Санкт–Петербургский государственный университет
Кафедра технологии программирования

Гнатюк Александр Валерьевич

Выпускная квалификационная работа бакалавра

Разработка API и средств интеграции для личного кабинета обучающегося СПбГУ

Направление 010300
Фундаментальная информатика и информационные технологии

Научный руководитель,
Старший преподаватель
Севрюков С.Ю.

Рецензент
Кандидат физ.–мат. наук,
доцент кафедры КММС
Корхов В.В.

Санкт–Петербург
2017

Оглавление

Введение

В последние годы наблюдается рост использования информационных технологий. Благодаря современным технологиям меняются способы взаимодействия людей друг с другом, за счет этого многократно возрастает эффективность их работы.

Учебные заведения, в частности, университеты, не являются исключением. Образовательное учреждение представляет собой множество различных подразделений, связанных друг с другом или независимых.

Эффективность работы университета можно увеличить путем внедрения современных информационных технологий, которые уже тесно проникли в нашу жизнь.

Информационные технологии в образовательном учреждении можно использовать в нескольких направлениях:

1) административно–управленческое, которое характеризуется автоматизацией планирования, контроля и организации;
2) научно–исследовательское, в котором специализированные технологии используются для какой-либо научной деятельности;
3) учебный процесс, в котором технологии направлены на улучшение качества обучения.

Для улучшения административно–управленческой деятельности успешно используются возможности единого интерфейса для доступа ко всему необходимому. Часто в качестве него используется веб-приложение [1]. Пользователь не задумывается о необходимости сложных манипуляций для получения нужных данных, а лишь пользуется разработанным приложением. Например, он может оперировать большим объемом информации, получать ее в агрегированном виде, не задумываясь о технической составляющей.

В век информационных технологий, почти у каждого человека, который пользуется Интернетом, есть один или несколько аккаунтов в социальных сетях. Студенты используют разные развлекательные и информационные ресурсы, не имея при этом возможности пользоваться всеми услугами и

сервисами современного учебного заведения XXI века в личном кабинете студента.

Постановка задачи

Целью данной работы является оптимизация расходов и ресурсов, используемых при работе с информационной системой.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- Осуществить сбор требований:
  - Ознакомиться с существующими решениями личного кабинета студента;
  - Выделить потенциально необходимую функциональность;
  - Изучить текущую архитектуру и реализацию личного кабинета;
  - Выявить сильные и слабые стороны решения;
- Произвести тестирование компонентов текущей реализации:
  - Изучить литературу, посвященную видам и способам тестирования;
  - Произвести тестирование базы данных;
- Ознакомиться с текущей реализацией компонента, на основе которого будет, создано API и средства интеграции:
  - Ознакомиться с архитектурой решения;
  - Выявить недостатки текущего решения и предложить способы их исправления;
- Ознакомиться со способом организации распределенных систем;
- Спроектировать архитектуру системы;
- Осуществить разработку и документацию API;
- Реализовать прототип средств интеграции, для интегрирования, созданного ранее API;
- Провести тестирование разработанного решения.

Обзор литературы

В этом разделе приведен обзор некоторых книг и статей, наиболее активно используемых автором при написании работы.

Грегор Хоп, Бобби Вульф, Шаблоны интеграции корпоративных приложений, 2016 [2]

Эта книга наиболее активно использовалась при написании работы.

В данной книге не рассматриваются конкретные технологии или продукты. Она посвящена интеграции корпоративных приложений с помощью обмена сообщениями. В ней рассматриваются существующие шаблоны интегрирования их слабые и сильные способы. Благодаря ей можно понять необходимость использования такого решения как очередь. В ней детально описываются всевозможные тонкости при работе с очередями.

Статья Грученков В. В. Новиков В. И. Микросервисная архитектура веб-приложений, 2016 [3]

В статье рассматривается такое явление, как микросервисная архитектура, объясняются различные варианты ее использования, возможности применения. Также в ней написано в каких случаях лучше использовать данную архитектуру и почему. В статье рассмотрены основные ее достоинства.

Пивень А.А., Скорин Ю.И. ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ, 2012 [4]

В статье рассмотрены и проанализированы виды тестирования программного обеспечения, а также представлена классификация тестирования по объекту тестирования. Данная статья использовалась при проведении тестирования базы данных и тестировании прототипов.

Глава 1. Сбор требований

В связи с ростом компьютеризации и информатизации, возникают ситуации, в которых появляется необходимость эффективного использования функциональности некоторого программного продукта в решениях, которые разрабатываются третьими лицами и с использованием других технологий. Для решения такого рода задача существует такая практика, как API.

Application Programming Interface — интерфейс, позволяющий взаимодействовать с системой. API используется для взаимодействия различных систем между собой. В современном мире существуют различные платформы и программные продукты, которые были разработаны разными людьми с использованием разных стандартах и спецификаций, но благодаря API имеется возможность наладить взаимодействие между ними.

На данный момент личный кабинет обучающегося не имеет ни внешнего, ни внутреннего API. Личный кабинет выступает в роли посредника, который взаимодействует со множеством систем в информационной сети университета. Если рассматривать его отдельно от всей системы, то сам по себе он принимает данные только на вход, потому что реализованная функциональность нацелена на работу с входными данными. Поэтому имеет смысл разработать API, которое помимо ввода данных будет выступать единой точкой взаимодействия с различными системами университета. Тем самым, стороннему разработчику не придется задумываться о взаимодействии с различными элементами ИС университета.

1.1 Анализ существующих реализаций личного кабинета

В данном разделе приводятся итоги анализа и сравнения существующих и доступных решений, обзор которых выполнен перед началом работы (детали обзора в приложении 1)

Функциональность	СПбГУ	ПМ-ПУ	ИТМО	УрФУ
Подача заявок	+	+	-	-
Получение документов	+	-	+	+
Успеваемость	+	-	+	-
Персональное расписание	-	-	+	-
Персональные новости	-	+	+	-
Отзывы о преподавателях и дисциплинах	-	+	-	-
Обмен сообщениями	-	-	+	-
Даты выплат стипендий	-	-	+	-

Таблица 1. Сравнение личных кабинетов

1.2 Итоги анализа

На основе анализа полученных результатов (табл. 1) можно сделать вывод, что личный кабинет в основном направлен на работу с данными. Следовательно, в существующей классификации API прежде всего будет разрабатываться REST API [5]. Такие же классы API, как RPC, использоваться не будут по причине отсутствия необходимости, так как RPC ориентирует разработчиков на работу с операциями [6]. Кроме того, это семейство протоколов и стандартов, которое использует HTTP как транспортный протокол, в то время как REST использует все тонкости протокола HTTP: манипулирование информацией, кэширование, аутентификация, сжатие трафика, передача данных в теле сообщений [7] [8].

На основе полученной информации проведен опрос студентов, благодаря которому удалось выделить функциональность, которая была бы потенциально полезной для студентов, в частности, личное расписание и даты выплат стипендий.

