Государственное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет имени Н. Э. Баумана»

(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Р А С Ч Ё Т Н О - П О Я С Н И Т Е Л Ь Н А Я З А П И С К А

к курсовой работе на тему:

Система телеметрии для графического редактора Крита

Студент _____ Капустин А.И.
(Подпись, дата)

Руководитель курсового проекта _____ Рогозин Н.О.
(Подпись, дата)

Москва 2017

Содержание

Введение 3
1 Аналитический раздел 4
1.1 Телеметрия 4
1.1.1 Как работает телеметрия 4
1.1.2 Преимущества телеметрии 4
1.1.3 Проблемы телеметрии 5
1.2 Графический редактор Krita 6
1.2.1 Функциональные возможности 7
2 Конструкторский раздел 8
2.1 Общая архитектура приложения 8
2.2 Клиентская часть 8
2.2.1 Библиотека KUserFeedback 8
2.2.2 Собираемые данные 10
2.2.3 Плагин для графического редактора 10
2.2.4 Динамическое изменение данных 12
2.2.5 Агрегация данных 12
2.3 Кэширование данных на фронтенде 14
3 Технологический раздел 16
3.1 Плагин 16
3.2 Backend-сервер 16
3.3 Фронтенд-сервер 16
3.4 База данных 17
Заключение 18
Список использованных источников 19

Введение

Графический редактор Крита содержит большое количество инструментов для художников, значительное количество настроек и функций. Их число увеличивается с каждым годом, но не все из них популярны. Разработчики хотят знать какие функции популярны, а какие нет. На основе полученной информации они смогут определить вектор дальнейшей работы, улучшить популярные, но не слишком хорошо реализованные функции, удалить старые, бесполезные функции. Использование телеметрии поможет узнать популярность различных вещей.

1 Аналитический раздел

1.1 Телеметрия

Телеметрия - это автоматическая запись и передача данных из удаленных или недоступных источников в ИТ-систему в другом месте для мониторинга и анализа. Данные телеметрии могут быть переданы с использованием радио, инфракрасного, ультразвукового, GSM, спутника или кабеля, в зависимости от приложения (телеметрия используется не только в разработке программного обеспечения, но также в метеорологии, разведке, медицине и других областях). В данном случае телеметрия - это слежение за действиями пользователя в приложении, установленном на персональном компьютере пользователя.

1.1.1 Как работает телеметрия

В общем случае телеметрия работает через датчики в удаленном источнике, который измеряет физические (например, осадки, давление или температуру) или электрические (такие как данные о токе или напряжении) или же через специальные программные средства).Совокупность данных, полученных из разных источников формируют поток данных, который передается по беспроводной среде, проводной или комбинации обоих. В контексте разработки программного обеспечения понятие телеметрии часто путается с журналированием. Но журнал - это инструмент, используемый в процессе разработки для диагностики ошибок и потоков кода, и он ориентирован на внутреннюю структуру веб-сайта, приложения или другого проекта разработки. Однако, как только проект будет выпущен, телеметрия - это то, что вы ищете для автоматического сбора данных из реального мира. Телеметрия - это то, что позволяет собирать все необработанные данные, которые становятся ценными, эффективными для аналитиков. В удаленном приемнике поток дезагрегирован, а исходные данные отображаются или обрабатываются на основе пользовательских спецификаций.

В мире разработки программного обеспечения телеметрия может дать представление о том, какие функции используют конечные пользователи, обнаружение ошибок и проблем, а также улучшенная видимость производительности без необходимости запрашивать обратную связь непосредственно от пользователей.

1.1.2 Преимущества телеметрии

Основным преимуществом телеметрии является способность конечного пользователя контролировать состояние объекта или окружающей среды, физически далеко от него. После того, как вы отправили продукт, вы не сможете физически присутствовать, просматривая плечи тысяч (или миллионов) пользователей, когда они взаимодействуют с вашим продуктом, чтобы узнать, что работает, что легко и что

громоздко. Благодаря телеметрии эти идеи могут быть доставлены непосредственно к вам на компьютер, чтобы вы могли анализировать и действовать.

