МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФГБОУ ВО ПСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

КОЛЛЕДЖ ПСКОВГУ

КУРСОВОЙ ПРОЕКТ

по МДК 03.02. Инструментальные средства разработки программного обеспечения

Тема: «Разработка программного обеспечения автоматизированной информационной системы «Кинотеатр»

по специальности

09.02.03 Программирование в компьютерных системах

Обучающегося группы 0314-36 ПО

Певцов А. П.

Проверил(а): Луканов С. Ю.

Оценка\_\_\_\_\_\_\_\_\_\_\_\_

Подпись\_\_\_\_\_\_\_\_\_\_\_

Псков

2021

Содержание

[Введение 3](#_Toc90985063)

[1. Теоретическая часть 4](#_Toc90985064)

[Системы документирования исходного кода 4](#_Toc90985065)

[Пять стратегий документирования кода 8](#_Toc90985066)

[Подход 1: размещаем “Как?” во внутренние комментарии, “Почему?” - во внешней документации. 8](#_Toc90985067)

[Подход 2: сопоставление комментариев в третьей колонке 10](#_Toc90985068)

[Подход 3: подход Лего 11](#_Toc90985069)

[Подход 4: подход Наутилус 12](#_Toc90985070)

[Подход 5: Интерактивный браузер 15](#_Toc90985071)

[2. Техническое задание 18](#_Toc90985072)

[3. Практическая часть 22](#_Toc90985073)

[3.1. Постановка задач 22](#_Toc90985074)

[3.2. Математическое теоретическое обоснование 22](#_Toc90985075)

[3.3. Схема данных 23](#_Toc90985076)

[3.4. Концептуальная схема базы данных 24](#_Toc90985077)

[3.5. Обоснования выбора инструментальных средств 25](#_Toc90985078)

[3.6. Выполнение задачи на основе выбранного инструментального средства 25](#_Toc90985079)

[4. Экспериментальное тестирование 29](#_Toc90985080)

[5. Эксплуатационные документы 31](#_Toc90985081)

[5.1. Руководства системного администратора 31](#_Toc90985082)

[5.2. Руководство программиста 33](#_Toc90985083)

[5.3. Руководство оператора 36](#_Toc90985084)

[5.4. Руководство к использованию 37](#_Toc90985085)

[6. Описание языка 39](#_Toc90985086)

[Заключение 41](#_Toc90985087)

[Список источников информации 42](#_Toc90985088)

## Введение

Целью данного курсового проекта является применение теоретических знаний об инструментальных средствах разработки программного обеспечения на практике для создания автоматизированной информационной системы.

Также целью является: научится использовать средства построения диаграмм и графиков, необходимых в программной документации к проекту. Таких как: Erwin data modeler, MS Office Visio и Modeler Navigator.

Для выполнения данной разработки необходимо реализовать автоматизированную информационную систему «Кинотеатр». С этой целью необходимо решить следующие задачи:

Создать логическую структуру;

Выбрать инструментальные средства;

Выполнить разработку;

Разработать интерфейс пользователя;

Протестировать работу программы;

Написание программной документации;

В данном задании необходимо в полной мере реализовать знания и навыки, касающиеся инструментальных средств разработки программного обеспечения, создания программной документации, создания моделей данных, стадий процесса разработки программного продукта и стадий жизненного цикла.

# Теоретическая часть

# Системы документирования исходного кода

Вероятнее всего, каждый из нас сталкивался с результатами работы различных генераторов документации. Общий принцип их работы, следующий: на вход такого генератора поступает специальным образом комментированный исходный код, а иногда и другие компоненты программы, а на выходе создаётся готовая документация для распространения и использования. Многие программисты не любят это дело и пытаются всячески избежать этой работы всеми доступными способами. Отсутствие документации кода приводит к плохой его читаемости и к сложному техническому обслуживанию для других членов команды.

Документация кода отличается от проектной документации, так как она в основном фокусируется на том, как работает система. Несмотря на то, есть несколько причин для написания документации, многие программисты, как правило, игнорируют это. Ниже я собрал ряд причин, по которым, все таки стоит писать документацию для своего кода:

* Ваш код будет поддерживаться и использоваться другими программистами в команде? Обслуживание кода становится большой проблемой, если он не был должным образом документирован.
* Вы хотите, чтобы другие программисты помогли вам, например, через открытый исходный код. Если вы планируете начать большую, коллективную работу, стоит начать документирование кода прямо сейчас!
* Документирование вашего кода делает логику гораздо более ясной, а также делает ваш код лучше.
* Недокументированный код тяжело не только передавать на сопровождение, а также порой тяжело сопровождать и самому

Даже с учетом всех вышеперечисленных преимуществ, документирование, в целом, является трудоемким процессом. Для того, чтобы обеспечить более быстрый процесс подготовки документации и последовательности стилей, вам стоит использовать специальные инструменты.

Эти инструменты помогут вам с документированием кода и в целом помогут стать на ступеньку выше. Давайте начнем!

Doxygen

Doxygen является отличным инструментом для генерации документации из исходного кода. Инструмент нацелен на документирование программного обеспечения, написанного на языке C++, однако на самом деле данная система поддерживает гораздо большое число других языков, таких как: C, Objective-C, C#, PHP, Java, Python, IDL, Fortran, VHDL, Tcl, и частично D. С помощью Doxygen, вы можете создать онлайн HTML документацию. Doxygen — консольная программа в духе классической Unix. Она работает подобно компилятору, анализируя исходные тексты и создавая документацию.

Самым большим преимуществом использования Doxygen является то, что вы будете иметь последовательность всей документации исходного кода. Она также может помочь вам создавать структуру кода с использованием недокументированных исходных файлов. Все, что вам нужно сделать, это настроить его соответствующим образом.

Sphinx

Sphinx это популярный инструмент позволяющий создавать текстовые документы и преобразовывать их в различные форматы. Это удобно при использовании систем управления версиями, предназначенных для отслеживания изменений. Он доступен по лицензии BSD и поддерживает несколько языков программирования, таких как Python, C и C++. Он может быть использован как для проектной документации, так и для документации кода. Sphinx избавляет от рутинных действий и предлагает автоматическую функциональность для решения типовых проблем, например, индексирования заголовков и специального выделения кода (например, при включении в документ фрагментов кода) с соответствующим выделением синтаксиса.

Pandoc

Pandoc не похож на другие инструменты для документации. Он действует как швейцарский нож и позволяет разработчику быстро конвертировать разметку из одного формата в другой. Pandoc имеет широкий спектр поддержки документов, в том числе textile, reStrcuturedText, LaTex, EPUB и т.д.

Кроме того, он предлагает несколько расширений синтаксиса разметки, в том числе списки определений, таблицы, сноски и т.д.

Dr. Explain

Frontend разработка также, в определенной степени, требует документирования. Создавать документацию как для обычных, так и онлайн-приложений, написанных на любом языке программирования, в любой среде разработки, с применением любого фреймворка поможет DR. EXPLAIN. Он фильтрует ключевые элементы интерфейса, а затем извлекает связанные с ним мета данные. После этого, вы можете изменить полученную информацию, чтобы быстро создать документацию интерфейса.

LaTex

LaTex является де-факто стандартом для документирования научных проектов. Тем не менее, он также может быть использован для других типов проектов, в том числе кода и проектной документации. Готовя свой документ, автор указывает логическую структуру текста (разбивая его на *главы*, *разделы*, *таблицы*, *изображения*) и позволяет LaTeX’у заботиться о том, как изобразить его.

Markdown

Markdown, творение Джона Грубера, очень простой и изящный *синтаксис разметки текста*, который поможет вам писать качественный код и документации. С технической точки зрения Markdown является инструментом преобразования текста в HTML для веб-писателей, но в равной степени он может быть использован и для документирования. Как разработчик, вы можете написать документацию в Markdown, а затем использовать Pandoc, чтобы преобразовать его в любой формат, который вам нужен.

GhostDoc

GhostDoc это расширение для Visual Studio, с помощью которого вы можете легко генерировать комментарии документа XML. Инструмент генерирует комментарии на основе нескольких факторов, в том числе имя, параметры, контекстную информацию, типы и т.д.

