Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

К ЗАЩИТЕ ДОПУСТИТЬ
Зав. каф. ЭВМ
_____ Б.Н. Никульшин

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к дипломному проекту
на тему
ПРОГРАММНОЕ СРЕДСТВО ДЛЯ ПРОСМОТРА И ОБРАБОТКИ ИНФОРМАЦИИ О РАСПИСАНИИ ЗАНЯТИЙ

БГУИР ДП 1–40 02 01 01 019 ПЗ

Студент А.В. Гуринович

Руководитель Д.В. Басак

Консультанты:

от кафедры ЭВМ Д.В. Басак

по экономической части В.Г. Горовой

Нормоконтролер Е.Е. Клинцевич

Рецензент

МИНСК 2023

РЕФЕРАТ

Дипломный проект предоставлен следующим образом. Электронные носители: 1 компакт-диск. Чертёжный материал: 6 листов формата А1. Пояснительная записка: 128 страниц, 10 рисунков, 33 таблицы, 14 литературных источников, 3 приложения.

Ключевые слова: Swift, SwiftUI, MVVM, CoreData, Xcode, App Store, адаптивный графический пользовательский интерфейс, iOS, iPadOS, Apple.

Предметной областью данного проекта является расписание занятий Белорусского государственного университета информатики и радиоэлектроники. Объектом разработки является нативное приложение для просмотра и обработки информации о расписании занятий Белорусского государственного университета информатики и радиоэлектроники.

Целью разработки данного дипломного проекта является создание приложения с высокой степенью удобства и быстродействия, которое будет использовать всю информацию, доступную через программный интерфейс интегрированной информационной системы Белорусского государственного университета информатики и радиоэлектроники, для отображения расписания, и иных данных. Среди этих данных: подразделения аудиторий и преподавателей, периоды проведения занятий, периоды проведения экзаменационной сессии, полные названия учебных дисциплин, статистическая информация и другое.

Для разработки данного проект использовалась интегрированная среда разработки Xcode, язык программирования Swift, база данных CoreData и фреймворк SwiftUI для создания пользовательского интерфейса.

В результате разработки создано программное средство для просмотра и обработки информации о расписании занятий для операционных систем iOS и iPadOS. Программное средство подготовлено для загрузки в магазин приложений Apple App Store.

Практическим применением разработки является удобное и быстрое получение структурированной информации для студентов и преподавателей университета. Пользователи смогут легко находить информацию о своих занятиях, преподавателях, группах, кабинетах, а также добавлять задания со сроком выполнения и управлять ими. Проект улучшает опыт обучения в университете для пользователей на платформах Apple, снижает время, затрачиваемое на получение и обработку информации.

Проект является эффективным с экономической точки зрения, при этом эффективность выражается не только в прямой прибыли от продаж, но и в повышении эффективности пользователей в процессе обучения.

Дипломный проект полностью завершён и подготовлен для выхода на рынок, в частности, в Apple App Store. Дальнейшему развитию приложения может поспособствовать открытый исходный код, который доступен на web-сервисе GitHub, где другие пользователи могут сообщать об ошибках, предлагать и реализовывать улучшения программного средства.

Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет: КСиС. Кафедра: ЭВМ.
Специальность: 40 02 01 «Вычислительные машины, системы и сети».
Специализация: 40 02 01-01 «Проектирование и применение локальных компьютерных сетей».

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____ Б.В.Никульшин
«____» _____ 2023 г.

ЗАДАНИЕ
по дипломному проекту студента
Гуриновича Андрея Викторовича

1 Тема проекта: «Программное средство для просмотра и обработки информации о расписании занятий» – утверждена приказом по университету от 24 марта 2023 г. № 743-с.

2 Срок сдачи студентом законченного проекта: 1 июня 2023 г.

3 Исходные данные к проекту:

3.1 Получение данных из интегрированной информационной системы Белорусского государственного университета информатики и радиоэлектроники.

3.2 Операционные системы: iOS и iPadOS.

3.3 Среда разработки: Xcode.

3.4 Язык программирования: Swift.

4 Содержание пояснительной записки (перечень подлежащих разработке вопросов):

Введение 1. Обзор литературы. 2. Системное проектирование. 3. Функциональное проектирование. 4. Разработка программных модулей. 5. Программа и методика испытаний. 6. Руководство пользователя. 7. Технико-экономическое обоснование разработки и реализации на рынке программного средства для просмотра и обработки информации о расписании занятий. Заключение. Список использованных источников. Приложения.

5 Перечень графического материала (с точным указанием обязательных чертежей):

5.1 Вводный плакат. Плакат.

5.2 Программное средство для просмотра и обработки информации о расписании занятий. Схема структурная.

5.3 Программное средство для просмотра и обработки информации о расписании занятий. Диаграмма классов.

5.4 Программное средство для просмотра и обработки информации о расписании занятий. Модель данных.

5.5 Представление списка групп. Диаграмма последовательностей

5.6 Представление расписания занятий. Диаграмма последовательностей

5.7 Заключительный плакат. Плакат.

6 Содержание задания по экономической части: «Технико-экономическое обоснование разработки и реализации на рынке средства для просмотра и обработки информации о расписании занятий».

ЗАДАНИЕ ВЫДАЛ _____ В.Г. Горовой

КАЛЕНДАРНЫЙ ПЛАН

Наименование этапов дипломного проекта	Объем этапа, %	Срок выполнения этапа	Примечания
Подбор и изучение литературы	10	23.03 – 29.03	
Структурное проектирование	10	29.04 – 05.04	
Функциональное проектирование	20	05.04 – 18.04	
Разработка программных модулей	30	18.04 – 08.05	
Программа и методика испытаний	10	08.05 – 11.05	
Расчёт экономической эффективности	10	11.05 – 15.05	
Оформление пояснительной записки	10	15.05 – 21.05	

Дата выдачи задания: 23 марта 2023 г.

Руководитель _____ Д.В. Басак

ЗАДАНИЕ ПРИНЯЛ К ИСПОЛНЕНИЮ _____ А.В. Гуринович

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Данный дипломный проект посвящён разработке программного средства для просмотра и обработки информации о расписании занятий Белорусского государственного университета информатики и радиоэлектроники (БГУИР).

Целью разработки данного дипломного проекта является создание приложения с высокой степенью удобства и быстродействия, которое будет использовать всю доступную через программный интерфейс интегрированной информационной системы БГУИР информацию для отображения расписания, и иной информации.

Для разработки данного проект использовалась интегрированная среда разработки Xcode, язык программирования Swift и фреймворк SwiftUI для создания пользовательского интерфейса.

В результате разработки создано программное средство для просмотра и обработки информации о расписании занятий для операционных систем iOS и iPadOS. Программное средство подготовлено для загрузки в магазин приложений Apple App Store.

Практическим применением разработки является удобное и быстрое получение структурированной информации для студентов и преподавателей университета. Пользователи смогут легко находить информацию о своих занятиях, преподавателях, группах, кабинетах, а также добавлять задания со сроком выполнения и управлять ими. Проект улучшает опыт обучения в университете для пользователей на платформах Apple, снижает время, затрачиваемое на получение и обработку информации.

Проект является эффективным с экономической стороны, при этом эффективно выражается не только в прямой прибыли от продаж, но и в повышении эффективности пользователей в процессе обучения.

Разработка программного средства для разных типов устройств является одной из основных проблем данного дипломного проекта, что требует создания большого количества шаблонов и использования стандартных библиотек SwiftUI.

В рамках разработки проекта отсутствует возможность изменения интерфейса программирования приложений интегрированной информационной системы БГУИР, что ставит дополнительной задачей создание особого функционала, который, как правило, должен находиться на серверной части, например, функции для обработки и композиции больших объёмов данных.

1 ОБЗОР ЛИТЕРАТУРЫ

1.1 Программный интерфейс интегрированной информационной системы БГУИР

Программный интерфейс (API) интегрированной информационной системы (ИИС) БГУИР предоставляет программные интерфейсы, предназначенные для получения информации о расписании групп и преподавателей, а также информацию о факультетах, кафедрах, специальностях и аудиториях [1].

API имеет возможность предоставления ответа на запрос как в виде JavaScript Object Notation (JSON), так и в виде так и в виде Extensible Markup Language (XML). Так как Swift имеет встроенную библиотеку для расшифровки JSON, в разработке проекта будет использованы ответы в нотации JSON. Для XML потребовалось бы использовать сторонний модуль расшифровки, либо разрабатывать его самостоятельно.

Далее в данном подразделе представлены структуры ответов API ИИС на различные запросы. В каждой структуре приведено названия ключей и типы значений. В случае, если значение является опциональным, это указывается в отдельном столбце таблицы. Если такой столбец отсутствует, то все значения являются обязательными. Полями, в данном подразделе, далее будут называться пары ключа и значения.

Типы данных также могут быть другими описываемыми в этом подразделе типами.

1.1.1 Факультет

Структура представлена в таблице:

Таблица 1.1 – Структура факультета из API ИИС

Ключ	Тип	Описание
id	Число	Идентификатор
name	Строка	Название
abbrev	Строка	Аббревиатура

1.1.2 Специальность

Некоторые специальности содержат идентификаторы факультета, которые не сопоставляются ни с одним из получаемых запросом факультетом объектов. Такое поведение вызвано тем, что в ответе на запрос специальностей содержатся также те специальности, обучение по которым более не производится, либо производится через Систему электронного обучения БГУИР. Также существуют специальности, которые ссылаются на более несуществующие факультеты, либо на их предыдущие идентификаторы.

Структура представлена в таблице:

Таблица 1.2 – Структура специальности из API ИИС

Название	Тип	Описание
id	Число	Идентификатор
name	Строка	Название
abbrev	Строка	Аббревиатура
code	Строка	Код
facultyId	Число	Идентификатор факультета

1.1.3 Кафедра

Ответ на данный запрос предоставляет только кафедры, другие подразделения, к которым, могут относится кабинеты, в нём не содержатся.

Структура представлена в таблице:

Таблица 1.3 – Структура кафедры из API ИИС

Ключ	Тип	Описание
id	Число	Идентификатор
name	Строка	Название
abbrev	Строка	Аббревиатура

1.1.4 Аудитория

Аудитория имеет две вложенных структуры: тип аудитории и номер здания. Типы аудитории принимают одно из следующих значений: лекционные занятия, практические занятия, лабораторные занятия, компьютерный класс.

Структура представлена в таблице:

Таблица 1.4 – Структура типа аудитории из API ИИС

Название	Тип	Описание
id	Число	Идентификатор
name	Строка	Название
abbrev	Строка	Аббревиатура

Номер здания представляет из себя структуру, в которой содержится только идентификатор и название здания. При этом идентификатор не совпадает с номером корпуса, это актуально только для корпусов с первого по пятый, поэтому значащим полем в первую очередь является название здания.

Даже аудитории, которые находятся вне университета, такие как, например, находящиеся в зданиях EPAM Systems в Парке высоких технологий, имеют присвоенное здание, в котором они де-факто не находятся. Например,

вышеупомянутые кабинетам присвоен второй корпус.