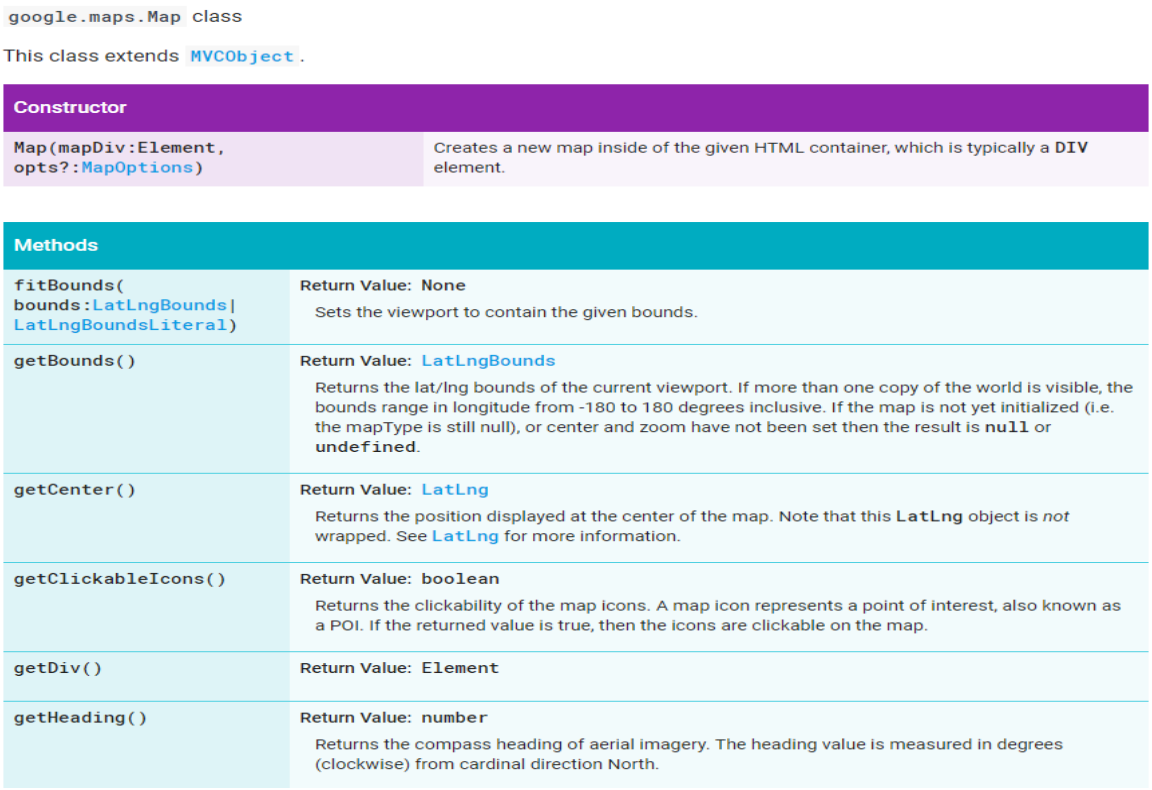
1.3 Характеристики приложения

Конечное приложение должно обладать характеристиками, которые свойственны для современных популярных решений: высокая производительность, отказоустойчивость и т.д. Аналогичная задача предоставления доступа к данным повсеместно встречается в различных сервисах.

Как было сказано ранее, API можно разделить на 2 типа: внешнее и внутреннее.

Внешнее API нужно для того, чтобы с информационной системой могли взаимодействовать различные внешние решения. Например, одним из примеров использования внешнего API является взаимодействие с API социальной сети. Сторонние разработчики могут использовать API для создания программного обеспечения, работающего с данными социальной сети, например, альтернативное клиентское приложение.

Одними из самых ярких примеров поставщиков API являются крупные IT–компании (Google, Yandex, Facebook). Их API предоставляет широкую функциональность. Например, на рисунке 1 представлен скриншот фрагмента документации API Google Maps.



`google.maps.Map` class

This class extends `MVCObject`.

Constructor	
`Map(mapDiv:Element, opts?:MapOptions)`	Creates a new map inside of the given HTML container, which is typically a **DIV** element.

Methods	
`fitBounds(bounds:LatLngBounds|LatLngBoundsLiteral)`	**Return Value:** None Sets the viewport to contain the given bounds.
`getBounds()`	**Return Value:** LatLngBounds Returns the lat/lng bounds of the current viewport. If more than one copy of the world is visible, the bounds range in longitude from -180 to 180 degrees inclusive. If the map is not yet initialized (i.e. the mapType is still null), or center and zoom have not been set then the result is **null** or **undefined**.
`getCenter()`	**Return Value:** LatLng Returns the position displayed at the center of the map. Note that this **LatLng** object is *not* wrapped. See LatLng for more information.
`getClickableIcons()`	**Return Value:** boolean Returns the clickability of the map icons. A map icon represents a point of interest, also known as a POI. If the returned value is true, then the icons are clickable on the map.
`getDiv()`	**Return Value:** Element
`getHeading()`	**Return Value:** number Returns the compass heading of aerial imagery. The heading value is measured in degrees (clockwise) from cardinal direction North.

Рис. 1. Фрагмент документации API Google Maps

На основе подобного API можно создавать свои собственные решения, которые будут работать с уже существующими инструментами. Например, если в разрабатываемом приложении требуется использовать онлайн–карты, то необходимость самому создавать сервис онлайн карт отпадает, потому что на рынке существует достаточное количество компаний, которые могу предложить свои сервисы карт: Google Maps, Яндекс.Карты. Используя такое API, разработчик не задумывается о поддержке и работоспособности используемых сервисов, так как этим заняты люди, которые разрабатывают и поддерживают используемое API.

В современном мире существует большое количество технологий для разработки программного обеспечения. Каждая из них оптимальна в определенных областях, и как следствие, в одной информационной системе несколько таких технологий. Чтобы установить взаимодействие между ними используется внутреннее API. API называется внутренним, потому что доступ к нему имеют только компоненты системы, и, в отличие от внешнего, им не может воспользоваться сторонний разработчик. Таким образом, API используется как средство интеграции.

В качестве примера можно рассмотреть покупку пользователем товара в интернет–магазине. Веб–клиент с помощью API отправляет на сервер запрос, в котором содержится информация о выбранном товаре. В свою очередь сервер, получив запрос, должен проверить наличие товара, информацию об оплате, сформировать заказ для отправки и т.д. Таким образом, покупка товара затрагивает несколько компонент системы. Эти компоненты могут быть написаны разными людьми и на разных языках программирования. Веб клиенту не нужно знать про внутреннее устройство: для него есть посредник, к которому он обращается с помощью API, который в свою очередь уже взаимодействует с этими системами при помощи другого, скрытого для клиента внутреннего API.

1.4 Выводы

В данной главе проведен анализ и сравнение существующих решений личного кабинета студента, были выдвинуты требования к потенциальной функциональности, а именно добавление личного расписания и даты выплаты стипендии. Также были выдвинуты требования к API, которые заключаются в том, чтобы разделить API на две составляющие: внешнее и внутреннее.

Глава 2. Разработка API личного кабинета

Решение разработки API принципиально делится на следующие части:

- разработка интерфейса (API);
- разработка интеграционных механизмов как часть внутреннего API;
- разработка и внедрение компонентов, используемых для авторизации доступа к данным.

Для разработки были использованы такие инструменты как WebAPI, Angular.js, swagger.io [9, 10, 11].

2.1 Разработка API для личного кабинета

Было разработано REST API, позволяющее взаимодействовать с внешними системами. Выбор сделан в пользу REST, потому что API планируется используется для обмена данными. Разработанное API может работать с данными как на ввод, так и на вывод.

Под выводом данных понимается запрос к системе на получение данных. Это может быть вывод данных о расписании, чтобы использовать их в своих целях, например, для решения логистической задачи, либо же вывод какой-либо информации о студенте.

Под вводом данных понимается запрос на добавление определенных данных. Это может быть использовано по-разному, начиная с заполнения заявления на материальную помощь и заканчивая отправкой фотографии или скана документа для нужд учебного отдела (например, справки).

В результате использования такого инструмента как swagger.io, любой разработчик, может ознакомиться с документацией данного API, и может использовать API по своему усмотрению, то есть выполнять взаимодействие с информационной системой университета, тем самым развивая информационную систему [11, 12]. Например, разработчик может создать мобильное приложение, которое будет отправлять уведомления о текущем занятии или изменении в расписании, либо же создать Telegram–бота, который также будет отправлять сообщение об изменении расписания [13].

Следующим шагом является интеграция данного API в информационную систему университета.

2.2 Безопасность API

Для полноценной работы API его необходимо обеспечить средствами безопасности и управления доступом, так как данные могут носить конфиденциальный характер.

Если говорить про разработанное API, то оно ориентировано на работу с данными, которые находятся в источнике данных под управлением MS SQL Server. Это значит, что для сервиса, который должен отвечать за аутентификацию (процесс подтверждения, что этот человек именно тот, за кого себя выдает), нет существенных ограничений по используемым технологиям.

На данный момент существует несколько вариантов аутентификации [14]:

- HTTP Basic Authentication
- HTTP Digest Authentication
- Аутентификация по сертификатам
- Аутентификация по ключам доступа
- Forms Authentication
- Token Authentication

Стоит поподробнее рассмотреть последние два варианта, так как в них есть схожесть и они являются популярными на данный момент.

