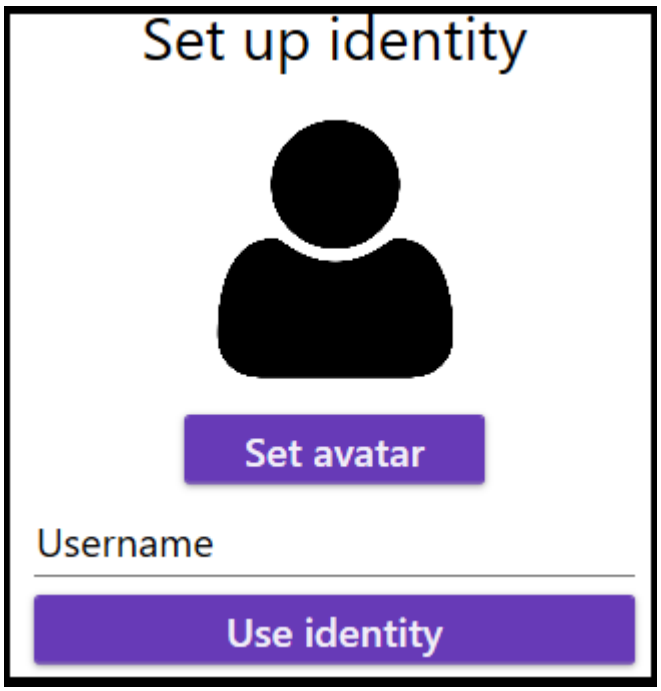

Рисунок 5.3 – Окно авторизации

Главное меню (рисунок 5.4) служит для навигации между другими окнами. Здесь пользователь может создать новое лобби, присоединится к существующему, поменять авторизационные данные, а также выйти из приложения.

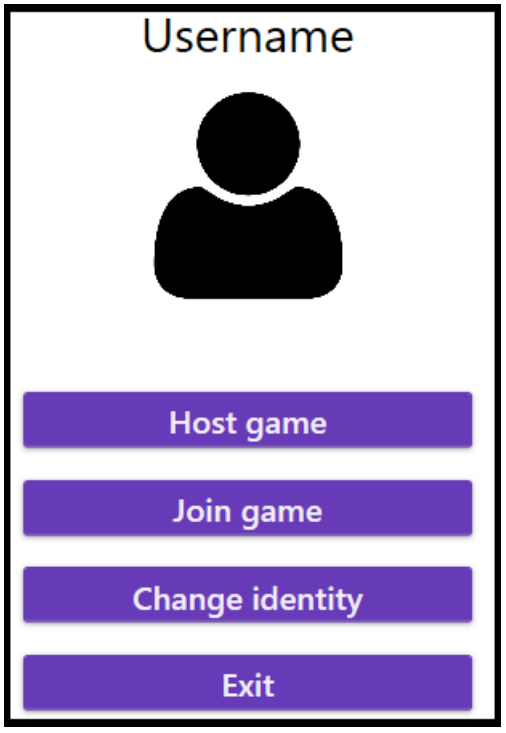

Рисунок 5.4 – Главное меню приложения

На экране создания лобби (рисунок 5.5) пользователь может стать ор-

ганизатором собственной викторины. Для этого достаточно выбрать набор вопросов из уже существующих, либо заранее создать его самостоятельно, а также задать имя лобби и, по желанию, пароль. Если пользователь попал сюда по ошибке то можно вернуться в меню с помощью кнопки «Back to menu».

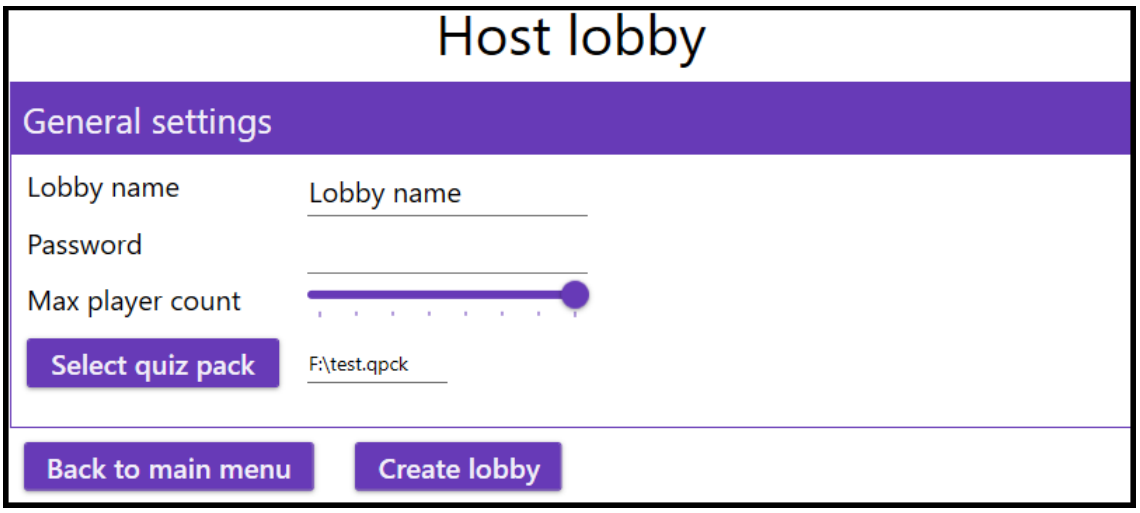

Рисунок 5.5 – Окно создания лобби

На экране присоединения к лобби (рисунок 5.6) пользователь может присоединиться к уже созданным кем-то лобби. Для этого достаточно выбрать лобби из списка и нажать «Join»/ если в лобби есть пароль, то его необходимо правильно ввести.

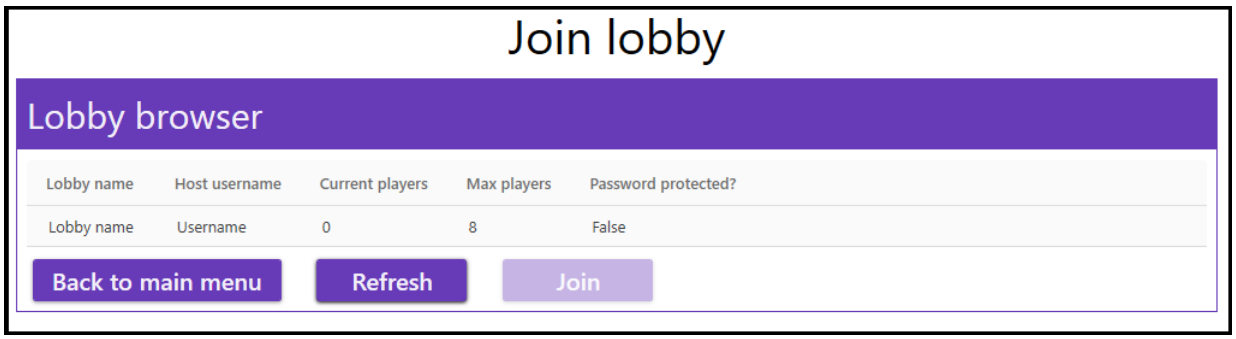

Рисунок 5.6 – Окно присоединения к лобби

На экране игрового лобби (рисунок 5.7) непосредственно просходит игровой процесс. Сюда можно попасть в качестве организатора создав лобби либо в качестве игрока присоединившись к нему.

