обеспечить основную функцию лечебно-профилактического учреждения – защиту здоровья граждан. Здесь надо искать компромисс. Надо, чтобы менеджеры от медицины осознали важность проблемы защиты персональных данных своих пациентов».

ВЫВОДЫ. 1. Чтобы минимально защитить конфиденциальные ПМД, нужно отказаться от всевозможных бумажных носителей медицинских данных и перейти на электронные. К сожалению, в части внедрения электронных медицинских карт Беларусь намного отстаёт от России. Отставание имеется и в части законодательства.

2. При исключении бумажной документации в медучреждении защиту ПМД в ЛВС медицинского учреждения традиционными методами (защита от несанкционированного доступа, защита от хакерских атак и т. д.)) можно считать достаточной, а, главное, дешёвой. Естественно, однако, что в этом случае стопроцентной гарантии исключения утечки данных для пациента через работников медучреждения нет.

3. Наиболее материально обеспеченные пациенты могут достичь более высокого уровня защиты своих ПМД аутентификацией за умеренную плату своей личности по радужке с помощью описанного в [5] архиватора. Воспользоваться аутентификацией по радужке могут и менее обеспеченные пациенты, если нарушение заданной характеристики безопасности ПМД, обрабатываемых традиционными методами, может привести к значительным негативным последствиям для таких пациентов.

Список использованных источников.

1. Защита персональных данных в медицинских учреждениях [Электронный ресурс]. – Режим доступа: www.bit-medic.ru/articles/zashita-personalnyh-dannyh/. – Дата доступа 25.03.2018.

2. О чём молчат сотрудники похоронных бюро? Беларускія навіны – [Электронный ресурс]. – Режим доступа: www.newsby.org/by/2011/08/11/text20890.htm. – Дата доступа: 25.03.2018.

3. Электронная медицинская карта. Введение в картотеку « e-Gov.by ... [Электронный ресурс]. – Режим доступа: e-gov.by/stroitelstvo-e-gov/elektronnaya-medicinskaya-karta-vvedenie-v-kartoteku. – Дата доступа 25. 03.2018.

4. Электронная медицинская карта (ЭМК) [Электронный ресурс]. – Режим доступа: swan-it.ru/elektronnoe_zdravoohranenie/elektronnaya_meditsinskaya_karta. – Дата доступа 25. 03.2018..

5. . Гивойно, А. А. Защита медицинских данных пациентов / А. А. Гивойно, В. Н. Ростовцев // Доклады БГУИР. – 2016. – № 7 (101). – С. 79–83.

6. Безопасное архивирование данных с помощью биометрических технологий / А. А. Гивойно, С. В. Нестерович, Г. В. Сечко, Т. Г., Таболич Т. Г. // Веснік сувязі. – 2013. – № 6 (122). – С. 25–28.

7. Обзор: ИТ в здравоохранении 2014, Денег на ИБ не хватает - CNews [Электронный ресурс]. – Режим доступа: www.cnews.ru/reviews/publichealth2014/articles/deneg_na_ib_ne_hvataet. – Дата доступа 25.03.2018.

МНОГОПОТОЧНОСТЬ В UNITY СРЕДСТВАМИ РЕАКТИВНЫХ РАСШИРЕНИЙ

Институт информационных технологий БГУИР,
г. Минск, Республика Беларусь

Соколов В.А. Скуратович Е.С.

Бакунова О.М., ст. преподаватель каф. ИСиТ, м.т.н.

В данной статье будут затронуты основные проблемы, возникающие при разработке многопоточных мобильных приложений на платформе Unity, а также способы их решения с использованием библиотеки и реактивных расширений UniRx.

С помощью языка C# можно создавать приложения, выполняющие несколько задач одновременно. Задачи, которые потенциально могут задержать выполнение других задач, выполняются в отдельных потоках; такой способ организации работы приложения называется многопоточностью или свободным созданием потоков. Приложения, которые используют многопоточность, более продуктивно реагируют на действия пользователя, так как пользовательский интерфейс остается активным, в то время как задачи, требующие интенсивной работы процессора, выполняются в других потоках. Многопоточные приложения на языке C# при использовании Mono разрабатываются с помощью ключевых слов: ThreadPool, Thread и асинхронных делегатов.

В качестве примера многопоточного приложения можно представить работу в ресторане. Предположим, что каждый сотрудник выполняет свои обязанности в одно время с другими сотрудниками. К примеру, один готовит еду, второй убирает столы и т.д. (и все это происходит одновременно). Это и есть наши потоки.

ThreadPool — реализация паттерна «пул объектов». Его смысл в эффективном управлении потоками: создании, удалении, назначении им какой-то работы. Возвращаясь к ресторанной аналогии, ThreadPool — это шеф-повар, который контролирует количество сотрудников на стройке и назначает каждому из них задания на день.

Класс, который позволяет создавать новые потоки внутри существующего приложения, называется Thread.

Асинхронные делегаты — асинхронный вызов метода с помощью делегата, который определен с такой же сигнатурой, что и вызываемый метод. Для асинхронного вызова метода необходимо использовать метод BeginInvoke. При таком подходе делегат берет из пула поток и в нем выполняет некоторый код.