Структура представлена в таблице:

Таблица 1.5 – Структура номера здания из API ИИС

Название	Тип	Описание
id	Число	Идентификатор
name	Строка	Название

Значение вместимости аудитории встречается очень редко, чаще всего у компьютерных классов, только 27 аудиторий имеют значение для этого ключа. Аудитории также могут не иметь присвоенной кафедры.

Структура представлена в таблице:

Таблица 1.6 – Структура аудитории из API ИИС

Название	Тип	Опциональность	Описание
id	Число	Нет	Идентификатор
name	Строка	Нет	Название
note	Строка	Да	Заметка
capacity	Число	Да	Вместительность
department	Кафедра	Да	В таблице 1.3
auditoryType	Тип аудитории	Нет	В таблице 1.4
buildingNumber	Номер здания	Нет	В таблице 1.5

1.1.5 Группа

В случае с группой и преподавателем существуют некоторые особенности, данные структуры предоставляются в двух представлениях: обычном, которое является ответом на запрос всех групп и расписания группы, и встроенном, которое является частью структуры занятия. При этом они имеют как общие поля, которые содержатся в обоих представлениях, так и содержащиеся исключительно в одном представлении поля.

Структура представлена в таблице:

Таблица 1.7 – Структура группы из API ИИС

Название	Тип	Представление		Описание
		Обычное	Встроенное	
1	2	3	4	5
id	Число	Да	Нет	Идентификатор
name	Строка	Да	Да	Номер
calendarId	Строка	Да	Нет	Идентификатор календаря Google Calendar

Продолжение таблицы 1.7

1	2	3	4	5
course	Число	Да	Нет	Курс
educationDegree	Число	Да	Да	Степень образования
numberOfStudents	Число	Нет	Да	Количество студентов
facultyId	Строка	Да	Нет	Идентификатор факультета
facultyAbbrev	Строка	Да	Нет	Аббревиатура факультета
specialityName	Строка	Да	Да	Название специальности
specialityAbbrev	Строка	Да	Нет	Аббревиатура специальности
specialityCode	Строка	Нет	Да	Код специальности
specialityDepartmentEducationFormId	Число	Да	Нет	Идентификатор специальности

1.1.6 Преподаватель

Структура преподавателя, имеет схожие со структурой группы особенности, однако в ней меньше непересекающихся полей, а встроенное представление вовсе содержит все поля.

Значение степени в обычном представлении имеет значение аббревиатуры степени, а в встроенном представлении же имеет значение полного названия степени.

Структура представлена в таблице:

Таблица 1.8 – Структура преподавателя из API ИИС

Название	Тип	Представление		Описание
		Обычное	Встроенное	
1	2	3	4	5
id	Строка	Да	Да	Идентификатор
firstName	Строка	Да	Да	Имя
middleName	Строка	Да	Да	Отчество
lastName	Строка	Да	Да	Фамилия
rank	Строка	Да	Да	Ранг
degree	Строка	Да	Да	Степень либо аббревиатура
degreeAbbrev	Строка	Нет	Да	Аббревиатура степени

Продолжение таблицы 1.8

1	2	3	4	5
urlId	Строка	Да	Да	Идентификатор для доступа через сайт
calendarId	Строка	Да	Да	Идентификатор календаря Google Calendar
photoLink	Строка	Да	Да	Ссылка на фотографию
academicDepartment	Строки	Да	Да	Список аббревиатур кафедр
email	Строка	Нет	Да	Адрес электронной почты
jobPositions	Строки	Нет	Да	Должности

Также дополнительно стоит отметить, что значения rank, degree, degreeAbbrev, email и jobPositions являются опциональными.

1.1.7 Расписание

Структуры группы и преподавателя существуют только для запроса соответствующего расписания. Данные структуры не могут встретиться в одном расписании вместе. Так как в структуре занятия отсутствует информация о дне недели, от предоставляется в контейнере занятий по дням недели. Занятия по дням недели включают экзамены только в том случае, если существуют обычные занятия.

Структура представлена в таблице:

Таблица 1.9 – Структура расписания из API ИИС

Название	Тип	Опциональность	Описание
1	2	3	4
schedules	Занятия по дням недели	Нет	Словарь, где ключом является день недели, а значением является массив занятий
exams	Занятия	Нет	В таблице 1.10
startDate	Строка	Да	Дата начала занятий
endDate	Строка	Да	Дата конца занятий

Продолжение таблицы 1.9

1	2	3	4
startExamsDate	Строка	Да	Дата начала сессии
endExamsDate	Строка	Да	Дата конца сессии
studentGroupDto	Группа	Да	В таблице 1.7
employeeDto	Преподаватель	Да	В таблице 1.8

1.1.8 Занятие

Структура занятия является наиболее сложной, так как содержит много опциональных значений, имеет в себе другие структуры, которые ссылаются далее на третье структуры. Именно занятие связывает аудитории, группы и преподавателей.

Опциональность аудиторий, групп и преподавателей обусловлена тем, что при запросах расписания группы, для занятия может не быть назначен преподаватель. Например, если преподаватель не является сотрудником университета, а занятия проходят на базе предприятия под руководством его преподавателей.

Аналогично и занятия со стороны расписания преподавателя могут быть временно не назначены группы.

Структура представлена в таблице:

Таблица 1.10 – Структура занятия из API ИИС

Название	Тип	Опциональность	Описание
1	2	4	3
subject	Строка	Нет	Аббревиатура названия дисциплины
subjectFullName	Строка	Нет	Название дисциплины
weekNumber	Числа	Нет	Список номеров недель,
startLessonTime	Строка	Нет	Время начала занятия
endLessonTime	Строка	Нет	Время окончания
numSubgroup	Число	Нет	Номер подгруппы
split	Булево значение	Нет	Определяет, является ли занятие разделённым
note	Строка	Да	Заметка
lessonTypeAbbrev	Строка	Нет	Аббревиатура типа занятия

Продолжение таблицы 1.10

1	2	3	4
announcement	Булево значение	Нет	Определяет, является ли это занятие объявлением
dateLesson	Строка	Да	Дата занятия
startLessonDate	Строка	Нет	Дата начала периода проведения данного занятия
endLessonDate	Строка	Нет	Дата окончания периода проведения данного занятия
auditories	Строки	Да	Названия аудиторий и здание
studentGroups	Группы	Да	Список групп,
employees	Преподаватели	Да	Список преподавателей

1.2 Обзор аналогов

На момент написания дипломной работы существуют только два схожих программных средства для iOS; одно – для iPadOS. Они распространяются в Apple App Store. Для macOS приложений, аналогичных разрабатываемому, не существует. Представленные версии приложений для не являются нативными для macOS и запускаются через системную прослойку. В пунктах приведён обзор существующих приложений, их основные характеристики, возможности, ограничения.

1.2.1 BSUIR Timetable

BSUIR Timetable представляет из себя приложение доступное для iOS версии 13,0 и выше и macOS 11,0 и выше с чипом Apple M1 и выше. [2] Приложение разработано на платформе UIKit. Средняя оценка приложения на основании 32 отзывов составляет 3,7 балла. Поддерживаются русский и английский языки. Монетизация происходит с использованием рекламы и встроенной покупки для её отключения. Стоимость данной покупки составляет 0,99 Долларов США.

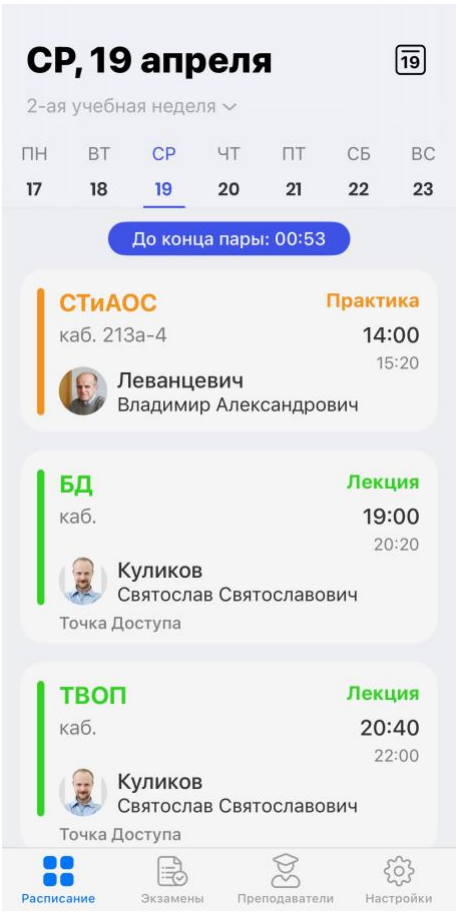
Основные возможности приложения представлены в таблице:

Таблица 1.11 – Основные возможности BSUIR Timetable

Секция	Возможность
Группа	Поиск по номеру
	Отображение списка всех групп
	Отображение расписания
	Отображение факультета
	Отображение специальности
	Выбор подгруппы
	Добавление в избранные
Преподаватель	Поиск по фамилии
	Отображение расписания
	Отображение списка всех преподавателей
	Отображение списка текущих преподавателей
Расписание	Отображение по датам
	Отображение времени до занятия
	Виджеты

Интерфейс данного приложения представлен на рисунке:

а)



б)

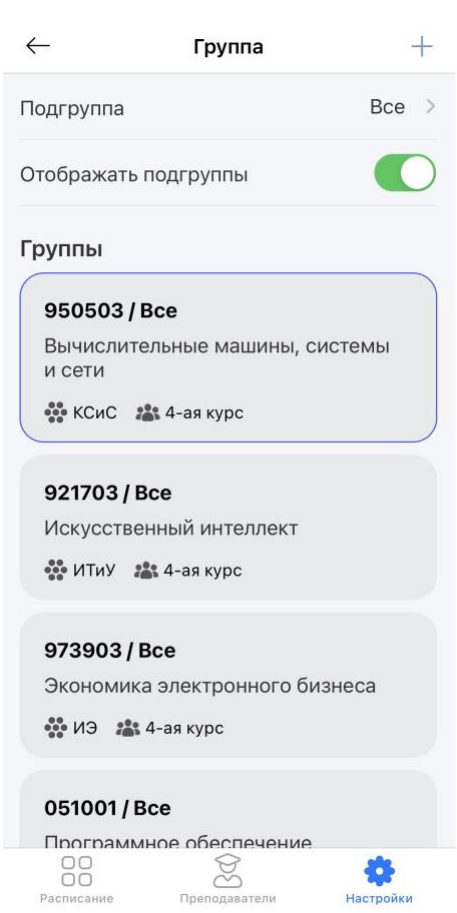


Рисунок 1.1 – интерфейс приложения BSUIR Timetable:
а – расписание группы; *б* – выбор из избранных групп

1.2.2 Bsuir Schedule

Bsuir Schedule является приложением, которое доступно для iOS и iPadOS версии 15,0 и выше и macOS 11,0 и выше с чипом Apple M1 и выше [3]. Приложение разработано на платформе SwiftUI. Средняя оценка приложения на основании 409 отзывов составляет 4,6 балла. Поддерживаются белорусский, украинский, русский и английский языки. Монетизация отсутствует, приложение распространяется на полностью безвозмездной основе. Исходные коды приложения доступны на платформе. GitHub [4].