Natural Docs

Natural Docs это еще один инструмент с открытым исходным кодом, который работает со многими языками программирования. Он поможет вам автоматизировать генерацию документации кода и преобразовать его в формат HTML. В настоящее время NATURAL DOCS поддерживает 19 языков программирования, среди них Python, C ++, PL / SQL, Actionscript и т.д.

PHPDocumentor

Если вы PHP разработчик и хотите сгенерировать документацию кода из исходного кода, стоит рассмотреть PhpDocumentor.

В основе работы системы лежит парсинг логической структуры PHP кода (классы, функции, переменные, константы) и привязка к ней комментариев, написанных по определенным стандартам. Инструмент также может помочь вам генерировать отчеты и графики и повысить общее качество кода.

Liveedu.TV

Как стриминговая платформа может помочь в документировании кода? Вы можете транслировать или хранить проектную работу непосредственно на Livecoding. Вы будете иметь возможность легко открыть доступ к важным разделам проекта для членов вашей команды. Несколько преимуществ использования видео в качестве инструмента для документирования кода. Некоторые из них приведены ниже:

* Нет писанины, но есть лучшее понимание контекста.
* Agile команды могут легко отслеживать изменения в проекте.
* Технические писатели могут использовать видео документацию, дабы понять проект лучше.
* Разработчики могут вкладывать сэкономленное время в реализацию других функциональностей проекта.

# Пять стратегий документирования кода

## Подход 1: размещаем “Как?” во внутренние комментарии, “Почему?” - во внешней документации.

После разделения, пояснения *Как?* встраиваются в код в виде комментариев (примерно на каждые 5-10 строк). Пояснения *Почему?* встраиваются во внешних разделах до или после кода. Разумеется, в пояснениях *Как?* можно получить подробную информацию о *Почему?* причинах, и наоборот, поэтому разделение этих двух вопросов на практике может быть не таким простым. Но такая общая закономерность, скорее всего, верна.

Кроме того, такой подход интересен тем, что он побуждает подумать над вопросом *Почему?*. Техническим писателям, документирующим код (который пишут другие), не всегда получается учитывать *Почему?*.Иногда бывает трудно понять, какие вопросы *Почему?* существуют. Зачем использовать этот класс вместо другого? Почему так, а не так? Часто существует много разных способов достижения цели, так почему же идти по этому пути, а не каким-то другим?

В общении с разработчиками о примерах кода, стоит задавать это вопрос: *Почему?*

* существуют другие подходы, от которых они отказались? Почему?
* могут ли пользователи реализовать какой-либо другой подход, или мы специально хотим, чтобы они следовали только одному методу? Почему?
* почему используется именно этот язык / инструмент / рамки / библиотеки, а не какой-либо другой?

Что касается вставки встроенных комментариев, best practice - это лучше всего вставлять короткие комментарии на каждые 5-10 строк кода. Получается не так много комментариев, что сделать код нечитаемым, но не так мало, чтобы не объяснять, что происходит.

Также стоит обратить внимание, что комментарии встроенного кода могут быть несколько спорными. Если просто объяснять, что делает код, это может оказаться лишним для знатока языка, на котором написан код. Некоторые разработчики считают, что простой код документирует сам себя - его значение понятно тем, кто знает язык, без необходимости объяснения. Но это утверждение про опытных пользователей и не распространяется на сложный код.

## Подход 2: сопоставление комментариев в третьей колонке

Лучшие практики для документации в целом (не только для документирования кода) включают размещение полезных инструкций рядом с областью, вызывающей затруднения, что в контексте документирования кода может означать добавление встроенных комментариев, пронизывающих весь код. Но предположим, что нужен большой комментарий о том, что происходит в коде (поскольку уровень сложности не может быть передан в коротком комментарии). Как сопоставить длинную концептуальную / пояснительную информацию с кодом?

Если ваш комментарий больше кода, есть риск сделать код нечитаемым. Если разместить комментарий в разделах, которые идут задолго до или после кода, есть риск создать пропасть между объяснением и кодом, так что читатели не будут знать, к каким частям кода относится объяснение.

Одним из решений такой проблемы размещения документации кода является создание дополнительного третьего столбца в макете. В среднем столбце находится концептуальное объяснение, а в правом столбце - код. Получается, код и его описание сопоставляются таким образом, что читатели могут взглянуть на код, читая концептуальные объяснения. Третий столбец поддерживает необходимый контекст между кодом и объяснениями.

Одной из проблем такого подхода является пространство экрана. Чтобы добавить третью колонку, нужно занять весь экран без полей.

Таким образом, код виден только частично. Код можно растянуть по горизонтали с возможностью прокрутки вправо, но, несомненно, дизайнерам тяжело дался пользовательский интерфейс, который включает в себя горизонтальную и вертикальную прокрутку.

Вдобавок, реализация размытия фокуса области экрана с кодом, на описании которого находится пользователь, должна быть технически сложной и громоздкой.

Другая проблема заключается в том, что код и пояснения к нему редко можно выровнять линейно. Предположим, есть метод в коде, который занимает всего одну строку, но описание этого метода занимает три длинных параграфа концептуального объяснения. К тому времени, когда пользователь достигает нижней части концептуального объяснения, код уже не виден. Пользователю приходится сделать много лишних манипуляций, чтобы найти соответствующий код. Дизайн пользовательского интерфейса для размещения всех этих движущихся частей не только кажется сложным, но и увеличивает нагрузку на пользователя.

Еще одна проблема, связанная с таким дизайном, заключается в том, что код часто относится к нескольким файлам. Отображение в колонке не указывает, отображается ли весь код в одном и том же файле Java или мы видим код из нескольких файлов Java. Представление вкладок в столбце кода требует еще более сложного формата документации. Сложно представить, что все это возможно сделать с помощью синтаксиса Markdown.

## Подход 3: подход Лего

Другой подход заключается в создании кода с нуля, уровня за уровнем, который можно назвать “подходом Лего”.

Подход Лего можно начать с пустого файла. Затем с каждым шагом добавлять все больше и больше кода, пока не получится полностью рабочий пример. Как и в случае с конструктором Лего - начинаем с основания, а затем добавляем блок за блоком, пока не получим желаемый результат.

Проблема с подходом Лего заключается в том, что технический писатель вряд ли воссоздаст логику построения, которой следовал разработчик. Скорее всего, разработчики просто отправят готовый кусок кода в документ, и тогда придется получать пояснения от разработчика о коде.

Поскольку код нелинейный, пошаговая работа с пояснениями кода не обязательно будет соответствовать подходу Лего. Код, отображаемый в верхней части, может показаться разработчикам вишенкой на торте - например, абстрагирование более сложных строк в переменные, которые используются для уменьшения сложности работы кода. Готовый код часто имеет логику, которая абстрагируется в переменные или другие ссылочные функции, так что определенные части конечного кода остаются чище и более краткими, а другие части становятся более непрозрачными. Готовый код часто слишком запутанный и сложный, чтобы документировать его любым доступным для обучения способом.

Несмотря на проблемы с подходом Лего, если нужно научить кого-то, как понимать код, нужно начать с простого и идти шаг за шагом от простого к сложному. Следующая техника объясняет такой подход от простого к сложному при помощи метафоры наутилуса.

## Подход 4: подход Наутилус

Подход Наутилус аналогичен подходу Лего, но вместо того, чтобы описывать куски работы, которые выполняются один за другим в порядке сборки, здесь описываются основные простые шаблоны, которые должны знать пользователи. Начинается с самого простого кода, а затем проект увеличиваться и увеличиваться по мере необходимости, как растущая спиральная структура оболочки наутилуса.

Пол Густафсон, управляющий компанией, занимающейся техническим обслуживанием письма, в Bay Area, которая называется экспертная поддержка, представляет подход с метафорой Наутилуса, описанный здесь. Пол говорит, что наутилус является хорошей метафорой для технической коммуникации, потому что наутилус представляет спиральную схему (последовательность Фибоначчи), которая начинает с малого и постепенно растет по мере необходимости.

