Технология разработки сервиса

Оглавление

3.1. Выбор операционной системы, языка и среды программирования, технологий и средств передачи данных.

Так как разрабатываемый сервис построен на клиент-серверной архитектуре, то необходимо выбрать инструменты, как для создания клиентской части, так и серверной.

Серверную часть можно выполнить как в виде скриптов, работающих на веб-сервере, так и в виде отдельного компилируемого приложения. Среди скриптовых языков построения серверной части наиболее популярными являются php, javascript, python, ruby. Основным их достоинством является то, что при изменении участка кода программы нет необходимости перекомпилировать всю программу, они кроссплатформенны и поддерживаются многими веб-серверами. Разрабатывать программное обеспечение на таких языках можно используя обычный текстовый редактор. Однако основным недостатком этих языков является скорость исполнения программы – так как каждый раз производится интерпретация написанного кода.

В отличие от интерпретируемых языков, компилируемые языки программирования не имеют такого недостатка. К компилируемым языкам можно отнести C, C++, Java, C#. Так же, для работы программы, написанной на компилируемом языке необязательно содержать веб-сервер.

На основании приведенного сравнения для реализации серверной части был выбран язык C#, так как в сравнении с другими языками он имеет большое преимущество.

C#, в отличие от C и C++ не имеет указателей и автоматически собирает мусор, что позволяет вести разработку проще и легче, уменьшает количество ошибок, связанных с утечкой и выходом за границы адресного пространства программы. Так как язык C# использует компиляцию программы во время её исполнения, то модули, неиспользуемые в данный момент не будут подгружены, что приводит к более рациональному использованию памяти и других ресурсов компьютера. Система сборок и

пространств имен в языке C# позволяет более просто поддерживать сложные программы.

Язык Java обладает всеми теми же преимуществами что и язык C# по сравнению с C и C++. Но, на данный момент, язык C# развивается намного быстрее Java, которая, по сути, ориентирована на корпоративный сектор. Технологии, используемые в Java и C# имеют много общего, но, в отличие от сред разработки на Java, среды для разработки на языке C# обладают набором инструментов, позволяющих автоматизировать большинство операций.

В качестве языка для разработки клиентской части так же был выбран язык C# и технология Silverlight. Преимуществом Silverlight является так же, что он компилируемый и, в поддерживаемых браузерах отображается одинаково без лишних ухищрений, в отличие от html, css и javascript, реализация которых сильно различается в различных браузерах. Так же Silverlight работает и в ОС семейства GNU/Linux при помощи плагина Moonlight. Преимущество Silverlight/Moonlight перед Flash – скорость работы, наличие бесплатных средств для разработки приложений, использование одного и того же языка и фреймворка для разработки клиентской и серверной частей.

В качестве среды программирования была выбрана Microsoft Visual Studio, так как на данный момент она обладает более широким спектром возможностей по отладке, профилированию и развертыванию приложений. Технологии, используемые при разработке, например объектно-ориентированная технология доступа к данным (ORM), имеют модули для автоматической генерации кода только для данной среды разработки. Так как необходима совместимость с Mono, то в процессе разработки программного обеспечения необходимо осуществлять проверку написанных компонентов под данной платформой. Помимо среды MonoDevelop только Visual Studio имеет профили для компиляции программы под Mono Framework. Так же, только для Visual Studio имеется набор утилит, позволяющий формировать

пакеты программ для GNU/Linux, производить запуск и отладку приложений на удаленном компьютере с Linux.

В качестве операционной системы была выбрана Microsoft Windows, так как Visual Studio работает только на данном семействе операционных систем.