Данное приложение является наиболее продвинутым среди аналогов в своей области. Его архитектура была разработана только для просмотра расписания занятий, из-за чего в текущей версии приложения нет функционала по сохранению информации о расписании между сеансами. Каждый раз, когда пользователь выходит из приложения, данные о расписании сбрасываются, что делает невозможным, например, просмотр расписания без доступа к сетевому подключению.

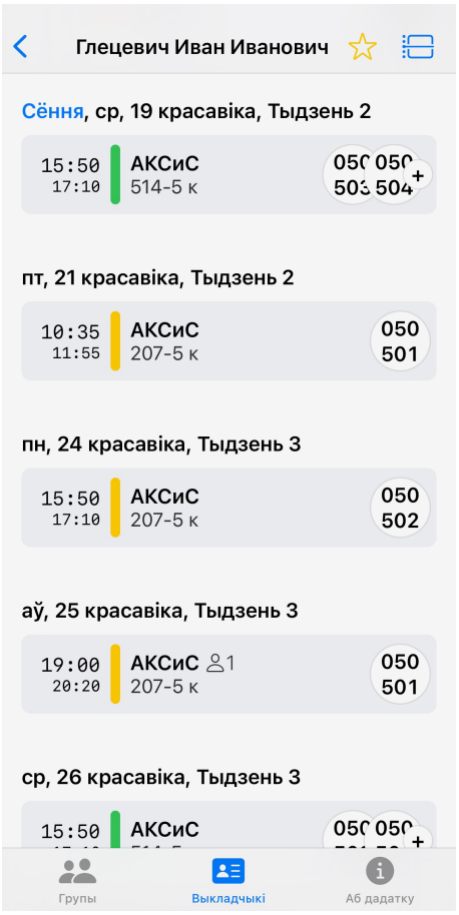
В данном дипломном проекте разрабатывается продвинутое приложение, основным фокусом разработки которого является обработка информации о расписании. Проект предполагает создание приложения, в котором особое внимание уделяется работе с информацией и другими предоставляемыми API ИИС БГУИР компонентами. Важно отметить, что данная информация о расписании и других элементах, предоставляемых API, хранится в постоянном хранилище приложения.

Таблица 1.12 – Основные возможности Bsuir Schedule

Секция	Возможность
Группа	Поиск по номеру
	Отображение списка всех групп
	Отображение расписания
	Добавление в избранные
Преподаватель	Поиск по фамилии, имени и отчеству
	Отображение списка всех преподавателей
	Отображение расписания
	Добавление в избранные
Расписание	Отображение по датам
	Отображение по неделям
	Выбор цвета для каждого типа занятия
	Виджеты
Другое	Изменение иконки приложения
	Очистка кэша

Интерфейс данного приложения представлен на рисунке:

а)



б)

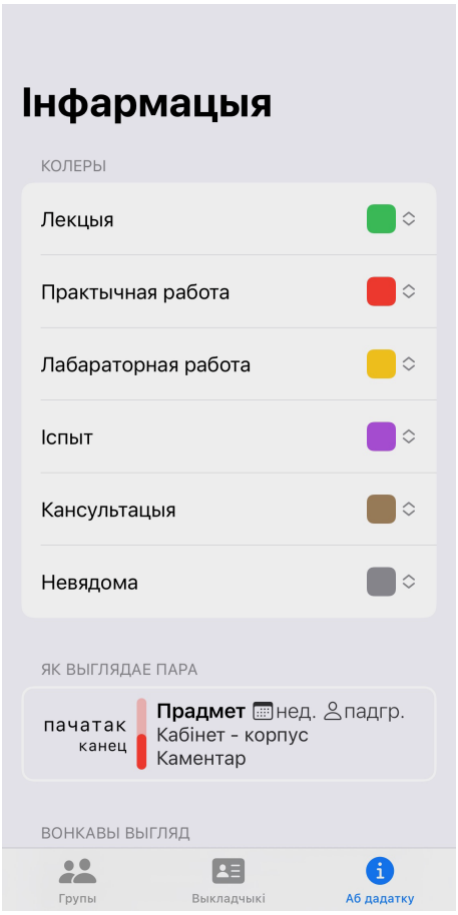


Рисунок 1.2 – интерфейс приложения Bsuir Schedule:
а – расписание преподавателя; *б* – настройки

1.3 Выбор фреймворка для разработки

Создание приложений на языке программирования Swift в первую очередь подразумевает использование двух платформ: UIKit или SwiftUI для всех операционных систем (ОС). Они используют соответственно паттерны Model-View-Controller и View-ViewModel-Model.

1.3.1 Model-View-Controller

Model-View-Controller (MVC) — это архитектурный паттерн, который является основой для разработки приложений. Он обеспечивает разделение приложения на три основных компонента: модель (Model), представление (View) и контроллер (Controller).

Использование паттерна MVC позволяет создавать гибкие и масштабируемые приложения, где каждый компонент отвечает за свою часть функциональности.

Структура паттерна представлена на рисунке:

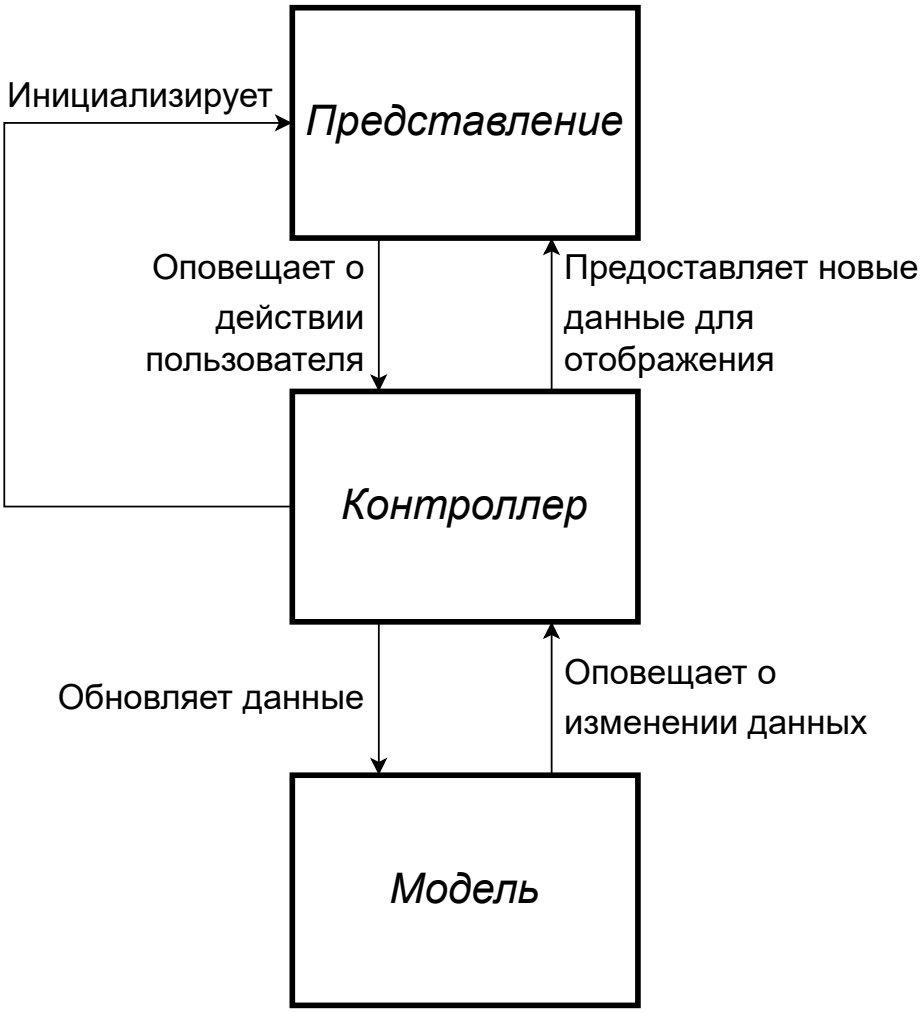


Рисунок 1.3 – схема структуры паттерна MVC

Модель отвечает за хранение данных и логику обработки данных: в ней описывается бизнес-логика, правила валидации и другие операции, связанные с данными. Модель не имеет информации о представлении и контроллером, то есть не имеет доступа к их полям, функциям и иному.

Представление отображает данные пользователю и предоставляет возможность взаимодействия с приложением через графический интерфейс. В ней определяется внешний вид и макет интерфейса, это происходит в графическом редакторе. Представление не имеет представления о существовании модели и контроллера.

Контроллер является посредником между моделью и представлением. Он получает информацию от представления, обрабатывает её и вносит соответствующие изменения в модель. Таким образом, контроллер обеспечивает взаимодействие и согласованность между моделью и представлением, делая возможным реализацию функциональности приложения. Чаще всего контроллер является наиболее наполненным исходным кодом элементом приложения.

1.3.2 UIKit

UIKit является основной платформой для создания пользовательских интерфейсов на iOS и iPadOS [5]. В платформе применяется в первую очередь императивное программирование. UiKit представляет собой набор готовых компонентов интерфейса, таких как кнопки, текстовые поля, таблицы, контейнеры, иные элементы интерфейса и контейнеры для них. UIKit использует паттерн проектирования архитектуры MVC.

В целом, UIKit и SwiftUI представляют собой две разные платформы для создания пользовательских интерфейсов в операционных системах Apple, каждая со своими плюсами и минусами. Разработчики должны выбирать платформу, которая лучше всего подходит для их задач и опыта в программировании.

UIKit и AppKit имеют следующую парадигму для создания пользовательских интерфейсов: используются файлы xib или Storyboard, в которых определяются все элементы интерфейса и их свойства, а уже в файлах Swift программируется поведение и особенности интерфейса, программный код в данном случае пишется в императивной парадигме. Обе платформы используют паттерн Model-View-Controller.

AppKit и UIKit очень похожи и имеют одинаковую структуру и логику. Элементы имеют одинаковые названия и одинаковые параметры, хоть и существуют особенные для платформ элементы. Несмотря на это, эти платформы не являются взаимозаменяемыми, хоть технически и существует возможность создания приложения сразу на двух платформах, по сути это два разных приложения, которые объединены только Моделью, а все элементы графического интерфейса необходимо имплементировать отдельно для каждой из платформ.

1.3.3 Model-View-ViewModel

Model-View-ViewModel (MVVM) является схожим с MVC паттерном, используемым для разделения компонентов приложения на три основных уровня: Модель, Представление и Модель Представления (ViewModel).

Одним из недостатков MVVM является некоторая его избыточность для простых приложений, которые имеют небольшое количеством логики и данных. MVVM требует больше кода, чем, например, простая парадигма Model-View-Controller. При этом эти недостатки нивелируются при разработке относительно больших приложений, так как MVVM предоставляет огромные возможности для повторного использования различных элементов Представления. На начальных этапах разработка приложение, использующего платформу MVVM, может быть затратным с точки зрения количества кода и времени разработки, однако повторное использование и универсальность стандартизированных представлений.