Пол пишет:

«Содействие пониманию, которое является основной технической связью, происходит наиболее эффективно, следуя аналогичной схеме….  
Когда ваше понимание невелико, вы учитесь лучше, когда первый урок дает информацию для небольшой, простой задачи с чертами, которые сродни первым камерам наутилуса.  
… Хорошая новость для наутилуса заключается в том, что маленькие камеры следуют тому же основному плану, что и большие камеры. Если первые задания, которые освоил учащийся, в основном аналогичны более сложным задачам в учебной программе, учащийся начинает понимать и применять эти шаблоны. Чем раньше новички научатся «думать о вещах правильно», тем быстрее они «поймут», а это именно то, чего хотят и преподаватель, и ученик. (Уроки головоногих)»

Следуя подходу Наутилус, мы начинаем с простых базовых шаблонов, а затем постепенно наращиваем объем кода вокруг него по мере необходимости. Мы не начинаем с описания сложной законченной работы с самого начала. Законченная работа, вероятно, включает в себя все виды абстракций кода и перестановки для чистого, готового продукта.

Проблема в том, что мы часто хотим объяснить, как работает готовый код, проводя пользователя через все от начала до конца. У нас может быть 500 строк кода, которые мы хотим сформулировать, но подход Nautilus заставил бы нас объяснить только несколько небольших фрагментов этого кода (по крайней мере, для начала). Следовательно, существует проблема типа A-против-Z: мы описываем A (самый простой шаблон ядра), но конечным продуктом является Z. Как именно мы добираемся от A до Z? Как мы придем от простых шаблонов, которые могут занимать 20 строк кода, до чудовищно сложного, законченного кода, который занимает 500 строк?

Для технического писателя, смотрящего на окончательный код, нет ясного понимания того, как разработчик попал туда. Мы часто не можем отделить подобную логику, с которой начинал разработчик, что привело его к этой более сложной цели. Все, что мы видим, это этот сложный конец. Как декомпилировать код, чтобы восстановить логику, с которой разработчик начал? Как понять, что это были за начальные паттерны, которые положили начало всему процессу? Если вы не разрабатывали код и не являетесь разработчиком, то будет практически невозможно восстановить шаблон Наутилуса, стоящий за кодом.

В качестве другой аналогии можно рассмотреть возможность научить рисованию. Предположим, задача - описать готовую картину потенциальному художнику. Вам необходимо описать, как рисовать Мону Лизу.

Чтобы задокументировать процесс создания такой картины, что лучше: начать с верха и идти к низу? Нет, это было бы смешно. Скорее всего, лучше начинать с создания овала головы. Потом, несколько общих набросков для глаз и так далее. Может быть, набросать линии перспективы и другую основную структуру. Никто сразу не приступает к цветам, освещению и теням, верно? То же самое с кодом. Начинаем с основы, а затем прокладываем путь к большему количеству завершающих деталей.

Однако, если вы не художник, как бы вы узнали, как описать процесс создания картины? Вам нужно знать логику художника - с чего начать и как идти к концу. Если вместо этого вы начинаете с конца и пытаетесь вернуться назад, руководство по рисованию, вероятно, будет безумно сложным.

## Подход 5: Интерактивный браузер

Изучение основных шаблонов смещает документацию в область руководства. С этим типом обучения связан интерактивный браузер, в котором действие сочетается с обучением. Интерфейсы, выполняемые браузером, имеют своей целью помочь пользователям лучше понять результаты входных и выходных данных, чтобы пользователи могли сами убедиться в том, как работает код, используя более практичный для себя подход.

Наиболее распространенный пример интерактивной документации для API - это пользовательский интерфейс Swagger.

Swagger предлагает оригинальное сочетание документации с активным взаимодействием, которые помогают пользователям изучать API (читая и выполняя). Но выполнение запросов с помощью конечных точек API REST обычно довольно простое. Более интенсивные руководства по коду будет сложнее сделать интерактивными в браузере. Тем не менее, некоторые «Notebooks» (как их часто называют) позволяют запускать код, в частности Jupyter Notebook. Описание Jupyter:

Jupyter Notebook - это веб-приложение с открытым исходным кодом, которое позволяет вам создавать и обмениваться документами, которые содержат живой код, уравнения, визуализации и текст повествования. Использование включает в себя: очистку и преобразование данных, численное моделирование, статистическое моделирование, визуализацию данных, машинное обучение и многое другое.

У Google есть несколько вариантов блокнота для совместной работы с документацией TensorFlow, в которой есть операции, которые можно выполнять на веб-страницах.

Хотя интерактивные записные книжки выглядят круто, они кажутся огромной работой для чего-то, что может быть легко достигнуто с помощью образца приложения. Вместо того, чтобы выяснить, как вы можете скомпилировать код Python или другой язык в браузере, почему бы просто не предоставить пример приложения, которое пользователи могут загрузить, а затем выполнить локально, используя свои собственные инструменты компиляции и настройки?

Конечно, пользователям может потребоваться установить на своем компьютере некоторые утилиты и платформы, чтобы запустить примеры приложений, а также IDE, но выполнение кода в браузере может информировать не полностью пользователей обо всех необходимых настройках и подготовке, которые в конечном итоге будут быть обязательными для них, чтобы заставить код работать локально. Браузеры имеют тенденцию быть несколько жесткими в том, что они могут сделать - у пользователей может быть больше свободы экспериментировать (и учиться на этих экспериментах) на примерах приложения.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФГБОУ ВО ПСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

КОЛЛЕДЖ ПСКОВГУ

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

Тема: «Разработка программного обеспечения автоматизированной информационной системы «Кинотеатр»

Выполнил обучающийся группы 0314-36 ПО

Певцов А. П.

Утвердил(а): Луканов С. Ю.

Псков

2021

## Техническое задание

**Введение**

Цель работы – обеспечение автоматизированной информационной системы «Кинотеатр». Программа предназначена для создания, управления содержимым базы данных, содержащим следующие данные:

1. Сведения о киносеансах (название фильма, жанр, в каком зале проходит, в какой день недели и во сколько)
2. Сведения о том на какие фильмы куплены билеты и кем
3. Отчет по таблицам БД
4. **Основания для разработки**

Основанием для разработки является задание на выполнение курсового проекта. Организация, утвердившая задание на разработку программного продукта: Колледж ПсковГУ.

Наименование работы: Автоматизированная информационная система «Кинотеатр»

1. **Назначение разработки**

Автоматизированная информационная система «Кинотеатр» предназначена для обобщения информации о наличии сеансов определенных фильмов и покупке билетов на них. Пользователями программы выступают администраторы кинотеатра, отдел учета, обычные посетители кинотеатра. Приобретение билетов осуществляется на основании договоров купли-продажи. Оформление и учет реализации билетов зависят от способа расчета за приобретенные в прокат кинофильмы между покупателем и продавцом.

1. **Требования к программе**
   1. **Требования к функциональным характеристикам.**

Автоматизированная система «Кинотеатр» должна обеспечивать выполнение функций:

- ввод, хранение, поиск и обработку информации по сеансам.

- ведение журнала регистрации приходных и расходных документов

- своевременное получение информации о наличии билетов на фильмы.

- формирование отчетов, необходимых менеджеру и бухгалтеру, содержащих все данные о прокатываемых фильмах.

Нормативно справочная информация автоматизированной информационной системы «Кинотеатра» представлена справочниками контрагентов, номенклатуры.

Первичные документы для учета прокатываемых фильмов:

- документы, содержащие дату реализации, перечень реализуемых фильмов, их количество и сумму продажи;

- документы на поступление фильмов от киностудий, содержащие следующую информацию: дата поступления, сведения о киностудии, перечень поступающих фильмов, количество, цена и общая сумма.

Выходными данными являются следующие виды отчетов:

- отчеты о поступлении фильмов за определенный период, содержащий сведения о киностудиях перечень поступивших фильмов, их количество, цену, сумму поступления по каждому наименованию и общую сумму.

- отчет о продажах за определенный период, содержащий перечень реализованной номенклатуры, их цену, количество и сумму продаж по каждому пункту и общую сумму реализации.