Поскольку телеметрия дает представление о том, насколько хорошо ваш продукт работает для ваших конечных пользователей, когда они его используют, это невероятно ценный инструмент для постоянного мониторинга и управления производительностью. Кроме того, вы можете использовать собранные вами данные из версии 1.0 для улучшения и обновления приоритетов для выпуска версии 2.0. Телеметрия позволяет вам отвечать на такие вопросы, как:

— Ваши клиенты используют функции, которые вы ожидаете? Как они взаимодействуют с вашим продуктом?

— Как часто пользователи взаимодействуют с вашим приложением и на какой срок?

— Какие параметры настройки для пользователей выбираются чаще всего?

— Предпочитают ли они определенные типы дисплеев, методы ввода, ориентацию экрана или другие конфигурации устройства?

— Что происходит при сбоях? Каков контекст, связанный с сбоями?

Очевидно, что ответы на эти и многие другие вопросы, на которые можно ответить с помощью телеметрии, неоценимы для процесса разработки, позволяя вам постоянно совершенствовать и внедрять новые функции, которые могут показаться конечным пользователям полученными из их мыслей(а вы, конечно, внедрили их благодаря телеметрии).

1.1.3 Проблемы телеметрии

Телеметрия - это, безусловно, фантастическая технология, но и она имеет свои недостатки. Самая важная проблема - и часто встречающаяся проблема - связана не с самой телеметрией, а с вашими конечными пользователями и их готовностью разрешить собирать данные. Некоторые видят телеметрию как шпионаж . Короче говоря, некоторые пользователи сразу же отключают телметрию, когда замечают её, то есть любые данные, полученные от использования вами вашего продукта, не будут собираться или сообщаться.

Это означает, что опыт этих пользователей не будет учитываться при планировании будущей дорожной карты, исправлении ошибок или решении других проблем в вашем приложении. Пользователи, которые склонны отказываться от сбора статистики, могут негативно повлиять на ваше планы разработки. Другие пользователи, с другой стороны, не обращают внимания на телеметрию, происходящую за кулисами, или просто игнорируют ее, если они это делают.

Это проблема не имеет четкого решения - и это не отрицает общую силу телеметрии для процесса разработки, но ее следует учитывать при анализе ваших данных.

1.2 Графический редактор Krita

Krita — бесплатный растровый графический редактор с открытым кодом, программное обеспечение, входящее в состав KDE. Ранее распространялось как часть офисного пакета Calligra Suite, но впоследствии отделилось от проекта и стало развиваться самостоятельно. Разрабатывается преимущественно для художников и фотографов, распространяется на условиях GNU GPL.

Начало разработки Криты было положены Матасом Этрикхом в 1998 года на конференции Linux 1998. Маттиас хотел показать возможно создать Qt GUI для уже существующего приложение и в качествея демо-приложения он выбрал Gimp. Его патч никогда не публиковался, но он вызвал споры в сообществе Gimp.

Не имея возможности работать вместе, люди в рамках проекта KDE решили запустить собственное приложение для редактора изображений. Основное внимание было уделено приложению, которое было частью KOffice, названного KImage, Майклом Кохом. Переименованный в KImageShop, это было началом Криты.

31 мая 1999 года проект KImageShop официально стартовал с почтовый рассылки Маттиаса Элтера. Основная идея тогда заключалась в том, чтобы сделать KImageShop оболочкой графического интерфейса вокруг ImageMagick. Они планировали, что это будет приложение на основе corba, с плагинами, совместимое с плагинами GIMP.

Название KImageShop не соответсвовало закону о товарных знаках в Германии, поэтому KImageShop был переименован в Krayon, который также, по-видимому, ущемлял существующий товарный знак, поэтому Krayon был окончательно переименован в Krita в 2002 году.

Первоначальная разработка была медленной; с 2003 года она перешла в активную фазу.В 2004 году был выпущен первый публичный релиз с в составле пакета KOffice 1.4. В 2005 году Krita получила поддержку цветовых моделей CMYK, Lab, YCbCr, XYZ и каналов с высокой битовой глубиной, а также поддержку OpenGL.