Структура паттерна представлена на рисунке:

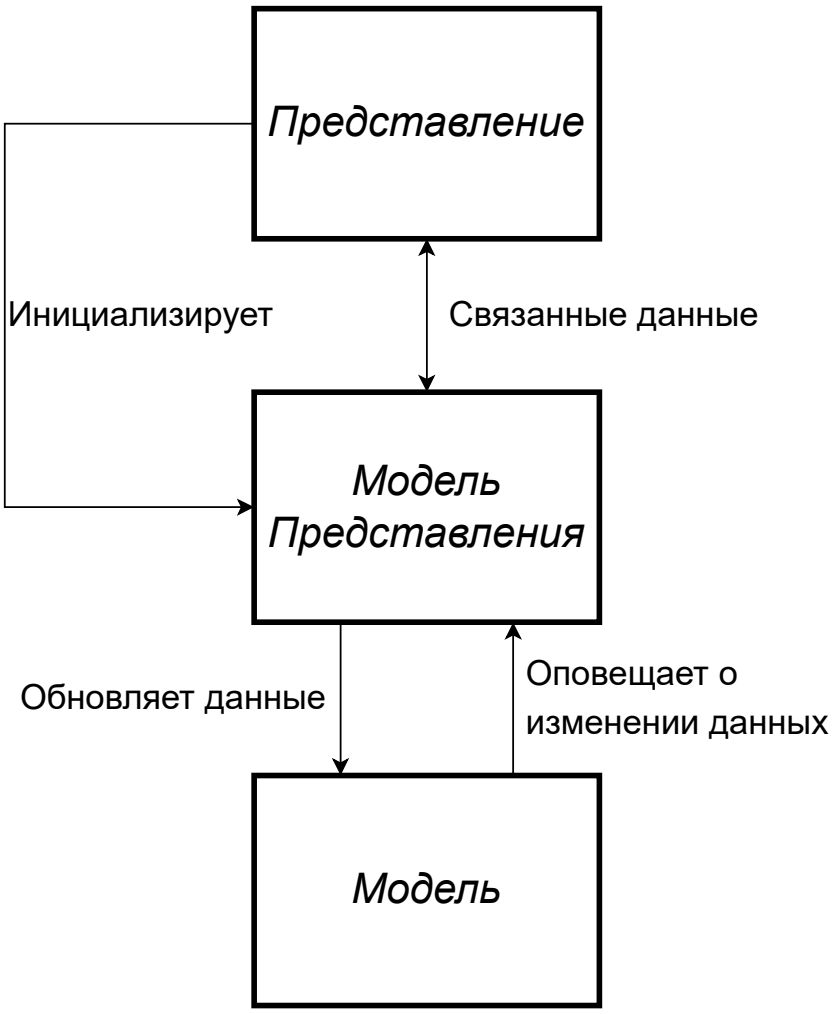


Рисунок 1.4 – схема структуры паттерна MVVM

Модель, аналогична модели MVC, не имеет информации о Представлением и модели представления.

Представление, частично похоже на MVC, отображает данные пользователю и обрабатывает пользовательский ввод. При это в отличии от MVC, представление напрямую обращается к модели представления, последняя чаще всего является полем представления.

Модель представления является посредником между моделью и представлением и чем-то схожа с контроллером MVC. Модель имеет общее с Представляем данные, которые автоматически обновляются и модели представления и в самом представлении при изменении их как стороны пользователя, так и со стороны модели. В MVC же контроллер при обновлении данных запускает метод обновления Представления. В каком-то смысле Представление совмещает контроллер и представление MVC. Модель представления также может обслуживать несколько представлений одновременно.

1.3.4 SwiftUI

SwiftUI является более современной платформа для создания пользовательских интерфейсов, которая была запущена в 2019 году [6]. В платформе применяется в первую очередь декларативное программирование. SwiftUI основан на декларативном подходе, где для создания интерфейса описывается его конечное представление в виде дерева вложенных представлений. SwiftUI паттерн проектирования архитектуры MVVM.

Одним из основных преимуществ SwiftUI является более простой и интуитивный синтаксис для описания пользовательского интерфейса. Кроме того, SwiftUI обладает рядом функций, которых нет в UIKit, таких как автоматические анимации и адаптивный дизайн, что упрощает создание интерфейсов, которые могут адаптироваться к разным размерам экранов.

Также преимуществом SwiftUI является возможность непосредственной работы с данными в представлениях, благодаря связыванию данных (Data Binding), который позволяет отслеживать изменения данных в Модели или Модели Представления и автоматически обновлять Представление, что значительно упрощает разработку приложений.

На данный момент SwiftUI не поддерживает все функции, которые доступны в UIKit, для SwiftUI 4.0 чаще всего это узкоспециализированные возможности. Для добавления элементов из UIKit в структуру SwiftUI существует структура UIViewControllerRepresentable [7], которая позволяет взаимодействовать с элементами UIKit и преобразовывать их в элементы путём некоторых манипуляций SwiftUI.

1.3.5 Сравнение UIKit с SwiftUI в контексте разрабатываемого программного средства

UIKit использует императивный подход к программированию, который требует явного указания всех деталей в коде, в то время как SwiftUI использует декларативный подход, который позволяет описывать только конечный результат, а не метод его достижения. Декларативная парадигма является более удобной для разработчика, однако может усложнить разработку при создании крайне специализированных элементов интерфейсов. Так как разрабатываемое программное средство не подразумевает создания таких элементов, SwiftUI является более подходящим для разработки.

Среди исходных данных к дипломному проекту приведена кроссплатформенность, а именно реализация приложения для iOS, iPadOS и macOS. Разработка приложения UIKit потребует большего количества времени, так как представление и контроллер необходимо разрабатывать отдельно для каждой платформы. Хоть это и были бы крайне схожие элементы, необходимость параллельной разработки значительно затруднила бы какие-либо изменения проекта. Однако, среди преимуществ UIKit можно выделить возможность создания более продвинуты и эстетически нестандартных

пользовательских графических интерфейсов.

SwiftUI обеспечивает адаптивный дизайн, который автоматически изменяет интерфейс в зависимости от размера Представления, а UIKit не имеет встроенной поддержки адаптивности, то есть требует отдельных расчётов для размеров элементов интерфейса разных устройств, что потребует большого количества времени в контексте разработки данного программного средства для разных типов устройств с разными размерами дисплеев.

С точки сторонних библиотек и инструментов: UIKit имеет более широкий выбор сторонних библиотек и инструментов для разработки, в то время как для SwiftUI являются более ограниченными.

Так как SwiftUI имеет декларативною парадигму программирования, автоматический адаптивный дизайн и возможности создания кроссплатформенных приложений для iOS и iPadOS без необходимости параллельной разработки Представления и Контроллера для каждой платформ, платформа оптимальной для разработки данного программного средства.

1.4 Интегрированная среда разработки Xcode

Xcode является интегрированной средой разработки (IDE) для приложений, предназначенных для платформ Apple: iOS, iPadOS, macOS, watchOS и tvOS. Xcode позволяет создавать приложения, тестировать их, отлаживать и развёртывать.

Xcode включает в себя различные редакторы кода, визуальные редакторы пользовательских интерфейсов, отладчики и профилировщики производительности, а также инструменты для автоматизации и управления проектами.

Возможно проводить установку и отладку приложений как в симуляторе, так и на устройстве, которое можно подключить как проводным, так и беспроводным образом. В режиме отладки Xcode выводит подробную информацию о работе устройства или симулятора. Например, возможно приостановить выполнение приложения и рассмотреть отображаемый графический пользовательский интерфейс как набор слоёв.

Xcode также включает в себя библиотеку функций, которые могут быть использованы для создания приложений: инструменты для работы с базами данных, мультимедийными файлами, сетевыми протоколами, графикой.

Мгновенный просмотр результатов компиляции графического пользовательского интерфейса, разработанного на платформе SwiftUI, позволяет мгновенно отображать последствия изменения кода реальном времени. Это разработчикам оперативно тестировать новые идеи и экспериментировать с различными вариантами оформления, не теряя время на компиляцию и запуск приложения в симуляторе или на устройстве. Также доступен мгновенный просмотр сразу для нескольких вариантов отображения: разные макеты устройств, ориентации, цветовые темы и иные модификации.

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Так как программное средство подразумевает большое количество функционала, в том числе и работу с базой данных и получение данных из API ИИС БГУИР, существует необходимость в создании предопределённой структуры.

Очевидно, что в первую очередь структуру можно представить как MVVM. Однако наиболее верным решением будет также дополнить её. Кроме модели, представления и модели представления, добавить блоки работы с базой данных (БД) и иные блоки, которые нельзя отнести к обычным Моделям.

Таким образом, структура проекта состоит из следующих блоков.

- модели;
- представления;
- модели представлений;
- блок извлечения;
- блок декодирования;
- постоянное хранилище;
- фоновые контексты;
- контекст представления.

Взаимосвязь между основными блоками проекта отражена на структурной схеме ГУИР.400201.019 С1.

2.1 Модели

Модели будут являться наиболее крупной частью проекта, так как необходима не только реализация моделей для всех данных из API ИИС, но и создание множества дополнительных моделей для обработки логики. При этом использование

Swift предполагает широкое использование расширений, то есть описание моделей в разных файлах и создание вместо новых моделей расширений к существующим, часто это происходит в файле другой модели. Например, для создания функции разделения массива занятий по датам их проведения принято не создавать глобальную функцию или метод для класса, в котором необходимо это действие, вместо этого создаётся расширение для массива элементом которого являются занятия. В таком случае функция будет вызываться от самого массива.

Написание расширений для модели позволяет создавать наиболее универсальные функции и не заполнять глобальное пространство имён специфичными элементами.

Модели имеют связи с моделями представления, которые создают и изменяют первых, а блок декодирования может создавать новые и обновлять существующие модели. Также модели имеют связь с фоновыми контекстами, где производятся фоновые вычисления.

2.2 Представления

Структура SwiftUI подразумевает существование некого корневого Представления, из которого начинается работа с приложением. Из этого Представления существуют ссылки на другие представления, и так далее. При это всегда существует возможность вернуться назад по пути представлений, также Представления путём ссылок могут приводить на самих себя.

В SwiftUI представлением является всё, что отображается в интерфейсе, то есть, например, как кнопка является представлением, которое описано программным декларативным образом, так и содержащий кнопку, например, список, также является представлением, то есть одно представление может включать в себя мужество других и даже самого себя. Благодаря этому свойству и возможности представлений принимать переменные при инициализации, появляется возможность широкого использования шаблонов.

Представления должны иметь одинаковый функционал на всех платформах, для которых ведётся разработка, и быть реализованными наиболее универсальным методом, который поддерживает адаптивный графический пользовательский интерфейс.

Представление связано с моделями представлений: получает из них данные и отправляет данные на обработку. Также считается приемлемым и выполнять некоторые простые вычисления сразу в представлениях, а в некоторых относительно простых или использующих большое шаблоны Представлениях вовсе не нужны модели представлений, они напрямую взаимодействуют с моделями. Это же касается и представлений, которым не нужна какая-либо обработка данных, например, ячейка таблицы или кнопка.

Представление также взаимодействует с контекстом представления, из которого получает Модели и их обновлениях.

2.3 Модели представлений

Модели представлений подразумеваются прослойкой между моделями и представлениями, в которых осуществляется специфичные для представления вычисления, неспецифичные же стоит выносить в сами модели, а именно в расширения моделей.

