**Методические указания**

**по организации занятий учебной практики**

**ПМ.05 Проектирование и разработка информационных систем**

**Специальность**

*09.02.07 «Информационные системы и программирование»*

Квалификация Разработка веб и мультимедийных приложений

Методические рекомендации по организации занятий учебной практики по **ПМ.05 Проектирование и разработка информационных систем** разработаны с учетом ФГОС среднего профессионального образования специальности 09.02.07 «Информационные системы и программирование», предназначены для студентов и преподавателей колледжа.

Методические указания определяют этапы выполнения работы на практическом занятии, содержат рекомендации по выполнению индивидуальных заданий, а также список рекомендуемой литературы.

Составитель (автор): Д.А. Косыченко преподаватель колледжа ЭУП

Рассмотрены на заседании предметной (цикловой) комиссии специальности 09.02.04 Информационные технологии (по отраслям), 09.02.05 Прикладная информатика (по отраслям) и 09.02.07 Информационные системы и программирование

Протокол № 9 от 30 июня 2022 г

Председатель П(Ц)К специальности \_\_\_\_\_\_\_\_\_\_\_ С.В. Шинакова

личная подпись

и одобрены решением учебно-методического совета колледжа.

Протокол № 8 от 30 июня 2022 г

Председатель учебно-методического совета колледжа

\_\_\_\_\_\_\_\_\_\_\_ С.В. Шинакова

личная подпись

Рекомендованы к практическому применению в образовательном процессе.

Рецензенты:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

место работы должность ФИО

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

место работы должность ФИО

# Тема 1 Применение основ проектирования информационных систем

Составление технического задания на разработку веб-сайта. Разработка информационной архитектуры и дизайна сайта.

Обучающийся должен

*знать:*

* основные виды и процедуры обработки информации, модели и методы решения задач обработки информации;
* основные модели построения информационных систем, их структуру, особенности и области применения;
* национальную и международную систему стандартизации и сертификации и систему обеспечения качества продукции, методы контроля качества;
* сервисно-ориентированные архитектуры;
* спецификации языка программирования, принципы создания графического пользовательского интерфейса (GUI), файлового ввода-вывода, создания сетевого сервера и сетевого клиента;

*уметь:*

* осуществлять постановку задачи по обработке информации;
* выполнять анализ предметной области;
* использовать стандарты при оформлении программной документации.

*выполнить:*

* анализ предметной области;
* выполнить работы предпроектной стадии;
* разработать проектную документацию на информационную систему;
* использовать стандарты при оформлении программной документации;
* подготовить графические элементы, аудио- и видеоматериалы.

## Задание 1.

Составление технического задания на разработку веб-сайта

Техническое задание – первый и самый важный шаг на пути создания сайта. Это документ на основе которого Исполнитель разрабатывает сайт, а Заказчик оценивает качество готового продукта.

Составление технического задания в соответствии со структурой, приведенной ниже (подробное содержание пунктов приведено в ГОСТ 34-602-89):

1. Общие сведения
   1. Наименование программы
   2. Шифр договора
   3. Наименование Разработчика и Заказчика системы и их реквизиты
   4. Перечень документов, на основании которых создается система, кем и когда они утверждены
   5. Плановые сроки начала и окончания работы по созданию системы
   6. Сведения об источниках и порядке финансирования работ
   7. Порядок оформления и предъявления заказчику результатов по созданию системы
2. Назначение и цели создания системы
   1. Назначение системы (вид автоматизируемой деятельности, перечень объектов автоматизации, на которых предполагается ее использовать)
   2. Цели создания системы (наименование и требуемые значения технических, технологических, производственно-экономических или других показателей объекта автоматизации, которые должны быть достигнуты в результате создания АС, и указывают критерии оценки достижения целей создания АС)
3. Характеристика объектов автоматизации
   1. Краткие сведения об объекте автоматизации или ссылки на документы, содержащие такую информацию
   2. Сведения об условиях эксплуатации объекта автоматизации и характеристиках окружающей среды
4. Требования к системе
   1. Требования к системе в целом
   2. Требования к функциям, выполняемым системой
   3. Требования к видам обеспечения
5. Состав и содержание работ по созданию системы
   1. перечень документов по ГОСТ 34.201. предъявляемых по окончании соответствующих стадий и этапов работ
   2. вид и порядок проведения экспертизы технической документации (стадия, этап, объем
   3. проверяемой документации, организация-эксперт);
   4. программу работ, направленных на обеспечение требуемого уровня надежности разрабатываемой системы (при необходимости);
   5. перечень работ по метрологическому обеспечению на всех стадиях создания системы с указанием их сроков выполнения и организаций – исполнителей (при необходимости).
6. Порядок контроля и приемки
   1. виды, состав, объем и методы испытаний системы и ее составных частей (виды испытаний в соответствии с действующими нормами, распространяющимися на разрабатываемую систему);
   2. общие требования к приемке работ по стадиям (перечень участвующих предприятий и организаций, место и сроки проведения), порядок согласования и утверждения приемочной документации;
   3. статус приемочной комиссии (государственная, межведомственная, ведомственная).
7. Требования к составу и содержанию работ по подготовке объекта автоматизации к вводу системы в действие
   1. приведение поступающей в систему информации (в соответствии с требованиями к информационному и лингвистическому обеспечению) к виду, пригодному для обработки с помощью ЭВМ;
   2. изменения, которые необходимо осуществить в объекте автоматизации;
   3. создание условий функционирования объекта автоматизации, при которых гарантируется соответствие создаваемой системы требованиям, содержащимся в ТЗ;
   4. создание необходимых для функционирования системы подразделений и служб;
   5. сроки и порядок комплектования штатов и обучения персонала.
8. Требования к документированию
   1. согласованный разработчиком и заказчиком системы перечень подлежащих разработке комплектов и видов документов, соответствующих требованиям ГОСТ 34.201 и Н'ГД отрасли заказчика; перечень документов, выпускаемых на машинных носителях; требования к микрофильмированию документации;
   2. требования по документированию комплектующих элементов межотраслевого применения в соответствии с требованиями ЕСКД и ЕСПД;
   3. при отсутствии государственных стандартов, определяющих требования к документированию элементов системы, дополнительно включают требования к составу и содержанию таких документов.
9. Источники разработки

документы и информационные материалы (технико-экономическое обоснование, отчеты о законченных научно-исследовательских работах, информационные материалы на отечественные, зарубежные системы-аналоги идр.), на основании которых разрабатывалось ТЗ и которые должны быть использованы при создании системы.

1. Приложения
   1. расчет ожидаемой эффективности системы;
   2. оценку научно-технического уровня системы.

## Задание 2.

Разработка информационной архитектуры и дизайна сайта

Пример. Описать логическую и физическую структуру сайта на тему «Туристическое агентство».

Сайт туристического агентства будет состоять из следующих групп и подгрупп на сайте (рисунок 1).

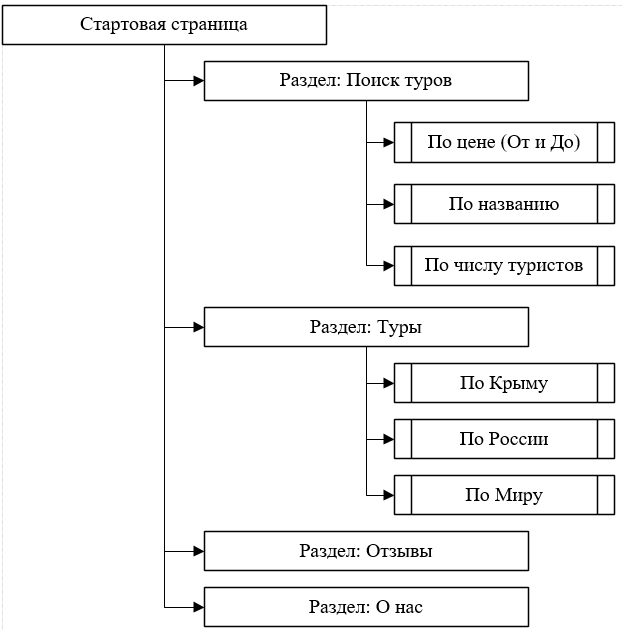


Рисунок 1 – Логическая структура сайта

Каждый тур содержит информацию: рейтинга тура, описания тура (название города),список мест посещения,стоимость билетов,что включено,трансфер, стоимость питания (всё включено!?),классность номера гостиницы (стоимость проживания),количество дней пребывания и т.д.,стоимость экскурсий по городу, дополнительная информация о туре.

Перейдя к выбору конкретного тура, пользователь может перейти к непосредственной покупке билетов на тур (название тура, краткая информация, количество билетов, стоимость и цена).

Физическая структура сайта:

На рисунке 2 показана физическая структура сайта.

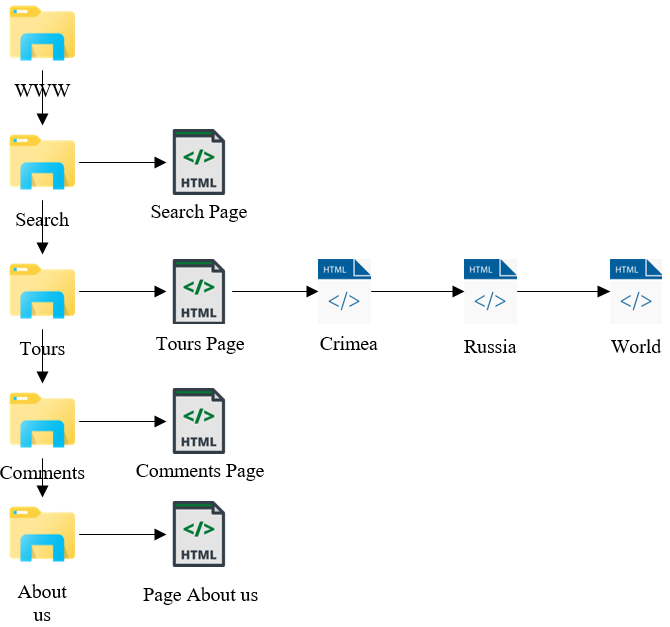
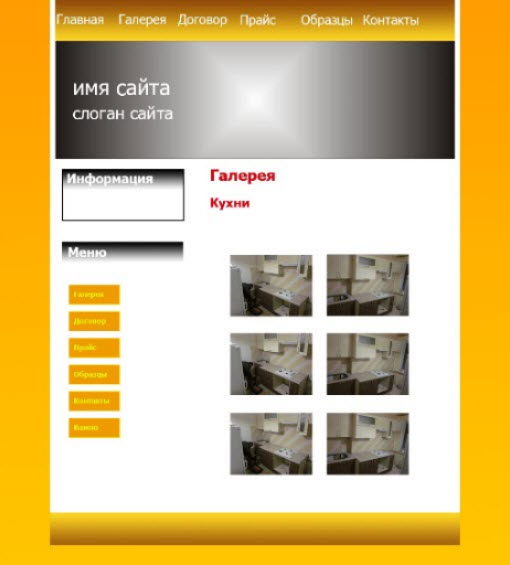


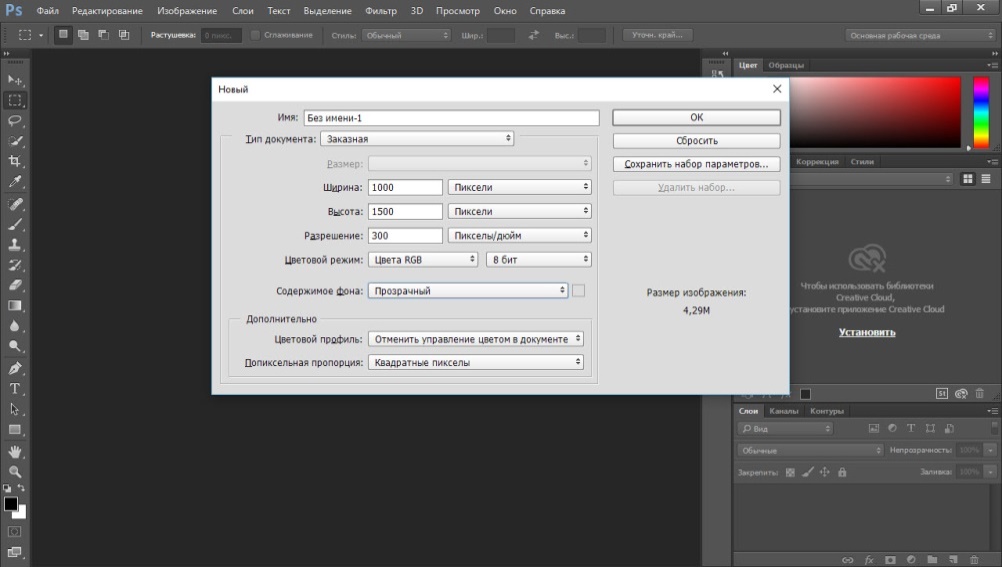
Рисунок 2 – Физическая структура сайта

Разработать дизайн сайта по примеру.



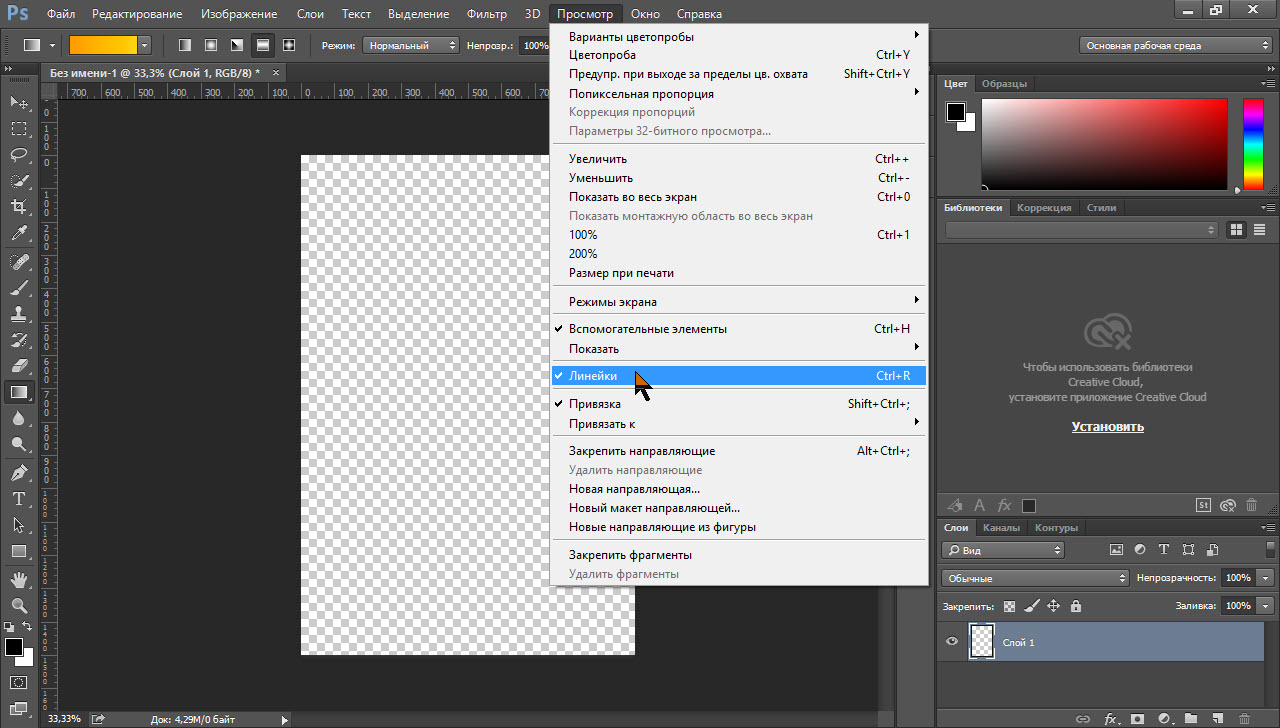
1. **Открываем Photoshop** и создаём в нём новый документ (**Файл -> Создать** или **Ctrl+N**).