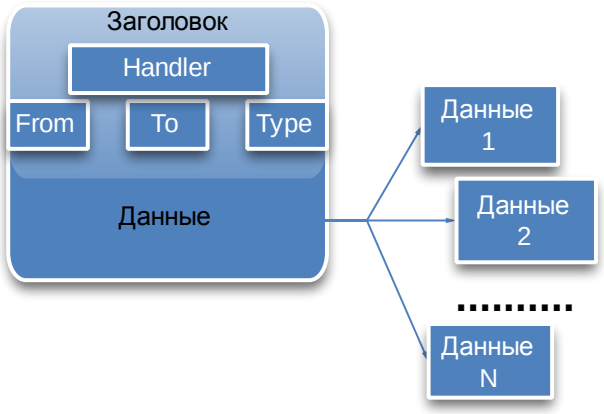
Для работы с базами данных была выбрана технология объектно-реляционного отображения, так как она позволяет абстрагироваться от используемой СУБД, отказаться от использования SQL-запросов в коде программы и производить работу с сущностями из базы данных как с локальными объектами. Данная технология позволяет менять используемую СУБД без изменения кода программы. В качестве реализации такой технологии был использован Mindscape Lightspeed. В отличие от Entity Framework Lightspeed работает и в среде Mono. А преимущество перед NHibernate состоит в наличии бесплатных средств построения модели базы данных и автоматической генерации объектного отображения.

В качестве самой СУБД была выбрана СУБД MySQL Server 5.5, так как она имеет хорошие показатели быстродействия и обладает всеми необходимыми функциями для обеспечения работоспособности проекта.

В качестве технологии передачи данных были выбраны асинхронные сокеты Беркли. Реализация WCF для платформы Mono находится в зачаточном состоянии и на данный момент работает нестабильно. Асинхронные сокеты были выбраны потому, что, в отличие от асинхронных, не блокируют вызвавший процедуру поток, чем экономят ресурсы компьютера.

3.2. Особенности программной реализации, генерации программного кода и отладки

Сервер обменивается данными с клиентской частью объектами класса, представляющего собой сообщение. Оно состоит из заголовка и тела данных. В заголовке содержится информация об отправителе, получателе и обработчике сообщения. Наглядно сервисные сообщения можно представить следующим образом:



Все подсистемы на стороне сервиса существуют в единственном экземпляре и для них необходимо наличие механизм идентификации клиента. В качестве такого механизма был разработан аналог сессий, используемых в php – при первом подключении пользователю присваивается идентификатор и создается окружение, содержащее некоторые пользовательские данные. При отправке следующих сообщений серверу, клиент вместе с необходимыми данными отправляет присвоенный ему в данный момент идентификатор. По этому идентификатору ядро сервера выбирает информацию о пользователе из оперативной памяти и автоматически добавляет его к данным сообщения, которые передаются подсистемам. При отключении клиента ресурсы, занимаемые окружением, освобождаются.

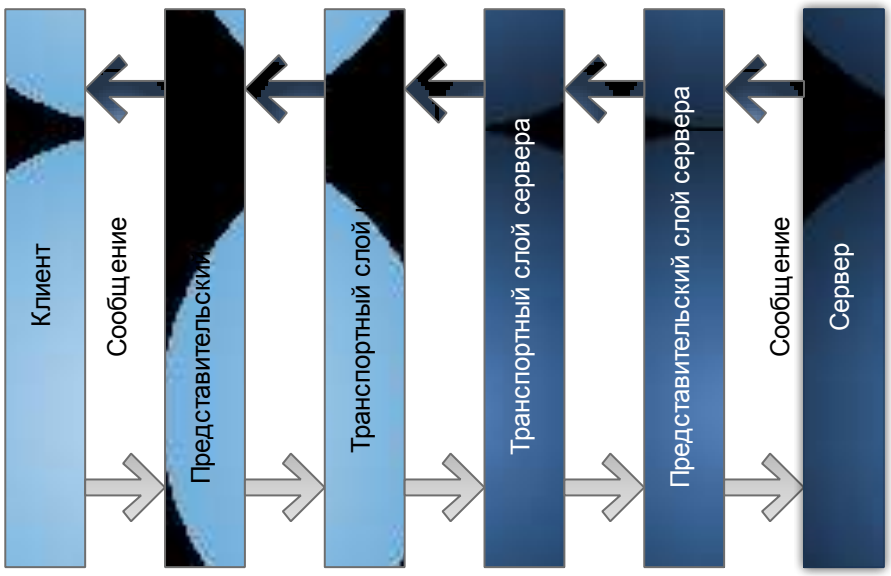
Данный механизм позволяет хранить часто используемые данные в оперативной памяти, что приводит к снижению количества запросов в базу данных и операций чтения/записи с внешних устройств.