Суть Forms Authentication заключается в том, что пользователь отправляет свои логин и пароль серверу для аутентификации. В случае успеха сервер возвращает токен, который записывается в клиенте и передается со всеми последующими запросами.

При использовании Token Authentication запрашиваемый сервис делегирует функцию проверки достоверности сведений о пользователе

другому сервису. Таким образом, провайдер услуг доверяет выдачу необходимых для доступа токенов токен–провайдеру.

Под токеном обычно понимается JWT [15]. JWT (JSON Web Token) — это стандарт, который определяет способ передачи данных о пользователе в формате JSON [16]. Токен состоит из трех частей: заголовок, данные, подпись. Заголовок содержит в себе информацию о типе токена и алгоритме его шифрования, тело токена содержит данные, а подпись служит для верификации токена

Есть ряд решений, использующих эти подходы. Одно из таких решений это IdentityServer4[17]. Это решение основано на протоколах OpenID Connect для аутентификации и OAuth 2.0 для авторизации и предназначено для ASP.NET Core. Такое решение предоставляет аутентификацию как сервис, который разворачивает разработчик. При использовании IdentityServer4 можно производить разделение прав доступа, выдавать токены и проводить их валидацию. Преимущество такого решения заключается в том, что оно может быть гибко настроено.

Также решение ответственное за безопасность и управление доступом может быть создано самим разработчиком средствами Visual Studio, как это было сделано на примере API подсистемы “Обучающиеся” ИС “Обучение” СПбГУ. В этом API была реализована выдача токена по логину и паролю, в котором в зашифрованном виде хранилась роль доступа. Каждый последующий запрос содержал этот токен, после расшифровки которого на стороне сервера, можно было понять, имеет ли права доступа пользователь к запрашиваемому ресурсу или нет. Недостаток такого решения заключается в том, что обновление токена происходит не автоматически, а вручную, путем введения логина и пароля еще раз.

Это самый очевидный список решений, но тем не менее существует также множество и других вариантов.

Предполагается реализовать сервис аутентификации и управлением доступа, благодаря которому API можно будет предоставить стороннему

разработчику, который будет обязан зарегистрироваться как потребитель API. Это требование необходимо учесть, чтобы избежать DDOS атаки через публичные методы API.

Глава 3. Разработка средств интеграции API

Цель интеграции приложений заключается в том, чтобы обеспечить эффективное и надежное взаимодействие между интегрируемыми приложениями.

Существует 4 основные стиля интеграции приложений:

- Передача файла
- Общая база данных
- Удаленный вызов процедуры
- Обмен сообщениями

Все они имеют как свои преимущества, так и недостатки. Не существует какого-то определенного подхода, который будет лучше, чем другие. Для каждой ситуации оптимален конкретный способ. Подробнее об этом можно прочитать в книге “Шаблоны интеграции корпоративных приложений” [2].

Вся работа над проектом была поделена на отдельные итерации. Первой итерацией было ознакомление с существующей реализацией личного кабинета, а именно настройка и запуск текущей реализации на локальной машине. Второй итерацией были выявлены слабые стороны существующего решения, связанные с буферной базы данных, ознакомление со способами тестирования баз данных и тестирование основной базы данных. Третья итерация заключалась в ознакомлении с реализацией функциональности, которая отвечает за дисциплины по выбору, и разработке API и веб–приложения. Четвертой итерацией было анализ интеграции текущего решения, выявление текущих недостатков, разработка и тестирование прототипа, показывающего вариант интеграции API в информационную систему университета. Наиболее подробно в этой главе описывается последняя итерации.

3.1 Текущая реализация интеграции

3.1.1 Интеграция в текущем решении

Университет имеет развитую информационную систему, состоящую из множества модулей, которым требуется взаимодействовать друг с другом.

Прежде чем стать студентом университета человек подает документы. Этот факт фиксируется в системе, и этому человеку присваивается уникальный идентификатор, с помощью которого можно узнать всю информацию про абитуриента. Если же поступающий становится студентом, его данные помещаются в основную базу данных студентов и преподавателей, он получает студенческий билет, в котором встроен специальной чип, в котором содержится идентификатор студента.

Доступ в здания университета предоставляется следующим образом: студент прикладывает свой студенческий билет к считывателю, а считыватель в свою очередь обращается к системе, в которой для каждого идентификатора определены свои права доступа в те или иные помещения. Данные идентификаторов, находящиеся в системе ответственной за пропуска, берутся из базы данных студентов, в которую заносят информацию о студенте при его поступлении. Функциональность информационной системы университета на этом не ограничивается: взаимодействие с библиотекой, выплата стипендий также включены в информационную систему университета.

Благодаря тому, что у студента есть свой уникальный идентификатор он так же может получить доступ к услугам, которые предоставляет не только университет. Например, для того чтобы зайти на репозиторий различных научных статей Scopus необходим логин и пароль, однако студентам и сотрудникам СПбГУ достаточно ввести свои данные [18]. Также студенты и сотрудники СПбГУ имеют подписку DreamSpark от Microsoft [19].

Для студентов разработан личный кабинет, благодаря которому все необходимое можно получить здесь и сейчас, так как все данные хранятся в одной системе, и они всегда актуальны. Функциональность личного кабинета

на этом не заканчивается, к примеру, имеется возможность сверить свои оценки, ознакомиться с различными нормативными документами, сформировать квитанцию на оплату обучения и так далее.

Взаимодействие компонентов информационной системы (рис. 2) достигается путем использования такого способа интеграции приложений как “Общая база данных”.

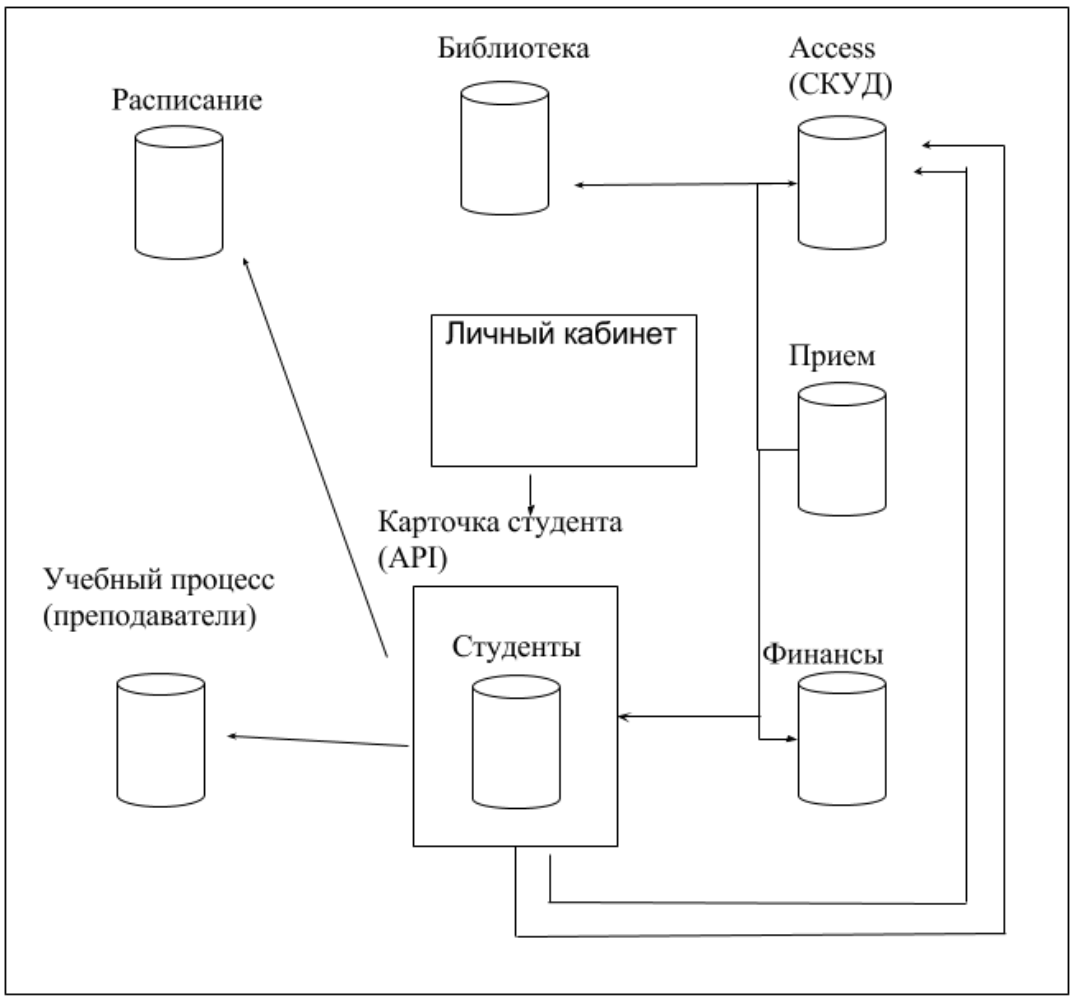


Рис. 2. Схема Информационной Системы СПбГУ

3.1.2 Тестирование

В текущей реализации личного кабинета используется такой стиль интеграции, как общая база данных, который имеет свои преимущества и недостатки [2]. К преимуществам такого стиля относится согласованность хранящейся в ней информации, а к недостаткам сложная разработка схемы

базы данных и большое число обращений [2]. Для того, чтобы снизить нагрузку к общей базе данных, в текущем решении используется буферная база данных.