Организатор следит за игровым процессом и контролирует правильность ответов. Игроки отвечают на вопросы и зарабатывают очки. Победитель определяется по исчерпанию всех вопросов на основе наибольшего количества очков.

Рисунок 5.7 – Окно игрового лобби

6 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО СРЕДСТВА ПРОВЕДЕНИЯ МНОГОПОЛЬЗОВАТЕЛЬСКИХ ВИКТОРИН

6.1 Характеристика программного средства

Разработка программ связана со значительными затратами ресурсов и всевозможными рисками. Перед разработкой проводится соответствующее технико-экономическое обоснование, позволяющее предварительно оценить затраты и выгоду от инвестиций в разработку программного средства. Это обоснование и описывается в данном разделе.

6.2 Описание функций, назначения и потенциальных пользователей ПО

Разработанное программное обеспечение служит для автоматизации проведения викторин на произвольные темы в игровой форме. Одним из видов использования является рекреационный, то есть получение удовольствия от игрового процесса. В то же время некоторые пользователи могут использовать формат викторин для повышения собственной эрудиции по наиболее интересным темам. Также допустим вариант проверки знаний по уже изученным темам в незамысловатой форме. Программное средство берет на себя большую часть организации хода проведения викторины и оставляет за ведущим лишь возможность принятия ключевых решений, что значительно ускоряет сам процесс игры и снижает нагрузку на организатора, что особенно заметно, если присутствует много игроков. Среди главных функций можно выделить несколько видов вопросов: стандартный, секрет, со ставкой и так далее, несколько форматов вопросов: текст, звук, изображение, видео, а также поддержку множества пользователей с работой по сети Интернет. Посчет результатов викторины и выявление победителя выполняется автоматически.

Основными потенциальными покупателями данного программного средства являются люди, желающие весело и пользой провести время с друзьями, причем местоположение друзей не имеет значения, так как работа приложения происходит по сети. Вторичными потенциальными покупателями являются учебные заведения, которые могу проводить образовательные мероприятия для детей или студентов в игровой форме. После анализа рынка было выяснено, что данный формат досуга пользуется достаточно боль-

шим спросом и существующие аналоги не способны удовлетворить его в полном объеме: в час пик качество услуг аналогов значительно снижается, вплоть до невозможности использования. Отчасти это обусловлено текущим переходом на удаленные виды работы и как следствие уменьшением количества встреч вживую. Поэтому данное программное средство может уверенно занять часть рынка развлечений. Сильной стороной данного формата развлечений является то, что он подходит для людей всех возрастов, что значительно повышает охват целевой аудитории.

В результате использования разработанного программного обеспечения пользователи удовлетворят свои рекреационные потребности, повысят свой уровень знаний, а также определят, кто в их окружении самый эрудированный.

6.3 Расчет затрат на разработку ПО

Расчет затрат на разработку программного обеспечения включает в себя:

- затраты на основную заработную плату разработчиков;
- затраты на дополнительную заработную плату разработчиков;
- отчисления на социальные нужды;
- прочие затраты.

Исходные данные сотрудников, которые будут использоваться при расчете сметы затрат, представлены в таблице 6.1. Для расчетов используются коэффициенты из таблицы 6.2

Для определения величины основной заработной платы участников команды сначала необходимо найти часовую заработную плату для каждого специалиста. Учитывая тот факт, что время работы и заработная плата между всеми специалистами одного рода распределены одинаково, то для каждого специалиста в своей области часовую ставку можно высчитать один раз. Формула и пример подсчета часовой ставки архитектора

$$З_{ч.i} = \frac{З_{м.i}}{168} = \frac{11500}{168} = 68{,}45, \qquad (6.1)$$

где $З_{м.i}$ — величина ежемесячной заработной платы специалиста в конкретной области;

168 — количество часов в одном рабочем месяце.

Трудоемкость работ позволяет оценить общее число часов, необходимое специалистам для полного выполнения работы. В случае, если специ-

Таблица 6.1 – Исходные данные сотрудников

Наименование показателя	Единицы измерения	Значение
Общее количество часов работы каждого бизнес-аналитика	ч.	240
Общее количество часов работы каждого системного архитектора	ч.	160
Общее количество часов работы каждого инженера-программиста	ч.	250
Общее количество часов работы каждого программиста	ч.	300
Общее количество часов работы каждого специалиста по тестированию программного обеспечения	ч.	350
Общее количество часов работы каждого дизайнера	ч.	160
Ежемесячная заработная плата бизнес-аналитиков	руб.	3250
Ежемесячная заработная плата системных архитекторов	руб.	11500
Ежемесячная заработная плата инженеров-программистов	руб.	6000
Ежемесячная заработная плата программистов	руб.	3500
Ежемесячная заработная плата специалистов по тестированию программного обеспечения	руб.	2350
Ежемесячная заработная плата дизайнеров	руб.	2500
Число сотрудников на должности бизнес-аналитика	чел.	1
Число сотрудников на должности системного архитектора	чел.	1
Число сотрудников на должности инженера-программиста	чел.	1
Число сотрудников на должности программиста	чел.	2
Число сотрудников на должности специалиста по тестированию программного обеспечения	чел.	3
Число сотрудников на должности дизайнера	чел.	1

Таблица 6.2 – Коэффициенты для вычислений

Наименование показателя	Условное обозначение	Значение
Коэффициент, учитывающий процент премий	$К_{пр}$	30%
Норматив дополнительной заработной платы	$Н_{д}$	15%
Норматив отчислений от фонда оплаты труда	$Н_{соц}$	34.6%
Норматив прочих затрат	$Н_{пз}$	25%
Норматив налога на добавленную стоимость	$Н_{дс}$	20%
Налог на прибыль для IT компаний	$Н_{п}$	13%
Цена за копию программного средства	Ц	15
Предполагаемое количество проданных копий	N	15000

алистов в отделе работает несколько, то трудоемкость является суммой затрат времени каждого специалиста по отдельности. Формула подсчета трудоемкости и работ и пример подсчета для программиста

$$T_i = \sum_{i=1}^{n} t_i = \sum_{i=1}^{2} 300 = 600, \quad (6.2)$$

где t_i — общее количество часов работы одного сотрудника конкретной специальности;

n — количество сотрудников выбранной специальности.

Сумму основной заработной платы без учета премий можно найти умножив часовую ставку сотрудника отдела на общий объем требуемой работы отдела. Эта величина показывает, сколько денег уйдет на выплату заработной платы всему отделу при условии, что премии еще не выплачивались. Пример расчета приведен для программистов. Для расчета суммы заработной платы всех сотрудников одной специальности без учета премии используется формула

$$З_i = T_i \cdot З_{ч.i} = 600 \cdot 20{,}83 = 12\,500{,}00, \quad (6.3)$$

где t_i — общее количество часов работы одного сотрудника конкретной специальности;

n — количество сотрудников выбранной специальности.