В окружении пользователя содержится небольшое количество данных, которые наиболее вероятно будут использованы в будущем. Так, например, когда один из участников проекта отправляет текстовое сообщение в чат, клиент посылает серверу соответствующее сервисное сообщение. Подсистема чата, обрабатывающая данное сообщение запишет сообщение чата в соответствующую таблицу базы данных, а так же в окружения всех других участников данного проекта, находящихся онлайн в данный момент времени. Когда клиентские приложения других участников проекта отправят сервисные сообщения о проверке входящих сообщений в чате, данные будут переданы им из их окружений на сервер, а не считаны из базы данных, после чего занимаемая данными текстовыми сообщениями память будет очищена.

Как видно из приведенного выше примера, данный механизм позволяет оптимизировать доступ к одному из самых медленных ресурсов компьютера – жесткому диску, храня часть наиболее востребованных данных в оперативной памяти.

Помимо сообщений чата, в окружении пользователя, хранятся так же актуальные изменения кода в редактируемом в данный момент файле другими пользователями, логин пользователя, дата входа в систему и другая информация.

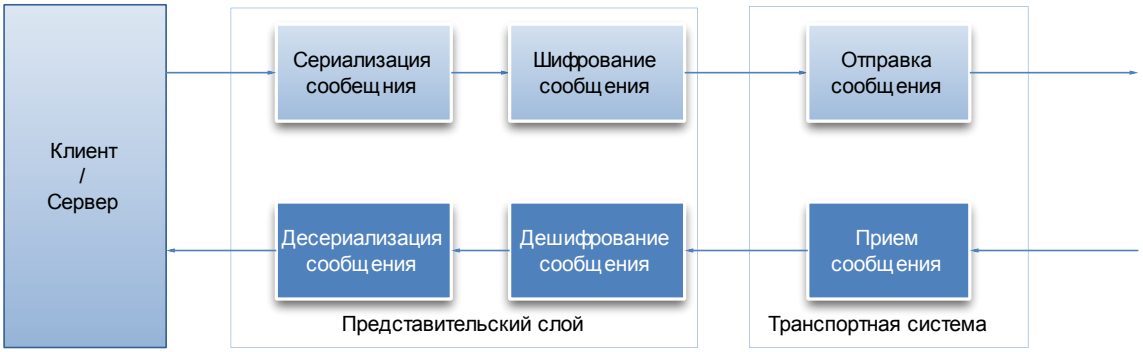
Взаимодействие между серверной и клиентской частью можно представить следующим образом:



Темные стрелки указывают на движение сообщений от сервера к клиенту, а светлые – от клиента к серверу.

Слой представления информации, как и в модели OSI, служит для кодирования/декодирования передаваемых данных. Транспортный слой отвечает за передачу данных по сети.

Данные слои обеспечивают один и тот же функционал для клиентской и серверной частей.