С 2004 по 2009 год разработка была сфокусирована на повторение функций Photoshop/Gimp. С 2009 года акцент в разработке был смещен на удоволетворение потребности художников. Сообщество Krita стремится сделать Krita лучшим приложением для рисования для карикатуристов, иллюстраторов и художников-концептов[1]

1.2.1 Функциональные возможности

Krita поддерживает работу в различных цветовых пространствах и с различными цветовыми моделями — RGB, CMYK, Lab, в режиме от восьми до 32 с плавающей точкой разрядов на канал. Кроме того, реализованы популярные фильтры (такие как нерезкое маскирование), корректирующие слои, маски и динамические фильтры, а также серия инструментов для ретуши.

Однако основным приоритетом разработчики ставят реализацию возможностей для художников. Для них Krita может предложить:

— полноценные инструменты для работы с покадровой анимацией, включая экспорт анимации с использованием ffmpeg

— широкий выбор кистей (в том числе смешивающие, фильтрующие, эффектные, спрей, кисти для заполнения объемов)

— большое количество режимов наложения

— управление динамикой кистей с помощью графического планшета

— имитацию бумаги и пастели

— поворот и зеркалирование холста

— псевдо-бесконечный холст

— поддержку горячих клавиш Photoshop

2 Конструкторский раздел

2.1 Общая архитектура приложения

В состав приложения будет входить плагин для графического редактора, backend-сервер и frontend. Так же на бекенде будет размещена база данных

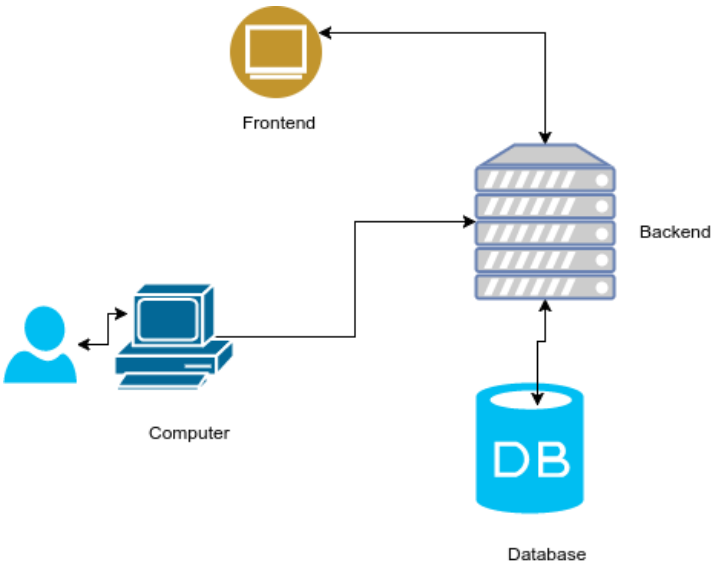

Рисунок 2.1 — Общая архитектура приложения

2.2 Клиентская часть

2.2.1 Библиотека KUserFeedback

В качестве вспомогательной библиотеки для сбора стастики используется библиотека KUserFeedback компании KDAB. Эта библиотека включает в себя C++ Qt клиентскую часть, а так же сервер, написанный на PHP[2] Нам нет требуется их сервер и значительная часть функций, мы будем использовать только часть собирающую телеметрию. KUserFeedBack позволяет собрать библиотеку по частям и линковать к нашему приложению только необхоимые модули. В программе используется модуль Core. Модуль Core содержит абстракный класс источника данных, от которого можно наследовать различные источники данных. Ниже представлено описание этого класса. В дальнейшем мы создадим классы, которые наследуются от этого абстрактого класса. В этих классах будут переопределеные все чисто виртуальноые функции. Ключевую роль будет играть переопределение функции *data()* - именно это переопределение несет смысловую нагрузку класса.