Сумма основной заработной платы без учета премии показывает сколько нужно выплатить всем отделам разработки. Расчет и формула имеют сле-

дующий вид

$$З_{общ} = \sum_{i=1}^{n} З_i = 4642{,}86 + 10\,952{,}38 + \ldots + 2380{,}95 = 54\,092{,}26, \qquad (6.4)$$

где $З_i$ — сумма основной заработной платы без учета премии конкретной специальности;

n — количество различных специальностей в команде разработчиков.

Премиальные отчисления необходимы для стимулирования сотрудников и повышения их заинтересованности в выполнении работы. Премиальная ставка показывает процент от первичной заработной платы, который выплачивается в качестве премии за хорошую работы. Величину премий для каждого отдела можно найти по формуле

$$Р_п = \frac{З_{общ} \cdot К_{пр}}{100\%} = \frac{54\,092{,}26 \cdot 30\%}{100\%} = 16\,227{,}68, \qquad (6.5)$$

где $З_{общ}$ — общая сумма выплат основной заработной платы без учета премии;

$К_{пр}$ — коэффициент, учитывающий размер премии.

Итоговая основная заработная плата содержит информацию о затратах на поддержку работы специалистов с учетом премии. Расчет проводится по формуле

$$З_о = З_{общ} + Р_п = 54\,092{,}26 + 16\,227{,}68 = 70\,319{,}94, \qquad (6.6)$$

где $З_{общ}$ — общая сумма выплат основной заработной платы без учета премии;

$Р_п$ — общая сумма премий.

Результаты расчетов основной заработной платы приведены в таблице 6.3.

В список расходов на разработку также входят затраты на дополнительную заработную плату разработиков. Сюда входят выплаты, предусмотренные законодательством о труде, расчеты производятся по формуле

$$З_д = \frac{З_о \cdot Н_д}{100\%} = \frac{70\,319{,}94 \cdot 15\%}{100\%} = 10\,547{,}99, \qquad (6.7)$$

Таблица 6.3 – Расчет затрат на основную заработную плату разработчиков

Наименование должности разработчика	Виды выполняемой работы	Месячная заработная плата, р.	Часовая заработная плата р.	Трудоемкость работ, ч.	Сумма, р.
Бизнес-аналитик	Сбор и анализ требований, изучение пожеланий пользователей	3250	19,35	240	4642,86
Системный архитектор	Проектирование архитектуры ПС	11500	68,45	160	10 952,38
Инженер-программист	Разработка важнейших компонентов ПС	6000	35,71	250	8928,57
Программист	Разработка основных компонентов ПС	3500	20,83	600	12 500,00
Специалист по тестированию программного обеспечения	Тестирование ПС	2350	13,99	1050	14 687,50
Дизайнер	Разработка пользовательского интерфейса	2500	14,88	160	2380,95
Итого					54 092,26
Премия (30%)					16 227,68
Всего основная заработная плата					70 319,94

где $З_о$ — сумма основной заработной платы с учетом премии;
$К_{пр}$ — норматив дополнительной заработной платы.

Нормы отсчислений на социальные нужны (в фонд социальной защиты населения и на обязательное страхование) устанавливаются государством и любой бизнес обязан их выплачивать. В соответствии с действующими законодательными актами.

$$Н_{соц} = \frac{(З_о + З_д) \cdot Н_{соц}}{100\%} = \frac{(70\,319{,}94 + 10\,547{,}99) \cdot 34.6\%}{100\%} = 27\,980{,}30, \quad (6.8)$$

где $З_о$ — сумма основной заработной платы с учетом премии;
$З_д$ — сумма дополнительной заработной платы;
$Н_{соц}$ — норматив отчислений от фонда оплаты труда.

Прочие затраты, включающие в себя лицензии на программное обеспечени, используемое в разработке можно рассчитать по формуле

$$Р_{пз} = \frac{З_о \cdot Н_{пз}}{100\%} = \frac{70\,319{,}94 \cdot 25\%}{100\%} = 17\,579{,}99, \quad (6.9)$$

где $З_о$ — сумма основной заработной платы с учетом премии;
$Н_{пз}$ — норматив прочих затрат.

Результаты подсчетов сведены в таблицу 6.4.

Таблица 6.4 – Затраты на разработку программного обеспечения

Наименование статьи затрат	Значение, руб.
Основная заработная плата разработчиков	70 319,94
Дополнительная заработная плата разработчиков	10 547,99
Отчисления на социальные нужды	27 980,30
Прочие затраты	17 579,99
Общая сумма инвестиций в разработку	126 428,22

6.4 Оценка результата от продажи ПО

Цена за копию продукта была выбрана на основе статистики по средним ценам игр категории "инди"(малобюджетные), собранной на самой популярной платформе для дистрибуции видеоигр Steam. Средняя цена в эквиваленте на белорусские рубли составляет около 15 рублей [18]. При этом количество проданных копий в сегменте "инди"обычно варьируется от 5-10 тысяч до сотни тысяч, а в особо успешных случаях количество копий может достигать полумиллиона и больше. Для расчетов возьмем среднее значение проданных копий, равное 15000[19].

Для расчета прибыли, которую получит разработчик программного обеспечения, сначала необходимо рассчитать налог на добавленную стоимость по формуле

$$НДС = \frac{(Ц \cdot N \cdot Н_{дс})}{100\% + Н_{дс}} = \frac{(15 \cdot 15000 \cdot 20\%)}{100\% + 20\%} = 37\,500{,}00, \qquad (6.10)$$

где Ц — цена за одну лицензионную копию программного средства;
N — количество проданных копий;
$Н_{дс}$ — коэффициент подоходного налога (в Беларуси равен 20%).

Имея значение налога на добавленную стоимость можно рассчитать прибыль, полученную разработчиком от реализации программного обеспечения без учета налога на прибыль

$$П = Ц \cdot N - НДС - З_р = 15 \cdot 15000 - 37\,500{,}00 - 126\,428{,}22 = 61\,071{,}78, \quad (6.11)$$

где $З_р$ — затраты на разработку программного обеспечения;
Ц — цена за одну лицензионную копию программного средства;
N — количество проданных копий;
$Н_{дс}$ — коэффициент подоходного налога (в Беларуси равен 20%).

Для IT-компаний в Беларуси введены налоговые льготы на прибыль, поэтому при расчете чистой прибыли используется льготная налоговая ставка равная 13% [20]

$$П_ч = П \cdot (1 - \frac{Н_п\%}{100\%}) = 61\,071{,}78 \cdot (1 - \frac{13\%}{100\%}) = 53\,132{,}45, \qquad (6.12)$$

где П — прибыль без учета налога на прибыль;
$Н_п$ — налоговая ставка на прибыль.

Рентабельность показывает насколько выгодны ивестиции в разработку программного обеспечения, для ее подсчета используется формула

$$У_р = \frac{П_ч}{З_р} \cdot 100\% = \frac{53\,132{,}45}{126\,428{,}22} \cdot 100\% = 42{,}03\%, \qquad (6.13)$$

где $П_ч$ — прибыль с учетом налога на прибыль;
$З_р$ — затраты на разработку программного обеспечения.

Учитывая тот факт, что средняя ставка по депозиту равна 19% [21], то можно сделать вывод, что инвестиции в разработку программного средства являются более выгодными, по сравнению с депозитным вкладом, а полученные результаты расчетов свидетельствуют об эффективности разработки проектируемого программного средства.