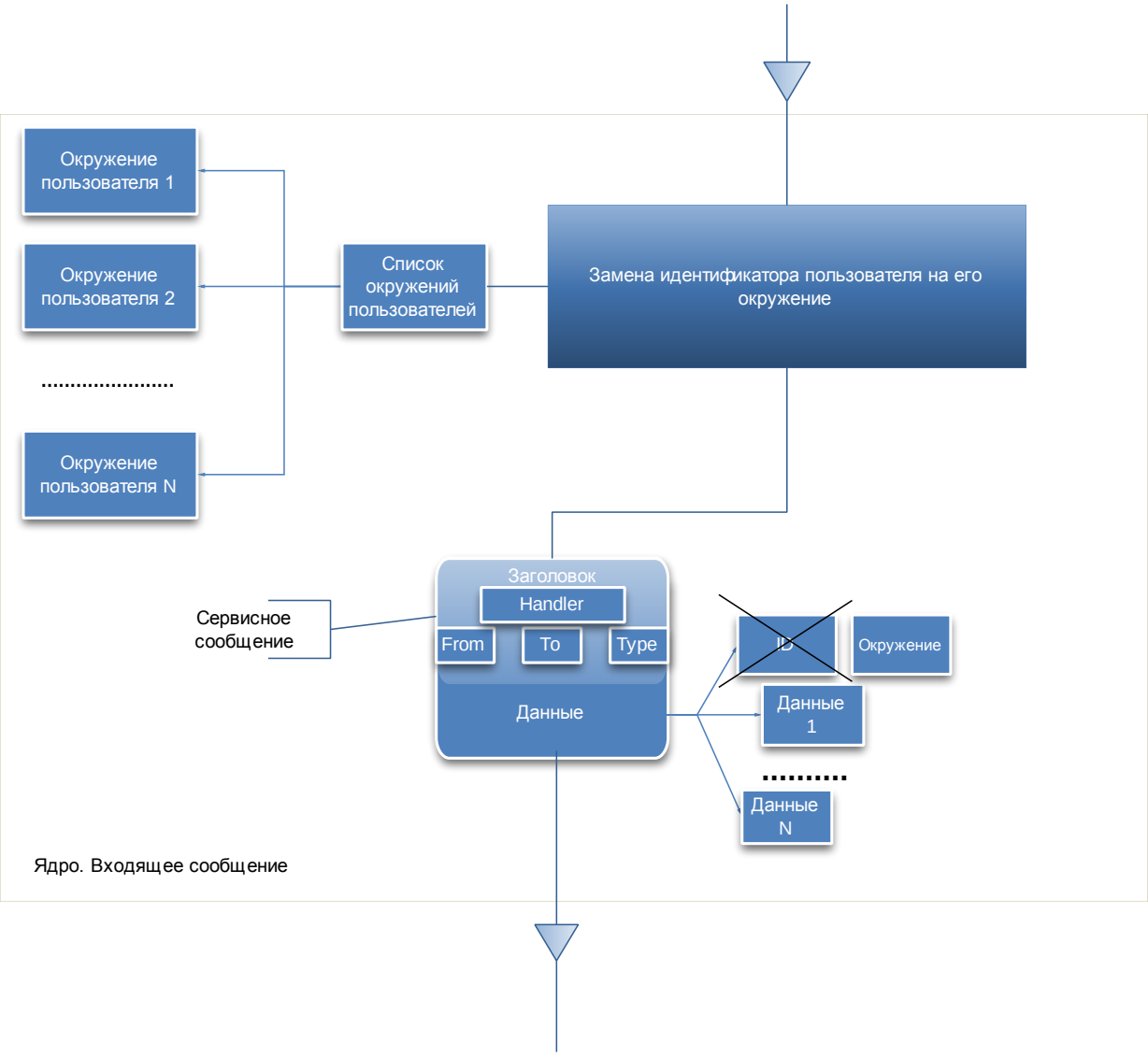


Транспортный слой реализован при помощи асинхронных сокетов беркли и отвечает за передачу данных между клиентом и сервером.

При отправке сообщения представительский слой осуществляет сериализацию объектов, представляющих сервисные сообщения, в xml формат и производит их шифрование.

При получении сообщения сначала производится дешифрования принятых данных, а потом их десериализация. В результате мы получаем сервисное сообщение, отправленное другой стороной.

Процесс подмены идентификатор пользователя его окружением производится в ядре, после десериализации принятого сообщения. Идентификатор пользователя находится в 1 блоке данных в теле данных сообщения. Таблица окружений пользователей является хранилищем типа «ключ-значение». Ключом в данной таблице выступает идентификатор пользователя, а значением – ссылка на его окружение.