2. **Устанавливаем начальные параметры.** На практике продумать дизайн с точностью до пикселя практически невозможно — в процессе вёрстки обязательно нужно будет что-то поменять, переместить, переделать. Поэтому размеры можно задавать приблизительные, причём ширину и высоту документа желательно указывать заведомо больше планируемых размеров сайта, чтобы в макете точно уместились все элементы. Наш документ сделаем 1000 пикселей шириной и 1500 пикселей высотой (планшет – 768 x 1024 пикселей, смартфон – 320 x 480 пикселей). Обратите внимание: пикселей, а не сантиметров. Другие параметры можно не трогать.



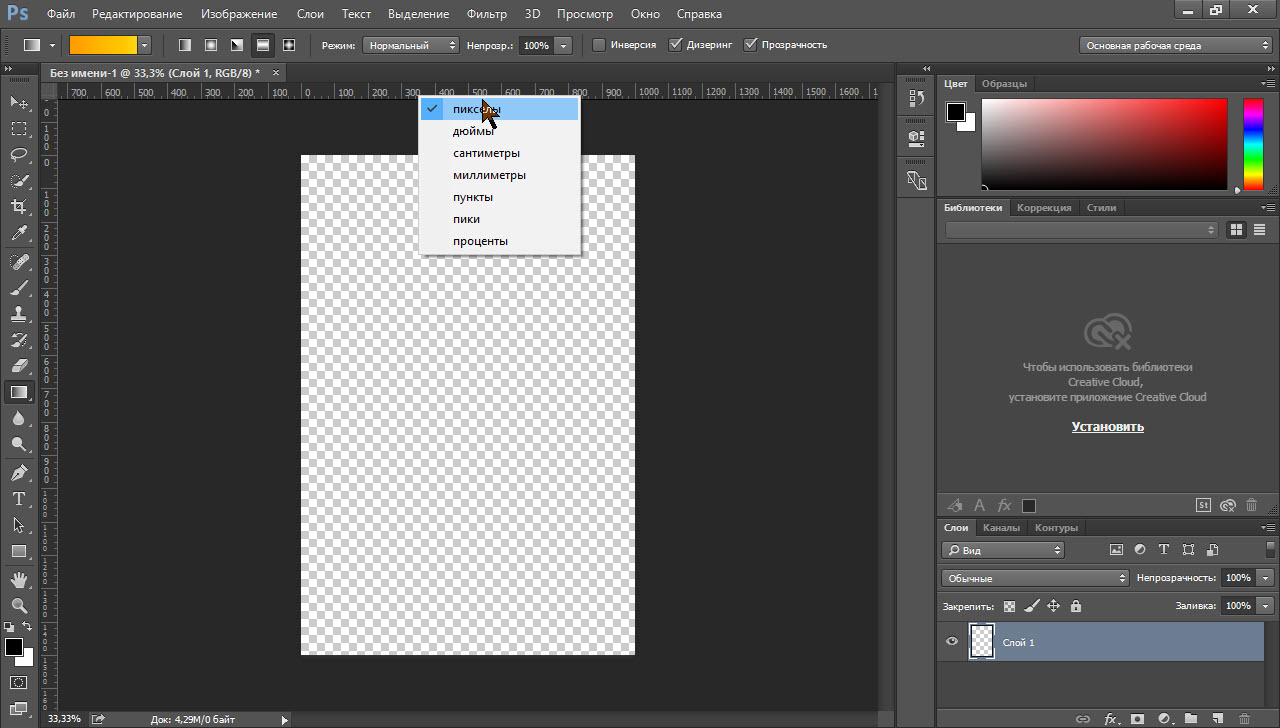
3. **Включаем линейки.** Во время работы они понадобятся, ведь линейки позволяют очень точно отмерять расстояния. Проверьте, включены ли линейки у вас. Если да, то вы увидите шкалы рядом с левой и под верхней панелью инструментов.

Если линеек нет — включите их (**Просмотр -> Линейки** или **Ctrl+R**).



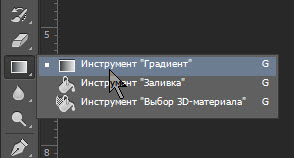
Линейки должны показывать величину в пикселях. Чтобы переключиться на них с другой меры длины, щёлкните на линейке правой кнопкой мыши и в открывшемся меню установите соответствующий флажок.

**Очень удобно использовать сетку Bootstrap, в которой заранее заданные размеры колонок, которые можно сразу же использовать, например ширина колонки 140 px (посмотреть видеоурок «Как в фотошопе сделать сетку бутстрап (bootstrap grid)» или Открыть Photoshop-Перейти на вкладку операции-Открыть папку**hello_html_m3abd2341.png**и перенести файл *960\_GRIDS* на вкладку Операции-после этого выбрать соответствующую сетку (12,16,24)).**



4. Проверяем, чтобы размер шрифта был указан в пикселях, а не в пунктах. Если настроено не так, идём в **Редактирование -> Настройки -> Основные**, в отобразившемся окне переходим на вкладку **Единицы измерения и линейки**, в выпадающем списке **Текст** выбираем **Пиксели** и нажимаем **OK**.

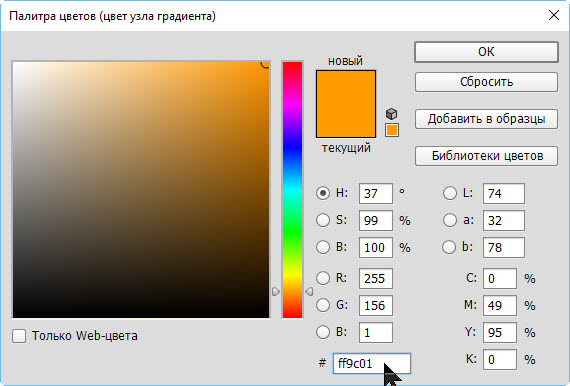
5. Сразу делаем подложку сайта. У нас это градиентная заливка оранжевого, переходящего в жёлтый цвет. На панели слева выбираем инструмент **Градиент**.



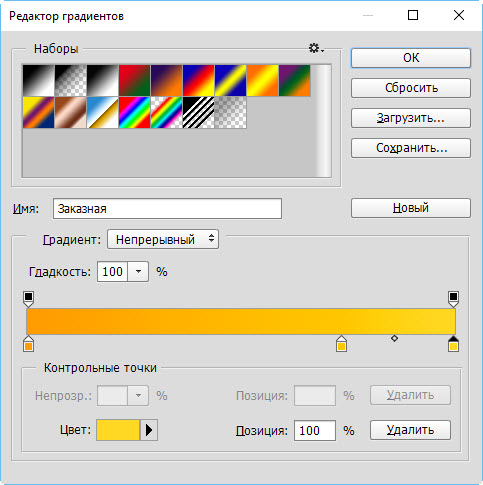
На появившейся вверху панели жмём кнопку **Зеркальный градиент**, выбираем цвет на палитре левее.

hello_html_498b882a.jpg

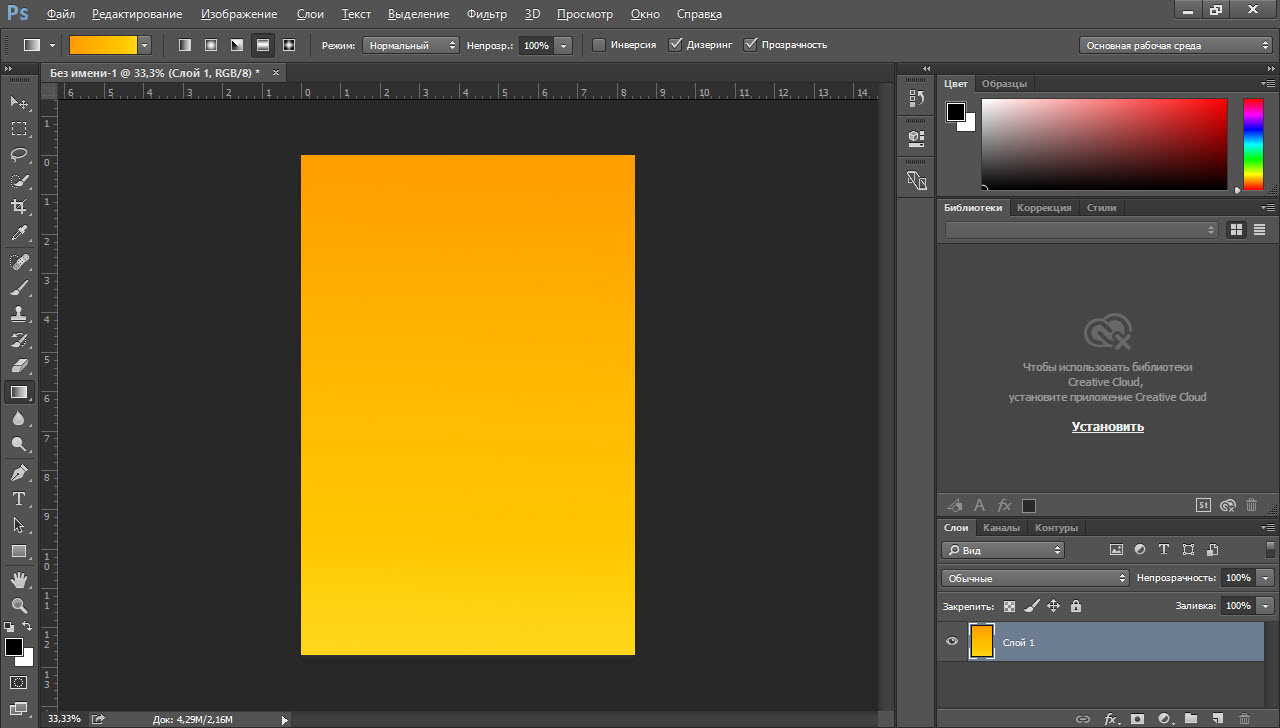
Используя инструменты открывшегося окна **Редактор градиентов**, выбираем нужные цвета. Чтобы задать точный цвет контрольной точки, щёлкните на ней, нажмите кнопку **Цвет** и в окне палитры цветов укажите его в формате RGB, HSB, CSS или любом другом из доступных.



В результате манипуляций градиент получился следующим.



Чтобы применить градиент к текущему слою, проводим над ним указателем, удерживая при этом нажатой левую кнопку мыши.



6. Сохраняем подложку в файл. То, что мы сделали, должно отображаться под основной страницей и занимать всё окно браузера целиком — своего рода подложка. Например, ширина сайта — 800 пикселей, а разрешение экрана у пользователя гораздо больше. Оставшееся пространство (всё, кроме тех самых 800px, которые будут заняты блоком страницы) заполнится градиентным фоном.

Так как разрешение экрана нельзя предугадать, из созданного фона можно вырезать полоску толщиной в один пиксель и сохранить как картинку. Браузер будет заполнять задний фон ею по всей ширине.

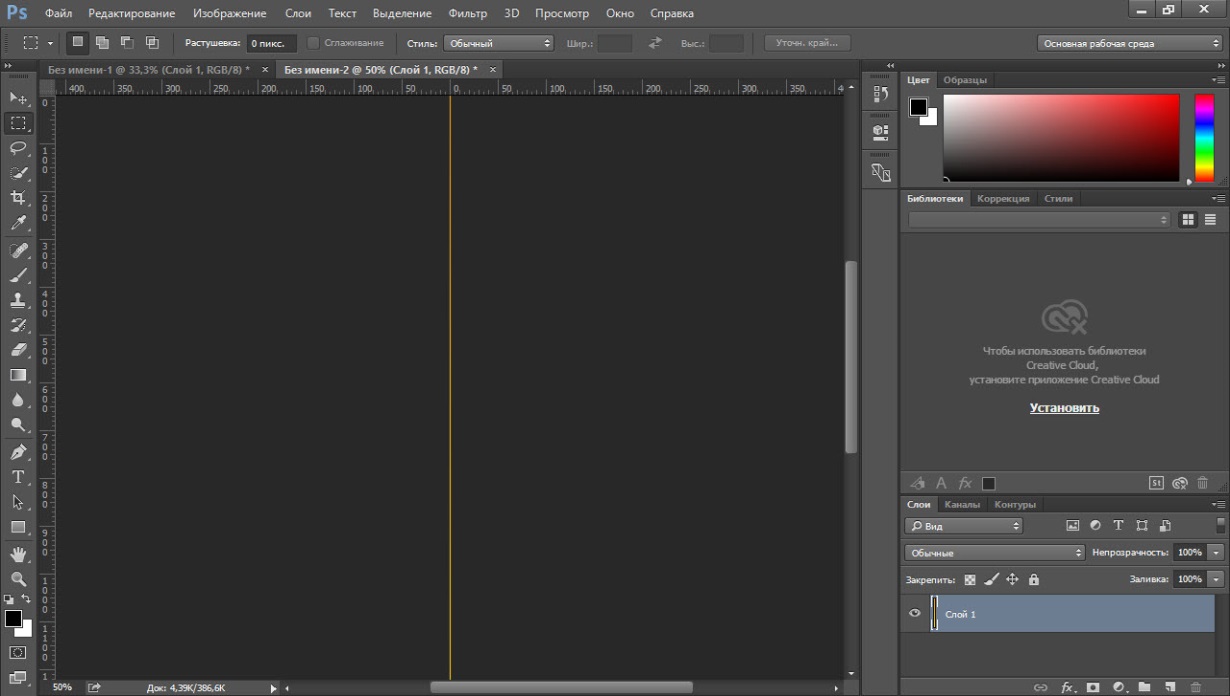
Сохранить такую узкую полоску несложно.

6.1. Выбираем инструмент **Прямоугольная область**.

6.2. Выделяем полоску произвольной ширины, но по всей длине слоя.

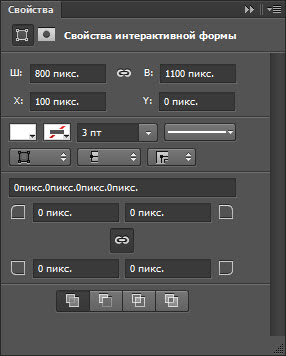
6.3. Копируем выделенную область (**Ctrl+C**).

6.4. Создаём новый документ (**Ctrl+N**), устанавливаем для него ширину 10 пикселей и вставляем скопированное (**Ctrl+V**).