ЗАКЛЮЧЕНИЕ

Предметной областью данного дипломного проекта является процесс проведения многопользовательских викторин для участников: игроков и организатора. Был проведен поиск существующих программные средств этого рода, по его результатам был сделан вывод о существовании частичных аналогов. Были выявлены недостатки существующих решений. Было предложено программное средство, которое должно усовершенствовать процесс проведения многопользовательских викторин.

На основании проведенного анализа предметной области были выдвинуты требования к программному средству. В качестве технологий разработки были выбраны наиболее современные существующие на данный момент средства, широко применяемые в индустрии. Спроектированное программное средство было успешно протестировано на соответствие спецификации функциональных требований. Исходя из анализа предметной области, а также фактов наличия множества недостатков в аналогах позволили сделать вывод о целесообразности проектирования и разработки программной системы. Результаты, полученные в ходе выполнения технико-экономического обоснования только подтвердили данный вывод.

Разработано программное средство проведения многопользовательских викторин, целевой платформой которого является настольные компьютеры и которое поддерживает следующие функции:

– создание и управление пакетами с вопросами;

– разделение пользователей на игроков и организаторов с поддержкой анонимной авторизации.

– создание, изменение, удаление игровых лобби;

– использование сервиса-координатора для установки соединения между клиентами;

– несколько видов вопросов повышающих уровень удовлетворения игровым процессом;

– система контроля ведущего за игровым процессом;

– автоматическое определение победителя в конце игры.

Для разработки программы по созданию пакетов с вопросами был использован WPF, а для сохранения результатов на диск использовалась бинарная сериализация. В качестве технологии для разработки сервиса-координатора использовался .NET Core, в свою очередь координатор работает через Docker. Для разработки клиента был использован WPF, для общения по сети между клиентами использовался протокол TCP/IP.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Макконел, С. Совершенный код. Мастер-класс / С. Макконел. — Издательско-торговый дом «Русская редакция», 2010. — 896 Р.

[2] Как правильно переходить границу: кроссплатформенность в мобильном приложении [Электронный ресурс]. — Электронные данные. — Режим доступа: https://habrahabr.ru/company/parallels/blog/254325/. — Дата доступа: 25.03.22.

[3] Richter, Jeffrey. CLR via C Sharp / Jeffrey Richter. — Издательство «Питер», 2021.

[4] Sheriff, Paul D. Designing for Web or Desktop? [Электронный ресурс]. — Электронные данные. — Режим доступа: https://msdn.microsoft.com/en-us/library/ms973831.aspx. — Дата доступа: 25.03.22.

[5] Shaaban, Sam. Web-Based vs “Desktop” Software [Электронный ресурс]. — Электронные данные. — Режим доступа: https://nurelm.com/web-based-vs-desktop-software/. — Дата доступа: 25.03.22.

[6] Bychkov, Dmitriy. Desktop vs. Web Applications: A Deeper Look and Comparison [Электронный ресурс]. — Электронные данные. — Режим доступа: http://www.seguetech.com/desktop-vs-web-applications/. — Дата доступа: 25.03.22.

[7] Summerfield, Jason. Mobile Website vs. Mobile App: Which is Best for Your Organization? [Электронный ресурс]. — Электронные данные. — Режим доступа: https://www.hswsolutions.com/services/mobile-web-development/mobile-website-vs-apps/. — Дата доступа: 25.03.22.

[8] Garlan, David. An Introduction to Software Architecture / David Garlan, Mary Shaw / Ed. by V Ambriola, G Tortora. — New Jersey : World Scientific Publishing Company, 1994. — Vol. I. — Режим доступа: http://www.cs.cmu.edu/afs/cs/project/able/ftp/intro_softarch/intro_softarch.pdf. — Дата доступа: 25.03.22.

[9] Team, Microsoft Patterns & Practices. Microsoft® Application Architecture Guide (Patterns & Practices) / Microsoft Patterns & Practices Team. — Microsoft Press, 2009.

[10] Marinescu, Floyd. Domain-Driven Design Quickly / Floyd Marinescu, Abel Avram. — LULU PR, 2007.

[11] Software Cost Estimation with Cocomo II / Barry W. Boehm [et al.]. — Prentice Hall, 2000.

[12] Walston, Claude E. A Method of Programming Measurement and

Estimation. / Claude E. Walston, Charles P. Felix // IBM Systems Journal. — 1977. — Vol. 16, no. 1. — Pp. 54–73.

[13] Введение в язык C# и .NET Framework [Электронный ресурс]. — Электронные данные. — Режим доступа: https://msdn.microsoft.com/ru-ru/library/z1zx9t92.aspx. — Дата доступа: 25.03.22.

[14] Pattern matching overview [Электронный ресурс]. — Электронные данные. — Режим доступа: https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/functional/pattern-matching. — Дата доступа: 25.03.22.

[15] Weil, Arnaud. Learn WPF MVVM - XAML, C# and the MVVM Pattern / Arnaud Weil. — Prentice Hall, 2016.

[16] Serialization (C#) [Электронный ресурс]. — Электронные данные. — Режим доступа: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/serialization/. — Дата доступа: 25.03.22.

[17] Различные виды тестирования ПО [Электронный ресурс]. — Электронные данные. — Режим доступа: https://www.atlassian.com/ru/continuous-delivery/software-testing/types-of-software-testing. — Дата доступа: 25.03.22.

[18] Steam Spy [Электронный ресурс]. — Электронные данные. — Режим доступа: https://steamspy.com. — Дата доступа: 25.03.22.

[19] Infographic: Indie game revenues on Steam [Электронный ресурс]. — Электронные данные. — Режим доступа: https://vginsights.com/insights/article/infographic-indie-game-revenues-on-steam. — Дата доступа: 25.03.22.

[20] О развитии цифровой экономики [Электронный ресурс]. — Электронные данные. — Режим доступа: https://president.gov.by/ru/documents/dekret-8-ot-21-dekabrja-2017-g-17716. — Дата доступа: 25.03.22.

[21] Процентные ставки по вкладам [Электронный ресурс]. — Электронные данные. — Режим доступа: https://belarusbank.by/site_ru/25271/bels-01-04-22.pdf. — Дата доступа: 25.03.22.

[22] О начисленной средней заработной плате работников в феврале 2017 г. [Электронный ресурс]. — Электронные данные. — Режим доступа: http://www.belstat.gov.by/ofitsialnaya-statistika/solialnaya-sfera/trud/operativnaya-informatsiya_8/o-nachislennoi-srednei-zarabotnoi-plate-rabotnikov/o-nachislennoy-sredney-zarabotnoy-plate-rabotnikov-v-fevrale-2017-nbsp-g/. — Дата доступа: 25.03.22.