Листинг 2.1 — struct file_operations

```
1
2 class KUSERFEEDBACKCORE_EXPORT AbstractDataSource
3 {
4 public:
```

```
5  virtual ~AbstractDataSource();
6
7  /* Возвращает имя ресурса
8  *  Этот метод используется только на сервере
9  *  и не должен показыватся пользователю
10 */
11 QString name() const;
12
13 //Возвращает человеко-читаемое описание ресурса.
14 virtual QString description() const = 0;
15
16 //Возвращает данные, собранные ресурсом
17 virtual QVariant data() = 0;
18
19 // Загружает постоянное состояние для этого источника данных.
20 virtual void load(QSettings *settings);
21
22 //Сохраняет постоянное состояние для этого источника данных.
23 virtual void store(QSettings *settings);
24
25 // Сбрасывает состояние источника данных
26 virtual void reset(QSettings *settings);
27
28 // Возвращает режим сбора данных
29 Provider::TelemetryMode telemetryMode() const;
30
31 //Изменяет режим сбора данных
32 void setTelemetryMode(Provider::TelemetryMode mode);
33
34 protected:
35 // Создает новый источник данных
36 explicit AbstractDataSource(const QString &name, Provider::TelemetryMode
      mode = Provider::DetailedUsageStatistics, AbstractDataSourcePrivate *dd
      = nullptr);
37
38 //Изменяет имя источника данных
39 void setName(const QString &name);
40
41 class AbstractDataSourcePrivate* const d_ptr;
42 private:
43 Q_DECLARE_PRIVATE(AbstractDataSource)
44 Q_DISABLE_COPY(AbstractDataSource)
45 };
46 }
```

2.2.2 Собираемые данные

Основые собираемые данные приведены в таблице 2.1:

2.2.3 Плагин для графического редактора

Был разработан плагин для графического редактора Krita, который позволяет собирать телеметрию с пользователей. Телеметрия отправляется через определенные интервалы времени на удаленный сервер. Записи о действиях и инструментах отправляются каждые n минут. Информация о компьютере отправляется 1 раз при установке Krita. После этого в файл настроек записывается информация о том, что больше информацию о компьютере отправлять не стоит. Это позволяет избежать "замусоривания"отправляемой информации за счёт отсуствия дублирования. Информация об ассертах отправляется по мере необходимости. Если программа находится debug-режиме, программа аварийно завершает свою работу после любого ассерта. После того как аварийный ассерт сработал, он записывается в файл конфига Krita. При следующей запуске программы, он будет прочитан оттуда и отправлен на удаленный сервер при старте программы. Если программа собрана в release-версии, то при ассерте, не приводящем к аварийному завершению программы, информация об ассерте в рантайме программы отправляется на удаленный сервер. Собираемые метрики агрегруются на стороне клиента в http-запрос. Тело запроса представляет собою JSON, пример этого JSON приведен ниже(форматирование нарушено в целях наглядности).

Листинг 2.2 — Тело http-запроса

```
{
"asserts": []
}

{
"Tools": [
{
"countUse": 1,
"timeUseMSeconds": 4581,
"toolName": "/Activate/KritaShape/KisToolLine"
},
{
"countUse": 4,
"timeUseMSeconds": 446,
"toolName": "/Use/KritaShape/KisToolBrush"
},
{
"countUse": 3,
"timeUseMSeconds": 417,
```

```
"toolName": "/Use/KritaShape/KisToolLine"
}]
}

{
"actions": [
{
"actionName": "file_save_as",
"countUse": 1,
"sources": "smth"
},
{
"actionName": "send_info",
"countUse": 2,
"sources": "smth"}]
}

{
"Images": [
{
"colorProfile": "RGB/Alpha",
"colorSpace": "sRGB-elle-V2-srgbtrc.icc",
"height": 1200,
"numLayers": 3,
"size": 0,
"width": 1600}]
}

{
"compiler": {
"type": "GCC",
"version": "5.4"
},
"cpu": {
"architecture": "x86_64",
"count": 4,
"family": 6,
"isIntel": true,
"model": 60
},
"general": {
"appLanguage": "ru",
"appVersion": "4.0.0-pre-alpha",
"systemLanguage": "en_US"
},
"opengl": {
"glslVersion": "1.30",
```

```
"renderer": "Haswell Mobile ",
"type": "GL",
"vendor": "Intel",
"vendorVersion": "Mesa 17.0.7",
"version": "3.0"
},
"platform": {
"os": "linux",
"version": "ubuntu-16.04"
},
"qtVersion": {
"value": "5.9.1"
},
"screens": [
{
"dpi": 101,
"height": 768,
"width": 1366}]
}
```

2.2.4 Динамическое изменение данных

Инструменты и действия(actions) могут меняться достаточно часто в процессе разработки графического редактора. Поэтому неразумно задавать статически эти метрики в коде. В коде бекенд-сервера реализована поддержка добавления новых элементов. Раз в сутки просыпается новая горутина(легковесный тред), которая пробегается по базе данных и ищет новые инструменты и действия. После этого она записывает их в текстовый файл. Подобное разумно применять не только для инструментов и действий, но и для любых часто изменяющихся данных. Поэтому подобная система реализована так же для версий приложения. В будущем возможно расширения этой системы.