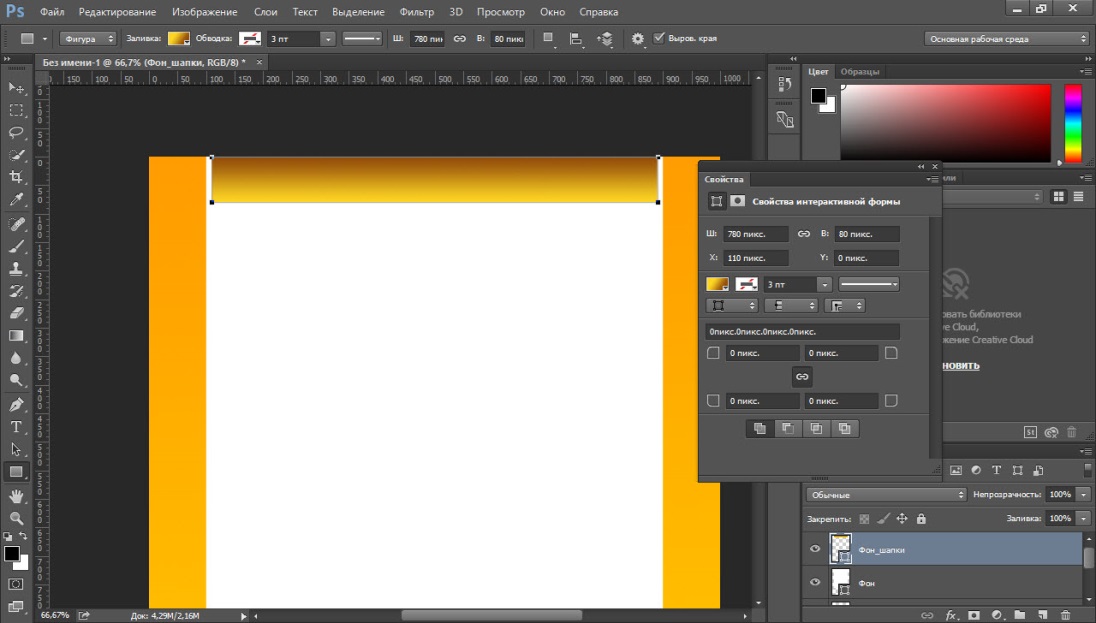


6.5. Сохраняем файл в **JPG**-формате.

7. Создаём фон страницы. Фоном будет простой белый цвет. Выбираем инструмент **Прямоугольник** и в окне свойств задаём нужные параметры. У нас получился прямоугольник 800x1100 пикселей, левый верхний угол которого лежит в точке 100,0.



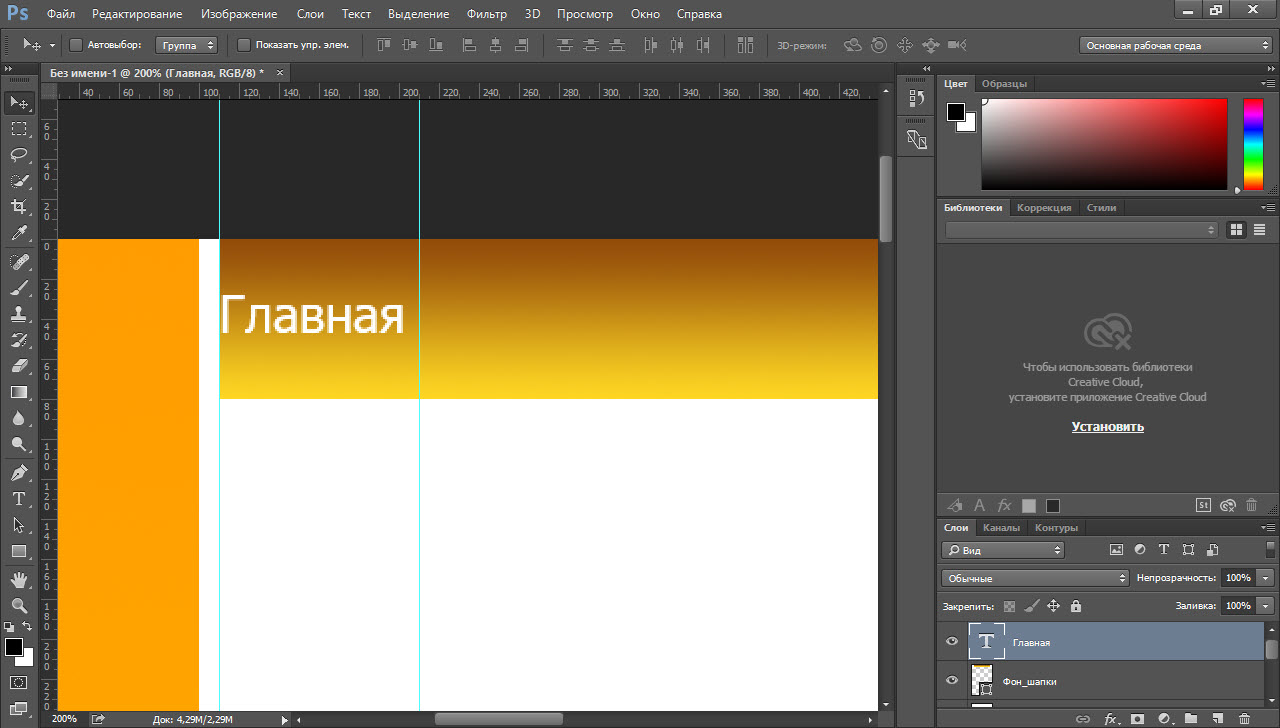
8. Делаем фон шапки. Градиентная заливка, похожая на подложку, размерами 780x80px.



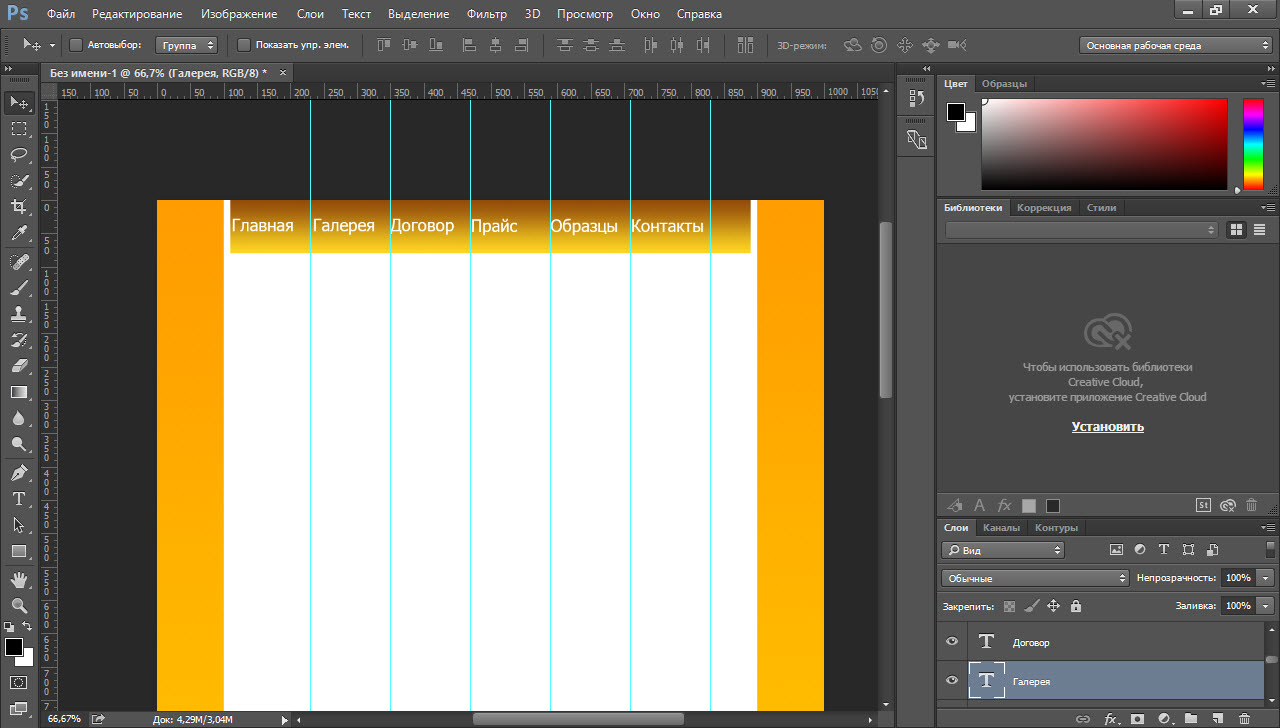
Сохраняем его отдельным графическим файлом шириной в 1 пиксель, как делали это с подложкой.

9. Создаём верхнее меню. С помощью инструмента Горизонтальный текст добавляем на макет первый пункт меню — **Главная**. Слой создастся и даже переименуется автоматически, так что с ним можно ничего не делать.

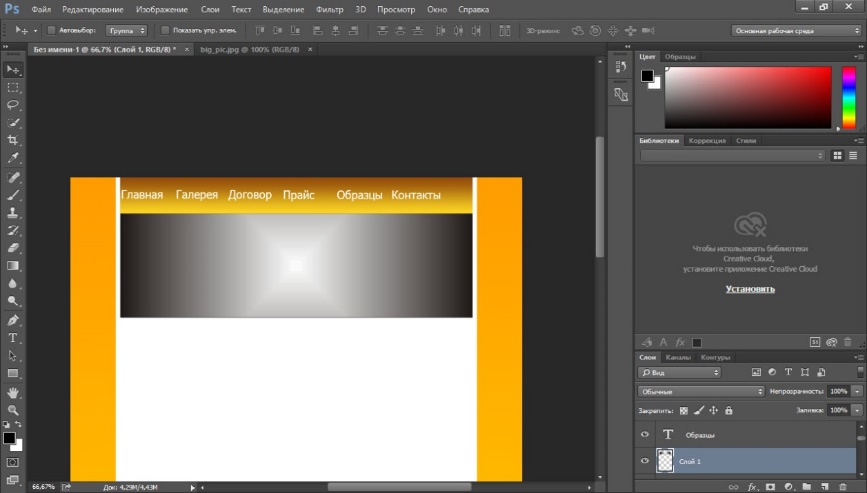
Здесь важно, чтобы все пункты распределились равномерно, поэтому крайне рекомендуется использовать линейки. Чтобы вытянуть вертикальную, проведите указателем, удерживая при этом нажатой кнопку мыши, слева направо. С помощью линеек отмеряйте расстояние, учитывая, что каждый пункт меню должен занимать в нашем случае 120 пикселей.



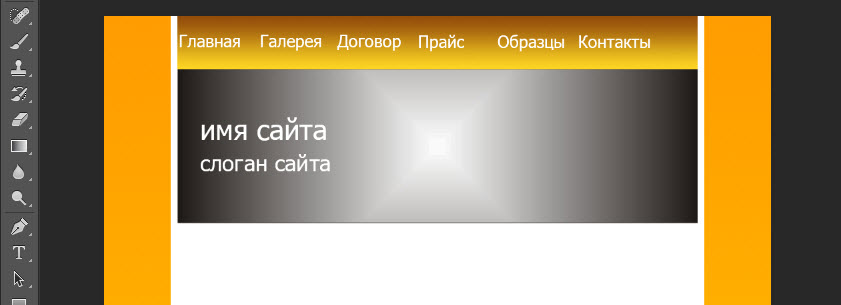
10. Аналогично вставляем остальные пункты меню.



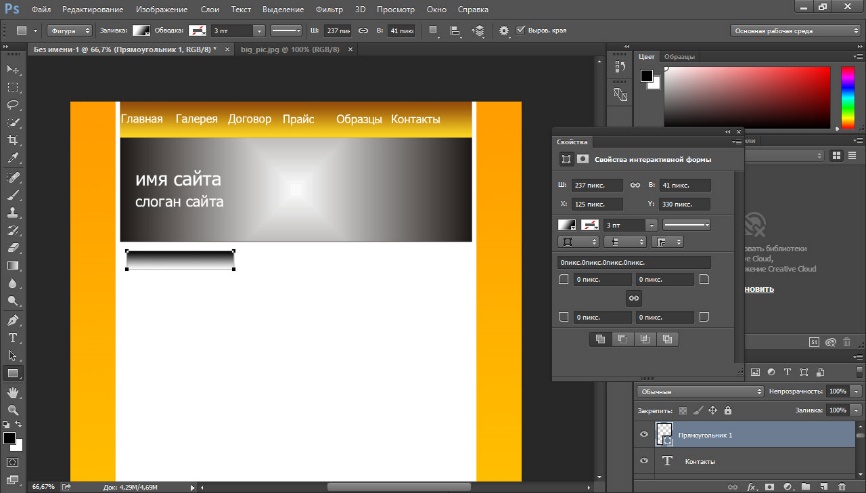
11. Добавляем логотип. У нас уже есть готовый, поэтому его остаётся только аккуратно вставить в макет. Для этого нажимаем **Файл -> Открыть**, затем щёлкаем на изображении и, удерживая нажатой кнопку мыши, перемещаем его на заголовок документа-шаблона, когда он откроется, перетаскиваем картинку в нужное место макета и отпускаем кнопку мыши.



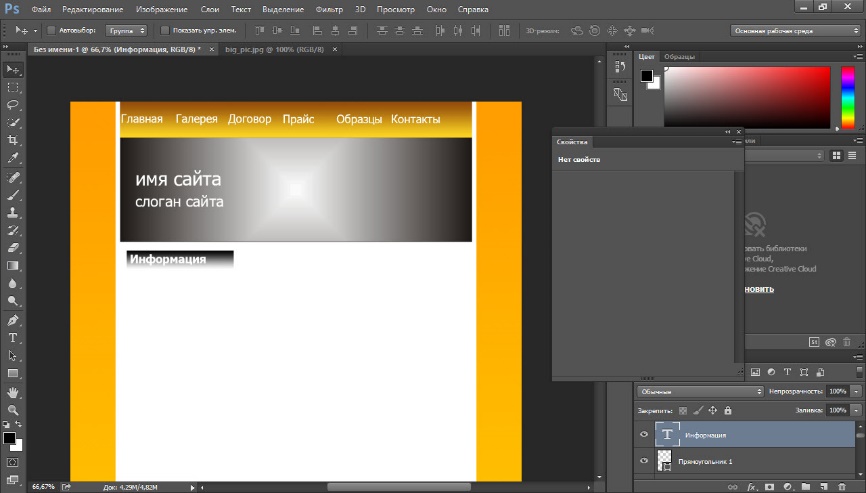
12. Пишем имя и слоган сайта. Уже известным нам инструментом **Горизонтальный текст** дополняем уже почти созданный логотип надписями.



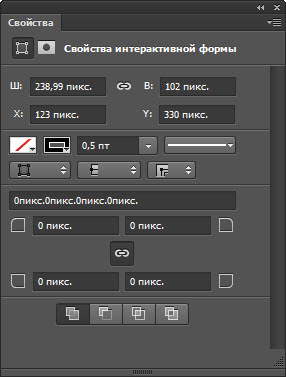
13. Переходим к боковой панели. Для начала создадим и сохраним отдельным файлом градиентную заливку для её заголовков. С инструментами знакомы, пояснения не требуются.



14. Добавляем на только что созданный градиент текст-заголовок информационного блока.