ПРИЛОЖЕНИЕ А
(обязательное)
Фрагменты исходного кода

```
using ProtoBuf;

namespace Jeopardy.Core.Data.Users
{
    [ProtoContract]
    [ProtoInclude(1, typeof(NetworkIdentity))]
    public class UserIdentity
    {
        [ProtoMember(2)]
        public string NetworkUserId { get; set; } = string.Empty;
        [ProtoMember(3)]
        public string Username { get; set; } = string.Empty;
        [ProtoMember(4)]
        public byte[] Avatar { get; set; } = Array.Empty<byte>();

        protected UserIdentity() { }

        public UserIdentity(string username, byte[]? avatar)
        {
            NetworkUserId = Guid.NewGuid().ToString();
            Username = username;
            Avatar = avatar ?? Array.Empty<byte>();
        }
    }
}

using Jeopardy.Core.Data.Quiz;
using Jeopardy.Core.Data.Quiz.Constants;
using Jeopardy.Core.Data.Utility;
using Jeopardy.Core.Data.Validation;
using Jeopardy.Core.Serialization;
using Jeopardy.Core.Wpf.Commands;
using Jeopardy.Core.Wpf.Viewmodels;
using Microsoft.Win32;
using System.Collections.ObjectModel;
using System.Windows.Input;

namespace Jeopardy.Desktop.Tester.App.Viewmodels
{
    internal class MainWindowViewmodel : ViewmodelBase
    {
```

```
public Quiz Quiz { get; set; } = new(new
   ObservableCollection<QuizRound>());
public ValidationResult ValidationResult { get; set; } =
   new();

public ICommand AddRoundCommand => new RelayCommand(
    () => Quiz.Rounds.Add(new(new ObservableCollection<
       QuizCategory>())),
    null
);

public ICommand AddCategoryCommand => new RelayCommand(
    () => Quiz.Rounds[SelectedRound].Categories.Add(new(
       new ObservableCollection<Question>())),
    IsRoundSelected
);

public ICommand AddQuestionCommand => new RelayCommand(
    () => Quiz.Rounds[SelectedRound].Categories[
       SelectedCategory].Questions.Add(new Question()),
    IsCategorySelected
);

public ICommand RemoveRoundCommand => new RelayCommand(
    () => Quiz.Rounds.RemoveAt(SelectedRound),
    IsRoundSelected
);

public ICommand RemoveCategoryCommand => new RelayCommand
   (
    () => Quiz.Rounds[SelectedRound].Categories.RemoveAt(
       SelectedCategory),
    IsCategorySelected
);

public ICommand RemoveQuestionCommand => new RelayCommand
   (
    () => Quiz.Rounds[SelectedRound].Categories[
       SelectedCategory].Questions.RemoveAt(
       SelectedQuestion),
    IsQuestionSelected
);

public ICommand SetContentCommand => new RelayCommand(
    () =>
    {
```

```
            OpenFileDialog openFileDialog = new();
            Question? question = Quiz.Rounds[SelectedRound].
               Categories[SelectedCategory].Questions[
               SelectedQuestion];
            ContentType contentType = question.ContentType;

            switch (contentType)
            {
                case ContentType.Image:
                    openFileDialog.Filter = "Image files (*.
                       png;*.jpg)|*.png;*.jpg";
                    break;
                case ContentType.Sound:
                    openFileDialog.Filter = "Sound files (*.
                       mp3;*.flac)|*.mp3;*.flac";
                    break;
                case ContentType.Video:
                    openFileDialog.Filter = "Video files (*.
                       mp4;*.avi)|*.mp4;*.avi";
                    break;
                default:
                    return;
            }

            if (openFileDialog.ShowDialog() == true)
            {

                question.ContentPath = openFileDialog.
                   FileName;
                question.ContentAccessType = Core.Data.Quiz.
                   Constants.ContentAccessType.Embedded;
            }
        },
        null //selected question is populated before
           invocation so its always good
    );

    public ICommand RunValidationsCommand => new RelayCommand
       (
        () =>
        {
            ValidationResult = QuizPacker.Validate(Quiz);
            OnPropertyChanged(nameof(ValidationResult));
        },
        null
    );
```

```
public ICommand NewQuizCommand => new RelayCommand(
    () =>
    {
        ValidationResult.FieldValidationResults.Clear();
        OnPropertyChanged(nameof(ValidationResult));
        Quiz = new(new ObservableCollection<QuizRound>())
            ;
        OnPropertyChanged(nameof(Quiz));
    },
    null
);

public ICommand SaveQuizCommand => new RelayCommand(
    () =>
    {
        SaveFileDialog dlg = new();
        dlg.Filter = "Question pack (*.qpck)|*.qpck";
        dlg.DefaultExt = "qpck";
        dlg.AddExtension = true;
        if (dlg.ShowDialog() == true)
        {
            ValidationResult = QuizPacker.Pack(Quiz);
            if (!ValidationResult.HasErrors())
            {
                BinarySerializer.SerializeToFile(Quiz,
                    dlg.FileName);
            }
        }
    },
    null
);

public ICommand OpenQuizCommand => new RelayCommand(
    () =>
    {
        OpenFileDialog openFileDialog = new();
        openFileDialog.Filter = "Question pack (*.qpck)
            |*.qpck";
        if (openFileDialog.ShowDialog() == true)
        {
            Quiz = BinarySerializer.DeserializeFromFile<
                Quiz>(openFileDialog.FileName);
            QuizPacker.Unpack(Quiz);
            OnPropertyChanged(nameof(Quiz));
        }
```

```
            },
            null
        );

        public int SelectedRound { get; set; } = -1;
        public int SelectedCategory { get; set; } = -1;
        public int SelectedQuestion { get; set; } = -1;

        private bool IsRoundSelected() => SelectedRound >= 0 &&
                    SelectedRound < Quiz.Rounds.Count;

        private bool IsCategorySelected() => IsRoundSelected() &&
                    SelectedCategory >= 0 &&
                    SelectedCategory < Quiz.Rounds[SelectedRound].
                        Categories.Count;

        private bool IsQuestionSelected() => IsCategorySelected()
            &&
                    SelectedQuestion >= 0 &&
                    SelectedQuestion < Quiz.Rounds[SelectedRound].
                        Categories[SelectedCategory].Questions.
                        Count;
    }
}

//QuizRound.cs

using ProtoBuf;

namespace Jeopardy.Core.Data.Quiz
{
    [ProtoContract]
    public class QuizRound
    {
        [ProtoMember(1)]
        public IList<QuizCategory> Categories { get; set; } = new
            List<QuizCategory>();
        [ProtoMember(2)]
        public string Name { get; set; } = "Round name";
        public bool HasUnplayedCategories => Categories.Any(c =>
            c.HasUnplayedQuestions);

        public QuizRound() => Categories = new List<QuizCategory
            >();

        public QuizRound(IList<QuizCategory> quizCategories) =>
```

```
            Categories = quizCategories;
        }
    }

    //QuizCategory.cs

    using ProtoBuf;

    namespace Jeopardy.Core.Data.Quiz
    {
        [ProtoContract]
        public class QuizCategory
        {
            [ProtoMember(1)]
            public IList<Question> Questions { get; set; } = new List
               <Question>();
            [ProtoMember(2)]
            public string Name { get; set; } = "Category name";
            public bool HasUnplayedQuestions => Questions.Any(q => q.
               Unplayed);

            public QuizCategory() => Questions = new List<Question>()
               ;

            public QuizCategory(IList<Question> quizQuestions) =>
               Questions = quizQuestions;
        }
    }

    //Quiz.cs

    using ProtoBuf;

    namespace Jeopardy.Core.Data.Quiz
    {
        [ProtoContract]
        public class Quiz
        {
            [ProtoMember(1)]
            public IList<QuizRound> Rounds { get; set; } = new List<
               QuizRound>();
            [ProtoMember(2)]
            public DateTime CreatedDate { get; set; } = DateTime.Now;
            public bool HasUnplayedRounds => Rounds.Any(r => r.
               HasUnplayedCategories);

            public Quiz() { }
```

```
        public Quiz(IList<QuizRound> quizRounds) => Rounds =
            quizRounds;
    }
}

//QuestionType.cs

using Jeopardy.Core.Localization.Locales;
using ProtoBuf;
using System.ComponentModel.DataAnnotations;

namespace Jeopardy.Core.Data.Quiz.Constants
{
    [ProtoContract]
    public enum QuestionType
    {
        [Display(Description = "Enum_Undefined", ResourceType =
            typeof(LocaleCommon))]
        Undefined = 0,
        //First to click receives the question
        [Display(Description = "QuestionType_Simple",
            ResourceType = typeof(LocaleCommon))]
        Simple = 1,
        ////Highest bidder receives the question
        //[Display(Description = "QuestionType_Auction",
            ResourceType = typeof(LocaleCommon))]
        //Auction = 2,
        ////Gifted person receives the question
        //[Display(Description = "QuestionType_Cat", ResourceType
            = typeof(LocaleCommon))]
        //Secret = 3,
        ////Double the prize and no point loss on wrong answer
        //[Display(Description = "QuestionType_Sponsored",
            ResourceType = typeof(LocaleCommon))]
        //Sponsored = 4
    };
}

//Question.cs

using Jeopardy.Core.Data.Quiz.Constants;
using ProtoBuf;

namespace Jeopardy.Core.Data.Quiz
{
    [ProtoContract]
```

```
    public class Question
    {
        [ProtoMember(1)]
        public byte[] RawContent { get; set; } = Array.Empty<byte
            >();
        [ProtoMember(2)]
        public QuestionType QuestionType { get; set; } =
            QuestionType.Simple;
        [ProtoMember(3)]
        public ContentType ContentType { get; set; } =
            ContentType.Text;
        [ProtoMember(4)]
        public ContentAccessType ContentAccessType { get; set; }
            = ContentAccessType.Embedded;
        [ProtoMember(5)]
        public int Price { get; set; }
        [ProtoMember(6)]
        public string CorrectAnswer { get; set; } = string.Empty;
        [ProtoMember(7)]
        public string TaskDescription { get; set; } = string.
            Empty;
        [ProtoMember(8)]
        public string ContentPath { get; set; } = string.Empty;
        [ProtoMember(9)]
        public bool Unplayed { get; set; } = true;
    }
}

//Player.cs

using Jeopardy.Core.Data.Users;