2.2.5 Агрегация данных

Было бы неразумно обращаться каждый раз к базе данных при запросе с фронтенда. Поэтому все данные предварительно кэшируются. Раз в n минут запускаются горутины, которые осуществляют запросы к базе данных и записывают полученные результаты в соответствующие структруры.

Листинг 2.3 — Агрегация инструментов

```
func AgregateTools() {
    file, err := os.Open("list_tools.txt")
    serv.CheckErr(err)
    defer file.Close()
```

```
5
6       var ToolUse md.ToolsCollected
7       var ToolActivate md.ToolsCollected
8       var ToolsUse []md.ToolsCollected
9       var ToolsActivate []md.ToolsCollected
10
11      scanner := bufio.NewScanner(file)
12      for scanner.Scan() {
13          ToolUse.Name = scanner.Text()
14          ToolUse.CountUse, ToolUse.Time = countToolsUse("/Use/" +
                ToolUse.Name)
15          ToolsUse = append(ToolsUse, ToolUse)
16
17          ToolActivate.Name = ToolUse.Name
18          ToolActivate.CountUse, ToolActivate.Time =
                countToolsUse("/Activate/" + ToolActivate.Name)
19          ToolsActivate = append(ToolsActivate, ToolActivate)
20      }
21      agregatedTools.ToolsActivate = ToolsActivate
22      agregatedTools.ToolsUse = ToolsUse
23      err = scanner.Err()
24      serv.CheckErr(err)
25  }
```

Эта функция требует пояснений. Во второй строчке открывается файл с предварительно сгенерированным списком инструментом. В цикле, который начинается со строчки 22, последовательно считываем названия инструментов из списка. После этого обращаемся к базе данных и считаем сколько раз инструмент использовался и сколько в среднем заняло время этого использования. Начиная со строчки 27 считаем аналогично для активации/деактивации инструмента. Внутри функции countToolUse скрыто обращение к базе данных. Пример этого обращения приведен ниже. Первый запрос считает количество использований, второй считает среднее время использования. Запрос осуществляется при помощи библиотеки mgo - неофициальной обертки для MongoDB для языка Golang[3]

Листинг 2.4 — Агрегация инструментов

```
1  pipe := c.Pipe([]bson.M{{"$unwind": "$tools"}, {"$match":
       bson.M{"tools.toolname": name}}, {"$group": bson.M{"_id":
       "$tools.toolname", "total_count": bson.M{"$sum": "$tools.countuse"}}}})
2  pipe2 := c.Pipe([]bson.M{{"$unwind": "$tools"}, {"$match":
       bson.M{"tools.toolname": name}}, {"$group": bson.M{"_id":
       "$tools.toolname", "total_count": bson.M{"$avg": "$tools.time"}}}})
```

2.3 Кэширование данных на фронтенде

Очевидно, что неразумно каждый раз обращаться к бекенду за новыми данными. Поэтому данные на фронтенде тоже кэшируются в структуры, аналогичные структурам на бекенде. Кэшировать данные на фронтенде и на бекенде может показаться излишним, однако, это помогает значительно увеличить скорость работы приложения, снизить нагрузку на сервер, позволить в дальнейшем легко масштабировать систему.