Ядро. Входящее сообщение

Ядро извлекает из данных сообщения идентификатор пользователя, затем осуществляет поиск окружения пользователя в таблице окружений

пользователей по данному идентификатору. Если окружение было найдено, то значит, пользователь прошел процедуру авторизации на сервере и его идентификатор заменяется ссылкой на окружение. Для обеспечения защиты от подмены идентификаторов для доступа к окружениям производится проверка IP адреса пользователя, от которого пришел запрос с указанным идентификатором и IP адреса, для которого было создано окружение.

Для обеспечения работы сервиса с базой данных была использована ORM технология с применением решений Mindscape Lightspeed.

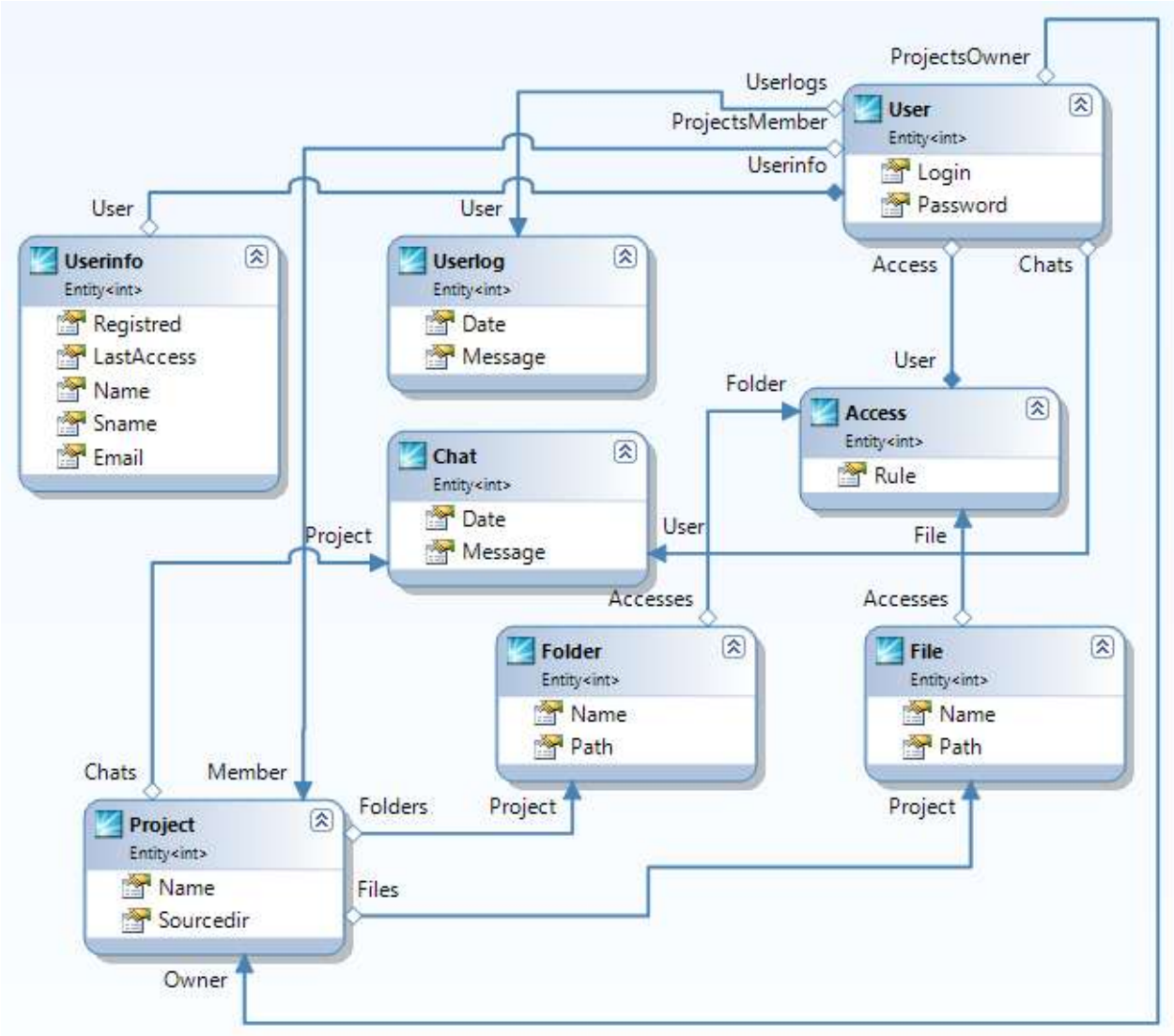
Как уже было упомянуто выше, Lightspeed, в отличие от Entity Framework, работает не только на платформе .Net, но и на Mono.