using ProtoBuf;

namespace Jeopardy.Core.Data.Gameplay
{
    [ProtoContract]
    public class Player
    {
        public string Username => NetworkIdentity.Username;
        public string NetworkUserId => NetworkIdentity.
            NetworkUserId;

        [ProtoMember(1)]
        public NetworkIdentity NetworkIdentity { get; set; } =
            new();
        [ProtoMember(2)]
```

```
        public int Score { get; set; }
        [ProtoMember(3)]
        public bool HasAnswerAttempt { get; set; } = true;
        [ProtoMember(4)]
        public bool IsWinner { get; set; } = false;

        public Player() { }
        public Player(UserIdentity userIdentity) =>
            NetworkIdentity = new(userIdentity);
    }
}

//NetworkIdentity.cs

using ProtoBuf;
using System.Net.Sockets;

namespace Jeopardy.Core.Data.Users
{
    [ProtoContract]
    public class NetworkIdentity : UserIdentity
    {
        public TcpClient? TcpClient { get; set; }

        public NetworkIdentity() : base() { }

        public NetworkIdentity(string username, byte[] avatar) :
            base(username, avatar)
        {
        }

        public NetworkIdentity(UserIdentity userIdentity) : base(
            userIdentity.Username, userIdentity.Avatar)
        {

        }
    }
}

//LobbyInfo.cs

using Jeopardy.Core.Cryptography;
using Jeopardy.Core.Data.Gameplay;
using ProtoBuf;

namespace Jeopardy.Core.Data.Matchmaker
{
```

```
    [ProtoContract]
    public class LobbyInfo
    {
        public static int MaxAllowedPlayerCount => 8;
        public static int MinAllowedPlayerCount => 1;

        [ProtoMember(1)]
        public GameState GameState { get; set; } = new();

        [ProtoMember(3)]
        public string NetworkLobbyId { get; set; } = Guid.Empty.
           ToString();
        [ProtoMember(4)]
        public DateTime LobbyCreationDate { get; set; } =
           DateTime.Now;
        [ProtoMember(5)]
        public string LobbyName { get; set; } = "Lobby name";
        //[ProtoMember(6)]
        public SecurePassword? Password { get; set; }
        [ProtoMember(7)]
        public int MaxPlayerCount { get; set; } =
           MaxAllowedPlayerCount;

        public int CurrentPlayerCount => GameState.Players.Count;
        public string OrganizerName => GameState.Host.
           NetworkIdentity.Username;
        public bool IsPasswordProtected => Password != null;

        public LobbyPreview ToLobbyPreview() => new()
        {
            NetworkLobbyId = NetworkLobbyId,
            LobbyName = LobbyName,
            OrganizerName = OrganizerName,
            MaxPlayerCount = MaxPlayerCount,
            CurrentPlayerCount = CurrentPlayerCount,
            IsPasswordProtected = IsPasswordProtected,
        };
    }
}

//LobbyPreview.cs

using ProtoBuf;

namespace Jeopardy.Core.Data.Matchmaker
{
```

```
    [ProtoContract]
    public class LobbyPreview
    {
        [ProtoMember(1)]
        public string NetworkLobbyId { get; set; } = string.Empty
            ;
        [ProtoMember(2)]
        public string? LobbyName { get; set; }
        [ProtoMember(3)]
        public string? OrganizerName { get; set; }
        [ProtoMember(4)]
        public int MaxPlayerCount { get; set; }
        [ProtoMember(5)]
        public int CurrentPlayerCount { get; set; }
        [ProtoMember(6)]
        public bool IsPasswordProtected { get; set; }
    }
}

//GameState.cs

using Jeopardy.Core.Data.Gameplay.Contexts;
using Jeopardy.Core.Data.Quiz;
using ProtoBuf;

namespace Jeopardy.Core.Data.Gameplay
{
    [ProtoContract]
    public class GameState
    {
        [ProtoMember(1)]
        public Quiz.Quiz Quiz { get; set; } = new();
        [ProtoMember(2)]
        public GameRules GameRules { get; set; } = new();
        [ProtoMember(3)]
        public Dictionary<string, Player> Players { get; set; } =
            new();
        [ProtoMember(4)]
        public Player Host { get; set; } = new Player();

        [ProtoMember(5)]
        public GameContext? GameContext { get; set; }
        [ProtoMember(6)]
        public bool IsPaused { get; set; } = true;
        [ProtoMember(7)]
        public string ControlledNetworkUserId { get; set; } =
```

```
    string.Empty; //0 – host, 1 – first player etc

public string CurrentStateDescription => GameContext
    switch
{
    null => "Waiting for host to start",
    PlayerSelectContext c => $"Waiting for player
        selection from " + (c.SelectorNetworkUserId ==
        Host.NetworkUserId ? $"{Host.Username}" : $"{
        Players[c.SelectorNetworkUserId].Username}"),
    SelectQuestionContext c => $"Waiting for question
        selection from {Players[c.SelectorNetworkUserId].
        Username}",
    SimpleQuestionContext => $"Simple question with
        reward of {CurrentQuestion?.Price}",
    WinnerContext => $"Congratulations to winners!!!",
    //SponsoredQuestionContext => $"Sponsored question
        with reward of {CurrentQuestion?.Price}",
    //SecretQuestionContext => $"Secret question with
        price of {CurrentQuestion?.Price * GameRules.
        SecretQuestionRewardMultiplier}",
    //AuctionQuestionContext => $"Auction started,
        initial bet is {CurrentQuestion?.Price}, max bet
        is {CurrentQuestion?.Price * GameRules.
        StakeQuestionMaxStakeMultiplier}, make your bets",
    PlayerAnswerContext c => $"Player {Players[c.
        AnsweringPlayerId].Username} is answering",
    _ => "Description for this state is not defined"
};

//[ProtoMember(8)]
//public string CurrentPlayerId { get; set; } = string.
    Empty; //0 – host, 1 – first player etc
[ProtoMember(9)]
public QuizRound? CurrentRound { get; set; }
[ProtoMember(10)]
public Question? CurrentQuestion { get; set; }

public void SetNextContext(string answeringPlayerId)
{
    Question? currentQuestion = CurrentQuestion;
    QuizRound? currentRound = CurrentRound;
    if (currentRound is not null)
    {
        if (currentQuestion is null || !Players.Values.
            Any(p => p.HasAnswerAttempt))
```

```
            {
                CurrentQuestion = null;
                if (currentRound.HasUnplayedCategories)
                {
                    GameContext = new SelectQuestionContext(
                        answeringPlayerId);
                }
                else
                {
                    if (!SetNextRound())
                    {
                        var maxScore = Players.Values.Max(p
                            => p.Score);
                        IEnumerable<Player>? winnerPlayers =
                            Players.Values.Where(p => p.Score
                            == maxScore);
                        foreach (Player? player in
                            winnerPlayers)
                        {
                            player.IsWinner = true;
                        }
                        var winners = winnerPlayers.Select(p
                            => p.NetworkUserId).ToList();
                        GameContext = new WinnerContext(
                            winners);
                    }
                }
            }
            else if (GameContext is PlayerAnswerContext ctx)
            {
                GameContext = ctx.QuestionContext;
            }
        }
    }

    private bool SetNextRound()
    {
        QuizRound? newRound = Quiz.Rounds.FirstOrDefault(r =>
            r.HasUnplayedCategories);
        if (newRound is not null)
        {
            CurrentRound = newRound;
            return true;
        }
        CurrentRound = null;
        return false;
```

```
        }
    }
}

//GameRules.cs

using ProtoBuf;

namespace Jeopardy.Core.Data.Gameplay
{
    [ProtoContract]
    public class GameRules
    {
        //public const int MaxSecretQuestionRewardMultiplier =
           10;
        //public const int MinSecretQuestionRewardMultiplier = 1;
        //public const int MaxStakeQuestionMaxStakeMultiplier =
           10;
        //public const int MinStakeQuestionMaxStakeMultiplier =
           1;
        public const int MaxQuestionAnswerTime = 60;
        public const int MinQuestionAnswerTime = 5;
        public const int MaxQuestionHangingTime = 60;
        public const int MinQuestionHangingTime = 5;

        //[ProtoMember(1)]
        //public int SecretQuestionRewardMultiplier { get; set; }
            = 2;
        //[ProtoMember(2)]
        //public int StakeQuestionMaxStakeMultiplier { get; set;
           } = 3;
        [ProtoMember(3)]
        public int QuestionAnswerTime { get; set; } = 30;
        [ProtoMember(4)]
        public int QuestionHangingTime { get; set; } = 60;
    }
}

//ContentType.cs

using Jeopardy.Core.Localization.Locales;
using ProtoBuf;
using System.ComponentModel.DataAnnotations;

namespace Jeopardy.Core.Data.Quiz.Constants
{
    [ProtoContract]
```

```
    public enum ContentType
    {
        [Display(Description = "Enum_Undefined", ResourceType =
           typeof(LocaleCommon))]
        Undefined = 0,
        //Simple text to show
        [Display(Description = "ContentType_Text", ResourceType =
            typeof(LocaleCommon))]
        Text = 1,
        //An image to show
        [Display(Description = "ContentType_Image", ResourceType
           = typeof(LocaleCommon))]
        Image = 2,
        //Sound to playback
        [Display(Description = "ContentType_Sound", ResourceType
           = typeof(LocaleCommon))]
        Sound = 3,
        //Video to playback
        [Display(Description = "ContentType_Video", ResourceType
           = typeof(LocaleCommon))]
        Video = 4
    };
}

//ContentAccessType.cs
using Jeopardy.Core.Localization.Locales;
using ProtoBuf;
using System.ComponentModel.DataAnnotations;
namespace Jeopardy.Core.Data.Quiz.Constants
{
    [ProtoContract]
    public enum ContentAccessType
    {
        [Display(Description = "Enum_Undefined", ResourceType =
           typeof(LocaleCommon))]
        Undefined = 0,
        //Access via embedded binary
        [Display(Description = "ContentAccessType_Embedded",
           ResourceType = typeof(LocaleCommon))]
        Embedded = 1,
        ////Access via link on the internet
        //[Display(Description = "ContentAccessType_Link",
           ResourceType = typeof(LocaleCommon))]
        //Link = 2
    }
}
```