* 1. **Требования к надежности**

Требования к обеспечению надежного функционирования программы

Надежное (устойчивое) функционирование программы должно быть обеспечено выполнением Заказчиком совокупности организационно-технических мероприятий, перечень которых приведен ниже:

а) организацией бесперебойного питания технических средств;

б) использованием лицензионного программного обеспечения;

в) регулярным выполнением рекомендаций Министерства труда и социального развития РФ, изложенных в Постановлении от 23 июля 1998 г. Об утверждении межотраслевых типовых норм времени на работы по сервисному обслуживанию ПЭВМ и оргтехники и сопровождению программных средств»;

г) регулярным выполнением требований ГОСТ 51188-98. Защита информации. Испытания программных средств на наличие компьютерных вирусов

Время восстановления после отказа, вызванного сбоем электропитания технических средств (иными внешними факторами), не фатальным сбоем (не крахом) операционной системы, не должно превышать 30-ти минут при условии соблюдения условий эксплуатации технических и программных средств.

Время восстановления после отказа, вызванного неисправностью технических средств, фатальным сбоем (крахом) операционной системы, не должно превышать времени, требуемого на устранение неисправностей технических средств и переустановки программных средств.

* 1. **Требования к составу и параметрам технических средств**

В состав технических средств должен входить IВМ-совместимый персональный компьютер (ПЭВМ), выполняющий роль сервера, включающий в себя:

Х86 совместимый процессор с тактовой частотой не менее 2Ггц;

Оперативную память объемом не менее 4 Гб;

Свободного места на SSD или HDD не менее 10 Гб;

Минимум встроенное графическое ядро в процессор или видеокарту;

операционную систему Windows 10;

.NET Framework 4.6.1 версии

* 1. **Требования к информационной и программной совместимости**

Программа должна работать в операционных системах Windows 10.

* 1. **Требования к транспортированию и хранению**

ПО поставляется в электронном виде. Программная документация поставляется в электронном виде.

* 1. **Специальные требования**

Программное обеспечение должно иметь дружественный интерфейс, рассчитанный на пользователя средней квалификации.

1. **Требования к программной документации**

В ходе разработки программы должны быть подготовлены текст программы**,** описание программы.

## Практическая часть

## Постановка задач

Разработать программное обеспечение автоматизированной информационной системы «Кинотеатр». База данных должна содержать все сведения о фильмах, поступающих для проката. В ней должны быть данные о сеансах (жанр, название, в каком зале, в какой день недели и в какое время), сведения о билетах (включая дату покупки, способ оплаты), а также сведения о пользователях (логин, пароль, имя, фамилия). Создать экранные формы для ввода и редактирования данных в таблицах и все необходимые выходные отчеты.

# Математическое теоретическое обоснование

Цель разработки данного программного обеспечение – уменьшение затрат времени на проведение мероприятий, связанных с документооборотом предприятия, а также с оптимизацией издержек на работу персонала кинотеатра.

Уменьшение затрат времени и оптимизация издержек достигается путем автоматизации рабочего места администратора кинотеатра. Она заключается в автоматизированной обработке данных поступления и прокат фильмов в кинотеатре.

# Схема данных

Структура базы данных состоит из 3 таблиц: сеансы, пользователи и заказы (Рисунок 1).

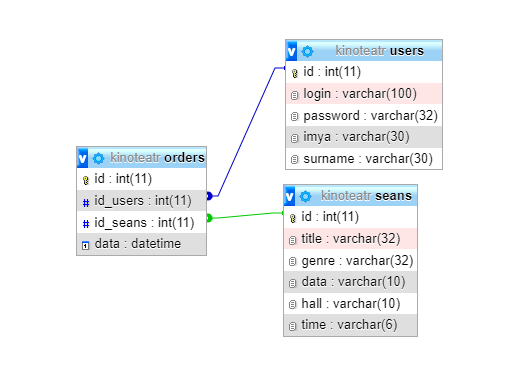


Рисунок 1. Схема используемых данных.

# Концептуальная схема базы данных

Концептуальная схема взаимодействия с базой данных (Рисунок 2):

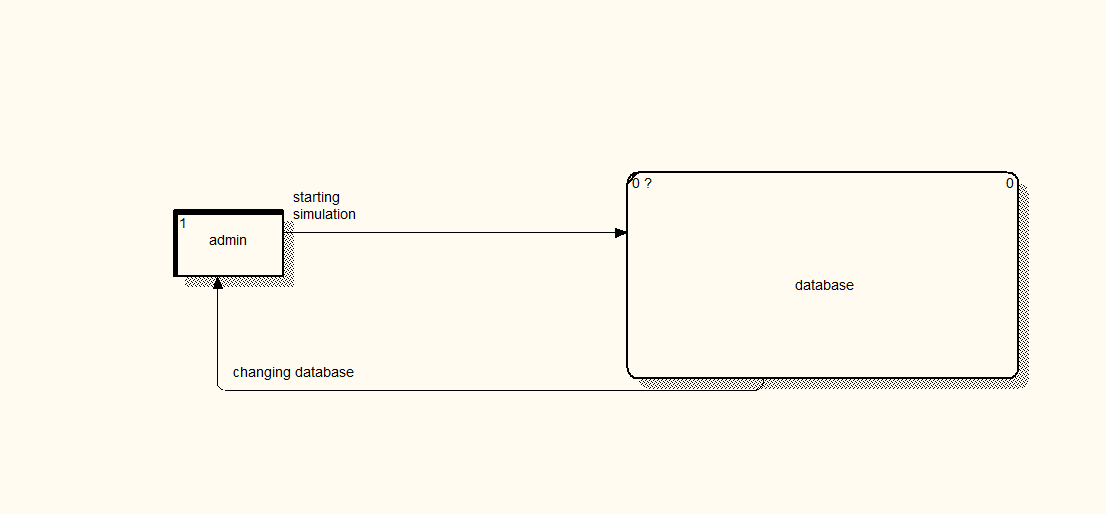


Рисунок 2. Схема взаимодействия с БД

Схема бизнес-процессов (Рис. 3):

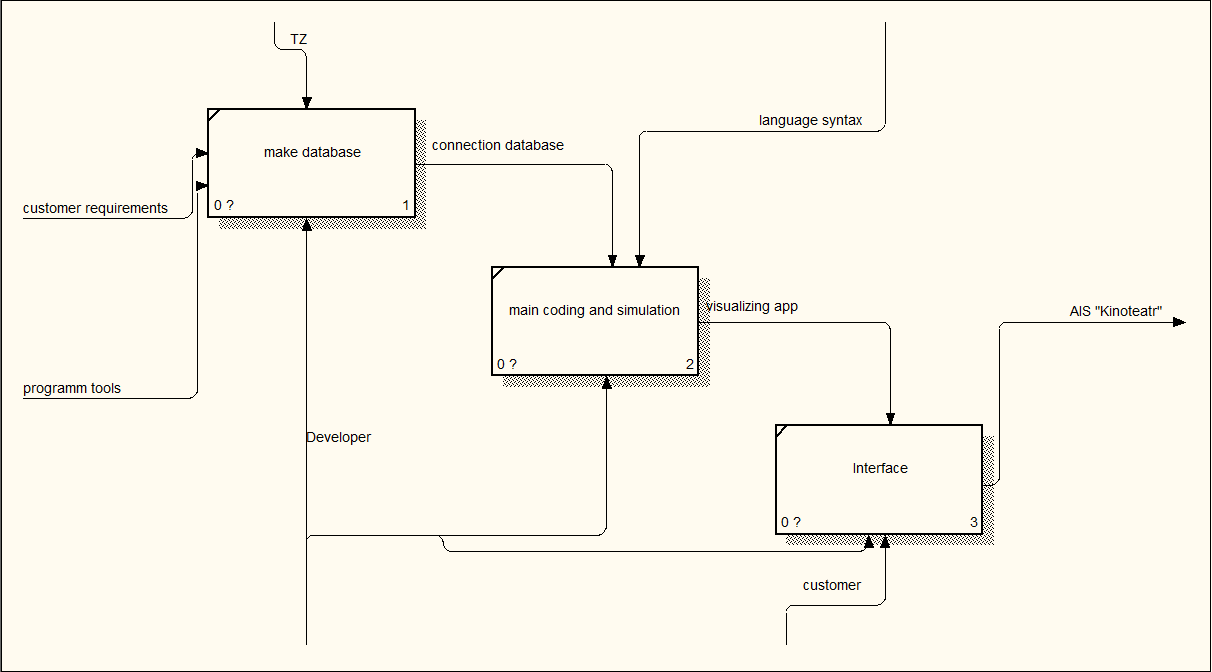


Рисунок 3. Схема бизнес-процессов

Архитектура приложения (Рис. 4):