Модели представлений чаще всего предназначены для обработки информации из моделей для приведения их в советующий конкретному Представлению формат, примерном такой функциональности может служить вычисления периода длительности сессии для групп:, когда в модели хранятся только даты начала и конца сессии, в модели представления из них вычитается длительность и уже этот высчитанный параметр передаётся для отображения пользователю в представление.

Модель представления имеет связь как с моделью, так и с Представлением.

2.4 Блок извлечения

Для получения необходимой информации из API ИИС БГУИР необходимо создание блока, который будет извлекать данные из API: отправлять запрос, получать и частично обрабатывать ответ.

Блок извлечения должен передавать полученные данные, либо информацию об ошибке, в блок декодирования.

2.5 Блок декодирования

Блок декодирования получает данные из блока извлечения. Он должен быть реализован в соответствии структурами, представленными в API ИИС БГУИР, подробное их описание приведено в подразделе 1.1. Блок создаёт модели в фоновых контекстах и заполняет их декодированными данными. Для обновления моделей должна быть реализована возможность получения существующих моделей и записи в них новых извлечённых данных.

2.6 Постоянное хранилище

Стандартным фреймворком для работы с БД CoreData является стандартным для iOS, iPadOS и macOS. Постоянное хранилище обеспечивает сохранение данных приложения в постоянную память устройства и их восстановление после выхода из приложения.

Постоянное хранилище является контейнером для хранения объектов CoreData, который позволяет сохранять, загружать и управлять данными приложения.

При использовании CoreData, данные приложения хранятся в базе данных SQLite, которая представляет собой один файл в контейнере приложения. Постоянное хранилище управляет этим файлом и предоставляет удобный интерфейс для взаимодействия с данными без использования SQL.

Данные из постоянного хранилища читаю и записывают различные контексты, которые могут автоматически обновляются при изменении данных в постоянном хранилище. При этом без выполнения операции сохранения контекста новые или изменённые данные не попадают не только в постоянное хранилище, но и в другие контексты.

Постоянное хранилище автоматически выполняет операции по объединению данных из сохранённого контекста постоянного хранилища в соответствии с заданными правилами. Например, если в постоянном хранилище имеется группа с определённым номером, и в одном из контекстов появилась группа с таким же номером, то, при сохранении контекста, хранилище может провести различные операции, которые зависят от настроек: полностью игнорировать данные из контекста; полностью игнорировать данные из хранилища; объединить объекты с приоритетом данных из хранилища; объединить объекты с приоритетом данных из контекста; выдать

ошибку сохранения.

Постоянное хранилище связано с контекстом представления и фоновыми контекстами.

2.7 Фоновые контексты

Фоновые контексты используются для обработки данных, которые напрямую не отображаются пользователю, то есть они используются для выполнения задач на фоне, например, для загрузки данных из сети.

Основным отличим между фоновым контекстом и контекстом представления является то, что первый работает в отдельном потоке, в то время как второй работает в основном потоке, что означает, что задачи, выполняемые фоновым контекстом, не блокируют пользовательский интерфейс.

Фоновые контексты имеют связь как с моделями, так и с блоком декодирования.

2.8 Контекст представления

В CoreData, главный контекст, он же контекст представления является контекстом, который обычно используется для работы с графическим интерфейсом в основном потоке приложения.

Контекст представления представляет собой корневой объект контекстов CoreData, который создаётся автоматически при инициализации стека CoreData.

Основной задачей контекста представления является предоставление доступа к объектам из постоянного хранилища представлениям и другим основных функциям приложения, которые непосредственно связанных с представлением. В отличие от других контекстов, контекст представления создаётся в главном потоке приложения и связан с главным потоком графического интерфейса, что позволяет ему выполняться в том же потоке, что и обработка графического интерфейса. Данный факт также накладывает и ограничения, контекст представления опасно использовать для загрузки и крупного изменения данных, так как при критической ошибке в нём велик шанс непредвиденного завершения всего приложения.

Контекст Представления обычно настроен таким образом, чтобы изменения, сделанные в нём, автоматически сохраняются в постоянном хранилище, что гарантирует, что данные, которые отображают представления, всегда будут соответствовать актуальным данным в постоянном хранилище.

Контекст представления связан только с представлением и постоянным хранилищем.

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В целом функциональная структура проекта аналогична структурной, однако из-за декларативного стиля написания в неё не входят представления. Также не описано функциональное проектирование связанных с БД блоков, кроме модели данных, так как они используют стандартные функции и методы. Взаимосвязь между основными компонентами представлена на диаграмме классов ГУИР.400201.019 РР.1.

Дополнительно стоит отметить использование протоколов. Протоколо-ориентированная парадигма – это парадигма, в которой основной упор делается на работу с протоколами (интерфейсами), а не на классах и наследовании. В основе парадигмы лежит использование протоколов для определения интерфейсов, а не классов, как это происходит в объектно-ориентированном программировании. Это означает, что в протоколо-ориентированной парадигме акцент смещается на то, что объекты могут делать, с того, чем они являются.

Swift поддерживает данную парадигму, предоставляя средства для определения протоколов, наследования протоколов, расширения протоколов и даже использования протоколов в качестве типов данных. Протоколы в Swift могут использоваться для определения интерфейсов, которые объекты должны реализовывать, и для расширения функциональности существующих типов данных.

Протоколы в Swift могут содержать объявления свойств, методов и инициализаторов, а также расширения, которые позволяют добавлять новую функциональность к типам данных, которые уже реализуют протокол. Более того, протоколы в Swift могут быть использованы в качестве типов данных, что позволяет создавать переменные и константы, тип которых определяется протоколом, а не конкретным классом или структурой.

Преимущества этой парадигмы включают в себя более лёгкую и быструю разработку программ, более простое тестирование кода, а также более лёгкое переиспользование кода. Эти преимущества особенно заметны в Swift, где протоколы могут использоваться в качестве типов данных и для расширения функциональности уже существующих типов данных.

Так как в проекте предусмотрено постоянное хранение данных, необходимо создать модель данных. Проект подразумевает наличие связи между данными, поэтому необходимо создать реляционную модель данных.

Основной связующей таблицей, очевидно, является занятие, которое связывает группы, преподавателей и кабинеты, при этом последние также имеют и другие связи с таблицами, которые описывают их особенности. Всего создано 13 основных таблиц и 7 служебных таблиц для описания связей типа многие-ко-многим. Таблицы и связи между ними представлены графически на модели данных ГУИР.400201.019 РР.2, а также подробно описаны в подразделе 3.1.

Таким образом, можно выделить следующую функциональную

структуру проекта:

- модель данных;
- протоколы;
- модели;
- модели представления;
- блок извлечения;
- блок декодирования.

3.1 Модель данных

В данном подразделе представлены таблицы моделей данных и связи между ними. Типы данных представлены в нотации Swift, также, как они описаны в исходном коде, то есть на опциональность значения указывает вопросительный знак после названия типа данных. При этом служебные таблицы не описываются, так как они создаются автоматически.

Описание модели состоит из двух частей:

- описание атрибутов таблицы;
- описание связей таблицы.

3.1.1 Auditorium

`Auditorium` является таблицей, описывающей аудиторию, её структура представлена в таблице:

Таблица 3.1 – Таблица `Auditorium`

Атрибут	Тип	Описание
building	Int16	Номер корпуса
name	String	Название
floor	Int16	Номер этажа
capacity	Int16?	Вместимость аудитории
favorite	Bool	Флаг, указывающий, является ли аудитория избранной
formattedName	String	Форматированное название аудитории, например «207б-4» или «epam 103 к1»
note	String?	Примечание или дополнительная информация о аудитории
outsideUniversity	Bool	Флаг, указывающий, находится ли аудитория вне корпусов университета

Первичными ключами являются `buiding`, `name`, `floor`. Таблица `Auditorium` имеет четыре опциональные связи с другими таблицами, что представлено в таблице:

Таблица 3.2 – Связи таблицы `Auditorium`

Название	Тип	Таблица
compoundSchedule	Многие-ко-многим	CompoundSchedule
department	Многие-к-одному	Department
lessons	Многие-ко-многим	Lesson
type	Многие-к-одному	AuditroiumType

3.1.2 AuditoriumType

Таблица `Auditorium` описывает тип аудитории. Структура представлена в таблице:

Таблица 3.3 – Таблица `AuditoriumType`

Атрибут	Тип	Описание
id	Int16	Уникальный идентификатор типа аудитории.
abbreviation	String	Краткое название типа аудитории.
name	String	Полное название типа аудитории.

Первичным ключом является `id`. Таблица имеет опциональную связь типа один-к-многим с таблицей `Auditroium`.

3.1.3 CompoundSchedule

`CompoundSchedule` является таблицей, которая имеет только один атрибут: `name` – который содержит уникальное название, которое является первичным ключом. Данная таблица имеет опциональные связи типа многие-ко-многим с `Auditorium`, `Employee` и `Group`, которые необходимы для просмотра объединённого расписания занятий.

3.1.4 Degree

Таблица `Degree` описывает учёную степень преподавателя, атрибуты, их типы и описания приведены в таблице:

Таблица 3.4 – Таблица `Degree`

Атрибут	Тип	Описание
abbreviation	String	Краткое название степени
name	String?	Полное название степени

Первичным ключом является `abbreviation`. Необходимость в данной таблице вытекает из указанной в пункте 1.1.6 особенности предоставления научной степени преподавателей, а именно того, что она разные представления названия степени содержатся в ответах на разные запросы к

API. Для предотвращения разного одной и той же степени, информация о ней вынесена в отдельную таблицу, которая имеет связь типа один-к-многим с `Employee`.

3.1.5 Department

Таблица `Department`, описывает подразделение университета, которым может быть как кафедра, так и, например, научно-исследовательская лаборатория или часть. Структура представлена в таблице:

Таблица 3.5 – Таблица `Department`

Атрибут	Тип	Описание
abbreviation	String	Краткое название подразделения
favorite	Bool	Флаг, указывающий, является ли подразделение избранным
id	Int16	Уникальный идентификатор подразделения
name	String?	Полное название подразделения

Первичным ключом являются `abbreviation`. Таблица имеет две опциональные связи с другими таблицами, что представлено в таблице:

Таблица 3.6 – Связи таблицы `Department`

Название	Тип	Таблица
auditories	Многие-к-одному	Auditorium
employees	Многие-ко-многим	Empoyee

3.1.6 EducationTask

`EducationTask` является таблицей, которая описывает задания, которые пользователи сами добавляют. Таблица не имеет первичного ключа. Структура приведена в таблице:

Таблица 3.7 – Таблица `EducationTask`

Атрибут	Тип	Описание
creation	Date	Дата создания задачи
deadline	Date	Дата, до которой необходимо выполнить задачу
images	[Data]?	Массив данных, содержащих изображения
note	String	Заметка к задаче
subject	String	Название предмета

3.1.7 EducationType

Таблица `EducationType` описывает тип получения образования:

Таблица 3.8 – Таблица `EducationType`

Атрибут	Тип	Описание
id	Int16	Уникальный идентификатор типа получения образования
name	String	Название типа получения образования

Первичным ключом является атрибут `id`. `EducationType` имеет опциональную связь типа один-к-многим с таблицей `Speciality`

3.1.8 Employee

Таблица `Employee` описывает преподавателя университета и все доступные для него атрибуты из API ИИС. Структура представлена в таблице:

Таблица 3.9 – Таблица `Employee`

Атрибут	Тип	Описание
id	Int32	Уникальный идентификатор преподавателя
educationEnd	Date?	Дата окончания образования преподавателя
educationStart	Date?	Дата начала образования преподавателя
examsEnd	Date?	Дата окончания экзаменов преподавателя
examsStart	Date?	Дата начала экзаменов преподавателя
favorite	Bool	Флаг, указывающий, является ли преподаватель избранным
firstName	String	Имя преподавателя
lastName	String	Фамилия преподавателя
lessonsUpdateDate	Date?	Дата последнего обновления списка занятий, которые проводит преподаватель
middleName	String?	Отчество преподавателя
photo	Data?	Фотография преподавателя
photoLink	String?	Ссылка на фотографию преподавателя
rank	String?	Звание преподавателя
urlID	String?	Уникальный идентификатор преподавателя в системе

Первичным ключом является `id`, Таблица `Emplpyee` имеет четыре опциональные связи с другими таблицами, что представлено в таблице:

Таблица 3.10 – Связи таблицы `Employee`

Название	Тип	Таблица
compoundSchedule	Многие-ко-многим	CompoundSchedule
degree	Многие-к-одному	Degree
departments	Многие-ко-многим	Department
lessons	Многие-ко-многим	Lesson

3.1.9 Faculty

Таблица `Faculty` описывает факультет университета и обладает атрибутами, представленными в таблице:

Таблица 3.11 – Таблица `Faculty`

Атрибут	Тип	Описание
id	Int16	Уникальный идентификатор факультета
abbreviation	String?	Краткое название факультета
name	String?	Полное название факультета

Первичным ключом является атрибут `id`. Таблица имеет опциональную связь типа один-к-многим с таблицей `Speciality`.

3.1.10 Group

Таблица `Group` содержит атрибуты учебной группы студентов. Структура представлена в таблице:

Таблица 3.12 – Таблица `Group`

Атрибут	Тип	Описание
name	String	Название группы студентов.
course	Int16	Номер курса, на котором учится группа.
educationDegreeValue	Int16	Ступень образования группы
educationEnd	Date?	Дата окончания обучения группы.
educationStart	Date?	Дата начала обучения группы.
examsEnd	Date?	Дата окончания экзаменационной сессии для группы.
examsStart	Date?	Дата начала экзаменационной сессии для группы.
favorite	Bool	Флаг, указывающий, является ли группа избранной
lessonsUpdateDate	Date?	Дата последнего обновления информации о занятиях для группы.
nickname	String?	Никнейм группы, если таковой имеется.
numberOfStudents	Int16	Количество студентов, учащихся в группе

3.1.11 Lesson

Таблица `Lesson` содержит атрибуты занятия. Данная таблица является связующей для всех остальных таблиц базы данных, через неё проходят связи между аудиториями, преподавателями и группами. Структура представлена в таблице:

Таблица 3.13 – Таблица `Lesson`

Атрибут	Тип	Описание
abbreviation	String	Краткое название учебной дисциплины занятия
auditoriesNames	[String]	Названия аудиторий
dateString	String	Дата проведения занятия
employeesIDs	[Int32]	Идентификаторы преподавателей
startLessonDateString	String	Дата начала периода проведения занятия
timeStart	String	Время начала занятия
weekday	Int16	День недели проведения занятия
weeks	[Int]	Номера недель, в которые проводится занятие
subgroup	Int16	Номер подгруппы
endLessonDate	Date	Дата окончания периода проведения занятия
note	String?	Примечание к занятию
subject	String?	Полное название учебной дисциплины
timeEnd	String	Время окончания занятия

Первичными ключами являются `abbreviation`, `auditoriesNames`, `dateString`, `employeesIDs`, `startLessonDateString`, `timeStart`. `weekday`, `weeks`. Таблица имеет четыре опциональные связи, что представлено в таблице:

Таблица 3.14 – Связи таблицы `Lesson`

Название	Тип	Таблица
auditories	Многие-ко-многим	Auditroium
employees	Многие-ко-многим	Employee
groups	Многие-ко-многим	Group
type	Многие-к-одному	LessonType

3.1.12 LessonType

Таблица `LessonType` описывает тип занятия и содержит атрибуты, которые представлены в таблице:

Таблица 3.15 – Таблица `LessonType`

Атрибут	Тип	Описание
id	String	Уникальный идентификатор типа занятия
abbreviation	String	Краткое название типа занятия
colorData	String	Строка, содержащая кодированные данные о цвете типа занятия
name	String	Полное название типа занятия

Атрибут `id LessonType` является первичным ключом, также таблица имеет одну опциональную связь типа один-к-многим с таблицей `Lesson`.

3.1.13 Speciality

Таблица `Speciality` описывает специальность студенческой группы, описание атрибутов приведено в таблице:

Таблица 3.16 – Таблица `Speciality`

Атрибут	Тип	Описание
id	Int32	Уникальный идентификатор специальности
abbreviation	String	Краткое название специальности
code	String?	Код специальности
name	String	Полное название специальности

3.2 Протоколы

Кроме стандартных протоколов, таких как `Decodable`, `ObservableObject`, `CaseIterable`, в проекте используются собственные протоколы, которые, в первую очередь. Используются для создания, универсальных представлений.

3.2.1 Favored

Протокол `Favored` описывает общие свойства и методы для классов, которые могут быть добавлены в избранное.

Класс, который реализует протокол `Favored`, должен реализовывать протокол `ObservableObject`, чтобы его объекты могли быть наблюдаемыми в SwiftUI, что позволяет графическому интерфейсу обновляться автоматически.

Класс также должен реализовывать протокол `NSManagedObject`, который определяет класс, который может быть сохранен и управляем в CoreData, что позволяет создавать универсальные функции для обработки изменения статуса соответствующего этому протоколу объекта.

Также в протоколе определено свойство `favorite` типа `Bool`, которое используется для хранения флага, который является ли объект избранным или нет. Данное свойство необходимо для отображение избранных элементов в представлении избранных элементов.

Таким образом, если класс, который хранится в CoreData и должен отображаться в SwiftUI и должен иметь возможность быть помеченным как избранный и сохранённым в CoreData, то он может реализовать этот протокол и использовать его свойства и методы.

3.2.2 SectionType

Протокол `SectionType` описывает тип, который должен быть перечислимым, имеет хешируемое значение, должен содержать описание типа `description`, а также должен содержать все возможные значения в `AllCases`, который должен быть массивом того же типа, что и сам протокол.

Перечислимые типы, которые соответствуют протоколу `SectionType`, должны реализовывать переменную `description`, которая содержит описание данного типа.

Протокол `SectionType` также должен реализовывать протокол `Hashable`, чтобы объекты данного типа можно было хешировать и использовать в качестве ключей в словарях.

Наконец, протокол `SectionType` требует реализации статического свойства `AllCases`, которое должно содержать все возможные значения перечисления. Это свойство используется, например, для создания списка всех возможных значений перечисления, которые могут быть использованы в пользовательском интерфейсе.

3.2.3 EducationBounded

Протокол `EducationBounded` описывает сущности, которые имеют определенный период обучения и сдачи экзаменов, а также позволяет получить даты занятий и экзаменов на основе указанных дат.

Он содержит следующие свойства:

- `educationStart`: дата начала обучения, если она была предоставлена API;
- `educationEnd`: дата окончания обучения, если она была предоставлена API;
- `examsStart`: дата начала экзаменов, если она была предоставлена API;
- `examsEnd`: дата окончания экзаменов, если она была предоставлена API;

Протокол также имеет расширения для `Group` и `Employee`, которые позволяют декодировать даты обучения и экзаменов из JSON, используя соответствующие ключи в `EducationDatesCodingKeys`.

Кроме того, в расширении `EducationBounded` используются методы `datesStride`, которые возвращают массив дат между двумя указанными датами включительно.

3.2.4 EducationRanged

Протокол `EducationRanged`, определяет свойства, связанные с периодом обучения, в частности, определяет два свойства: `educationRange` и `educationDates`.

Свойство `educationRange` определяет диапазон дат, который содержит

период обучения. Этот диапазон вычисляется на основе дат начала и конца обучения, а также дат начала и конца экзаменов, если они указаны.

Свойство `educationDates` определяет массив дат, который содержит все даты в периоде обучения, вычисленные на основе `educationRange`.

Кроме того, протокол `EducationRanged` имеет метод `dividedEducationDates`, который разделяет даты на две группы: прошлые даты и будущие даты. Этот метод вычисляет текущую дату и затем использует её для разделения всех дат на две группы.

Первое расширение протокола `EducationRanged` добавляет реализацию свойств и метода. Второе расширение расширяет последовательность `Sequence`, которая содержит элементы типа `Lesson`. Оно определяет свойства и методы, аналогичные тем, что определены в расширении `EducationRanged`

3.2.5 LessonsRefreshable

Протокол `LessonsRefreshable` позволяет классам, реализующим его, обновлять список занятий из API.

Протокол объявляет требования, которые должны быть реализованы классами, которые используют этот протокол. Он содержит ассоциированный тип (`associatedtype`), который позволяет определить, какой тип данных будет использоваться для обновления занятий.

Свойства:

- `lesson` представляющее собой множество (`NSSet`) занятий;
- `lessonsUpdateDate` представляющее дату последнего обновления занятий.

Метод `update` метод, который возвращает результат типа, заданного через ассоциированный тип (`AssociatedType`).

Метод `fetchLastUpdateDate` возвращает дату последнего обновления занятий. Этот метод должен быть реализован в классах, которые используют протокол `LessonsRefreshable`.

Для Employee и Group также реализованы расширения, которые реализуют эти методы.

3.2.6 Scheduled

`Scheduled` является ключевым протоколом, которому соответствуют `Auditorium`, `Group` и `Employee` и `CompoundSchedule`. Именно этот протокол используется для `ScheduleView`, который отображает расписание.

Протокол `Scheduled` описывает сущности, которые имеют расписание, могут быть отмечены как избранные и имеют определенный диапазон дат обучения. Протокол наследуется от трех других протоколов: `Favored`, `EducationRanged` и `DefaultLessonViewSettings`.

Свойства протокола:

- `lessons`: необязательное свойство, которое представляет занятия, связанные с этой сущностью;
- `title`: обязательное свойство типа `String`, которое представляет заголовок для этой сущности.

Протокол `Scheduled` не имеет собственной реализации и определяет только требования, которые должны быть реализованы классами, которые будут его принимать.

3.3 Модели

Некоторые модели проекта имеют соответствующие таблицы в БД, описание их атрибутов приводиться не будет, а основной акцент сделан на функциональности.

3.3.1 Auditorium

Расширение `Auditorium` для соответствия протоколу `Comparable` позволяет сравнивать объекты типа `Auditorium` между собой. Он принимает два параметра типа `Auditorium` и возвращает значение типа `Bool`, указывающее, меньше ли левый операнд по отношению к правому.