ПРИЛОЖЕНИЕ Б
(обязательное)
Описание команд сетевого общения

```
//StartGameAction.cs

using Jeopardy.Core.Data.Gameplay.Contexts;
using ProtoBuf;

namespace Jeopardy.Core.Data.Gameplay.Actions
{
    [ProtoContract]

    public class StartGameAction : GameAction
    {
        public override void Execute(GameState gameState)
        {
            gameState.CurrentRound = gameState.Quiz.Rounds[0];
            gameState.GameContext = new PlayerSelectContext(
               gameState.Host.NetworkUserId);
            gameState.IsPaused = false;
        }

        public override bool CanExecute(GameState gameState) =>
           gameState.CurrentRound is null && gameState.
           GameContext is null;
    }
}

//SkipQuestionAction.cs

using Jeopardy.Core.Data.Gameplay.Contexts;

namespace Jeopardy.Core.Data.Gameplay.Actions
{
    public class SkipQuestionAction : GameAction
    {
        public override bool CanExecute(GameState gameState) =>
           gameState.GameContext is QuestionContext;
        public override void Execute(GameState gameState)
        {
            if (gameState.GameContext is QuestionContext ctx)
            {
                gameState.CurrentQuestion = null;
                gameState.SetNextContext(ctx.
                   SelectorNetworkUserId);
```

```
            }
        }
    }
}

//RequestAnswerAttemptAction.cs

using Jeopardy.Core.Data.Gameplay.Contexts;
using ProtoBuf;

namespace Jeopardy.Core.Data.Gameplay.Actions
{
    [ProtoContract]

    public class RequestAnswerAttemptAction : GameAction
    {
        [ProtoMember(1)]
        public string RequestorNetworkUserId { get; set; } =
           string.Empty;

        public override bool CanExecute(GameState gameState) =>
           gameState.GameContext is QuestionContext;
        public override void Execute(GameState gameState)
        {
            if (gameState.GameContext is QuestionContext ctx)
            {
                gameState.GameContext = new PlayerAnswerContext {
                    QuestionContext = ctx, AnsweringPlayerId =
                   RequestorNetworkUserId };
                gameState.Players[RequestorNetworkUserId].
                   HasAnswerAttempt = false;
            }
        }
    }
}

//QuestionSelectAction.cs

using Jeopardy.Core.Data.Gameplay.Contexts;
using Jeopardy.Core.Data.Quiz;
using Jeopardy.Core.Data.Quiz.Constants;
using ProtoBuf;

namespace Jeopardy.Core.Data.Gameplay.Actions
{
    [ProtoContract]
```

```
public class QuestionSelectAction : GameAction
{
    [ProtoMember(1)]
    public string SelectorNetworkUserId { get; set; } =
       string.Empty;
    [ProtoMember(2)]
    public int RoundId { get; set; }
    [ProtoMember(3)]
    public int CategoryId { get; set; }
    [ProtoMember(4)]
    public int QuestionId { get; set; }

    public override bool CanExecute(GameState gameState) =>
       gameState.GameContext is SelectQuestionContext;

    public override void Execute(GameState gameState)
    {
        Question? question = gameState.Quiz.Rounds[RoundId].
           Categories[CategoryId].Questions[QuestionId];
        question.Unplayed = false;
        gameState.CurrentQuestion = question;

        foreach (Player player in gameState.Players.Values)
        {
            player.HasAnswerAttempt = true;
        }

        gameState.GameContext = question.QuestionType switch
        {
            QuestionType.Simple => new SimpleQuestionContext(
               SelectorNetworkUserId),
            //QuestionType.Auction => new
               AuctionQuestionContext(),
            //QuestionType.Secret => new
               SecretQuestionContext(),
            //QuestionType.Sponsored => new
               SponsoredQuestionContext(),
            _ => new SimpleQuestionContext(
               SelectorNetworkUserId)
        };
    }

    private QuestionSelectAction() { }
    public QuestionSelectAction(string selectorNetworkUserId,
        int roundId, int categoryId, int questionId)
    {
```

```
            SelectorNetworkUserId = selectorNetworkUserId;
            RoundId = roundId;
            CategoryId = categoryId;
            QuestionId = questionId;
        }
    }
}

//PlayerSelectAction.cs

using Jeopardy.Core.Data.Gameplay.Contexts;
using ProtoBuf;

namespace Jeopardy.Core.Data.Gameplay.Actions
{
    [ProtoContract]

    public class PlayerSelectAction : GameAction
    {
        [ProtoMember(1)]
        public string SelectedNetworkUserId { get; set; } =
           string.Empty;

        public override void Execute(GameState gameState) =>
           gameState.GameContext = new SelectQuestionContext(
           SelectedNetworkUserId);
        public override bool CanExecute(GameState gameState) =>
           gameState.GameContext is PlayerSelectContext;


        private PlayerSelectAction() { }
        public PlayerSelectAction(string selectedNetworkUserId)
           => SelectedNetworkUserId = selectedNetworkUserId;
    }
}

//JudgeAction.cs

using Jeopardy.Core.Data.Gameplay.Contexts;
using ProtoBuf;

namespace Jeopardy.Core.Data.Gameplay.Actions
{
    [ProtoContract]

    [ProtoInclude(5001, typeof(ApproveAnswerAction))]
    [ProtoInclude(5002, typeof(DenyAnswerAction))]
```

```
    public abstract class JudgeAction : GameAction
    {
        [ProtoMember(1)]
        public string AnsweringPlayerId { get; set; } = string.
           Empty;

        public override bool CanExecute(GameState gameState) =>
           gameState.GameContext is PlayerAnswerContext;

    }
}

//GameAction.cs

using ProtoBuf;

namespace Jeopardy.Core.Data.Gameplay.Actions
{
    [ProtoContract]
    [ProtoInclude(5001, typeof(PlayerSelectAction))]
    [ProtoInclude(5002, typeof(StartGameAction))]
    [ProtoInclude(5003, typeof(QuestionSelectAction))]
    [ProtoInclude(5004, typeof(RequestAnswerAttemptAction))]
    [ProtoInclude(5005, typeof(JudgeAction))]
    [ProtoInclude(5006, typeof(SkipQuestionAction))]
    public abstract class GameAction
    {
        public abstract void Execute(GameState gameState);

        public abstract bool CanExecute(GameState gameState);
    }
}

//DenyAnswerAction.cs

namespace Jeopardy.Core.Data.Gameplay.Actions
{
    public class DenyAnswerAction : JudgeAction
    {
        public override void Execute(GameState gameState)
        {
            gameState.Players[AnsweringPlayerId].Score -=
               gameState.CurrentQuestion?.Price ?? 0;
            gameState.SetNextContext(AnsweringPlayerId);
        }
    }
}
```