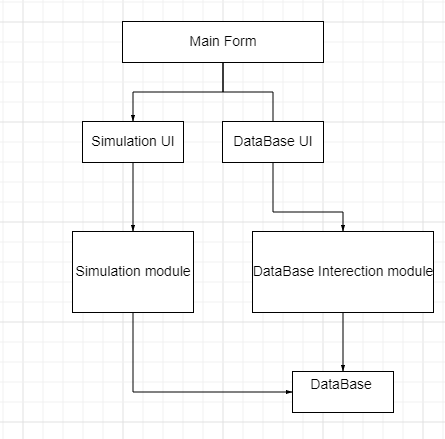


Рисунок 4. Архитектура приложения

## Обоснования выбора инструментальных средств

Для выполнения поставленной задачи была выбрана программная платформа .NET Framework в связи с тем, что она имеет высокий уровень интеграции с ОС Windows, основным языком платформы – является язык программирования высокого уровня C#.

В качестве системы управления базами данных была выбрана программа MAMP и phpMyAdmin.

Важными обстоятельствами для выбора данной СУБД являются удобство и простота использования.

## Выполнение задачи на основе выбранного инструментального средства

* + 1. **Добавление базы данных**

Начальная стадия разработки базы данных начинается с добавления новой базы используя либо SQL-код, либо конструктор phpMyAdmin (Рис.5-6).

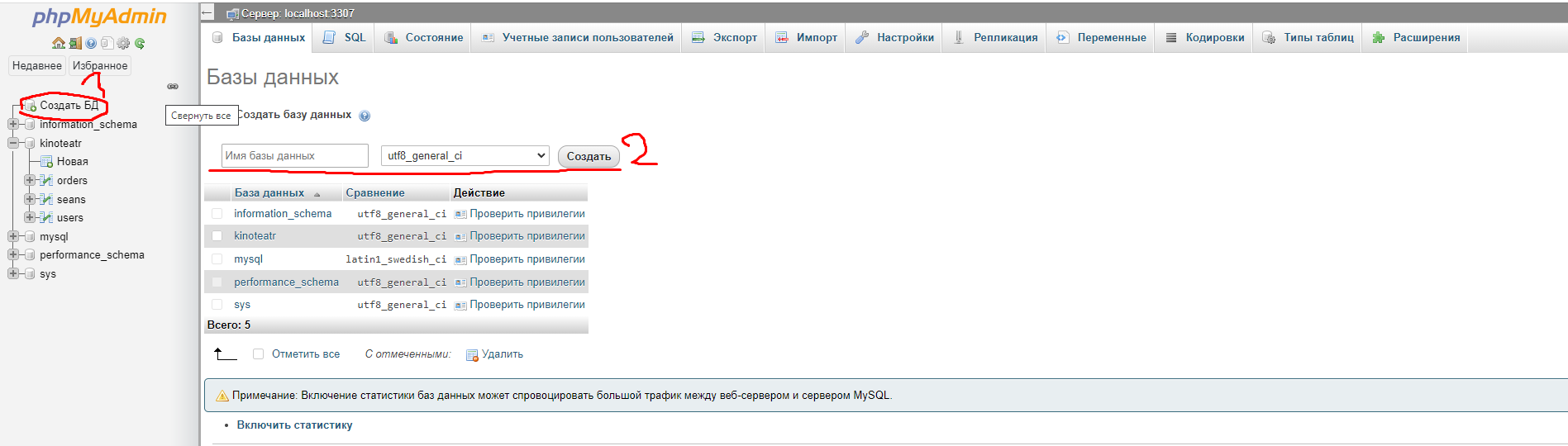


Рисунок 5. Добавление новой базы данных с помощью конструктора phpMyAdmin

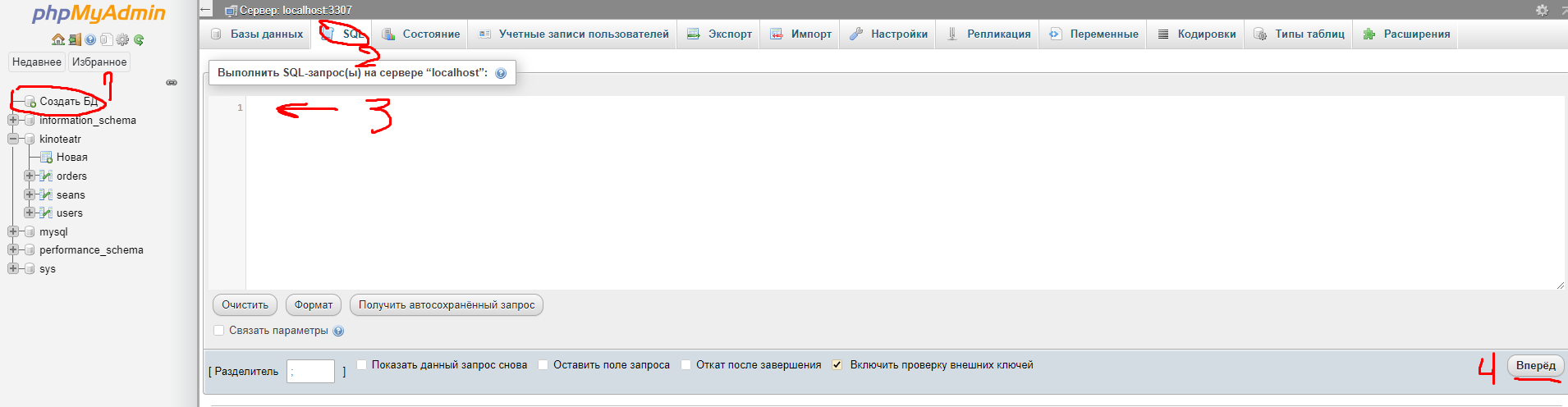


Рисунок 6. Добавление новой базы данных с помощью SQL-кода

После создания БД, ее структура выглядит следующим образом (Рис. 7).

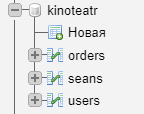


Рисунок 7. Структура БД

* + 1. **Решение поставленной задачи**

Для решения поставленной задачи были созданы:

* Приложения авторизации и регистрации (для регистрации сделана симуляция пользователей)
* Приложение «Сеансы», в котором симуляцией генерировались сеансы в соответствии с требованиями
* Приложение покупка, в котором выводилась таблица «orders» из БД, основанная на зависимости внешних ключей с таблицами «seans» и «users»
* Приложение отчет, в котором выводится запрос по БД.
  + 1. **Пользовательский интерфейс**

Для улучшения зрительного восприятия и повышения скорости работы в клиентском приложении была реализована система вкладок (Рис. 8)



Рисунок 8. Подсистемы конфигурации.

Результатом проделанной работы служит простой, интуитивно понятный пользовательский интерфейс (Рисунок 9).

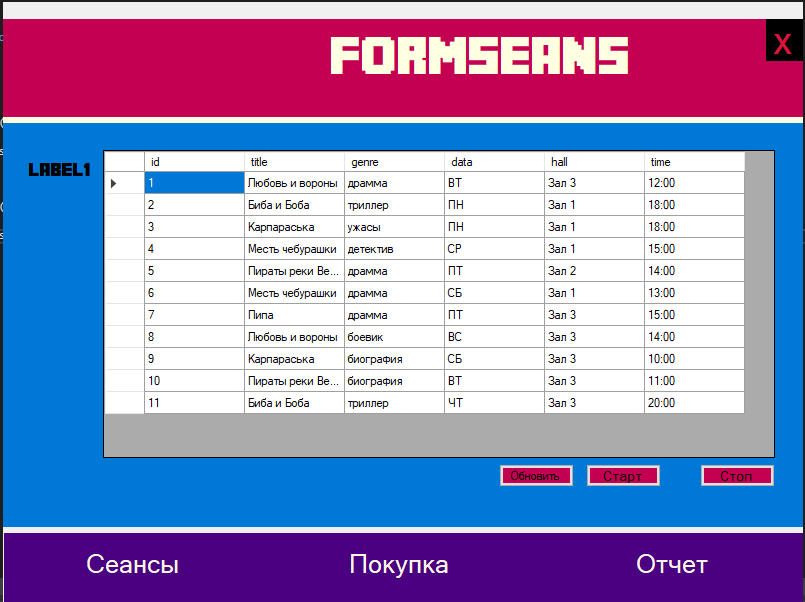


Рисунок 9. Интерфейс готовой программы.