Это решение может показаться спорным, потому что буферная база данных это реляционная база под управлением промышленной СУБД MS SQL Server, которая находится на том же хосте что и основная база. Поэтому в разрабатываемом прототипе будет использован такой инструмент как очередь. Производительность очередей значительно выше чем производительность баз данных под управлением промышленных СУБД [20] [21]. Однако для того чтобы удостовериться, что разрабатываемое решение подтверждает заявленные со стороны исследователей утверждения, и в последующем сделать тестирование аналогичного компонента, в разрабатываемом решении произведен замер производительности базы данных с применением нагрузочного тестирования.

Относительно базы данных можно проводить нагрузочное тестирование, стресс-тестирование и тестирование производительности [4]. Нагрузочное тестирование направлено на установление, либо получение характеристик производительности и времени отзыва устройства в ответ на внешний запрос. Стресс-тестирование, в свою очередь, направлено на оценку надежности и устойчивости системы в условиях нагрузки, которые превышают норму. Тестирование производительности определяет, как быстро система работает под нагрузкой. Для достижения поставленной цели выбрано нагрузочное тестирование, благодаря которому можно получить необходимые характеристики.

Для проведения тестирования, в первую очередь необходимо получить скрипты, которые будут выполнятся в результате различных бизнес-сценариев. Для этого использовалось приложение «Profiler», поставляемый вместе с MS SQL Server [22]. Для тестирования были проведены замеры скорости выполнения запросов, и выбрано три наиболее часто выполняемых и тяжелых запроса на:

- Открытие главное страницы;
- Открытие формы мат. помощи;
- Отправка формы мат. помощи.

Для замеров производительности используется утилита SqlQueryStress [23]. Этот инструмент позволяет выполнить заданный пользователем запрос к базе данных заранее заданное количество раз.

Произведены измерения, представленные в таблице 2.

Количество (итерации: пользователи)	Главная страница (сек)	Открытие формы матпомощи (сек)	Отправка формы матпомощи (сек)
1:1	0.3262	0.3264	0,0010
1:5	0.4966	0.4916	0.0015
1:100	0.3910	0.4189	0.0018
1:200	0.4166	0.3967	0.0021

Таблица 2. Измерения полученные в ходе тестирования

Использование данных результатов и теории массового обслуживания (теорию очередей) позволяет просчитать показатели эффективности для базы данных, указанные далее.

Теория массового обслуживания имеет дело с процессами, для которых характерна следующая структура. В систему массового обслуживания (СМО) в случайные моменты времени поступают заявки (или требования). Заявки на обслуживание образуют входной поток (рис. 3). Если есть свободные каналы, то требование выполняется обслуживания. Если все каналы обслуживания заняты, то требование становится в очередь по определенным правилам или без обслуживания покидает систему. Выполненные требования образуют выходной поток. [24]

Системы массового обслуживания делятся на два вида.

- Одноканальная СМО:

        - СМО с отказами;
        - СМО с очередью:
            - СМО с ограниченной очередью;
            - СМО с неограниченной очередью;
- Многоканальная СМО:
        - СМО с отказами;
        - СМО с очередью:
            - СМО с ограниченной очередью;
            - СМО с неограниченной очередью.

Очереди могут ограничиваться по длине (по числу входящих заявок), заявки могут выполняться как в порядке поступления, так и в любом другом, заранее заданным пользователем. Для случая с MS SQL Server следует использовать одноканальную СМО с ограниченной очередью (рис. 3). Такая система имеет один обслуживающий канал. На вход, потоком, поступают заявки с интенсивностью λ. Если обслуживающий канал занят в тот момент, когда поступает заявка, она встает в очередь и ожидает начала обслуживания. Число мест заранее ограничено и известно. Если обслуживающий канал занят и в очереди нет свободных мест, то входящая заявка уходит из системы не обслуженной. Время обслуживания заявки — это случайная заявка, которая подчиняется экспоненциальному закону распределения с параметром μ. Среднее время обслуживания одной заявки вычисляется по формуле:

$$t = \frac{1}{\mu}$$

Существует несколько возможных состояний системы

- S_0 канал свободен
- S_1 канал занят, очереди нет
- S_{1+1} канал занят, в очереди одна заявка
- S_{1+2} канал занят, в очереди две заявки
- S_{1+m} канал занят, в очереди м заявок

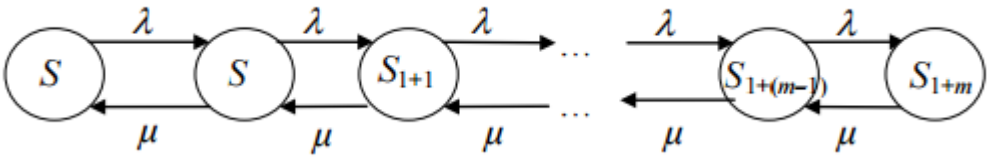


Рисунок 3. Размеченный граф состояний системы

Выделяют следующие показатели эффективности работы СМО:

1) Вероятность того, что обслуживающий канал свободен:

$$p_0 = \frac{1-p}{1-p^{m+2}}$$

2) Вероятность отказа $p_{\text{отк}}$ (вероятность того, что заявка покинет СМО не обслуженной) $p_{\text{отк}} = p_n$;

3) Вероятность того, что канал занят, в очереди k заявок:

$$p_{1+k} = p^{1+k} p_0$$

4) Вероятность отказа (в очереди нет свободных мест):

$$p_{\text{отк}} = p^{1+m} p_0$$

5) Относительная пропускная способность Q (отношение среднего числа обслуживаемых в единицу времени заявок к среднему числу поступивших за это время заявок):

$$Q = 1 - p_{\text{отк}}$$

6) Абсолютная пропускная способность A (среднее число заявок, которое СМО может обслужить в единицу времени);

$$A = \lambda Q$$

7) Среднее число заявок в очереди:

$$L_{\text{оч}} = p^2 \frac{1 - p^m(m+1-mp)}{(1-p^{m+2})(1-p)}$$

8) Среднее время нахождения заявки в очереди:

$$T_{\text{оч}} = \frac{1}{\lambda} L_{\text{оч}}$$

9) Среднее число занятых обслуживанием каналов:

$$L_{\text{обсл}} = 1 - p_0$$

10) Среднее число заявок в системе:

$$L_{\text{сис}} = L_{\text{оч}} + L_{\text{обсл}}$$

11) Среднее время нахождения заявки в системе:

$$\mathrm{T}_{\text{сис}} = \frac{1}{\lambda} L_{\text{сис}}$$

После проведения подсчетов были получены следующие значения:

- Максимальное количество элементов в очереди: **5000** [25]
- Интенсивность: **1,38** (запроса в сек.)
- Среднее время выполнения запроса: **0,4166** (сек.)
- Вероятность того, что канал свободен: **0,425**
- Вероятность отказа (канал занят, в очереди нет свободных мест): $\mathbf{1,137 \times e^{-247}}$
- Относительная пропускная способность Q (отношение среднего числа обслуживаемых в единицу времени заявок к среднему числу поступивших за это время заявок): **0,999**
- Абсолютная пропускная способность A (среднее число заявок, которое СМО может обслужить в единицу времени): **1,37**
- Среднее число заявок в очереди: **0,77794**
- Среднее время нахождения заявки в очереди: **0,5637**
- Среднее число заявок, находящихся под обслуживанием (среднее число занятых каналов): **0,575**
- Среднее число заявок в системе: **1,35294**
- Среднее время нахождения заявки в системе: **0,9803** (сек.)