Плагин Lightspeed для Visual Studio 2010 позволяет осуществлять проектирование базы данных прямо в среде разработки. Так же позволяет осуществлять реверс-инжиниринг отдельных сущностей и всей базы данных целиком. При проектировании модели базы данных можно настроить ограничения целостности.

При изменении модели базы данных плагин автоматически генерирует отображение сущностей базы данных в объекты программы.

Фреймворк Lightspeed работает с множеством поставщиков данных к базам данных, что позволяет изменить используемую базу данных без перекомпиляции программы, просто переконфигурировав приложение и подключив сборку соответствующего поставщика данных.

В ходе проектирования базы данных была создана следующая модель:



База данных состоит из 8 таблиц:

1. User – представляет пользователя. Содержит имя для входа пользователя (Login) и пароль (Password)
2. Userinfo – представляет дополнительную информацию о пользователе. Содержит информацию о дате регистрации пользователя (Registred), времени последнего доступа к системе (LastAccess), имя пользователя (Name), фамилию (Sname) и адрес электронной почты (Email)
3. UserLog – содержит информацию о действиях пользователя. Содержит поля для хранения времени происхождения события (Date) и описания события (Message)
4. Chat – предназначена для хранения сообщений чата проектов. Состоит из поля даты отправки сообщения (Date), сообщения (Message),

идентификатора пользователя, отправившего сообщение (User), и идентификатора проекта чата (Project)

5. Access – содержит информацию о доступе правах доступа пользователей к ресурсам проекта. Состоит из правила (Rule), идентификатора пользователя, для которого применяет данное правило (User) и ресурса (File или Folder)
6. File – таблица, содержащая информацию о всех файлах проектов. Содержит имя файла (Name), путь к файлу (Path), идентификатор прав доступа к файлу (Access), идентификатор проекта, которому принадлежит файл (Project)
7. Folder – аналогично таблице описания файлов, представляет информацию о папках проектов. Состоит из имени папки (Name), пути к папке (Path), списку прав доступа (Access), проекту, в котором находится папка (Project)
8. Project – содержит информацию о проектах. Состоит из имени проекта (Name), пути к файлам проекта (SourceDir), идентификаторы владельца проекта (Owner), списков участников проекта (Members), файлов (Files), папок (Folders), чатов (Chats)

После завершения проектирования модели базы данных плагин Lighspeed позволяет автоматически сгенерировать скрипт SQL для создания таблиц базы данных, определения всех ключей, индексов и ограничений целостности для выбранной СУБД.

Защита данных, с которыми работает сервер, осуществляется путем шифрования входящих/исходящих сообщений, а защита данных, хранимых в базе данных, осуществляется СУБД. Так как все данные записываются в базу данных и в окружение пользователей, то при возникновении непредвиденных ситуаций, либо аппаратных или программных сбоев верная копия данных всегда будет содержаться в базе данных.

3.3. Тестирование и отладка сервиса.

Для отладки серверной и клиентской части используется встроенный в Visual Studio отладчик. Помимо пошаговой отладки он позволяет ставить точки останова, ставить условия останова, просматривать значения локальных и глобальных переменных в текущий момент жизни программы, стека вызова методов, список активных потоков и т.д.