Интерфейс авторизации и регистрации (Рис. 10-11).

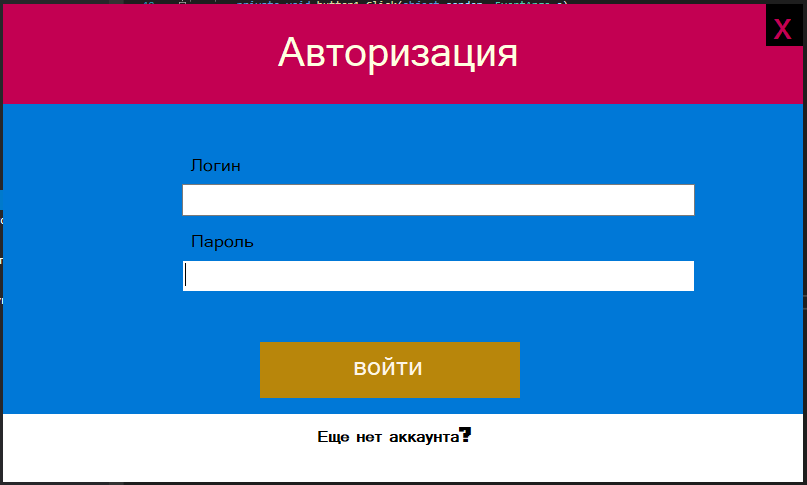


Рисунок 10. Интерфейс авторизации

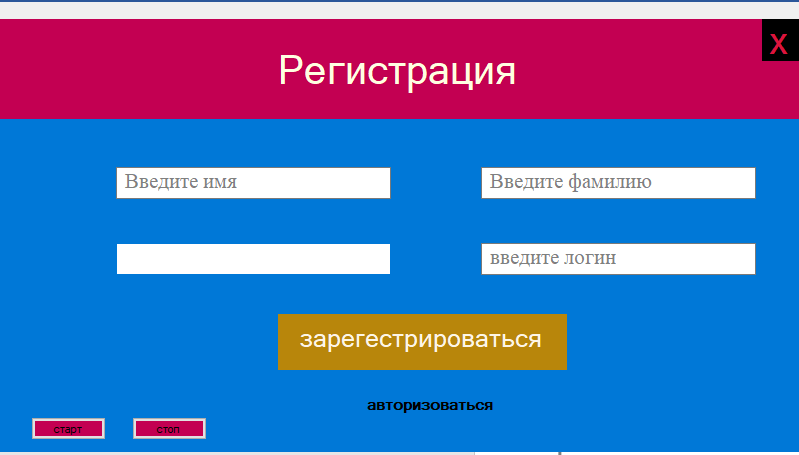


Рисунок 11. Интерфейс регистрации.

## Экспериментальное тестирование

Для проверки работоспособности автоматизированной информационной системы будет совершена попытка входа под несуществующим логином и паролем и запуск и проверка работы симуляции сеансов (Рис. 12-15).

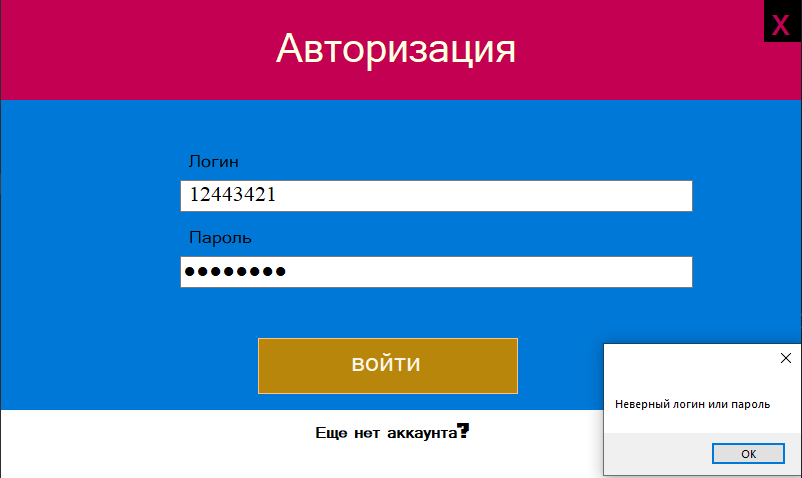


Рисунок 12. Попытка входа под неверным логином и паролем.

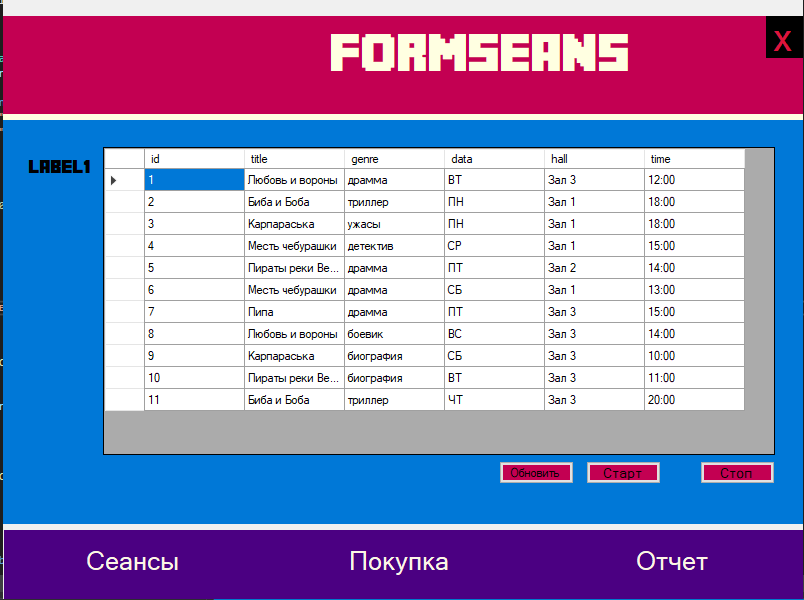


Рисунок 13. Приложение до запуска симуляции, имеется 11 сеансов.

После запуска симуляции по счетчику можно понять, что сгенерировалось 3 сеанса (Рис. 14). Для того чтобы проверить добавились ли они нажмем кнопку «Обновить» и увидим результат. Добавилось 3 сеанса их ID 12, 13, 14 (Рис. 15).

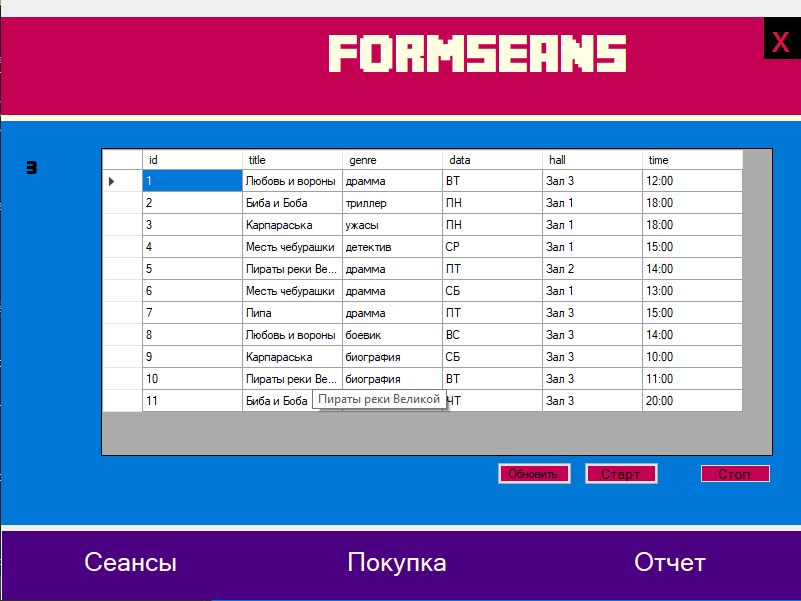


Рисунок 14. Приложение после запуска симуляции.