Для подтверждения полученных результатов проведены практические испытания системы. Тестировался Microsoft SQL Server с основной базой данных.

Полученные замеры производительности базы данных позволят использовать их для последующего сравнения с тестированием аналогичного компонента, в разрабатываемом решении.

3.2 Разработка API для сервиса дисциплин по выбору

Использование для интеграции приложений вместо буферной базы данных подхода обмена сообщениями обеспечивает доступ как к общим данным, так и к общей функциональности [2]. Дополнительно, для разработки распределенного приложения предлагается использовать архитектуру микросервисов [3].

Для отработки технологии интеграции разработанного ранее API в информационную систему университета, а также для проверки возможности удовлетворения требования к высокой производительности и отказоустойчивости разработан прототип, обеспечивающий функциональность выбора студентом вариативной дисциплины.

3.2.1 Текущая реализация выбора вариативной дисциплины

В личном кабинете реализован функционал, позволяющий пользователям выбрать доступный им курс по выбору. С технической точки зрения, он имеет следующую реализацию.

Имеются следующие компоненты–участники (рис. 4): студент, личный кабинет, карточка студента, сервис дисциплин по выбору, источник дисциплин по выбору. Пользователь личного кабинета запрашивает список доступных ему дисциплин по выбору. Личный кабинет обращается к сервису и передает идентификатор студента, запрашивая список дисциплин, для студента с определенным идентификатором. Сервис, в свою очередь, не имеет никаких данных о студенте, и чтобы понять, дисциплины какого направления, курса, профиля и т.п. нужно запрашивать, он просит у карточки студента данные по идентификатору студента. Карточка студента знает данные всех студентов, и возвращает информацию о студенте, которая соответствуют идентификатору. Используя эти данные, сервис дисциплин по выбору обращается к источнику, и запрашивает нужные дисциплины по выбору для определенного студента. Полученный список возвращается в личный кабинет,

который в свою очередь формирует на основе этого списка форму, используя которую, студент должен заполнить ее и отправить в личный кабинет, выбранные дисциплины записываются в карточку студента, и если запись прошла успешно, то возвращается уведомление о успешной записи.

Рисунок 4. Текущая реализация списка дисциплин по выбору

3.2.2 Существующие проблемы и варианты их решения

В существующем решении присутствует ряд недостатков, избавившись от которых можно существенно улучшить приложение.

Во–первых, это синхронность. Все операции происходят синхронно, и пользователю необходимо ждать пока все компоненты “договорятся” с друг другом. Проблему синхронности можно решить добавлением асинхронной связи между компонентами.

Во–вторых, существует тонкое горлышко в виде запроса данных по студенту. Каждый раз приходится обращаться к подсистеме “Обучающиеся” ИС “Обучение” СПбГУ (карточка студента), чтобы получить данные об обучающемся. На данный момент, эту проблему решили таким способом, что при входе пользователя в систему, вне необходимые данные сохраняются в сессии. При запросе эти данные передаются, а сервис уже использует их для получения дисциплин по выбору.

В–третьих, это запрос дисциплин. Необходимо для каждого пользователя обращаться к базе данных, чтобы получить список дисциплин

доступных для него. В текущем решении не учитывается тот факт, что пользователи объединены в учебные потоки, и список доступных для них дисциплин будет одинаковый. Если рассматривать негативный сценарий, то в самый последний момент все пользователи, состоящие в одной учебной группе, будут запрашивать одни и те же дисциплины, тем самым будет произведена нагрузка на базу данных, которую можно было бы избежать. Отметим, что при запросе дисциплин идентифицирующих данных нет, поэтому в данном случае можно использовать кэширование. Например, при одной и той же выборке из базы данных имеет смысл хранить данные в кэше [26].

В–четвертых, еще одно узкое горлышко — это запись данных. Подсистема “Обучающиеся” ИС “Обучение” СПбГУ (карточка студента) делит свои ресурсы между чтением и записью, что не является хорошей практикой, потому что любое чтение в момент записи блокируется и наоборот, любая запись может привести к построению индексов, что в свою очередь может снижать скорость чтения. Для того чтобы избавиться от проблемы с записью, предлагается использовать очередь [2]. Для случая с очередью в идеальном варианте необходимо использовать отдельную систему или отдельный жесткий диск. Очередь может быть реализована в виде отдельной базе данных, либо же другими средствами. Очередь должна выступать не только в качестве объекта, в который мы пишем и из которого мы читаем, но и обладать неким интеллектом, и записывать в базу данных, в то время, когда она ничем не загружена.

3.2.3 Архитектура прототипа

Прототип представляет собой распределенную систему и веб приложение. Взаимодействие с системой происходит с помощью внешнего API, а взаимодействие компонентов происходит благодаря внутреннему API и сервису обмена сообщениями между элементами. Система, с помощью специальных сервисов, должна производить чтение и запись из базы данных.

Также, в прототипе отсутствуют недостатки текущей реализации, путем использования решений, предложенных в разделе “Существующие проблемы и варианты их решения”. Основным требованием к прототипу является разгрузка базы данных, которая достигается благодаря использованию промежуточного звена, основанного на использовании метода обмена сообщениями.

Архитектура прототипа была основана на микросервисах. Это означает, что он состоит из множества мелких, независимых друг от друга приложений, способных работать в своем собственном пространстве памяти [27]. Такая архитектура имеет как положительные, так и отрицательные стороны.

Положительные стороны:

- Грамотная и понятная организация архитектуры;
- Лучшая масштабируемость;
- Повышает изоляцию сбоев [28];
- Исключает долгосрочную приверженность единого стека технологий;
- Доступность.

Отрицательные стороны:

- Трудоемкая разработка распределенных систем;
- Сложности при тестировании.

Используя такую архитектуру, взаимодействие между модулями будет происходить с помощью «Внутреннего API». На данный момент, личный кабинет не имеет такого API.

На рисунке 5 показана архитектура приложения, и указаны взаимодействия между компонентами.

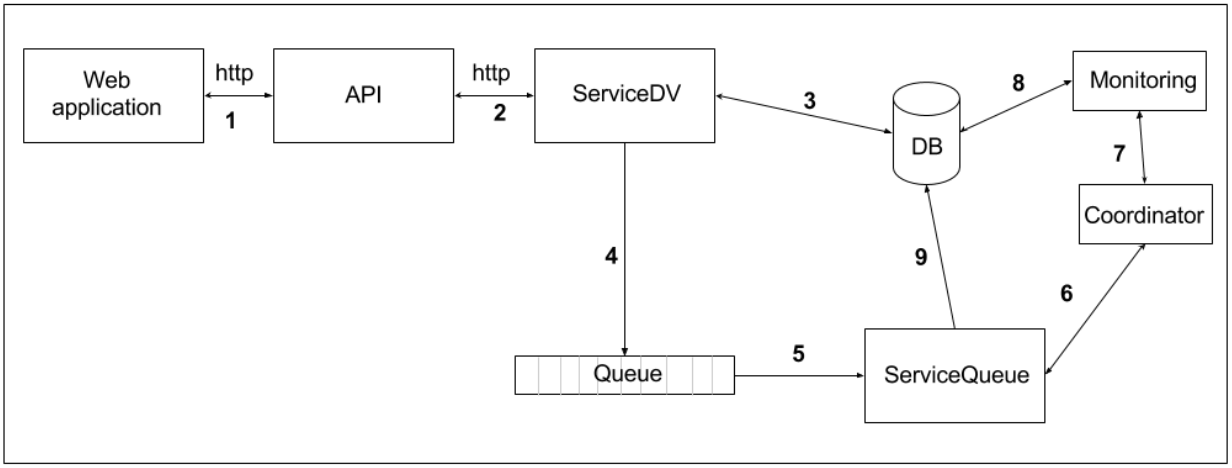


Рис. 5. Архитектура прототипа

Стоит подробнее рассмотреть технические аспекты реализации.