Так же имеются инструменты для определения узких мест программы, основываясь на некоторых метриках – объем памяти, время работы, количество строк кода, используемые типы данных и т.п.

При помощи профилировщиков можно посмотреть, сколько % процессорного времени заняло исполнение тех или иных процедур и на основании этих результатов определить места для оптимизации алгоритмов.

Например, информация, собранная при запуске сервера:

Function Name	Number of Calls	Elapsed Inclusive T...	Elapsed Inclusive T...
IDEService.exe	0	422,78	100,00
IDEService.Program.Main(string[])	1	422,78	100,00
IDEService.Core.BootLoader.Init()	1	395,42	93,53
IDEService.Core.BootLoader.LoadSubsystems()	1	393,21	93,01
IDEService.Core.BootLoader.LoadSubsystem	8	392,11	92,75
System.Object.ToString()	8	58,45	13,82
IDEService.Core.SubsystemFactory.GetSu	8	201,26	47,61
IDEService.Core.DataBaseSubsystem.	1	77,36	18,30
Mindscape.LightSpeed.LightSpee	1	68,82	16,28
IDEService.Core.Kernel.RegisterSubsyste	8	35,91	8,49
IDEService.Core.Configure.get_Cfg()	8	86,42	20,44

В данной таблице отображено количество вызовов каждой функции, время её выполнения в миллисекундах и процент от времени выполнения всей программы. Исходя из результатов, представленных в таблице, видно, что весь запуск программы занимает 422мсек, из которых 393(93%) – загрузка подсистем ядра. Аналогично можно собрать данные обо всей работе сервиса и выявить методы, которые слишком долго, либо слишком часто выполняются для проведения последующей оптимизации.

При разработке сервиса применятся 2 вида тестирования:

1. Юнит тестирование, позволяющее проверить корректность работы отдельных модулей разрабатываемого сервиса

2. Нагрузочное тестирование, позволяющее проверить корректность работы серверной части при больших нагрузках

Для модульного тестирования применяется NUnit. Юнит тестами покрываются все типы сообщений, передаваемых подсистемам и все методы модулей этих подсистем.

Так как модули подсистем не существуют отдельно от своих подсистем (например, если модуль обращается к базе данных, то при инициализации он посылает сообщение ядру на получение объекта работы с базой данных), то для тестирования методов модулей были реализованы специальные классы, реализующие те же алгоритмы, но не зависящие от других подсистем (например, в данных классах создается отдельный объект для работы с базой, не опрашивая ядро).

При успешном прохождении всех тестов модулем-клоном, содержимое его методов копируется в основной класс-модуль.

Тесты для методов составляются таким образом, что бы покрыть все диапазоны всех аргументов метода. Для значений строкового типа обычно выполняется 2 теста – правильное значение параметра и неправильное. Для числовых значений 4 теста – правильное, неправильное, левый граничный элемент и правый граничный элемент.

После того, как модуль прошел все тесты, происходит тестирование подсистемы, управляющей этим модулем.

Для подсистемы составляются тесты для всех типов обрабатываемых ею сообщений. Для каждого сообщения строятся тесты с правильными и неправильными наборами значений так, что бы максимально покрыть комбинации правильных и неправильных значений.

Для проведения нагрузочного тестирования для клиентской части был разработан специальный класс, который имитирует случайные действия пользователя с небольшими задержками. Одновременно запускаются несколько клиентских приложений с аккаунтами разных пользователей, на них запускаются модули загрузочного тестирования, а для серверной части

включается профилировщик. Он позволяет определить участки программы, которые необходимо изменить. Если изменение данных участков не возможно либо не оправданно, например, операции, связанные с изменением данных в базе данных, то с помощью профилировщика можно сделать вывод о максимально возможном количестве пользователей сервиса на определенном оборудовании, либо, в данном примере, попробовать использование другого коннектора или СУБД.

3.4. Особенности внедрения и сопровождения сервиса