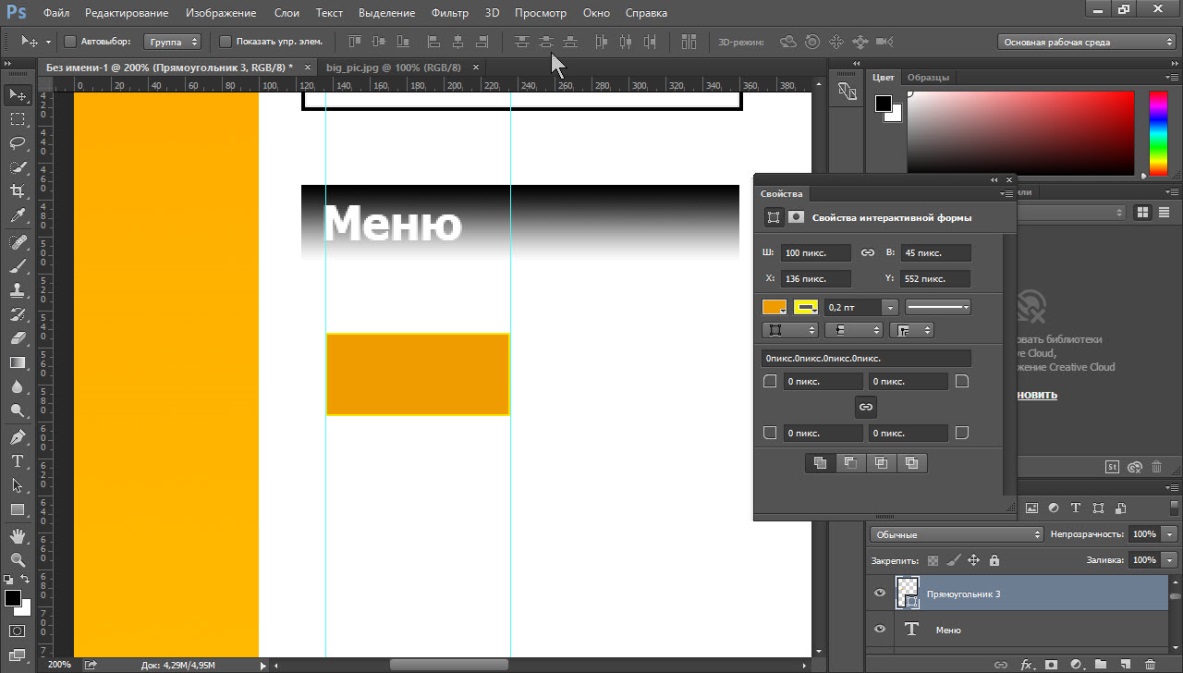


15. Рисуем обрамление области. Для этого достаточно использовать прозрачный прямоугольник с чёрными линиями контура. Выбираем инструмент Прямоугольник, задаём тип заливки фигуры Нет цвета, щёлкаем значок Задать тип штриха фигуры и выбираем чёрный цвет, иначе линий не будет. Если контур получился слишком толстым, устанавливаем ширине линий значение 0,5 пт.

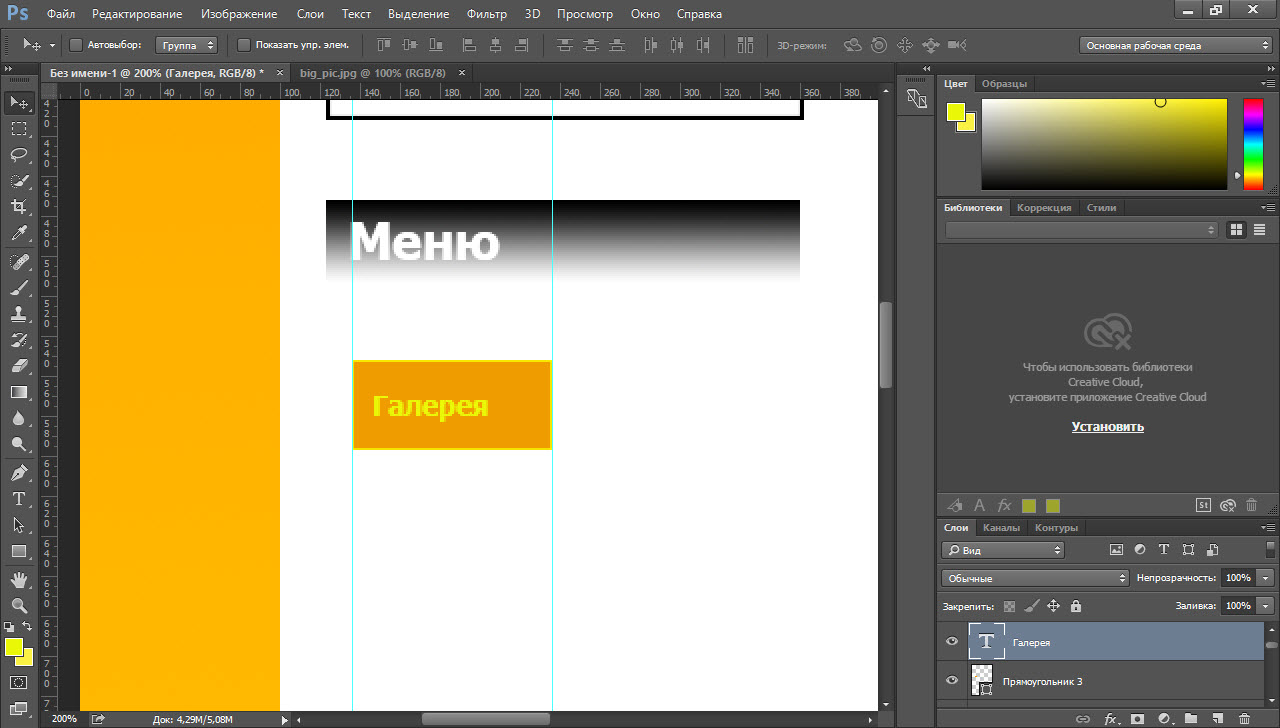


16. Ниже добавляем заголовок блока с градиентной заливкой, как в п. 12-13.

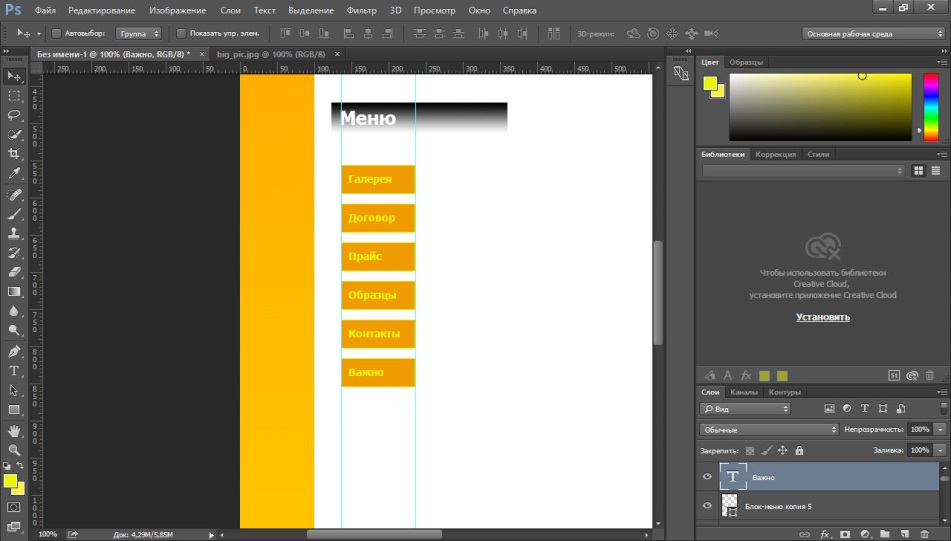
17. Создаём блок меню левой панели. Добавляем оранжевый прямоугольник шириной 100px с жёлтым контуром 0,2 пт.



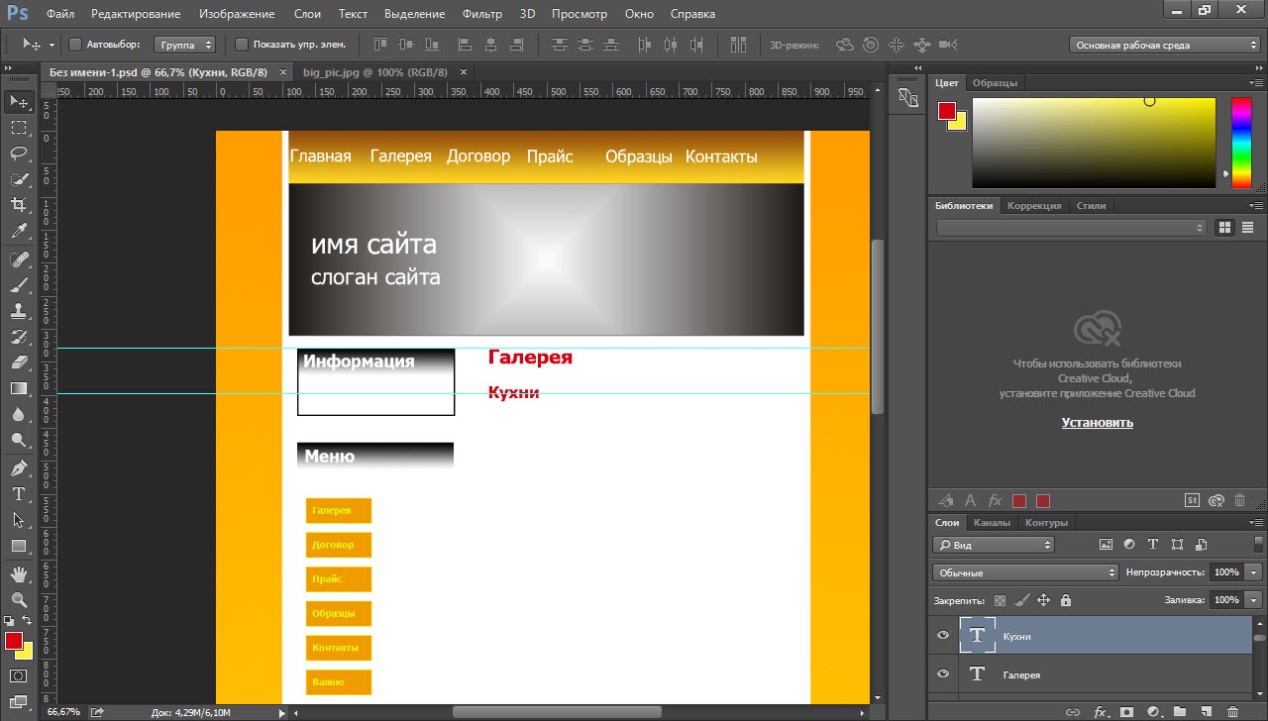
18. Добавляем на него текст.



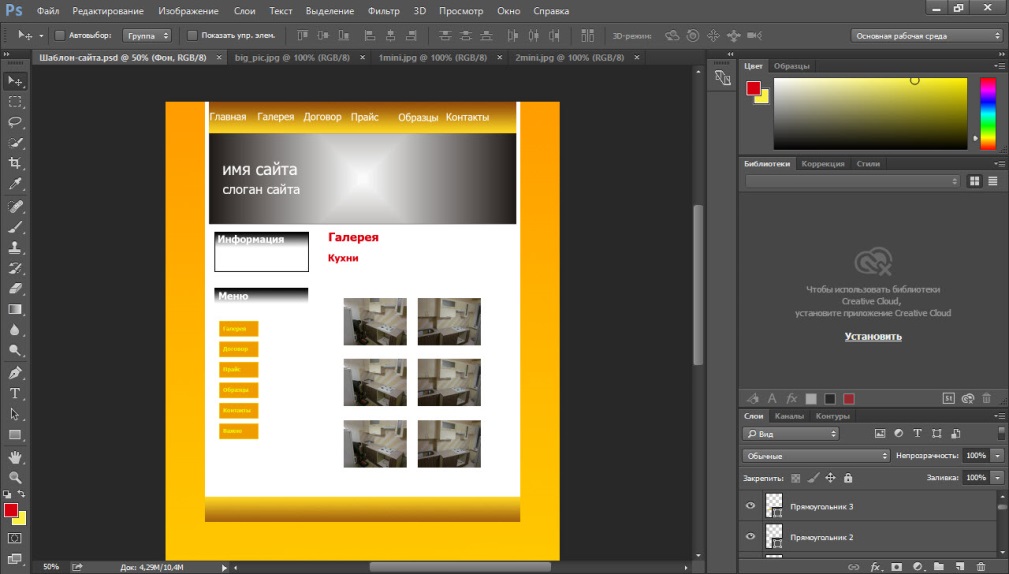
19. С помощью дублирования слоёв и линеек создаём ещё пять пунктов меню левой панели.



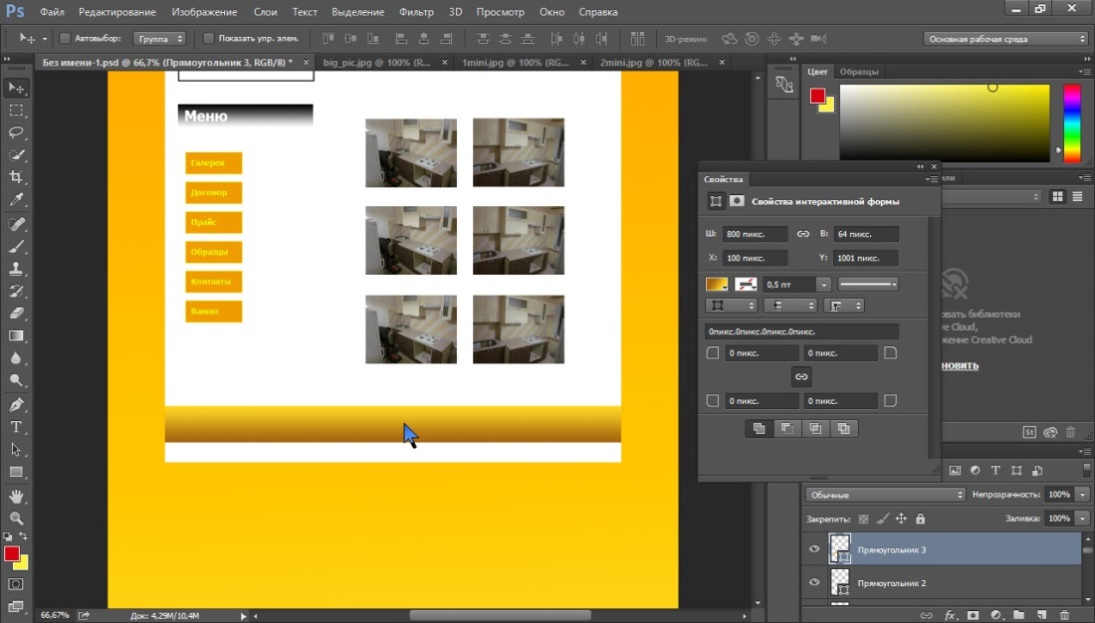
20. Прописываем текст в основной части страницы, используя всё тот же инструмент **Горизонтальный текст**.



21. Добавляем фото в основную часть страницы, как мы делали это с логотипом. Для копирования изображения просто перемещайте его мышью, удерживая при этом нажатой клавишу **Alt**. Если вдруг картинка не подходит по размеру, используйте инструмент Трансформация (**Ctrl+T**).

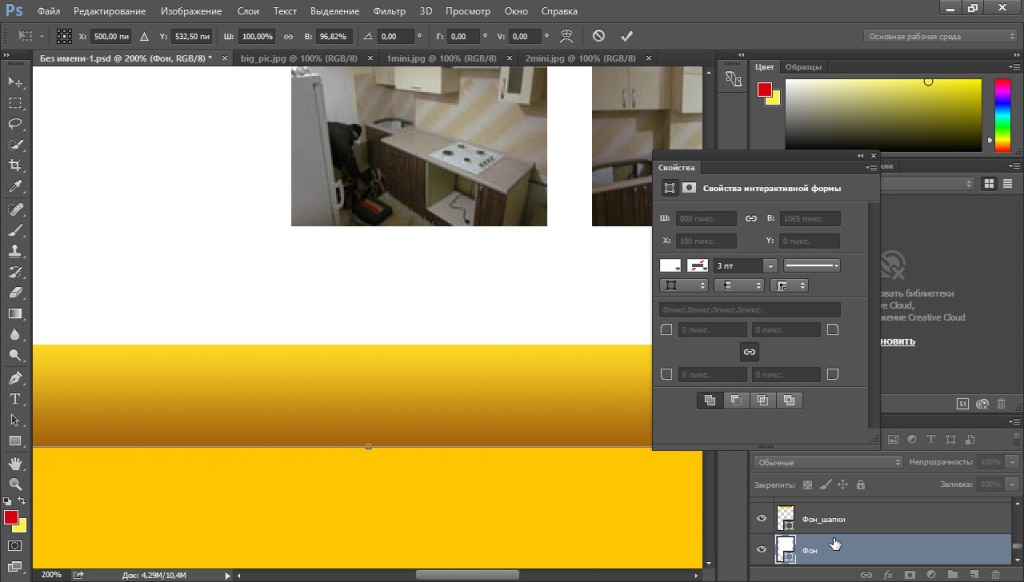


22. Рисуем фон нижней части сайта — оранжевый градиент длиной 64 пикселя.