Высчитываемая переменная `groups` добавляет свойство `groups` к объектам типа `Auditorium`. Это свойство возвращает массив `Group`, связанных с аудиторией через занятия (представленные в виде массива `lessons`). Если массив пуст или равен `nil`, то метод возвращает `nil`. Если массив не пуст, то метод компактно отображает его в массив групп, затем объединяет их в один массив и сортирует по возрастанию `name`. В итоге метод возвращает массив `Group`, связанных с аудиторией в отсортированном порядке.

Перечисление `AuditoriumSectionType` представляет тип секций, которые могут быть созданы на основе последовательностей `Auditroium`. Оно наследует протокол `SectionType`, который содержит только свойство `description` для текстового описания типа секции. `AuditoriumSectionType` содержит три случая:

- `building` – для секций, сгруппированных по зданию;
- `buildingAndFloor` – для секций, сгруппированных по зданию и этажу;
- `department` – для секций, сгруппированных по подразделению.

Для представления Auditorium в виде секций существует расширения `Sequence` для создания секций с разной сортировкой.

Метод `sections` принимает значение типа `AuditoriumSectionType` и возвращает массив секций `NSManagedObjectsSection<Auditorium>`. Суть метода заключается в группировке аудиторий по зданиям, этажам и подразделениям.

Метод `buildingBasedSections` принимает значение типа `Bool`, определяющее, должны ли секции группироваться по зданию и этажу или

только по зданию. Затем метод создаёт массив секций NSManagedObjectsSection<Auditorium> на основе группировки аудиторий по зданию и, если floorSectioned равно true, по этажам. Секции возвращаются в порядке возрастания номера здания и этажа.

Метод departmentBasedSections возвращает массив секций NSManagedObjectsSection<Auditorium>, сгруппированных по подразделениям, на основе свойства department каждой аудитории. Секции возвращаются в порядке возрастания аббревиатуры подразделения. Если у аудитории нет подразделения, она попадает в секцию «Без подразделения».

3.3.2 Employee

Вычисляемая переменная formattedName возвращает отформатированное имя преподавателя в виде строки, объединяя имя и отчество, если оно есть.

Вычисляемая переменная departmentsArray возвращает массив всех подразделений, к которым привязан данный преподаватель. Если таких подразделений нет, то возвращает nil.

Вычисляемая переменная departmentsAbbreviations возвращает массив сокращений названий отделов, к которым привязан данный преподаватель. Если таких отделов нет, то возвращает nil.

Печисление EmployeeSectionType, которое реализует протокол SectionType, используется для определения типа секции в списке преподавателей. Перечисление EmployeeSectionType определяет четыре типа секций:

- firstLetter – группирующее преподавателей по первой букве фамилии;
- department – группирующее преподавателей по отделам, с которыми они имеют связи;
- rank – группирующее преподавателей по их рангу;
- degree – группирующее преподавателей по их степени.

Перечисление также реализует вычисляемое свойство description, которое возвращает строковое представление описания типа секции. В зависимости от значения перечисления, вычисляемая переменная description возвращает соответствующее ему описание в виде строки.

Метод sections является расширением для коллекции элементов типа Employee, которое реализует протокол Sequence. Метод принимает аргумент типа EmployeeSectionType, определяющий как будут сгруппированы элементы коллекции.

Для значения аргумента .firstLetter, метод сначала группирует элементы по первой букве фамилии, используя метод sectioned, затем сортирует полученные секции в алфавитном порядке, и конвертирует их в массив объектов типа NSManagedObjectsSection, где каждый объект

представляет одну секцию, содержащую заголовок – символ первой буквы фамилии, ипреподователей, у которых фамилия начинается на эту букву.

Для значения аргумента `.department`, метод вызывает приватный метод `departmentSections`, который группирует элементы по подразделениям, используя массив всех подразделений, отсортированных по их сокращению. Для каждого подразделения создаётся отдельная секция, содержащая заголовок – сокращение названия подразделения, и элементы – работники, которые относятся к этому подразделению. Кроме того, создаётся дополнительная секция для работников, которые не относятся ни к одному подразделению.

Для значения аргумента `.rank`, метод группирует элементы по рангу, используя метод `sectioned`, затем сортирует полученные секции в алфавитном порядке по наименованию должности, и конвертирует их в массив объектов типа `NSManagedObjectsSection`, где каждый объект представляет одну секцию, содержащую заголовок – название ранга, и преподавателей, которые занимают эту должность. Если у преподавателя не указан ранг, он будет отображён в отдельной секции «Без ранга».

Для значения аргумента `.degree`, метод группирует элементы по учёным степеням, используя метод `sectioned`, затем сортирует полученные секции в алфавитном порядке по краткомк названию учёной степени, и конвертирует их в массив объектов типа `NSManagedObjectsSection`, где каждый объект представляет одну секцию, содержащую заголовок – название степени, и преподавателей, которые имеют эту степень. Если у работника не указана учёная степень, он будет отображён в отдельной секции «Без степени».

3.3.3 Group

Вычисляемая переменная `groups` является методом расширения для класса `Employee` и возвращает массив групп, у которых преподаёт данный преподаватель. Вычисляемая переменная использует свойство `lessons` данного преподавателя, чтобы получить все занятия, в которых он участвует. Затем метод использует компактный отображённый массив, чтобы получить все группы, связанные с каждым занятием. Метод объединяет все группы в единый набор (`Set`) и сортирует их по возрастанию `name`. Метод возвращает отсортированный массив `Group` или `nil`, если у преподавателя нет занятий.

Вычисляемая переменная `flow` является методом расширения для класса `Group` и возвращает массив групп того же потока (тех же специальностей и курса), что и данная группа. Вычисляемая переменная использует свойство `speciality` данной группы, чтобы получить все группы в той же специальности. Затем вычисляемая переменная удаляет все группы, которые не соответствуют тому же курсу или не являются данной группой. Метод сортирует массив групп по возрастанию `id` и возвращает его, или `nil`, если нет групп того же потока.

Метод description является методом расширения для массива групп Array<Group> и возвращает строковое описание этого массива. Метод использует сортировку по номеру каждой группы, затем метод объединяет их в строку, используя дефис для групп, которые являются частью последовательности номеров, а для остальных групп используется запятая. Метод возвращает строковое описание массива групп или пустую строку, если массив групп пустой.

Перечисление GroupSectionType, которое реализует протокол SectionType представляет типы секций, используемые для отображения данных в представлении. В перечислении определены четыре типа секций:

- specialityAbbreviation – краткое название специальности;
- specialityName – полное название специальности;
- faculty – факультет;
- flow – поток.

Каждый элемент перечисления имеет вычисляемую переменную description, которое возвращает соответствующую строку для отображения

3.3.4 Lesson

Метод id принимает опциональный параметр sectionID и возвращает строку, которая состоит из значений различных свойств объекта Lesson, таких как краткое название учебной дисциплины abbreviation, время начала timeStart, номер подгруппы subgroup, наименование аудиторий auditoriesNames и идентификаторы преподавателей employeesIDs. Если передан sectionID, то он добавляется в начало строки.

Вычисляемая переменная description возвращает строковое представление объекта Lesson, содержащее сокращение названия предмета, тип занятия, время начала и окончания, список групп, которые проходят занятие, и идентификаторы преподавателей.

Метод filtered(abbreviation:) принимает строку abbreviation и возвращает новую последовательность Lesson, содержащую только те элементы, у которых свойство abbreviation содержит переданную строку.

Метод filtered(subgroup:) принимает опциональный параметр subgroup типа Int и возвращает новую последовательность Lesson, содержащую только те элементы, у которых свойство subgroup совпадает с переданным параметром или равно 0, то есть общие занятия.

Вычисляемая переменнаяdate возвращает объект Date, полученный из строки dateString объекта Lesson с помощью форматирования, заданного в DateFormatters.short.

Вычисляемая переменная dateRange возвращает диапазон дат объекта Lesson, который задается через свойства startLessonDate и endLessonDate.

Вычисляемая переменная timeRange возвращает диапазон времени объекта Lesson, который задаётся через свойства timeStart и timeEnd.

Вычисляемая переменная weeksDescription возвращает строку, содержащую описание недель, в которые проходит занятие. Если свойство `weeks` объекта `Lesson` пустое, то метод возвращает `nil`. Если в свойстве `weeks` находится только одно значение, то метод возвращает его, увеличенное на 1. В противном случае метод разбивает список недель на группы, состоящие из последовательных значений, и возвращает строку, содержащую диапазоны недель.

3.3.5 LessonViewConfiguration

Класс `LessonViewConfiguration` представляет конфигурацию представления занятия. Он содержит следующие свойства:

- `showFullSubject`: флаг, указывающий, нужно ли отображать полное название учебной дисциплины.
- `showGroups`: флаг, указывающий, нужно ли отображать группы, связанные с занятием.
- `showEmployees`: флаг, указывающий, нужно ли отображать преподавателей, связанных с занятием.
- `showWeeks`: флаг, указывающий, нужно ли отображать номера недель.
- `showDates`: флаг, указывающий, нужно ли отображать даты.
- `showDate`: флаг, указывающий, нужно ли отображать дату.

Инициализатор класса принимает значения по умолчанию для некоторых свойств, чтобы облегчить создание экземпляра класса с помощью значения по умолчанию.

Класс `LessonViewConfiguration` также подписан на протокол `ObservableObject`, чтобы быть наблюдаемым в SwiftUI, и наследует его функциональность.

Расширение класса `LessonViewConfiguration` реализует протокол `Equatable`, чтобы обеспечить возможность сравнения двух экземпляров класса.

Протокол `DefaultLessonViewSettings` определяет метод `defaultLessonSettings`, которая должна быть реализована всеми классами, которые имеют настройки представления занятия по умолчанию.

Расширения классов `Auditorium`, `Employee` и `Group` реализуют протокол `DefaultLessonViewSettings`, предоставляя значения по умолчанию для настроек представления занятия, основанных на типе группы/преподавателя/аудитории.

3.3.6 Speciliaty

Метод `formattedID` формирует идентификатор специальности, который состоит из краткого названия специальности, краткого названия факультета, идентификатора типа получения образования и уникального идентификатора

специальности. Возвращает строку с сформированным идентификатором.

Метод `formattedDescription` Метод формирует описание специальности, которое состоит из названия специальности, типа обучения и краткого названия факультета. При необходимости, метод может использовать сокращённое название специальности вместо полного названия. Метод принимает значение `useSpecialityName` типа `Bool`, который указывает, следует ли использовать полное название специальности или сокращённое. Метод возвращает строку с сформированным описанием.

3.3.7 ScheduleSection

Класс `ScheduleSection` предназначен для представления отдельной секции расписания.

Метод `init` инициализирует объект класса, принимая значения типа `ScheduleSectionType`, даты, номера недели и дня недели, а также последовательность `Lesson`, и упорядочивает занятия по подгруппам и времени начала. Также устанавливает свойства `title` и `today`, определяет относительность секции по отношению к текущей дате и времени, и запускает таймер для обновления свойства `title`, если секция относится к текущему дню.

Метод `addTimerCancellable` создаёт таймер, который будет каждую секунду вызывать метод `updateTitle`, а также проверять относительность секции по отношению к текущей дате и времени.

Метод `checkRelativity`: вычисляет относительность секции по отношению к текущей дате и времени в днях или в днях и часах, в зависимости от типа секции.