Инструменты для синхронизации потоков

Язык C# предоставляет инструменты для синхронизации потоков. Эти инструменты представлены в виде Monitor и lock. Они используются для того, чтобы работа блока кода не осуществлялось одновременно несколькими потоками. Но есть одно различие. Использование этих инструментов может привести к deadlock'у (взаимоблокировке потоков). Это происходит следующим образом: поток А ждёт, когда поток В вернет управление, а поток В, ждет, когда поток А выполнит заблокированный код. Поэтому синхронизацию потоков и многопоточность необходимо использовать осторожно.

Проблемы встроенных механизмов многопоточности в Unity.

Основной проблемой, с которой сталкиваются программисты при разработке однопоточных приложений — это UI-зависания, вызванные выполнением сложных операций в основном потоке. В Unity имеется механизм распараллеливания задач, представленный, как coroutine (корутин), но он работает в одном потоке, и, если запустить в корутине что-либо «тяжеловесное», то произойдет зависание. Если от программиста требуется параллельное выполнение функций в основном потоке, то можно использовать корутины. Однако, хотелось бы напомнить, что корутины — это итераторы, которые в Unity работают следующим образом:

- сначала идет регистрация корутина,
- далее, после каждого вызова Update и перед вызовом LateUpdate, Unity опрашивает все зарегистрированные корутины и обрабатывает код, который описан внутри метода, имеющий возвращаемый тип IEnumerator.

Помимо своих преимуществ, корутины также имеют и недостатки:

1. Нельзя получить возвращаемое значение
2. Обработка ошибок
3. Проблемы с callback'ами
4. Нельзя обрабатывать тяжеловесные методы в корутинах

Как мы уже отметили ранее, что корутины работают в основном потоке. По этой причине мы получаем зависание, запуская в них громоздкие методы. Все эти недостатки запросто устраняются с помощью реактивных расширений, которые в дальнейшем принесут много различных возможностей и упростят разработку.

Реактивные расширения — набор библиотек, которые позволяют работать с событиями и асинхронными вызовами наподобие Linq запросов. Цель подобных расширений — сделать процесс написания кода быстрее, в котором участвует асинхронное взаимодействие. В Unity применяется библиотека UniRx, которая реализует базовый функционал реактивных расширений для Unity на базе .NET Reactive Extensions. Почему же нельзя использовать эту родную реализацию? Дело в том, что стандартные RX в Unity не работают. Библиотека UniRx поддерживается на платформах PC, Mac, Android, iOS, WP8, WindowsStore и является кроссплатформенной.

Основой реактивных расширений являются интерфейсы IObservable и Iobserver. Они предоставляют типизированный механизм для push-уведомления, также известный как шаблон проектирования «Наблюдатель».

- Интерфейс IObservable представляет класс, который отправляет уведомления (поставщик). Интерфейс Iobserver представляет класс, который их получает (наблюдатель). Так же существует класс «T», предоставляющий информацию для уведомлений. Реализация IObserver подготавливает к получению уведомлений от поставщика (реализация IObservable), передавая свой экземпляр методу поставщика IObservable. Subscribe. Этот метод возвращает объект IDisposable, который может использоваться для отказа от подписки наблюдателя до того, как поставщик завершит отправку уведомлений. Интерфейс IObserver определяет три следующих метода, которые должен реализовать наблюдатель:
- Метод OnNext, который вызывается поставщиком для предоставления наблюдателю новых данных или сведений о состоянии
- Метод OnCompleted, который вызывается поставщиком для подтверждения завершения отправки уведомлений наблюдателю.
- Метод OnError, который вызывается поставщиком для указания того, что данные являются недоступными, поврежденными, или у поставщика возникли другие ошибки.

Также в UniRx реализован Scheduler — главный компонент, посредством которого реализована многопоточность. Базовые временные операции (Timer, Interval) в UniRx реализованы с помощью MainThread. Это означает, что большинство операций, кроме Observable.Start, работают в основном потоке и потокобезопасностью. Observable.Start по умолчанию использует ThreadPoolScheduler — это означает, что будет создан поток. С основными понятиями и теоретическими знаниями мы ознакомились, теперь рассмотрим пример создания наблюдателя

В данном примере мы попытаемся получить данные из какого-либо интернет-ресурса с помощью библиотеки UniRx. Для загрузки данных с помощью реактивных расширений нам необходимо создать наблюдателя и использовать класс ObservableWWW, который является оберткой над стандартным классом WWW Unity. Метод Get использует корутины и возвращает IObservable, к которому мы присоединим наблюдателя. Данный подход позволяет избежать проблем и ошибок.

Выводы.

Использование реактивных расширений при разработке приложений на Unity имеет множество преимуществ. Основным из них является упрощение синтаксиса для построения многопоточных приложений. Количество костылей с корутинами значительно уменьшается, приложение становится более гибким и быстрым. Также при построении многопоточного приложения с помощью UniRx необходимо помнить, что любая часть данных должна быть защищена от изменения их значений множеством потоков.

Список использованных источников.

1. Документация Unity [Электронный ресурс] – Режим доступа: https://https://docs.unity3d.com/. – Дата доступа: 16.03.2018.

2. Документация по C# [Электронный ресурс] – Режим доступа:https://msdn.microsoft.com/ru-ru/.– Дата доступа: 16.03.2018.