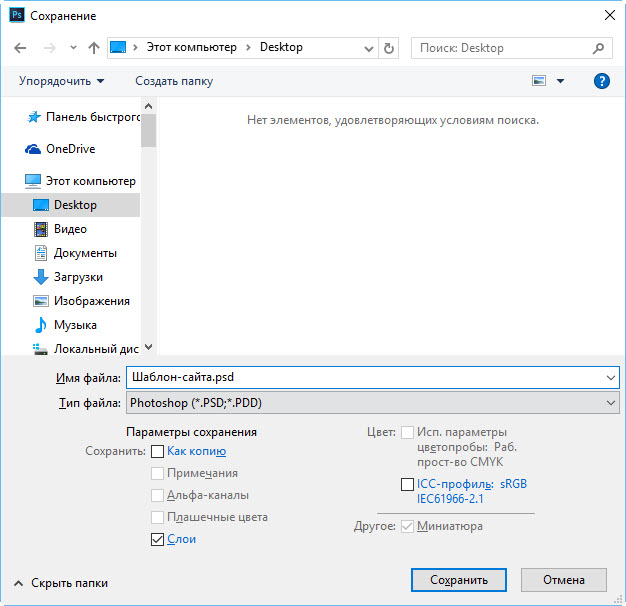


23. Сохраняем полоску нижнего фона шириной 1 пиксель в отдельный графический файл.

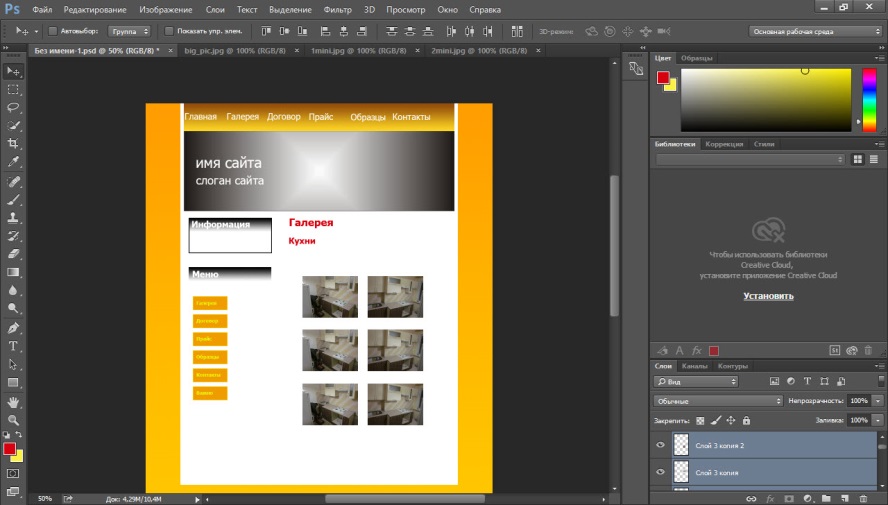
24. Уменьшаем высоту страницы. Получилось так, что все элементы уже прорисованы, а лишнее место ещё осталось. Вот тут-то нам и пригодились осмысленные имена слоёв. Среди прочих выбираем фоновый слой (у нас он так и называется — Фон) и с помощью инструмента Трансформация уменьшаем высоту нашего белого прямоугольника до нижнего края футера страницы.



25. Сохраняем шаблон в файл формата**.PSD** (**Файл -> Сохранить**).



26. Результатом сего действа и стал ещё простой, но уже нормально выглядящий шаблон сайта.



Теперь из PSD-макета остаётся средствами HTML/CSS сверстать шаблон сайта.

# Тема 2 Применение системы обеспечения качества информационных систем

Реинжиниринг методом интеграции. Разработка требований безопасности информационной системы

Обучающийся должен

*знать:*

* методы и средства проектирования, разработки и тестирования информационных систем;
* реинжиниринг бизнес-процессов.

*уметь:*

* разрабатывать требования безопасности;

*выполнить:*

* реинжиниринг методом интеграции;
* разработать требования безопасности.

## Задание 1.

Реинжиниринг методом интеграции

1. Построить в одной из нотаций диаграмму «as-is» отобра- жающую исходное содержание заданного бизнес-процесса.

2. Выделить деятельности (процессы второго уровня) интегра- ция которых возможна.

3. Сформировать интегрированные деятельности, на основе выявленных групп интегрируемых процессов. Описать данные про- цессы.

4. Построить диаграмму «to-be» отображающую полученное содержания рассматриваемого бизнес-процесса.

## Задание 2.

Разработка требований безопасности информационной системы

Анализ деловой ситуации:

Охарактеризуйте позицию организации на рынке.

1. Проведите аудит бизнес-процессов.

2. Оцените уровень непротиворечивости бизнес-требований к модулям информационной системы.

3. Какие инновационные технологии сферы ИТ требуется внедрить в бизнес- процессы организации?

4. Каковы перспективные направления реинжиниринга отдельных бизнес-процессов на предприятии?

1. Описать технико-экономические характеристики предприятия.

2. Разработка стратегии предприятия. Привести бизнесс-направления деятельности предприятия. Определить необходимость применения реинжиниринга.

3. Разработка предложений по усовершенствованию системы управления предприятием. Описание всех бизнес-процессов отдела или предприятия. Представить технологическую карту бизнес-процессов.

Технологическая карта бизнесс-процессов

|  |  |  |
| --- | --- | --- |
| Процесс | Информация | Механизм |
| Управление предприятием | Информация о внутренней и внешней среде предприятия | Директор |
| Управление снабжением | Информация о рынке поставщиков сырья и материалов. План деятельности предприятия | Служба снабжения |
| Управление финансами | Финансовая информация о предприятии. План деятельности предприятия. План закупок | Директор  Бухгалтерия |
| Управление производством | Заявки от покупателей, заказы. План закупленных сырья и материалов, годных к использованию. План деятельности предприятия | Директор  Служба производства  Отдел разработок |
| Управление сбытом | Запросы потребителей. Счет-фактура. План деятельности предприятия | Коммерческий отдел  Отдел сбыта и снабжения |

4. Построение функциональной модели, модели потоков данных существующих бизнес- процессов.

5. Анализ «узких мест».

6. Отбор бизнес-процессов для реинжиниринга.

7. Построение функциональной модели, модели потоков данных бизнес-процессов с предложениями по реинжинирингу.

8. Анализ экономической эффективности.

# Тема 3 Разработка документации информационных систем

Проектирование спецификации информационной системы. Разработка руководства по инсталляции программного средства. Разработка руководства пользователя программного средства

Обучающийся должен

*знать:*

* национальной и международной систему стандартизации и сертификации и систему обеспечения качества продукции, методы контроля качества.

*уметь:*

* разрабатывать проектную документацию на эксплуатацию информационной системы;

*выполнить:*

* разработку спецификации ИС;
* разработку руководства по инсталляции;
* разработать руководство пользователя.

## Задание 1.

Составить спецификацию ИС в соответствии со структурой, приведенной ниже:

1. Введение
   1. Цели
   2. Соглашения о терминах
   3. Предполагаемая аудитория и последовательность восприятия
   4. Масштаб проекта
   5. Ссылки на источники
2. Общее описание
   1. Видение продукта
   2. Функциональность продукта
   3. Классы и характеристики пользователей
   4. Среда функционирования продукта (операционная среда)
   5. Рамки, ограничения, правила и стандарты
   6. Документация для пользователей
   7. Допущения и зависимости
3. Функциональность системы
   1. Функциональный блок X (таких блоков может быть несколько)
      1. Описание и приоритет
      2. Причинно-следственные связи, алгоритмы (движение процессов, workflows)
      3. Функциональные требования
4. Требования к внешним интерфейсам
   1. Интерфейсы пользователя (UI)
   2. Программные интерфейсы (SI)
   3. Интерфейсы связи и коммуникации (CI)
5. Нефункциональные требования
   1. Требования к производительности
   2. Требования к сохранности (данных)
   3. Требования к безопасности системы
6. Приложения

## Задание 2.

Составить руководство по инсталляции в соответствии со структурой, приведенной ниже:

1. Общие сведения
   1. Назначение
   2. Минимальный состав технических средств
   3. Минимальный состав программных средств
2. Состав дистрибутива
3. Установка программы
4. Настройка программы
5. Проверка программы
6. Возможные ошибки

## Задание 3.

Составить руководство пользователя в соответствии со структурой, приведенной ниже:

1. Введение
2. Вход в систему
3. Ведение первичных данных
   1. Общие сведения
   2. Вход в подсистему просмотра и редактирования данных
   3. Ведение реестра автотранспортных предприятий
   4. Ведение реестра остановочных пунктов
   5. Ввод, редактирование, удаление информации
4. Просмотр информации

# Тема 4 Использование основных инструментов для создания, исполнения и управления информационной системой

Построение диаграммы Вариантов использования и диаграммы Последовательности. Построение диаграммы Кооперации и диаграммы Развертывания. Построение диаграммы компонентов. Построение диаграмм потоков данных и генерация кода

Обучающийся должен

*знать:*

* важность рассмотрения всех возможных вариантов и получения наилучшего решения на основе анализа и интересов клиента;
* методы и средства проектирования информационных систем;
* основные модели построения информационных систем, их структура.

*уметь:*

* работать с инструментальными средствами обработки информации;
* осуществлять выбор модели и средства построения информационной системы и программных средств;

*выполнить:*

* построение диаграмм вариантов использования;
* построение диаграмм кооперации и развертывания;
* построение диаграмм потоков данных;
* сгенерировать код.

## Задание 1.

Построение диаграммы Вариантов использования и диаграммы Последовательности

Ознакомиться с методологией построения диаграммы вариантов использования основе языка UML. Постройте диаграмму вариантов использования для заданной предметной области. Проанализируйте диаграмму вариантов использования.

Ознакомиться с методологией построения диаграммы последовательности основе языка UML. Постройте диаграмму последовательности. Проанализируйте диаграмму последовательности.

## Задание 2.

Построение диаграммы Кооперации и диаграммы Развертывания

Ознакомиться с методологией построения диаграммы кооперации основе языка UML. Постройте диаграмму кооперации для заданной предметной области. Проанализируйте диаграмму кооперации.

Ознакомиться с методологией построения диаграммы Развертывания основе языка UML. Постройте диаграмму Развертывания. Проанализируйте диаграмму Развертывания.

## Задание 3.

Построение диаграммы компонентов

Ознакомиться с методологией построения диаграммы компонентов основе языка UML. Постройте диаграмму компонентов. Проанализируйте диаграмму компонентов.

## Задание 4.

Построение диаграммы компонентов. Построение диаграмм потоков данных и генерация кода

Ознакомиться с методологией построения диаграмм потоков данных. Постройте диаграмму потоков данных. Проанализируйте диаграмму потоков данных.

# Тема 5 Разработка и модификация информационных систем

Построение архитектуры проекта. Шаблон проекта. Формирование репозитория проекта, определение уровня доступа в системе контроля версий. Распределение ролей. Разработка графического интерфейса пользователя. Отладка приложений. Организация обработки исключений.

Обучающийся должен

*знать:*

* основные платформы для создания, исполнения и управления информационной системой;
* особенности программных средств, используемых в разработке ИС;

*уметь:*

* работать с инструментальными средствами обработки информации;
* использовать языки структурного, объектно-ориентированного программирования и языка сценариев для создания независимых программ;
* разрабатывать графический интерфейс приложения;
* решать прикладные вопросы программирования и языка сценариев для создания программ;
* проектировать и разрабатывать систему по заданным требованиям и спецификациям;

*выполнить:*

* построение архитектуры проекта;
* сформировать репозиторий проекта;
* разработать графический интерфейс.

## Задание 1.

Построение архитектуры проекта. Шаблон проекта

Разработать проект архитектуры программного средства в соответствии с ГОСТ Р ИСО/МЭК 12207

Алгоритм выполнения работы

С целью реализации начальных этапов разработки ПС в соответствии с техническим заданием:

−выполнить подготовительную работу;

−провести анализ требований к ПС;

−выполнить проектирование архитектуры ПС на высоком уровне.

## Задание 2.

Формирование репозитория проекта, определение уровня доступа в системе контроля версий. Распределение ролей

Создать новый репозиторий.

Создайте на рабочем столе папку под названием git\_exercise. Для этого в окне терминала введите:

$ mkdir Desktop/git\_exercise/

$ cd Desktop/git\_exercise/

$ git init

1

2

3

$ mkdir Desktop/git\_exercise/

$ cd Desktop/git\_exercise/

$ git init

Командная строка должна содержать, например, следующее:

Initialized empty Git repository in

/home/user/Desktop/git\_exercise/.git/

1

Initialized empty Git repository in

/home/user/Desktop/git\_exercise/.git/

Это значит, что репозиторий был успешно создан, но пока он пуст. Теперь создайте текстовый файл под названием hello.txt и сохраните его в директории

git\_exercise.

4. Определить состояние репозитория.

Status — это еще одна важнейшая команда, которая показывает информацию о текущем состоянии репозитория: актуальна ли информация на нем, нет ли чего-то нового, что поменялось и т. д. Запуск git status на нашем недавно созданном репозитории выдаст:

$ git status

On branch master

Initial commit

Untracked files:

(use "git add ..." to include in what will be committed)

hello.txt

1

2

3

4

5

6

$ git status

On branch master

Initial commit

Untracked files:

(use "git add ..." to include in what will be committed) hello.txt

Сообщение говорит о том, что файл hello.txt неотслеживаемый. Это значит, что файл новый и система еще не знает, нужно ли следить за изменениями в файле или можно просто игнорировать его. Чтобы начать отслеживать новый файл, нужно его специальным образом объявить.

5. Подготовить файлы. В Git есть концепция области подготовленных файлов. Можно представить ее как холст, на который наносят изменения, нужные в коммите.

Коммит — состояние репозитория в определенный момент времени.

Первоначально он пустой, но затем мы добавляем на него файлы (или части файлов, или одиночные строки) командой add и наконец коммитим все нужное в репозиторий (создаем слепок нужного нам состояния) командой commit.

В нашем случае у нас только один файл, так что добавим его:

$ git add hello.txt

1

$ git add hello.txt

Если нам нужно добавить все, что находится в директории, мы можем использовать:

$ git add -A

1

$ git add -A

Проверим статус снова, на этот раз мы должны получить другой ответ:

$ git status

On branch master

Initial commit

Changes to be committed:

(use "git rm --cached ..." to unstage)

new file: hello.txt

1

2

3

4

5

6

$ git status

On branch master

Initial commit

Changes to be committed:

(use "git rm --cached ..." to unstage)

new file: hello.txt

Файл готов к коммиту. Сообщение о состоянии также говорит о том, какие изменения относительно файла были проведены в области подготовки, в данном случае это новый файл, но файлы могут быть модифицированы или удалены.