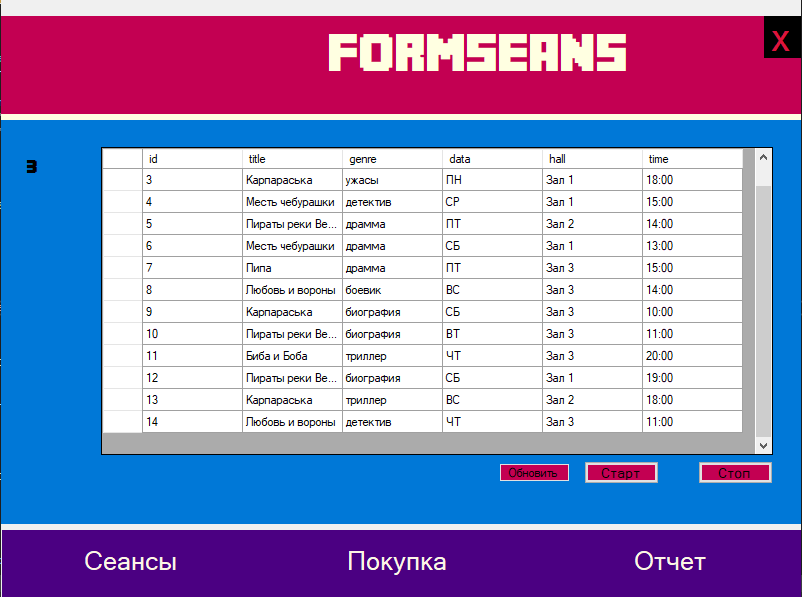


Рисунок 15. Приложение после обновления.

Проверка работоспособности базы данных «Кинотеатр» ошибок в ее работе не выявила.

В результате проверок базы данных ошибок выявлено не было, так как при создании автоматизированной информационной системы использовалась .NET Framework данная платформа не позволяет запустить программный продукт, в котором находятся критические ошибки.

## Эксплуатационные документы

## Руководства системного администратора

1. **Функции программы**

Автоматизированная информационная система «Кинотеатр» предназначена для обобщения информации о прокате фильмов.

1. **Минимальный состав технических средств**

Х86 совместимый процессор с тактовой частотой не менее 2Ггц;

Оперативную память объемом не менее 4 Гб;

Свободного места на SSD или HDD не менее 10 Гб;

Минимум встроенное графическое ядро в процессор или видеокарту;

Операционную систему Windows 10;

.NET Framework 4.6.1 версии

1. **Требования к персоналу (системному программисту)**

Операторы ЭВМ компании заказчика, управляющие должны пройти курс подготовки по работе с АИС «Кинотеатр».

В перечень задач, выполняемых операторами ЭВМ компании заказчика, будут входить:

1. Ведение данных в формы «FormSeans», «FormTicket», «FormOtchet»;

2. Обновление базы данных.

1. **Настройка программы**
   1. **Настройка на состав технических средств**

База данных «Кинотеатр» не требует каких либо настроек на состав технических средств.

* 1. **Настройка на состав программных средств**

Настройка на состав программных средств осуществляется путём установки операционной системы Microsoft® Windows® 10 на серверной и клиентской части АИС «Кинотеатр» и СУБД MAMP.

* 1. **Описание способов проверки**
* Структурное тестирование базы данных - это касается таблицы и столбца тестирование, тестирование схемы, хранимых процедур и тестирования взглядов, проверка триггеров и т.д.
* Функциональное тестирование - Это включает в себя проверку функциональности базы данных с точки зрения пользователя. Наиболее распространенный тип функционального тестирования являются Белая коробка и тестирование черного ящика.
* Нефункциональных Тестирование - Она включает в себя нагрузки тестирование, тестирование риска в базе данных, стресс - тестирование, минимальные системные требования, а также сделки с производительностью базы данных.

1. Проверка работоспособности программы
2. Проверка наличия платформы СУБД MAMP
3. Проверка имени базы данных
4. Проверка устройства данных, устройство дампа
5. Проверка, если хватает места, выделенного для каждой базы данных
6. Проверка настройки параметра базы данных
7. Первичный ключ для каждой таблицы
8. Внешние ключи для каждой таблицы
9. Типы данных между столбца внешнего ключа и столбца в других индексов таблицы, кластерные или не кластерные уникальным или не является уникальным.

## Руководство программиста

1. **Назначение и условия применения программы**
   1. **Назначение программы**

Программа предназначена для создания, управления содержимым базы данных, содержащим следующие данные:

1. Сведения о сеансах (название фильма, жанр, в каком зале, в какой день, в какое время)
2. Сведения о пользователях (логин, пароль, имя, фамилия)
3. Сведения о том на какие сеансы, когда и кем были приобретены билеты
   1. **Функции, выполняемые программой**

Автоматизированная информационная система «Кинотеатр» предназначена для обобщения информации о сеансах фильмов и покупке билетов на них.

* 1. **Условия, необходимые для выполнения программы**
  2. **Объем оперативной памяти**
* 32-разрядный сервер: 1024 мб и выше
* 64-разрядный сервер: 2048 мб и выше
  1. **Требования к составу периферийных устройств**

Программа «Кинотеатр» не предъявляет.

* 1. **Требования к параметрам периферийных устройств**

Программа «Кинотеатр» не предъявляет.

* 1. **Требования к программному обеспечению**

Компьютер конечного пользователя:

* Операционная система Microsoft® Windows® 10
* Клиентская часть АИС «Кинотеатр»
  1. **Требования к персоналу**

Операторы ЭВМ компании заказчика, управляющие должны пройти курс подготовки по работе с АИС «Кинотеатр».

В перечень задач, выполняемых операторами ЭВМ компании заказчика, будут входить:

1. Ведение данных в формы «FormSeans», «FormTicket», «FormOtchet»;

2. Обновление базы данных

1. **Характеристика программы**
   1. **Описание основных характеристик программы**

Система должна автоматизировать процессы регистрации, отслеживания покупки билетов и проходящих сеансов. Все данные должны храниться в БД на стороне сервера.

* 1. **Режим работы программы**

Файловый вариант работы обеспечивает легкость установки и эксплуатации системы. При этом для работы с информационной базой требуется установить СУБД MAMP.

Файловый вариант работы обеспечивает целостность информационной базы и простое создание резервных копий. Исключена ситуация, когда пользователь может по ошибке (например, при копировании информационной базы) перепутать различные файлы информационной базы и привести, таким образом, систему в неработоспособное состояние.

Резервное копирование может осуществляться при помощи SQL команды в терминале СУБД.

* 1. **Средства контроля правильности выполнения программы**

Корректное отображение данных, возможность обновлять и сохранять данные.

* 1. **Описание основных особенностей программы**

Программа практична, легка в пользовании, имеет дружественный интерфейс и подходит как для новичков, так и для бывалых пользователей.

* 1. **Самовосстанавливаемость программы**

Все данные хранятся на сервере и в случае непредвиденного поведения клиентской части АИС «Кинотеатр» данные не пострадают.

1. **Обращение к программе**
   1. **Загрузка и запуск программы**

Загрузка и запуск программы «Кинотеатр» осуществляется путём запуска исполняемого файла серверной части с правами администратора и запуска исполняемого файла клиентской части с правами обычного пользователя с доступом к сети Интернет.

* 1. **Выполнение функции сохранения обновлений базы данных**

Функция сохранения обновлений базы данных осуществляется автоматически после того, как были внесены в неё новые данные.

1. **Входные и выходные данные**
   1. **Организация используемой входной информации**

Входная информация будет содержать сведения о пользователе.