Таблица 2.1 — Описание собираемых данных

Информация о	Источник данных	Данные
Инструменты	KisTool::activate, KisTool:deactivate	CountUse float64 Time float64 ToolName string
Действия	KisMainWindows actioncollection()	CountUse float64 TimeUse float64 ActionName string
Свойства изображений	KisDocument::saveFile()	ColorProfile string ColorSpace string Height float64 Width float64 Size float64 NumLayers float64
Информация о компьютере	KisDocument::saveFile()	AppVersion string CompilerVersion string CompilerType string CpuArchitecture string CpuCount float64 CpuFamily float64 CpuIsIntel bool CpuModel float64 LocaleLanguage string OpenGlGlslVersion string OpenGlRenderer string OpenGlVendor string PlatformOs string PlatformVersion string QtVersion string ScreenDpi string ScreenHeight string ScrennWidth string
Ассерты	kis_assert_common()	AssertFile string AssertLine float64 AssertText string Count float64 IsFatal bool

3 Технологический раздел

3.1 Плагин

Для написания плагина к графическому редактору был выбран язык C++ в связи с тем, что сам графический редактор написан на этом языке. Преимущества C++:

а) Компилируемый язык со статической типизацией.
б) Сочетание высокоуровневых и низкоуровневых средств.
в) Реализация ООП.
г) Наличие удобной стандартной библиотеки шаблонов

3.2 Backend-сервер

Для написания бекенд-сервера был выбран язык Golang вместе с библиотекой mgo для доступа к базе данных . Плюсы этого языка:

а) Скорость разработки
б) Производительность
в) Удобная реализация легковесных потоков [4]

3.3 Фронтенд-сервер

В качестве языка разработки фронтенд-сервера был выбран Python. Хорошо известно, что Python является одним из самых используемых языков программирования благодаря простоте в изучении, дизайну и гибкости, что делает его практически совершенным языком программирования[5] Существует ряд причин, по которым его можно называть такими громкими словами.

а) Простота в изучении
б) Чистота и читаемость
в) Разносторонность
г) Быстрота написания
д) Цельный дизайн

В качестве вспомогательного фрейворка был выбран Django. Плюсы Django:

а) Быстрота: Django был разработан, чтобы помочь разработчикам создать приложение настолько быстро, на сколько это возможно. Это включает в себя формирование идеи, разработку и выпуск проекта, где Django экономит время и ресурсы на каждом из этих этапов.

б) Полная комплектация: Django работает с десятками дополнительных функций, которые заметно помогают с аутентификацией пользователя, картами сайта,

администрированием содержимого, RSS и многим другим. Данные аспекты помогают осуществить каждый этап веб разработки.

в) Безопасность: Работая в Django, вы получаете защиту от ошибок, связанных с безопасностью и ставящих под угрозу проект. Я имею ввиду такие распространенные ошибки, как инъекции SQL, кросс-сайт подлоги, clickjacking и кросс-сайтовый скриптинг. Для эффективного использования логинов и паролей, система пользовательской аутентификации является ключом.

г) Масштабируемость: фреймворк Django наилучшим образом подходит для работы с самыми высокими трафиками. Следовательно, логично, что великое множество загруженных сайтов используют Django для удовлетворения требований, связанных с трафиком.

3.4 База данных

В связи с возможной изменчивостью данных, приходящих с клиента, было решено использовать MongoDB. Плюсы MongoDB:

а) Бесструктурность

б) Возможность масштабировать сервера "из коробки"

в) Удобный NoSql синтаксис запросов

Заключение

В результате выполнение курсового проекта была создана система телеметрии для графического редактора Krita. Была создан плагин для графического редактора, frontend-сервер и backend-сервера. Был собран тестовый релиз для операционной системы windows. Этот релиз был опробовал несколькими пользователями.

Список использованных источников

1. *Kazakov, Dmitry.* Krita history / Dmitry Kazakov. — https://krita.org/en/about/history, 2014.

2. *Krause, Volter.* KUserFeedback / Volter Krause. — https://github.com/KDE/kuserfeedback, 2017.

3. *feeeper.* Шпаргалка по работе с JSON в Golang / feeeper. — https://ashirobokov.wordpress.com/2016/09/22/json-golang-cheat-sheet/, 2016.

4. *Зубач, Николай.* О плюсах и минусах Go / Николай Зубач. — https://habrahabr.ru/post/229169/, 2014.

5. *Azerus, Monty.* Плюсы и минусы Django / Monty Azerus. — https://python-scripts.com/django-obzor, 2017.