6. Зафиксировать изменения (коммит). Чтобы зафиксировать изменения, нужно хотя бы одно изменение в области подготовки (мы только что создали его при помощи git add), после которого мы можем коммитить:

$ git commit -m "Initial commit."

1

$ git commit -m "Initial commit."

Эта команда создаст новый коммит со всеми изменениями из области подготовки (добавление файла hello.txt). Ключ -m и сообщение "Initial commit." — это созданное пользователем описание всех изменений, включенных в коммит. Считается хорошей практикой делать коммиты часто и всегда писать содержательные комментарии.

## Задание 3.

Разработка графического интерфейса пользователя

Реализовать пользовательский интерфейс: возможность просмотра информации, оформления заказа (при необходимости), справочного меню, карты (при необходимости).

## Задание 4.

Отладка приложений. Организация обработки исключений

Механизм исключений, предусмотренный в C#, упрощает обработку ошибок. Вместо того чтобы проверять значение, возвращаемое функциями и методами, вы можете использовать для обнаружения и обработки ошибок структурные операторы, такие как try и catch.

Методы стандартных библиотек классов возбуждают исключения при возникновении ошибочных ситуаций. Заключив "ненадежный" с точки зрения возникновения ошибок код в блок try, вы можете перехватить и обработать исключения в блоке catch.

Заметим, что исключения возникают и в таких распространенных ситуациях, как выход за пределы массива или строки в процессе индексации, деление на нуль, попытка использования ссылки со значением null, недопустимое преобразование классов и так далее. С помощью исключений все подобные ошибки нетрудно обнаружить и обработать во время выполнения приложения.

К механизму обработки исключений в следующие ключевые слова: try, catch, throw и finally. Схема работы этого механизма следующая. Вы пытаетесь (try) выполнить блок кода, и, если при этом возникает ошибка, система возбуждает (throw) исключение, которое в зависимости от его типа вы можете перехватить (catch) или передать обработчику по-умолчанию (finally).

Ниже приведена общая форма блока обработки исключений.

try {

// блок кода }

catch (ТипИсключения1 е) {

// обработчик исключений типа ТипИсключения1 }

catch (ТипИсключения2 е) {

// обработчик исключений типа ТипИсключения2

throw(e) // повторное возбуждение исключения }

finally {

}

Пример:

try {

line = Сonsole.КeadLine();

if (line.Length() == 0)

throw new EmptyLineException("Строка, считанная с консоли, пустая!");

console.printLine("Привет, %s!" % line);

}

catch (EmptyLineException exception) {

console.printLine("Привет!");

}

catch (Exception exception) {

console.printLine("Ошибка: " + exception.message());

}

else {

console.printLine("Программа выполнилась без исключительных ситуаций");

}

finally {

console.printLine("Программа завершается");

Операторы try можно вкладывать друг в друга аналогично тому, как можно создавать вложенные области видимости переменных. Если у оператора try низкого уровня нет раздела catch, соответствующего возбужденному исключению будут проверены разделы catch внешнего оператора try. Вот пример, в котором два оператора try вложены друг в друга посредством вызова метода.

class MultiNest {

static void procedure() {

try {

int c[] = { 1 };

c[42] = 99;

}

catch(IndexOutOfRangeException e) {

System.out.println("array index oob: " + e);

} }

public static void main(String args[]) {

try {

int a = args.length();

System.out.println("a = " + a);

int b = 42 / a;

procedure();

}

catch (ArithmeticException e) {

System.out.println("div by 0: " + e);

}

} }

Хотя встроенные исключения обрабатывают большинство частых ошибок, вероятно, вам потребуется создать ваши собственные типы исключений для обработки ситуаций, специфичных для ваших приложений. Это достаточно просто сделать: просто определите подкласс Exception.

В следующем примере объявляется новый подкласс Exception, который затем используется для сигнализации об ошибочной ситуации в методе. Он переопределяет метод toString (), позволяя отобразить тщательно настроенное описание исключения.

Пример:

class MyException extends Exception {

private int detail;

MyException(int a) {

detail = a;

}

public String toString() {

return "MyException!" + detail ();"]";

}

}

class ExceptionDemo {

static void compute(int a)

{

System.out.println("Вызван compute(" + a + ")");

if(a > 10)

throw new MyException(a);

System.out.println("Нормальное завершение");

}

public static void main(String args[]) {

try {

compute(1);

compute(20);

}

catch (MyException e) {

System.out.println("Перехвачено " + e) ;

}

}

}

В этом примере определен подкласс Exception с именем MyException. Этот подкласс достаточно прост: он имеет только конструктор и перегруженный метод toString. Класс ExceptionDemo определяет метод под названием compute (), который возбуждает объект MyException. Это исключение возбуждается, когда целочисленный параметр compute () принимает значение больше 10

Метод main () устанавливает обработчик исключений MyException, затем вызывает compute () с правильным параметром (меньше 10), и с неправильным, чтобы продемонстрировать оба пути выполнения кода. Ниже показан результат.

Вызван compute(1)

Нормальное завершение

Вызван compute(20)

Перехвачено MyException[20]

Выше были рассмотрены способы обработки исключений и передача исключений в вызывающий метод. Однако имеется возможность совмещения этих подходов, т.е. исключение может быть обработано и выброшено в вызывающий метод:

public String сформироватьОписание() {

try {

...

if (nazvanie = = || nazvanie.equals(“”)) throw new MyException (“Не указано название

дерева”);

...

} catch (MyException ex) {

System.out.println(ex.getMessage());

throw ex; // повторный выброс исключения (операция return отсутствует)

}

}

# Тема 6 Отладка и тестирование информационных систем

Разработка тестового сценария проекта. Использование инструментария анализа качества. Функциональное тестирование. Нагрузочное тестирование, стрессовое, конфигурационное тестирование

Обучающийся должен

*знать:*

* объектно-ориентированное программирование.

*уметь:*

* использовать языки структурного, объектно-ориентированного программирования и языка сценариев для создания независимых программ;

*выполнить:*

* разработку тестового сценария;
* функциональное, нагрузочное, стрессовое, конфигурационное тестирование.

## Задание 1.

Разработка тестового сценария проекта

Написать тестовый сценарий из не менее 10 шагов, соответствующий полученному варианту задания. Сценарий должен включать в себя не только основной вариант использования функционала, но и ошибочный (например: ввод пустого/неверного пароля в примере). Обратите внимание, что все предварительные действия, необходимые для прохождения шага, должны быть явно описаны. Например, нельзя требовать от тестировщика банкомата ввести ПИН код до того, как он вставил карту.

## Задание 2.

Использование инструментария анализа качества

Оценить эффективность разработанной программы:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Исходная программа | | Улучшенная программа | |
|  | Недостатки | Количественная оценка | Улучшения | Количественная оценка |
| Время выполнения |  |  |  |  |
| Оперативная память |  |  |  |  |
| Внешняя память |  |  |  |  |

Оценить качество разработанной программы:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Правильность | Универсальность | Проверяемость | Точность результатов |
| Недостатки |  |  |  |  |
| Оценка |  |  |  |  |

## Задание 3.

Функциональное тестирование. Нагрузочное тестирование, стрессовое, конфигурационное тестирование. Тестирование безопасности

**Часть 1**

Разработать тестовые наборы для функционального тестирования.

Провести тестирование программы и представить результаты в виде таблицы.

|  |  |  |  |
| --- | --- | --- | --- |
| Тест (значения для входных данных) | Ожидаемый результат (значения для выходных данных) | Фактический результат (полученные значения выходных данных) | Результат тестирования (успешно/неуспешно) |
|  |  |  |  |

Выработать рекомендации для корректировки тестируемой программы.

**Часть 2**

**Нагрузочное тестирование или тестирование производительности**

**Нагрузочное тестирование** или **тестирование производительности** – это автоматизированное тестирование, имитирующее работу определенного количества бизнес пользователей на каком-либо общем (разделяемом ими) ресурсе.

Основные виды нагрузочного тестирования:

* Тестирование производительности;
* Стрессовое тестирование;
* Объемное тестирование;
* Тестирование стабильности или надежности.

**Тестирование производительности** (Performance testing)

Задачей тестирования производительности является определение масштабируемости приложения под нагрузкой, при этом происходит:

* измерение времени выполнения выбранных операций при определенных интенсивностях выполнения этих операций;
* определение количества пользователей, одновременно работающих с приложением;
* определение границ приемлемой производительности при увеличении нагрузки (при увеличении интенсивности выполнения этих операций);
* исследование производительности на высоких, предельных, стрессовых нагрузках.

**Стрессовое тестирование** (Stress Testing)

Стрессовое тестирование позволяет проверить насколько приложение и система в целом работоспособны в условиях стресса и также оценить способность системы к регенерации, т.е. к возвращению к нормальному состоянию после прекращения воздействия стресса. Стрессом в данном контексте может быть повышение интенсивности выполнения операций до очень высоких значений или аварийное изменение конфигурации сервера. Также одной из задач при стрессовом тестировании может быть оценка деградации производительности, таким образом, цели стрессового тестирования могут пересекаться с целями тестирования производительности.

**Объемное тестирование** (Volume Testing)

Задачей объемного тестирования является получение оценки производительности при увеличении объемов данных в базе данных приложения, при этом происходит:

* измерение времени выполнения выбранных операций при определенных интенсивностях выполнения этих операций;
* может производиться определение количества пользователей, одновременно работающих с приложением.

**Тестирование стабильности или надежности** (Stability / Reliability Testing)

Задачей тестирования стабильности (надежности) является проверка работоспособности приложения при длительном (многочасовом) тестировании со средним уровнем нагрузки. Время выполнения операций может играть в данном виде тестирования второстепенную роль. При этом на первое место выходит отсутствие утечек памяти, перезапусков серверов под нагрузкой и другие аспекты, влияющие именно на стабильность работы.

**Определение целей тестирования производительности**

Прямыми или косвенными целями любого тестирования, так или иначе затрагивающего вопросы производительности, является:

* определение "узких мест" системы (функций программно-аппаратного комплекса, обращение к которым приводит к наибольшему падению показателей производительности);
* определение лучшей архитектуры системы, выбор наилучшей платформы, средств и языков реализации;
* определение оптимального способа хранения файлов;
* оценка и оптимизация схемы базы данных в контексте повышения производительности;
* оценка максимальной и минимальной производительности системы и условий их достижения;
* определение характера увеличения времени отклика системы при увеличении нагрузки;
* определение максимального числа одновременно работающих пользователей, превышение которого делает использование системы невозможным;
* определение влияния конфигурации системы на производительность;
* оценка показателей масштабируемости системы;
* оценка соответствия сетевой инфраструктуры требованиям производительности.

**Основные тесты производительности**

В рамках нагрузочного тестирования и тестирования производительности, как правило, выполняются следующие основные тесты.

**Тест на определение максимальных возможностей системы** (capacity test) позволяет определить т.н. "точку насыщения системы" (system saturation point) – уровень нагрузки, при котором дальнейшее наращивание числа пользователей ведёт к увеличению времени отклика системы либо ухудшению стабильности системы, но не к увеличению в единицу времени количества полезных операций, обработанных системой. Данный тест направлен на оценку производительности системы как аппаратно-программного комплекса, поскольку учитывает доступные аппаратные ресурсы и эффективность их использования.

Проведение нескольких тестов на определение максимальных возможностей системы с добавлением аппаратных ресурсов позволяет определить показатели масштабируемости (scalability) системы, которая определяется как способность приложения увеличивать производительность пропорционально добавлению аппаратных ресурсов системы.

**Низко-, средне- и высоконагруженная работа** (low-, mid-, high-load tests) – позволяет оценить время отклика (response time) системы в некоторых заданных диапазонах нагрузки. Данная информация может быть использована при составлении перечня требований к условиям эксплуатации системы.

**Тест на выживаемость** (longevity test) показывает способность системы работать длительное время под высокой нагрузкой. Одной из наиболее опасных проблем, выявляемых данным тестом, является утечка памяти и иное снижение эффективности использования аппаратных ресурсов из-за накапливающихся со временем ошибок в работе приложения.

**Тест "часа пик"** (rush hour test) позволяет оценить реакцию системы на резкое изменение нагрузки. Во время теста проверяется способность системы выдержать скачкообразное увеличение нагрузки во время "часа пик", а также способность системы вернуться к изначальным показателям производительности после завершения "часа пик" (восстановления исходных показателей нагрузки на систему). Такой тест позволяет выявить проблемы с синхронизацией выполнения отдельных участков кода, а также проблемы с управлением всеми видами межкомпонентного взаимодействия (в т.ч. сетевых и локальных соединений) на всех уровнях системы.

**Тест "точки рандеву"** (rendezvous point test) подразумевает такую настройку профиля нагрузки и поведения виртуальных пользователей, чтобы в некоторый момент все они одновременно выполняли одну и ту же операцию: как правило, синхронную операцию сохранения, записи, и т.п. В отличие от теста "часа пик" этот тест не подразумевает увеличения числа одновременно работающих с системой пользователей, а подразумевает исследование ситуации конкуренции пользователей за некоторые ресурсы, совместное использование которых не представляется возможным или сопряжено с повышенной нагрузкой на системные ресурсы. В частности, этот тест позволяет выявить проблемы с разделением ресурсов на уровне баз данных.

**Основные показатели (метрики) производительности**

Одним из результатов, получаемых при нагрузочном тестировании и используемых в дальнейшем для анализа, являются показатели производительности приложения. Основные из них разобраны ниже.

1. **Потребление ресурсов центрального процессора (CPU, %)**

Метрика, показывающая, сколько времени из заданного определённого интервала было потрачено процессором на вычисления для выбранного процесса. В современных системах важным фактором является способность процесса работать в нескольких потоках, для того, чтобы процессор мог производить вычисления параллельно. Анализ истории потребления ресурсов процессора может объяснять влияние на общую производительность системы потоков обрабатываемых данных, конфигурации приложения и операционной системы, мультипоточности вычислений, и других факторов.

1. **Потребление оперативной памяти (Memory usage, Mb)**

Метрика, показывающая количество памяти, использованной приложением. Использованная память может делиться на три категории:

**Virtual** — объём виртуального адресного пространства, которое использует процессор. Этот объём не обязательно подразумевает использование соответствующего дискового пространства или оперативной памяти. Виртуальное пространство конечно и процесс может быть ограничен в возможности загружать необходимые библиотеки.

**Private** — объём адресного пространства, занятого процессом и не разделяемого с другими процессами.

**Working Set** — набор страниц памяти, недавно использованных процессом. В случае, когда свободной памяти достаточно, страницы остаются в наборе, даже если они не используются. В случае когда, свободной памяти остается мало, использованные страницы удаляются.

При работе приложения память заполняется ссылками на объекты, которые, в случае неиспользования, могут быть очищены специальным автоматическим процессом, называемым «сборщиком мусора» (англ. *Garbage Collector*). Время затрачиваемое процессором на очистку памяти таким способом может быть значительным, в случае, когда процесс занял всю доступную память (в Java — так называемый «постоянный Full GC») или когда процессу выделены большие объёмы памяти, нуждающиеся в очистке. На время, требующееся для очистки памяти, доступ процесса к страницам выделенной памяти может быть заблокирован, что может повлиять на конечное время обработки этим процессом данных.