Существуют следующие компоненты:

- Веб приложение, написанное на AngularJS, популярный продукт, который обладает модульностью и простотой тестирования [29];
- RESTful API, реализованное с помощью WebAPI. Оно выступает посредником между веб приложением и сервисом;
- Сервис Дисциплин по выбору (сервис ДВ);
- Очередь RabbitMQ [30];
- Сервис чтений из очереди и записи в базу данных;
- База данных;
- Средство мониторинга активности базы данных;
- Координатор сервиса очереди, использующий данные мониторинга.

Связь между внешним приложением и API происходит через http запросы. API, в свою очередь, обращается к сервису дисциплин по выбору, который берет информацию из базы или помещает данные в очередь. Сервис очереди в зависимости от загрузки базы данных записывает данные или нет.

Стоит рассмотреть, как компоненты взаимодействуют друг с другом.

Взаимодействие 1:

Веб–приложение позволяет асинхронно запрашивать и отправлять данные путем отправки GET\POST запросов. API позволяет получить доступ

с помощью http запроса, который будет перенаправлен сервису дисциплин по выбору.

Взаимодействие 2:

Сервис Дисциплин по выбору, благодаря внутреннему API, может получать http запросы и в зависимости от запроса совершать определенные действия

Взаимодействие 3:

В случае запроса на получение данных сервис в первую очередь проверит наличие необходимых данных в кэше, и если их там нет, то сделает запрос к базе, поместив при этом ответ в кэш.

Взаимодействие 4:

В случае POST запроса полученные данные записываются в очередь, которая реализована с помощью платформы RabbitMQ. Сервис дисциплин по выбору выступает в роли поставщика событий. Положительная сторона такого подхода заключается в том, что на поставщика может быть подписано несколько подписчиков, тем самым передаваемая информация может быть использована более чем одним сервисом.

Взаимодействие 5:

Это взаимодействие нужно для того, чтобы сервис мог получать сообщения из очереди. Сервис выступает в роли подписчика события.

Взаимодействие 6, 7, 8:

Данные взаимодействия следует рассматривать вместе, потому что именно их совокупность представляет собой ценность для решения. Мониторинг нужен для того, чтобы отслеживать состояние базы данных, и передавать их координатору, который будет принимать решение об загрузке данных в базу в момент ее наименьшей занятости.

Взаимодействие 9:

Благодаря этому взаимодействию происходит загрузка данных в базу.

3.2.4 Тестирование прототипа

Для того, чтобы проверить эффективность использования кэша и выполнение требования увеличения производительности, проведено нагрузочное тестирование прототипа средствами Visual Studio Enterprise [31]. Для теста были использованы следующие настройки:

- Количество пользователей: 25
- Прогрев: 1 минута
- Работа: 5 минут

В результате теста были получены следующие показатели:

- При использовании кэша было выполнено 9650 запросов за 5 минут (рис. 6).
- Без использования кэша было выполнено 7493 запроса за 5 минут (рис. 7).

Итогом является сравнительная таблица (табл. 3).

	Количество запросов за 5 минут
С использованием кэша	9650
Без использования кэша	7493

Таблица 3. Таблица сравнения производительности

Полученные результаты говорят о том, что конечное решение в показанной мере удовлетворяет требованию к производительности.

3.3 Итог разработки прототипа

При разработке прототипа проведены:

1. Ознакомление и изучение способов интеграции приложений в систему;
2. Проектирование архитектуры прототипа, позволяющую отработать методы интеграции;
3. Тестирование компонентов прототипа.

Благодаря асинхронности при коммуникации компонентов, удалось уменьшить время ожидания ответа и обеспечить их слабое связывание. В ходе работы учтены и устранены недостатки существующего решения.

Глава 4. Заключение

В рамках выпускной квалификационной работы проведен обзор и анализ существующих решений, на основе которого выявлена наиболее востребованная функциональность для реализации.

Разработан прототип API, с помощью которого сторонние разработчики смогут взаимодействовать с информационной системой университета. Для отработки технологии интеграции приложений разработан и протестирован прототип, в который было интегрировано API. Стоит отметить, что API находится на стадии оснащения полноценной системой безопасности.

Литература

1. Определение веб–приложения. https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5
2. Грегор Хоп, Бобби Вульф, Шаблоны интеграции корпоративных приложений, 2016
3. Грученков В. В. Новиков В. И., Микросервисная архитектура веб–приложений, 2016
4. Пивень А.А, Скорин Ю.И., ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ, 2012
5. Определение REST API. https://ru.wikipedia.org/wiki/REST
6. Определение RPC. https://ru.wikipedia.org/wiki/%D0%A3%D0%B4%D0%B0%D0%BB%D1%91%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B2%D1%8B%D0%B7%D0%BE%D0%B2_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D0%B4%D1%83%D1%80
7. Сравнение REST и SOAP. https://habrahabr.ru/post/131343/
8. Сравнение REST и SOAP. https://blog.zwezdin.com/2008/soap-vs-rest/
9. Официальная страница WebAPI. https://msdn.microsoft.com/en-us/library/hh833994(v=vs.108).aspx
10. Офциальная страница Angular.js. https://angularjs.org/
11. Официальная страница swagger.io. http://swagger.io/
12. Документация созданного API. https://app.swaggerhub.com/apis/AlexGnatuyk/SpbuAPI/1.0.0
13. Документация для создания Telegram–ботов https://core.telegram.org/bots
14. Обзор способов и протоколов аутентификации в веб-приложениях https://habrahabr.ru/company/dataart/blog/262817/

15.Официальная страница стандарта JWT. https://jwt.io/
16.Статья про проблемы производительности SQL сервера. https://habrahabr.ru/post/216309/
17.Официальный сайт IdentityServer4. http://identityserver4.readthedocs.io/en/release/
18.Официальный сайт Scopus. https://www.scopus.com/
19. Официальный сайт портала DreamSpark https://dspark.spbu.ru/
20.Статья про сервер очередей. https://habrahabr.ru/company/mailru/blog/216363/
21.База данных как анти–паттерн очереди. http://mikehadlow.blogspot.ru/2012/04/database-as-queue-anti-pattern.html
22.Приложение SQL Server Profiler https://msdn.microsoft.com/ru-ru/library/ms181091.aspx
23. Репозиторий SqlQueryStress. https://github.com/ErikEJ/SqlQueryStress
24. Теория про теорию очередей. http://kopilka77.ru/docs/gsv/disciplini/IMOD/lec_SMO.pdf
25. Основные счетчики IIS. http://devopshub.net/osnovnyie-schetchiki-iis/
26.Кэширование оперативной памяти. http://www.studfiles.ru/preview/5881240/page:5/
27. Сильные и слабые стороны микросервисов. https://trello.com/c/YUDwlR03/79-what-are-the-pros-and-cons-of-monolithic-vs-microservice-architectures
28. Статья про микросервисную архитектуру. http://cloudacademy.com/blog/microservices-architecture-challenge-advantage-drawback/
29.Angular.js настоящее модульное тестирование. http://savepearlharbor.com/?p=233705
30.Официальный сайт RabbitMQ. http://www.rabbitmq.com/
31.Запуск нагрузочных тестов. https://msdn.microsoft.com/ru-ru/library/ms184776(v=vs.110).aspx

Приложение 1. Обзор существующих решений личного кабинета

Санкт Петербургский Государственной Университет

Личный кабинет СПбГУ (рис. 1) имеет следующие разделы:

- Мои данные
    - Личные данные (ФИО, дата рождения, ИНН, студенческая почта, Адрес прописки, контактный телефон)
    - Сведения об обучении (Направление обучения, форма, курс, основа)
    - Есть возможность сформировать справку о статусе обучения
- Мои документы
    - Данные паспорта
- Мои оценки
    - Раскрывающийся список по семестрам
- Мои заявки
    - Список ранее поданных заявок (Дата подачи, номер заявки, статус заявки)
- Подать заявку
    - Возможность подать заявку на матпомощь и на военную кафедру
- Квитанция на оплату обучения
    - Доступна студентам, обучающимся на платной основе
- Нормативные документы
    - Различные документы
- Оставить отзыв
    - Возможность оставить отзыв о работе сайта

К достоинствам данного личного кабинета можно отнести возможность подать заявку на материальную помощь и на военную кафедру, посмотреть статус своей заявки.