ПРИЛОЖЕНИЕ В
(обязательное)
Конфигурационные файлы сервера

```
#See https://aka.ms/containerfastmode to understand how Visual
    Studio uses this Dockerfile to build your images for faster
    debugging.

FROM mcr.microsoft.com/dotnet/runtime:6.0 AS base
WORKDIR /app

EXPOSE 50000

FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build
WORKDIR /src
COPY ["Jeopardy.Web.Matchmaker.Service/Jeopardy.Web.Matchmaker.
    Service.csproj", "Jeopardy.Web.Matchmaker.Service/"]
COPY ["Jeopardy.Core.Network/Jeopardy.Core.Network.csproj", "
    Jeopardy.Core.Network/"]
COPY ["Jeopardy.Core.Serialization/Jeopardy.Core.Serialization.
    csproj", "Jeopardy.Core.Serialization/"]
COPY ["Jeopardy.Core.Data/Jeopardy.Core.Data.csproj", "Jeopardy.
    Core.Data/"]
COPY ["Jeopardy.Core.Cryptography/Jeopardy.Core.Cryptography.
    csproj", "Jeopardy.Core.Cryptography/"]
COPY ["Jeopardy.Core.Localization/Jeopardy.Core.Localization.
    csproj", "Jeopardy.Core.Localization/"]
RUN dotnet restore "Jeopardy.Web.Matchmaker.Service/Jeopardy.Web.
    Matchmaker.Service.csproj"
COPY . .
WORKDIR "/src/Jeopardy.Web.Matchmaker.Service"
RUN dotnet build "Jeopardy.Web.Matchmaker.Service.csproj" -c
    Release -o /app/build

FROM build AS publish
RUN dotnet publish "Jeopardy.Web.Matchmaker.Service.csproj" -c
    Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "Jeopardy.Web.Matchmaker.Service.dll"]
```