1. **Потребление сетевых ресурсов**

Эта метрика не связана непосредственно с производительностью приложения, однако её показатели могут указывать на пределы производительности системы в целом.

|  |
| --- |
| **Пример:**  *Серверное приложение обрабатывая запрос пользователя, возвращает ему видео-поток, используя сетевой канал в 2 мегабит. Требование гласит, что сервер должен обрабатывать 5 запросов пользователей одновременно.*  *Нагрузочное тестирование показало, что эффективно сервер может предоставлять данные только 4 пользователям одновременно, так как мультимедиа-поток имеет битрейт в 500 килобит. Очевидно, что предоставление этого потока 5 пользователям одновременно невозможно в силу превышения пропускной способности сетевого канала, а значит, система не удовлетворяет заданным требованиям производительности, хотя при этом потребление ей ресурсов процессора и памяти может быть невысоким.* |

1. **Работа с дисковой подсистемой (I/O Wait)**

Работа с дисковой подсистемой может значительно влиять на производительность системы, поэтому сбор статистики по работе с диском может помогать выявлять узкие места в этой области. Большое количество чтений или записей может приводить к простаиванию процессора в ожидании обработки данных с диска и в итоге увеличению потребления CPU и увеличению времени отклика.

1. **Время выполнения запроса (request response time, ms)**

Время выполнения запроса приложением остаётся одним из самых главных показателей производительности системы или приложения. Это время может быть измерено на серверной стороне, как показатель времени, которое требуется серверной части для обработки запроса; так и на клиентской, как показатель полного времени, которое требуется на сериализацию / десериализацию, пересылку и обработку запроса. Надо заметить, что не каждое приложение для тестирования производительности может измерить оба этих времени.

**Профилирование**

***Профилирование*** — это сбор характеристик программы, таких как время выполнения отдельных ее фрагментов (функций, строк).

Существует два основных типа инструментов для определения полного профиля приложения: *профилирование* и *трассировка*. Важно различать профилирование и трассировку, так как они несхожие, но взаимодополняющие инструменты. Профилирование приложения обычно означает агрегирование или суммирование статистики применения во время работы. Трассировка же собирает данные, архив событий, в то время как выполняется приложение, и представляет их как динамику изменений. Профилирование обычно предоставляет довольно небольшое количество данных, в то время как трассировка может предоставлять большое количество информации.

Трассировка предоставляет данные о проведённом времени в определённом режиме или о наборе вложенных циклов. Профилирование же выходит за рамки этого и проводит мониторинг системы во время работы приложения, которая действительно контролирует "события", происходящие в системе. Например, вы можете измерить количество различных промахов кэша или хитов, промахи буфера ассоциативной трансляции (TLB), ошибочность прогнозирования ветвления, количества команд, количество памяти нагрузок / блоков памяти, количество операций с плавающей точкой в секунду и так далее. Как правило, эти данные представлены в виде динамики изменений (то есть участка в зависимости от времени).

Инструменты мониторинга могут быть поделены на две главные группы — встраиваемые блоки и отдельные независимые программы.

Для первой категории существуют такие инфраструктуры как ETW (Event Tracing for Windows) и счетчики производительности. Для второй – библиотеки, которые встраиваются в ваш код, например, New Relic, и полностью готовые системы мониторинга, которые не требуют вмешательства программистов.

**Профилировщик Visual Studio**

1. **Дискретный профилировщик**

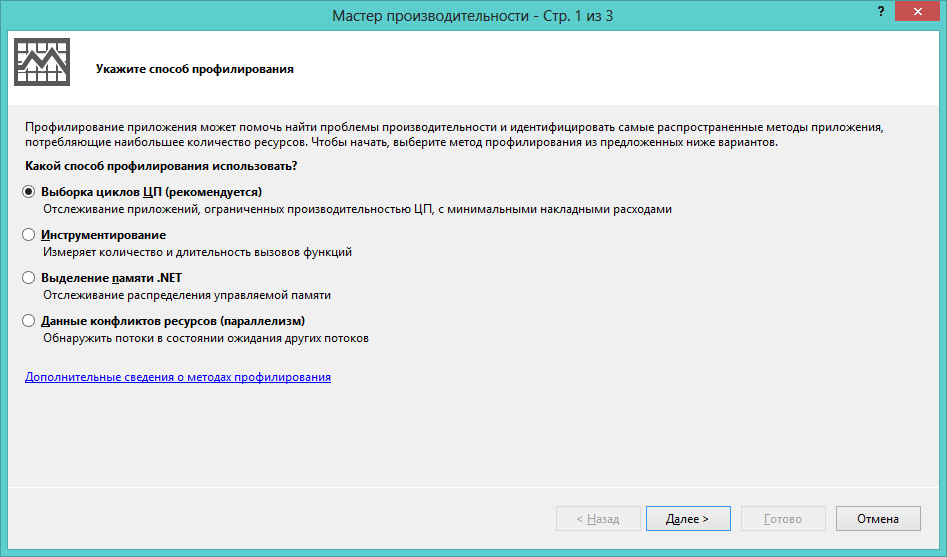
Он периодически прерывает работу приложения и сохраняет информацию о стеке вызовов для каждого процессора, где в настоящий момент протекает выполнение приложения или потока.

Сбор информации с использованием профилировщика Visual Studio обходится достаточно недорого, и если интервал следования событий достаточно широк (по умолчанию он составляет 10 000 000 тактов часов CPU), накладные расходы в общем времени выполнения приложения будут составлять менее 5%. Кроме того, дискретный подход к сбору информации позволяет подключиться к выполняющемуся процессу, собрать данные в течение некоторого времени и затем отключиться от процесса, чтобы проанализировать их. Из-за этой особенности, выявление проблем производительности рекомендуется начинать с поиска участков программы, оказывающих самую большую нагрузку на процессор - методов, на выполнение которых тратится больше всего процессорного времени.

По окончании сеанса сбора информации, профилировщик генерирует сводные таблицы, где каждый метод характеризуется двумя значениями: **числом исключительных попаданий** (*exclusive samples*), когда в момент отбора очередной порции данных на CPU выполнялся данный метод, и **числом включительных попаданий** (*inclusive samples*), когда выполнялся данный метод или вызванный им. Методы с большим количеством исключительных попаданий оказывают самую большую нагрузку на процессор; методы с большим количеством включительных попаданий не используют процессор непосредственно, но вызывают другие методы, суммарно занимающие значительную часть процессорного времени. (Например, в однопоточных приложениях метод Main() будет иметь 100% включительных попаданий.)

Запуск дискретного профелировщика:

1. В Visual Studio выберите пункт меню Analyze -> Launch Performance Wizard (Анализ --> Запустить мастер производительности).
2. На первой странице мастера отметьте радиокнопку CPU sampling (Выборка циклов ЦП (рекомендуется)) и щелкните на кнопке Next.

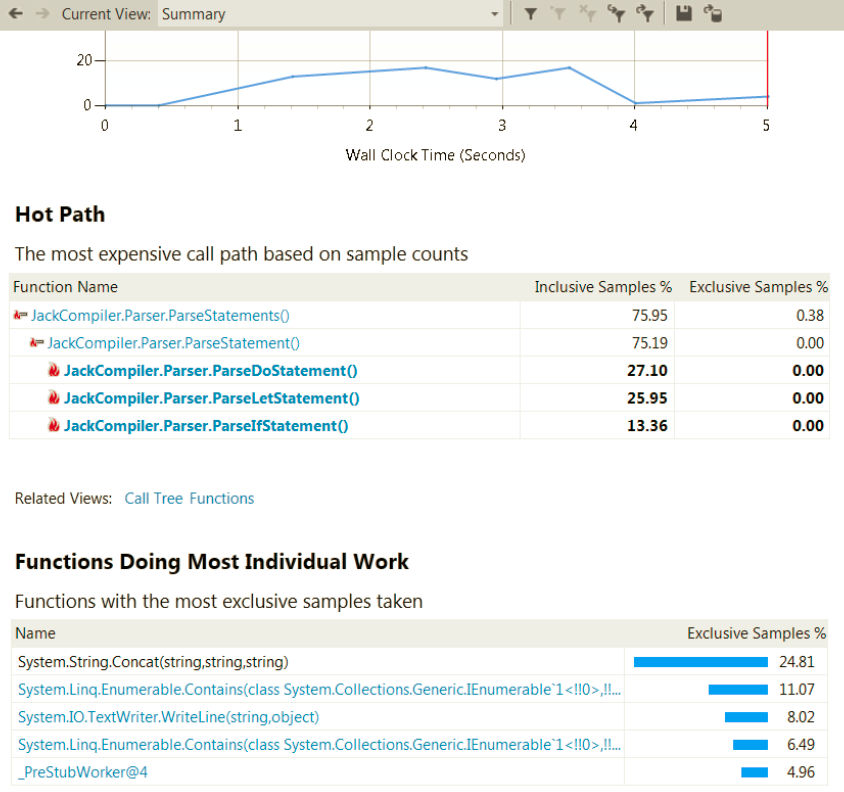


**Рисунок 1. Выбор способа профилирования**

1. Если проект для профилирования загружен в текущем решении, щелкните на радиокнопке One or more available projects (Один или несколько доступных проектов) и выберите проект из списка. В противном случае щелкните на радиокнопке An executable (.EXE file) (Исполняемый файл). Щелкните на кнопке Next.
2. Если на предыдущем шаге вы выбрали параметр An executable (.EXE file), укажите профилировщику путь к выполняемому файлу и аргументы командной строки, если они необходимы, затем щелкните на кнопке Next. (Если у вас нет собственного подходящего приложения, используйте JackCompiler.exe из исходников.)
3. Оставьте флажок Launch profiling after the wizard finishes (Запустить профилирование после завершения работы мастера) отмеченным и щелкните на кнопке Finish (Готово).
4. Если среда разработки Visual Studio была запущена не с привилегиями администратора, вам будет предложено повысить привилегии профилировщика.
5. Когда приложение завершит выполнение, откроется отчет профилирования. Используйте раскрывающийся список Current View (Текущее представление) для навигации между представлениями, отображающими информацию, собранную в процессе профилирования.

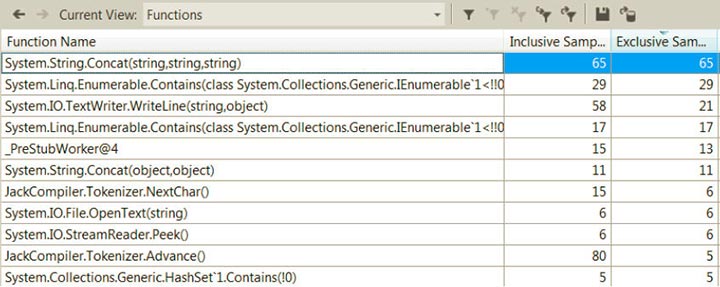
По окончании сеанса профилирования можно также воспользоваться обозревателем производительности - Performance Explorer, чтобы запустить его, выберите пункт меню Analyze -> Windows -> Performance Explorer (Анализ --> Окна --> Обозреватель производительности). С его помощью можно изменить параметры профилирования (например, выбрать другой интервал опроса или установить другой критерий), изменить целевой файл и сравнить результаты разных сеансов профилирования.

На рисунке ниже показано окно профилировщика с результатами, где можно видеть ветви в стеке вызовов, на выполнение которых было затрачено больше всего времени, и функции с наибольшим количеством исключительных попаданий:



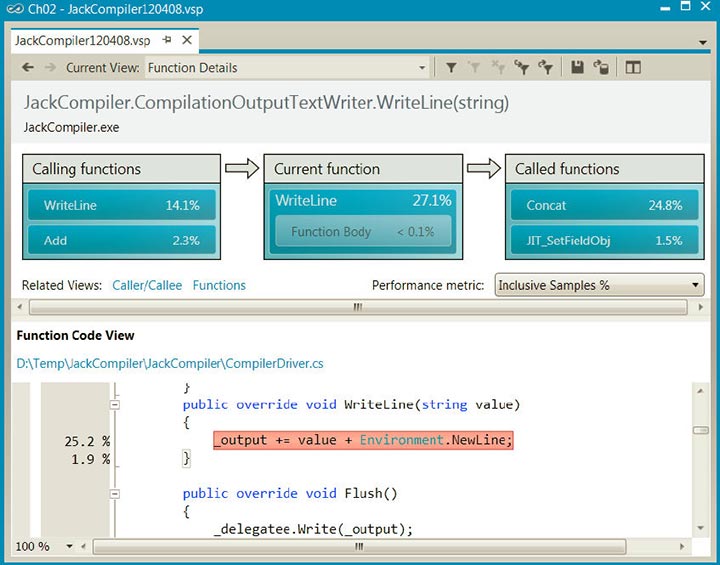
**Рисунок 2. Результаты дискретного профилирования из представления Summary**

На следующем рисунке показан подробный отчет, где перечислено несколько методов, на выполнение которых затрачено больше всего процессорного времени (имеющих наибольшее количество исключительных попаданий):



**Рисунок 3. Результаты дискретного профилирования из представления Functions**

Как видно, функция System.String.Concat имеет, как минимум, в два раза больше попаданий, чем любая другая функция. Если выполнить двойной щелчок на методе в списке, откроется окно детализации со строками исходного кода в приложении, в которых обнаружено наибольшее число попаданий:



**Рисунок 4. Результаты дискретного профилирования из представления Function Details**

Результаты дискретного профилировщика следует рассматривать не как точную картину распределения процессорного времени, а как общую схему, где выделены возможно узкие места.

Помимо количества исключительных/включительных попаданий для каждого метода, профилировщик Visual Studio предлагает массу другой ценной информации. Например, представление Call Tree (Дерево вызовов) отображает иерархию вызовов методов в приложении, представление Lines (Строки) отображает информацию о попаданиях на уровне строк, а представление Modules (Модули) группирует методы по сборкам, и может помочь быстро выяснить, в каком направлении двигаться в поисках узких мест.

Так как при дискретном мониторинге требуется, чтобы поток выполнения в приложении был активен в моменты сбора информации, нет никакого способа получить информацию о потоках, простаивавших в ожидании выполнения операций ввода/вывода или на блокировках механизмов синхронизации. Дискретный мониторинг отлично подходит для анализа приложений, занимающихся преимущественно вычислениями, но для приложений, основная работа которых заключается в выполнении операций ввода/вывода, необходимы иные подходы, опирающиеся на использование других, более глубинных механизмов профилирования.

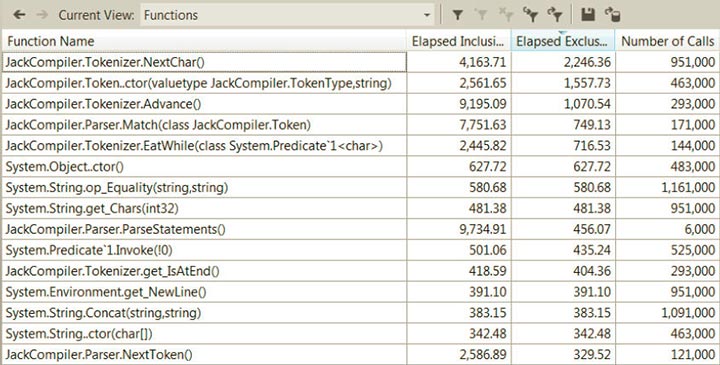
1. **Инструментированный профилировщик**

Он позволяет измерить общее время выполнения, а не только процессорное время, хорошо подходит для профилирования приложений, выполняющих большое количество операций ввода/вывода или интенсивно использующих механизмы синхронизации.

В режиме инструментированного профилирования профилировщик изменяет целевой выполняемый файл и внедряет в него код, выполняющий измерения и сообщающий профилировщику точную информацию о времени выполнения и количестве вызовов каждого оцениваемого таким способом метода.

Инструментированный профилировщик генерирует отчет с представлением Summary (Сводка), содержащим информацию о наиболее затратных ветвях в стеке вызовов и отдельных функциях, на выполнение которых тратится больше всего времени. Информация в отчете будет основана на точном хронометраже, проводимом инструментированным кодом.