К недостаткам относится отсутствие расписания, возможности посмотреть информацию об определенном преподавателе и отсутствие полной автоматизации подачи заявок.

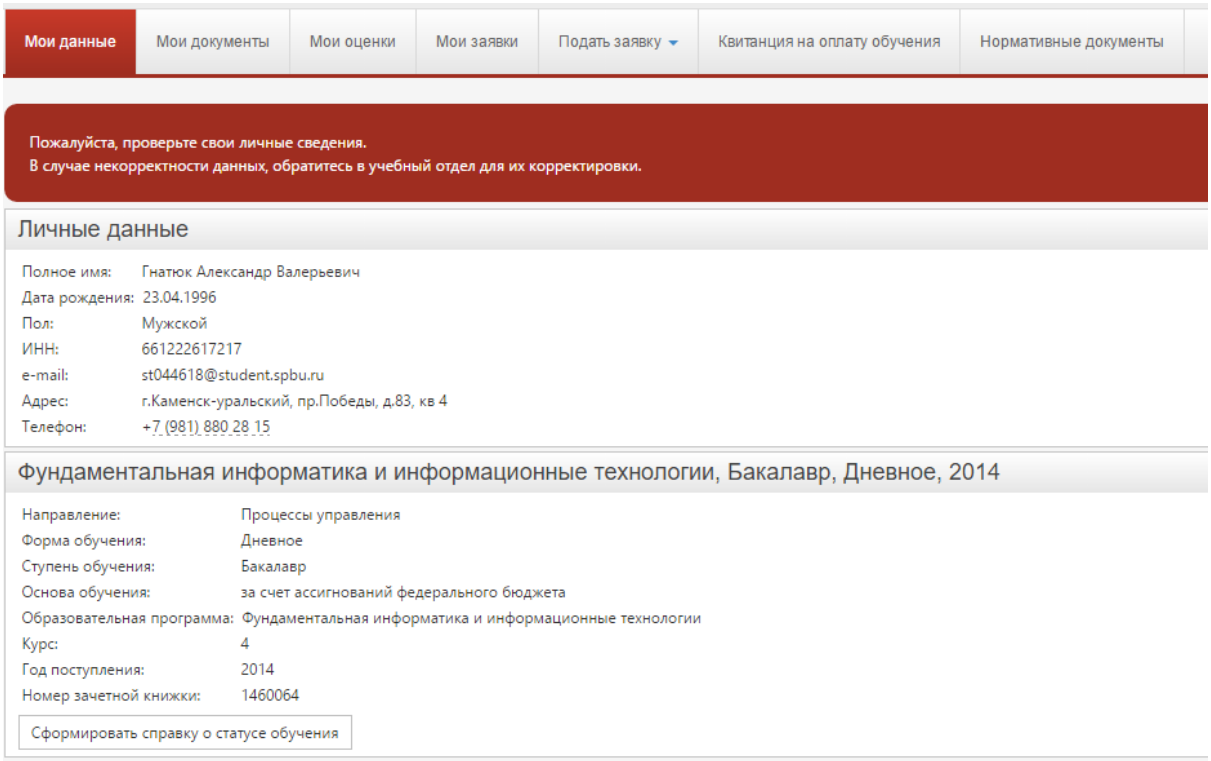


Рис. 1. Личный кабинет студента СПбГУ

Факультет Прикладной Математики — Процессов Управления

Личный кабинет (рис. 2) имеет следующие разделы:

- Моя Страница
  - Информация о студенте (ФИО, курс, группа)
  - Новости факультета
- Редактировать
  - Редактировать личную информацию
- Преподаватели
  - Рейтинг преподавателей по различным категориям
- Отзыв о прослушанных дисциплинах
  - Возможность оставить отзыв о прослушанных дисциплинах
- Выход

К достоинствам данного личного кабинета можно отнести то, что в нем можно посмотреть рейтинг преподавателей факультета, основанный на анонимном голосовании студентов, а также возможность ознакомиться с новостями.

К недостаткам относится достаточно малый функционал, отсутствие современного дизайна.

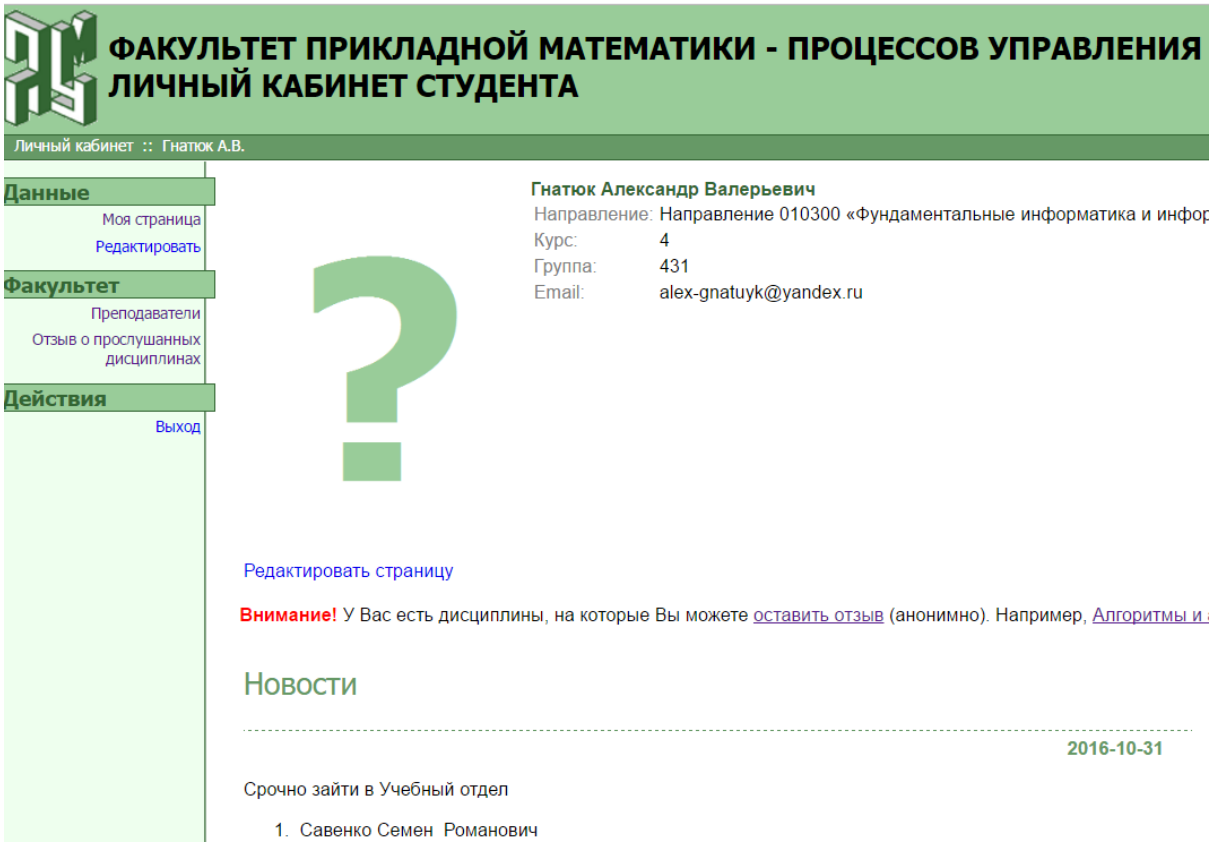


Рис. 2. Личный кабинет студента факультета ПМ–ПУ

Санкт–Петербургский национальный исследовательский Университет информационных технологий, механики и оптики

Личный кабинет ИТМО (рис. 3) имеет следующие разделы:

- Дом
  - Новостная
- Портфолио
  - Профиль (Информация о студенте)
  - Научная деятельность

- Сообщения
    - Входящие
    - Отправленные
    - Контакты
- Учебно методическая документация
- Календари
    - Календарь задач
- Площадки
- Расписание
    - Учебный план
    - Оценки (Оценки по семестрам)
    - Стипендия (Список с датой и суммой выплаты)
    - История обучения
    - Договоры

К достоинствам относится достаточно большой функционал, такой как: возможность просматривать новости, расписание, дату выплаты стипендии, а также отправлять сообщения.