* 1. **Организация используемой выходной информации**

Выходная информация будет содержать сведения о сеансах, пользователях и на какие сеансы куплены билеты.

## Руководство оператора

1. **Назначение программы**
   1. **Функциональное назначение программы**

Автоматизированная информационная система «Кинотеатр» предназначена для обобщения информации о сеансах, пользователях и билетах.

* 1. **Эксплуатационное назначение программы**

Пользователями программы выступают администратор и пользователь.

* 1. **Условия выполнения программы**

Минимальный состав аппаратных средств:

Х86 совместимый процессор с тактовой частотой не менее 2Ггц;

Оперативную память объемом не менее 4 Гб;

Свободного места на SSD или HDD не менее 10 Гб;

Минимум встроенное графическое ядро в процессор или видеокарту;

операционную систему Windows 10;

.NET Framework 4.6.1 версии

* 1. **Минимальный состав программных средств**

Для работы программного обеспечения базы данных «Кинотеатр» должна быть установлена платформа СУБД MAMP.

* 1. **Требования к персоналу (пользователю)**

Операторы ЭВМ компании заказчика и управляющие должны пройти курс подготовки по работе с АИС «Кинотеатр». В перечень задач, выполняемых операторами ЭВМ компании заказчика, будут входить:

1. Ведение данных в формы «FormSeans», «FormTicket», «FormOtchet»;

2. Обновление базы данных

## Руководство к использованию

1. **Назначение программы**
   1. **Назначение программы**

Автоматизированная информационная система «Кинотеатр» предназначена для обобщения информации о сеансах, пользователях и билетах.

* 1. **Возможности программы**

База данных позволяет хранить большой объём данных, обновлять их.

* 1. **Основные характеристики программы**

База данных позволяет производить учет прокатываемых фильмов, пользователей и билетов.

* 1. **Ограничения, накладываемые на область применения программного обеспечения**

База данных предназначена только для использования в кинотеатре. Свободное распространение по сети Интернет запрещено.

1. **Условия применения**
   1. **Требования к техническим (аппаратным) средствам**

Х86 совместимый процессор с тактовой частотой не менее 2Ггц;

Оперативную память объемом не менее 4 Гб;

Свободного места на SSD или HDD не менее 10 Гб;

Минимум встроенное графическое ядро в процессор или видеокарту;

операционную систему Windows 10;

.NET Framework 4.6.1 версии

* 1. **Общие характеристики входной информации**

Входная информация будет содержать сведения о пользователе.

* 1. **Общие характеристики выходной информации**

Выходная информация будет содержать сведения о сеансах, пользователях и на какие сеансы куплены билеты.

.

* 1. **Требования и условия организационного характера**

Для обеспечения актуальной работы базы данных персонал должен систематически обновлять данные, заносить изменения в документы, отвечающие за сеансы.

* 1. **Требования и условия технического характера**

Для работы программного обеспечения базы данных «Кинотеатр» должна быть установлена платформа СУБД MAMP .

1. **Описание задачи**
   1. **Определение задачи**

Определением задачи является задание на выполнение курсового проекта. Организация, утвердившая задание на разработку программного продукта: Колледж ПсковГУ.

Наименование работы: Автоматизированная информационная система «Кинотеатр»

* 1. **Методы решения задачи**

Решение задачи путём создания автоматизированной информационной системы «Кинотеатр» на платформе .Net Framework.

1. **Входные и выходные данные**
   1. **Сведения о входных данных**

Выходная информация будет содержать сведения о сеансах, пользователях и на какие сеансы куплены билеты.

* 1. **Сведения о выходных данных**

Выходная информация будет содержать сведения о сеансах, пользователях и на какие сеансы куплены билеты.

.

1. Описание языка

C# – это современный объектно-ориентированный и типобезопасный язык программирования. C# позволяет разработчикам создавать разные типы безопасных и надежных приложений.

Программы C# выполняются в .NET, виртуальной системе выполнения, вызывающей общеязыковую среду выполнения (CLR) и набор библиотек классов. Среда CLR — это реализация общеязыковой инфраструктуры языка (CLI), являющейся международным стандартом, от корпорации Майкрософт. CLI является основой для создания сред выполнения и разработки, в которых языки и библиотеки прозрачно работают друг с другом.

C# относится к широко известному семейству языков C, и покажется хорошо знакомым любому, кто работал с C, C++, Java или JavaScript.

C# – это объектно- и компонентно-ориентированный язык программирования. Он предоставляет языковые конструкции для непосредственной поддержки такой концепции работы. Благодаря этому C# подходит для создания и применения программных компонентов. С момента создания язык C# обогатился функциями для поддержки новых рабочих нагрузок и современными рекомендациями по разработке ПО.

C# позволяет создавать надежные и устойчивые приложения. Среда выполнения имеет встроенную сборка мусора - автоматически освобождает память, занятую недоступными или неиспользуемыми объектами. Типы, допускающие значение null, обеспечивают защиту от переменных, которые не ссылаются на выделенные объекты. Обработка исключений предоставляет структурированный и расширяемый подход к обнаружению ошибок и восстановлению после них. Лямбда-выражения поддерживают приемы функционального программирования. Синтаксис LINQ создает общий шаблон для работы с данными из любого источника. Поддержка языков для асинхронных операций предоставляет синтаксис для создания распределенных систем. В C# действует единая система типов. Все типы C#, включая типы-примитивы, такие как int и double, наследуют от одного корневого типа object. Все типы используют общий набор операций, а значения любого типа можно хранить, передавать и обрабатывать схожим образом. Более того, C# поддерживает как определяемые пользователями ссылочные типы, так и типы значений. C# позволяет динамически выделять объекты и хранить упрощенные структуры в стеке. C# поддерживает универсальные методы и типы, обеспечивающие повышенную безопасность типов и производительность. C# предоставляет итераторы, которые позволяют разработчикам классов коллекций определять пользовательские варианты поведения для клиентского кода.

В C# особое внимание уделяется управлению версиями для обеспечения совместимости программ и библиотек при их изменении. Вопросы управления версиями существенно повлияли на такие аспекты разработки C#, как раздельные модификаторы virtual и override, правила разрешения перегрузки методов и поддержка явного объявления членов интерфейса.

Подводя итоги, можно сказать, что язык C# имеет множество преимуществ начиняя от отсутствия необходимости следить за выделенной памятью и заканчивая лёгкостью и скоростью написания программ.

## Заключение

Для выполнения данного курсового проекта использовались инструментальные средства разработки программного обеспечения, такие как:

Erwin Data modeler для создания логической структуры программы;

Платформа .NET Framework для создания автоматизированной системы;

Так же в ходе выполнения курсового проекта разработан интерфейс пользователя, произведено тестирование и отладка готовой программы и написана программная документация.

Все поставленные задачи выполнены, знания и навыки, касающиеся инструментальных средств разработки программного обеспечения, создания программной документации, создания моделей данных, стадий процесса разработки программного продукта и стадий жизненного цикла, применены в полном объеме.

### Список источников информации

#### Основные

1. Гриффитс Иэн. Программируем на C# 8.0. Разработка приложений. – Издательский дом "Питер", 2021-944c.

2. Прайс Марк. C# 8 и .NET Core. Разработка и оптимизация, 2021-816c.

3. Джейсон Аллс. Чистый код в C#, 2020-500c.

#### Дополнительные источники

1. Котенко В. Н., Котенко Ю. В. Программирование на языках низкого уровня. Курс лекций. – 2016-621c.

2. Терехов А.Н. RTST-технология программирования встроенных систем реального времени //Записки семинара кафедры системного программирования" CASE-средства RTST. – 1998. – №. 1. – С. 3-17-235c.

3. Документация Microsoft Visual Studio [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/ru-ru/visualstudio/windows/?view=vs-2022

4. Habr «Инструментарий для анализа и отладки .NET приложений» [Электронный ресурс]. – Режим доступа: https://habr.com/ru/company/clrium/blog/463305/

5. Habr «12 инструментов для отладки .NET-приложений по производительности и по памяти» [Электронный ресурс]. – Режим доступа: https://habr.com/ru/company/jugru/blog/328294/