На рисунке ниже показано представление Functions, где доступны включительное и исключительное время, измеренные в миллисекундах, а также счетчики вызовов функций:



**Рисунок 5. Результаты инструментированного профилирования из представления Functions**

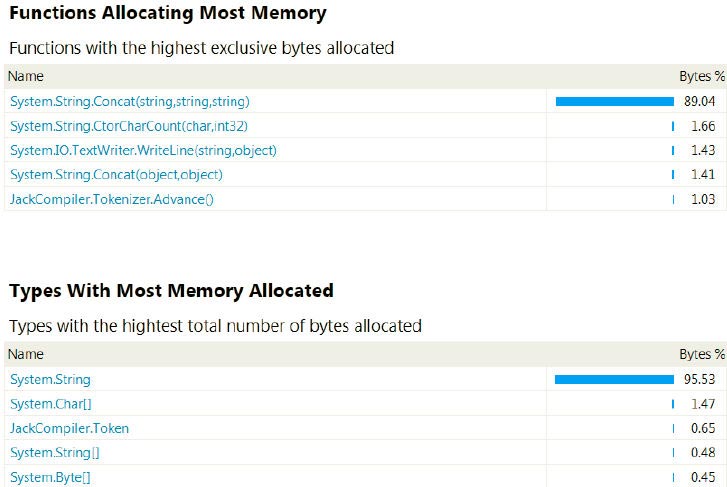
Инструментированное профилирование выглядит более точным, но на практике рекомендуется использовать дискретное профилирование, если приложение в основном решает вычислительные задачи. Прием инструментирования имеет ограниченную гибкость из-за необходимости изменять выполняемый файл приложения перед запуском и невозможности подключить профилировщик к уже запущенному процессу. Кроме того, инструментированное профилирование имеет немалые накладные расходы - объем выполняемого кода существенно увеличивается и в процессе выполнения часть времени затрачивается на отбор информации в точках входа и выхода из методов.

Как всегда, будет ошибкой безоговорочно доверять результатам инструментированного профилирования. Разумеется, количество вызовов того или иного метода не изменится из-за того, что приложение подвергается инструментированному профилированию, но информация о времени все еще может существенно искажаться из-за накладных расходов, несмотря на все попытки профилировщика сместить все дорогостоящие вычисления в конец. При внимательном подходе, дискретное и инструментированное профилирование могут помочь понять, где приложение проводит больше всего времени, особенно если сопоставлять множество отчетов и обращать внимание на результаты, полученные в результате оптимизации.

1. **Профилировщик выделения памяти**

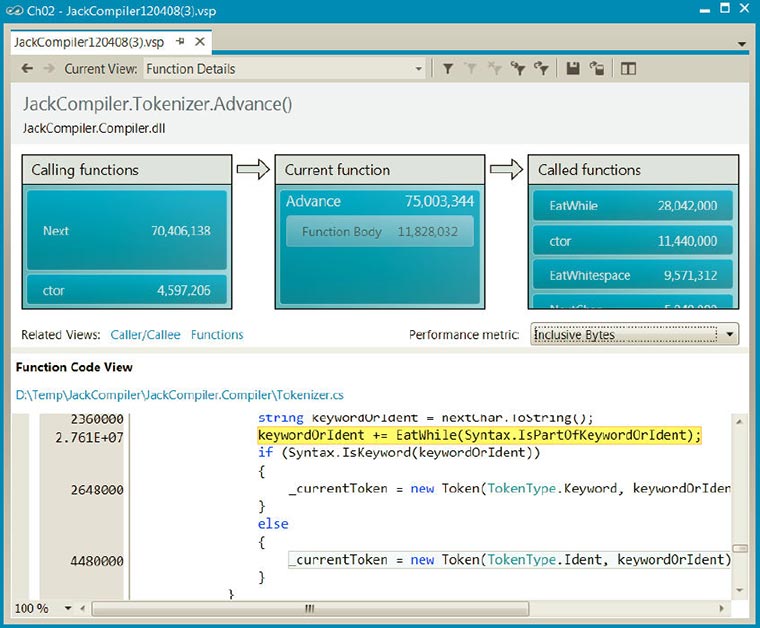
Профилировщик Visual Studio способен собирать информацию об операциях выделения памяти и жизненном цикле объектов (которые освобождаются сборщиком мусора) в обоих режимах, дискретном и инструментированном. В дискретном режиме профилировщик собирает информацию о выделении памяти в приложении в целом. В инструментированном режиме информация собирается только из инструментированных модулей.

Для того, чтобы запустить профилировщик выделения памяти, в мастере настройки профилировщика производительности нужно выбрать ***радиокнопку .NET memory allocation*** (Выделение памяти .NET). В конце сеанса профилирования, в представлении Summary (Сводка), будут показаны функции, выделившие памяти больше всего:



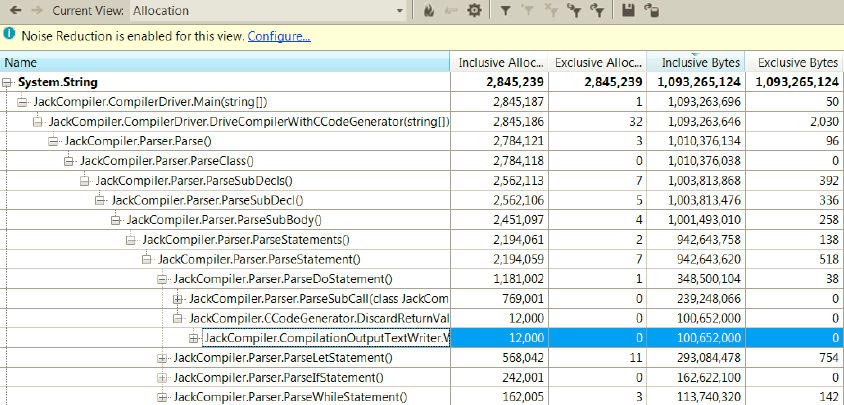
**Рисунок 6. Результаты профилирования выделения памяти из представления Summar**

В представлении Functions для каждого метода будет указано количество объектов и количество байтов памяти, выделенных методом (как обычно, включительные и исключительные значения). В представлении Function Details (Сведения о функции) будет представлена информация о вызывающих и вызываемых функциях, а также указаны строки кода с объемами выделенной ими памяти в поле слева:



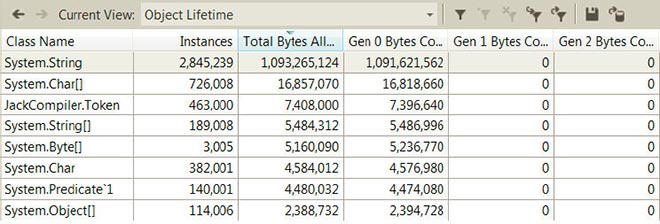
**Рисунок 7. Результаты профилирования выделения памяти из представления Function Details**

Но самая интересная информация содержится в представлении Allocation (Выделение), показывающем, какие ветви в стеке вызовов выделили памяти больше всего:



**Рисунок 8. Результаты профилирования выделения памяти из представления Allocation**

Представление Object Lifetime (Жизненный цикл объектов), сообщает, в каком поколении объекты были утилизированы. Это представление поможет увидеть, имеются ли объекты, пережившие слишком много циклов сборки мусора. На рисунке ниже можно видеть, что все объекты строк, созданные приложением (и занимающие более 1 Гбайта памяти!) были утилизированы в нулевом поколении, а это означает, что ни одному из них не удалось прожить дольше одного цикла сборки мусора:

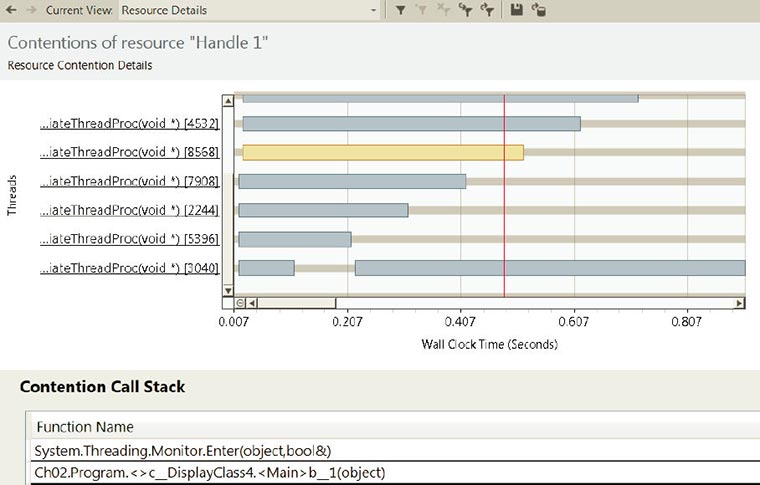


**Рисунок 9. Результаты профилирования выделения памяти из представления Object Lifetime**

1. **Профилировщик конкуренции**

Профилировщик Visual Studio, в режимах Concurrency (Данные конфликтов ресурсов (параллелизм)) и Concurrency Visualizer (Визуализатор параллелизма), использует события ETW для мониторинга производительности многопоточных приложений и предоставляет отчет с несколькими весьма полезными представлениями, упрощающими узкие места, отрицательно сказывающиеся на масштабируемости и производительности. Он имеет два режима работы.

В режиме **Concurrency** (Данные конфликтов ресурсов (параллелизм)) выявляются ресурсы, такие как управляемые блокировки, на которых потоки выполнения в приложении останавливаются в ожидании их освобождения. Первая часть отчета описывает сами ресурсы и потоки выполнения, которые были заблокированы на них - она поможет найти и устранить недостатки, мешающие масштабированию:

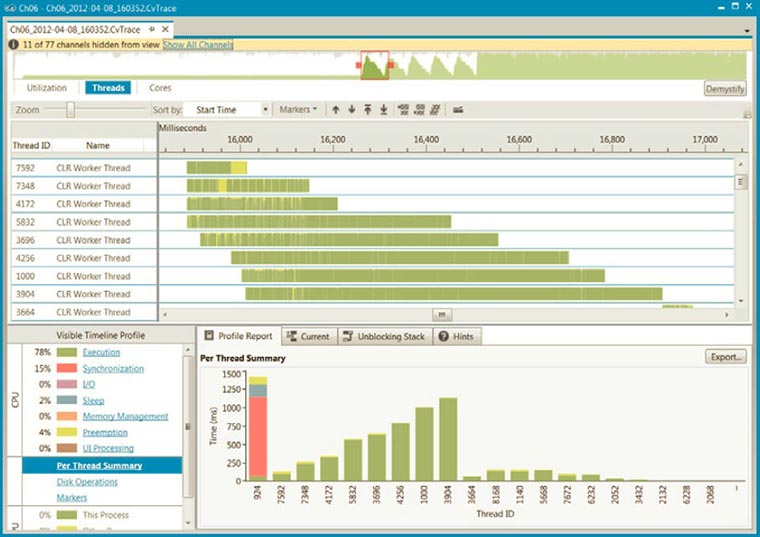


**Рисунок 10. Результаты профилирования конкуренции из представления Resource Details**

Как видно на рисунке, существует несколько потоков выполнения, ожидающих доступа к ресурсу. При выборе потока выполнения, в нижней части отображается его стек вызовов.

Другая часть отчета отображает информацию о конфликтах для конкретного потока выполнения, то есть перечень различных механизмов синхронизации, на которых поток выполнения вынужден был простаивать - она поможет уменьшить задержки на пути выполнения определенного потока.

Чтобы запустить профилировщик в этом режиме, нужно выбрать пункт меню Analyze --> Launch Performance Wizard (Анализ --> Запустить мастер производительности) и затем выбрать ***радиокнопку Concurrency*** (Данные конфликтов ресурсов (параллелизм)).



**Рисунок 11. Сводная информация по потокам в режиме визуализация параллелизма**

На основании этого отчета можно заключить, что работа распределяется между потоками неравномерно.

В режиме **визуализации параллелизма** отображается график выполнения всех потоков приложения, где цветом обозначается текущее состояние. Каждый поток имеет несколько переходных состояний - блокировка на операции ввода/вывода, ожидание на механизме синхронизации, выполнение - которые фиксируются профилировщиком, а также стек вызовов. Эти отчеты очень удобно использовать, чтобы понять особенности работы потоков выполнения и выявить причины низкой производительности, такие как чрезмерное количество потоков выполнения, недостаточное количество потоков выполнения и чрезмерное использование механизмов синхронизации.

В графике имеется также встроенная поддержка механизмов Task Parallel Library, таких как параллельные циклы (parallel loops) и механизмы синхронизации CLR. Чтобы запустить профилировщик в этом режиме выберите нужный пункт в подменю Analyze --> ***Concurrency Visualizer*** (Анализ --> Визуализатор параллелизма).

**Порядок выполнения лабораторной работы**

1. Написать многопоточную программу в соответствии с вариантом (количество потоков должно соответствовать возможному количеству потоков для вычислительной машины).
2. Провести профилирование с помощью **дискретного профилировщика**. В отчет занести результаты из представлений Summary, Functions. Для функций, на выполнение которых затрачено больше всего процессорного времени, добавить в отчет информацию из представления Function Details. Для данных в каждом из этих представлений написать комментарии.
3. Провести профилирование с помощью **инструментированного профилировщика**. В отчет занести результаты из представлений Summary, Functions. Для функций, у которых наибольшее исключительное время, добавить в отчет информацию из представления Function Details. Для данных в каждом из этих представлений написать комментарии.
4. Провести профилирование с помощью **профилировщика выделения памяти**. В отчет занести результаты из представлений Summary, Functions. Для функций, которые выделили больше всего памяти (исключительное значение), добавить в отчет информацию из представления Function Details. Также в отчет включить информацию из представления Allocation, а именно какие ветви в стеке вызовов выделили памяти больше всего. Для данных в каждом из этих представлений написать комментарии.
5. Провести профилирование с помощью **профилировщика конкуренции**. Для режима Concurrency в отчет занести результаты из представления Resource Details, Threads Details. Для режима Concurrency Visualizer – сводную информацию. Добавить комментарии.
6. На основании полученной информации сделать вывод о причинах временных затрат, большого выделения памяти и т.п. при выполнении программы. Можно собрать информацию и о других событиях, если это требуется для обоснования вывода.

# ПРИЛОЖЕНИЕ А

**Образец титульного листа отчета обучающегося по учебной практике**



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

Колледж экономики, управления и права

**ОТЧЕТ ПО УЧЕБНОЙ ПРАКТИКЕ**

ПМ.05 Проектирование и разработка информационных систем

Обучающегося \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (ФИО обучающегося)

Курс \_\_\_\_ Группа \_\_\_\_\_\_

Специальность 09.02.07 Информационные системы и программирование

Место прохождения практики Колледж ЭУП

Период прохождения практики \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка по практике \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_» \_\_\_\_\_\_\_\_\_ 202\_\_ г.

Руководитель практики от колледжа:

\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись ФИО преподавателя

# ПРИЛОЖЕНИЕ Б

**Образец титульного листа отчета студента по теме**

**Отчет по учебной практике**

**ПМ.05 Проектирование и разработка информационных систем**

**Тема \_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

*(наименование темы)*

Обучающийся должен

*знать:*

* …;
* ….

*уметь:*

* …;
* ….

*выполнить:*

* …;
* ….

Работу сдал обучающийся группы \_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(подпись) (Инициалы, фамилия)*

Работу принял преподаватель \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(подпись) (Инициалы, фамилия)*

Оценка \_\_\_\_\_\_\_\_\_\_\_\_ Дата \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_