К недостаткам можно отнести неудобный и не интуитивно понятный интерфейс, а также отсутствие возможности подавать заявки.

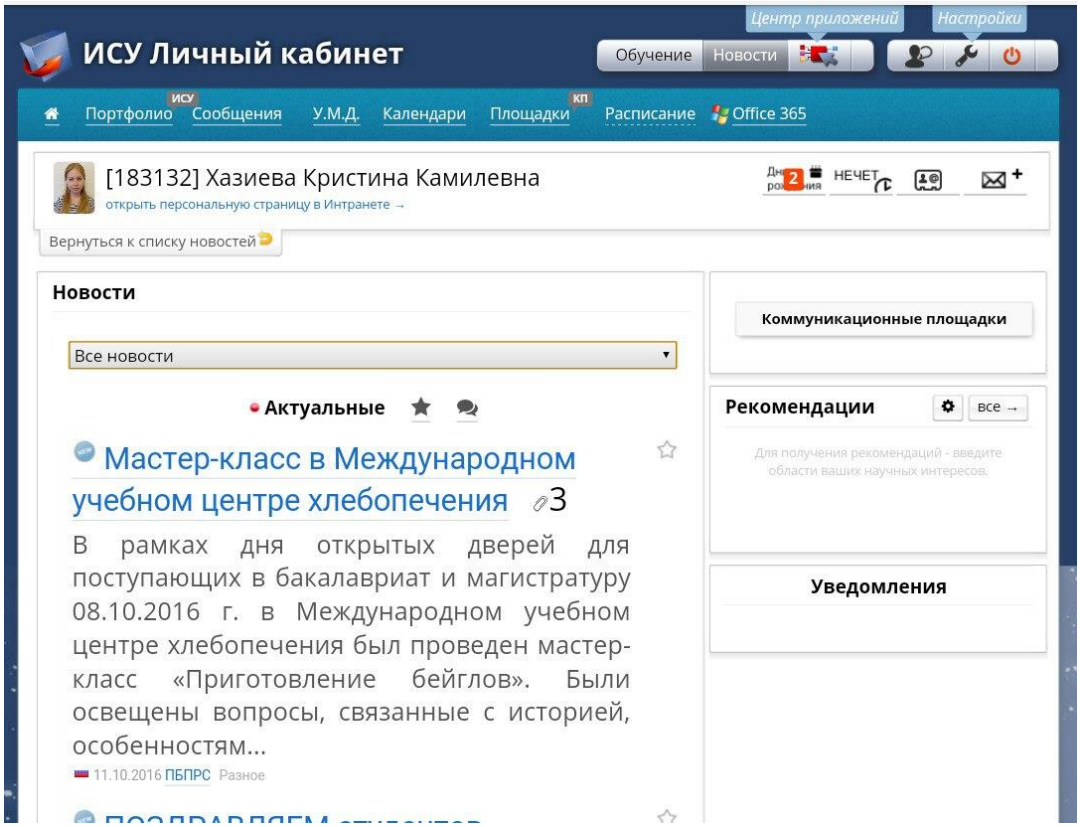


Рис. 3. Личный кабинет студента ИТМО

Уральский Федеральный Университет

Личный кабинет УрФУ (рис. 4) имеет следующие подразделы:

- Учеба
  - Выбор майнора
  - Индивидуальная траектория студента
  - Сервис информирования студента о баллах БРС
  - Расписание занятий
  - ГИПЕРМЕТОД
  - Электронные образовательные ресурсы
  - Выборка первых 10 % студентов института (филиала)
  - Учебный план
  - Запись на пересдачу НТК
  - Учебные фильмы
- Наука

    - Портал молодежной науки
    - Электронные ресурсы по подписке
    - Заказ литературы из электронного каталога
    - Электронный каталог
    - Электронные библиотечные системы
    - Электронные ресурсы свободного доступа
    - Электронный архив
- Внеучебная жизнь
    - Лучший преподаватель глазами студента
    - Сайт центра взаимодействия с работодателями
    - Центр трудоустройства студентов УрФУ
    - Рейтинг внеучебной деятельности студентов
- Кампус
    - Лицензионное программное обеспечение Microsoft
    - ИТ поддержка сервисов
    - Получение учетной записи для входа в личный кабинет
- Финансы
    - Целевой капитал на развитие УрФУ
- Документы
    - Формы документов для студентов (На самом деле просто нормативные документы)

К достоинствам данного личного кабинета можно отнести красивый внешний вид.

К недостаткам относится то, что большая часть функций недоступна, и вместо необходимых страниц используются “заглушки”. Также сюда относится отсутствие возможности сформировать различные заявки.

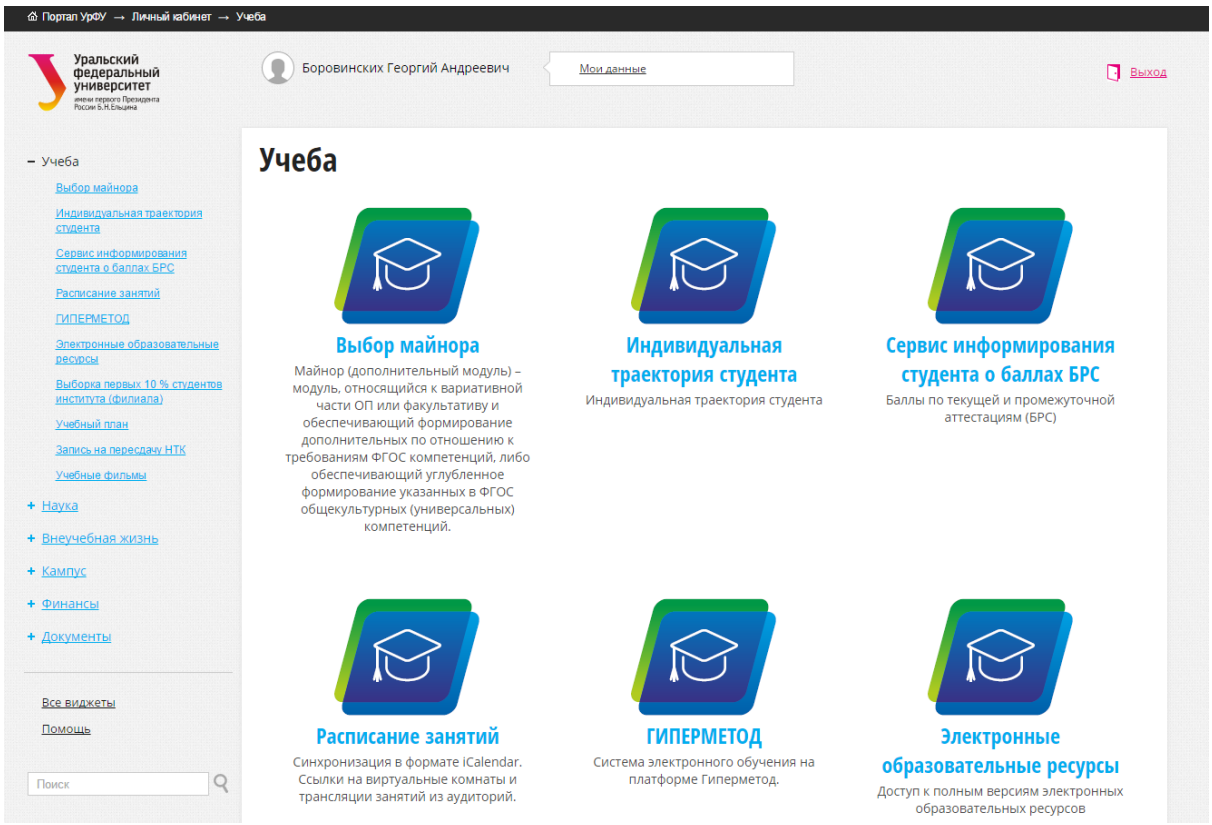


Рис. 5. Личный кабинет студента УрФУ