Метод `updateTitle` обновляет свойство "title" секции в зависимости от текущего времени и относительности секции по отношению к текущей дате и времени.

Метод `closestLesson`: возвращает ближайшее занятие из списка занятий, если такой есть.

Метод `isToday` возвращает `Bool`, который отписывает, относится ли секция к сегодняшнему дню.

Метод `todayTitle` возвращает строку, описывающую текущее время относительно ближайшего занятия, если такой есть.

Метод `dateTitle` возвращает заголовок, предназначенный дня секцией отсортированных по неделям.

Метод `weekTitle` возвращает заголовок, предназначенный дня секцией отсортированных по неделям.

Свойство `title` возвращает название секции, которое зависит от текущего времени и относительности секции по отношению к текущей дате и времени.

Вычисляемое свойство `today` типа `Bool`, которое устанавливается в

истинное значение, если секция относится к текущему дню.

ScheduleSectionType, которое советует типу SectionType, определяет два возможных значения: .date и .week.

Также имеется ряд связанных с ScheduleSection расширений.

Первое расширение добавляет два протокола: Identifiable и Equatable – к типу ScheduleSection. Протокол Identifiable гарантирует наличие у ScheduleSection идентификатора id, возвращающего строку, это используется для идентификации элементов в SwiftUI. Протокол Equatable гарантирует, что ScheduleSection может быть сравнен с другим ScheduleSection.

Второе расширение добавляет свойство описания description, которое возвращает строку, описывающую ScheduleSection и все его занятия.

Третье расширение добавляет функцию isToday(), которая определяет, является ли секция сегодняшней. Если секция имеет тип .date, функция возвращает true, если дата секции является сегодняшней датой. Если секция имеет тип .week, функция возвращает true, если неделя и день секции совпадают с текущей неделей и днём.

Четвёртое расширение добавляет функцию sections к типу Sequence, который содержит элементы типа Lesson. Функция принимает тип ScheduleSectionType и массив дат educationDates и возвращает массив ScheduleSection. Если тип ScheduleSectionType равен .date, функция создает ScheduleSection для каждой даты в educationDates. Если тип равен .week, функция создает ScheduleSection для каждой недели и каждого дня недели. Функции dateSections() и weekSections() создают ScheduleSection для соответствующих типов.

Пятое расширение добавляет функцию filtered к типу Sequence, который содержит элементы типа ScheduleSection. Функция фильтрует занятия в каждой секции с помощью метода filtered (abbreviation :) и filtered (subgroup :) у типа Lesson. Затем функция возвращает новый массив ScheduleSection, содержащий только отфильтрованные занятия.

Шестое расширение добавляет функцию closest к типу Array, который содержит элементы типа ScheduleSection. Функция принимает дату и тип ScheduleSectionType и возвращает ближайшую ScheduleSection к заданной дате. Если тип ScheduleSectionType равен .date, функция находит первую секцию, которая имеет дату не меньше, чем заданная дата. Если секция является сегодняшней, функция возвращает секцию только в том случае, если в ней есть ближайшее занятие. Если тип равен .week, функция находит первую секцию, которая имеет учебную неделю и день недели большей либо равный сегодняшней и имеет ближайшее занятие.

3.4 Модели представлений

Чаще всего модели представлений обладают малым количеством

функций, которые обеспечивают преобразование данных для представления.

3.4.1 AuditoriumViewModel

Класс `AuditoriesViewModel` соответствует протоколу `ObservableObject`, который отвечает за управление данными в представлении для отображения списка аудиторий. Класс имеет несколько свойств, которые публикуются `@Published`, чтобы сигнализировать об изменении свойствам, которые используются в представлении SwiftUI.

Свойства класса:

- `predicate`: свойство типа `NSPredicate`, которое используется для фильтрации списка аудиторий в соответствии с выбранным типом аудитории и поисковым запросом;
- `searchText`: свойство типа `String`, которое содержит текст для поиска в списке аудиторий;
- `selectedSectionType`: свойство типа `AuditoriumSectionType`, которое определяет, выбран ли раздел здания или тип аудитории;
- `selectedAuditoriumType`: свойство типа `AuditoriumType`, которое содержит выбранный тип аудитории;
- `cancellables`: свойство типа `Set<AnyCancellable>`, которое используется для хранения подписок, которые необходимо отменить, когда объект больше не нужен.

Класс содержит два приватных метода `addSearchTextPublisher` и `addSelectedAuditoriumTypePublisher`, которые используют `Combine` для отслеживания изменений в свойствах `searchText` и `selectedAuditoriumType` соответственно. Когда происходят изменения, методы вызывают метод `calculatePredicate`.

Метод `calculatePredicate` конструирует предикат для фильтрации списка аудиторий на основе выбранного типа аудитории и поискового запроса. Этот метод создаёт список предикатов и объединяет их оператором логическй конъюнкции, чтобы создать окончательный предикат для фильтрации.

Класс также содержит статический метод `update`, который обновляет список всех аудиторий. Этот метод использует `async` и `await` для асинхронного получения данных о всех аудиториях из API.

3.4.2 DepartmentsViewModel

Класс `DepartmentsViewModel` представляет модель данных для отображения списка отделов в пользовательском интерфейсе. В данном классе используется подписка на изменения переменной `searchText`, чтобы автоматически фильтровать список отделов при изменении поискового запроса.

Описание свойств класса:

- predicate: свойство типа NSPredicate, которое определяет предикат для фильтрации списка отделов. По умолчанию, это значение равно nil;

- searchText: свойство типа String, которое хранит поисковый запрос;

- cancellables: свойство типа Set<AnyCancellable>, которое хранит все подписки на изменения.

Метод addSearchTextPublisher добавляет подписку на изменение переменной searchText. При изменении переменной searchText, вызывается метод calculatePredicate.

Метод calculatePredicate вычисляет предикат для фильтрации списка отделов на основе значения переменной searchText. Если значение searchText пусто, предикат равен nil. В противном случае, создается предикат, который фильтрует список подразделений по полному и краткому названиям.

Статический метод update загружает данные из API и обновляет список отделов.

3.4.3 EmployeesViewModel

Класс EmployeesViewModel представляет модель представления для списка преподавателей. Он соответствует протоколу ObservableObject, который отвечает за управление данными в представлении для отображения списка аудиторий. Класс позволяет фильтровать список преподавателей по фамилии, имени, названию отдела или его аббревиатуре, а также выбирать тип секции для разделения списка.

Основные свойства класса:

- predicate: предикат для фильтрации списка преподавателей. При изменении фильтра вызывается метод calculatePredicate;

- searchText: текст поискового запроса для фильтрации списка преподавателей;

- selectedSectionType: тип секции, выбранный пользователем для разделения списка преподавателей;

- showDepartments: флаг, указывающий, должны ли отображаться отделы преподавателей;

- scrollTargetID: идентификатор преподавателя, к которому нужно прокрутить список при отображении.

Метод calculatePredicate используется для вычисления предиката фильтрации списка преподавателей. Предикат строится на основе текста поискового запроса и включает фильтрацию по фамилии, имени, названию отдела или его аббревиатуре. Если текст поискового запроса пуст, метод устанавливает predicate в nil.

Метод update обновляет список преподавателей из API и вызывает метод updateEmployees для обновления данных преподавателей.

Приватный метод addSearchTextPublisher предназначен для

добавления подписчика на изменения текста поискового запроса. Подписчик использует оператор `debounce` для того, чтобы задержать обработку изменений на 0,5 секунды, что оптимизирует работу алгоритма, так как изменения предиката происходит только через 0,5 секунды после окончания изменения свойства `searchText`, а не после ввода каждого нового символа.

3.4.4 EducationTaskDetailedViewModel

Класс `EducationTaskDetailedViewModel` является моделью представления для детального представления о задании. Он наследуется от `ObservableObject`, чтобы обновлять представление при изменении значений свойств, отмеченных как `@Published`. Переменная `educationTask` типа `EducationTask` является `@Published`, так как изменения в этом свойстве должны обновлять пользовательский интерфейс, использующий этот объект `EducationTaskDetailedViewModel`.

Переменная `backgroundContext` типа `NSManagedObjectContext` используется для работы с CoreData.

Переменная `withDeadline` и `deadline` используются для хранения информации о наличии или отсутствии срока выполнения для данной образовательной задачи.

Переменная `noteText` предназначена для хранения текстового описания образовательной задачи.

Переменная `photosPickerItems` и `imagesData` используются для хранения информации о фотографиях, прикреплённых к задаче.

Переменная `lessons` содержит массив объектов типа `Lesson` для данной задачи.

Переменная `deadlineDates` используется для хранения дат и времени для каждого срока выполнения, вычисленного на основе расписания занятий для заданных занятий.

Метод `addPhotosPickerItemsSubscriber` используется для добавления подписчика на свойство `photosPickerItems` и сохранения фотографий в `imagesData`.

Метод `createDeadlineDates` используется для вычисления сроков выполнения на основе расписания занятий для данной образовательной задачи.

Метод save сохраняет все изменения в CoreData, а именно изменения в переменных `educationTask`, `withDeadline`, `deadline`, `noteText`, `imagesData`.

Класс использует асинхронный подход и сопрограммы `Task` для выполнения длительных операций, таких как загрузка изображений или вычисление дат дедлайнов. Кроме того, он использует `Combine` для создания подписок на изменение данных и выполнения более эффективных асинхронных операций.

3.4.5 EducationTaskViewModel

Класс `EducationTaskViewModel` представляет модель представления задачи обучения, которая отвечает за отображение информации о дате сдачи задачи. Класс содержит следующие свойства:

- `deadline`: дата срока выполнения задачи;
- `deadlineString` – строковое представление относительного срока выполнения задачи, имеет атрибут `@Published`;
- `timerCancellable`: объект, используемый для хранения и отмены таймера.

Метод `addTimerCancellable` создаёт таймер, который будет обновлять `deadlineString` каждую секунду. Он также сохраняет этот таймер в свойство `timerCancellable`.

Метод `createDeadlineString` создаёт строковое представление даты срока выполнения задачи с помощью форматирования и возвращает это значение.

Когда таймер обновляет `deadlineString`, метод `withAnimation` используется для создания плавного эффекта анимации изменения текста в представлении экране.

3.4.6 GroupDetailedViewModel

Класс `GroupViewModel` отвечает за управление данными группы, включая, а именно её псевдонимом.

Свойство `nicknameString`, содержит псевдоним группы, отображаемый пользователю. При изменении этого свойства автоматически обновляется пользовательский интерфейс, так как оно имеет атрибут `@Published`.

Метод `submitNickname` принимает экземпляр `Group` и устанавливает ей псевдоним. Если `nicknameString` пуста, то псевдоним группы удаляется, то есть выставляется в nil. Метод использует CoreData для сохранения изменений и работает в фоновом контексте.

Когда пользователь изменяет псевдоним группы, `nicknameString` обновляется, и пользовательский интерфейс автоматически отражает это изменение. Затем пользователь может вызвать метод `submitNickname`, чтобы сохранить этот псевдоним в БД. Если псевдоним изменяется успешно, `submitNickname` сохраняет изменения в фоновом потоке и обновляет модель данных группы. В результате изменения пользовательского интерфейса группы сохраняются и переживают перезапуск приложения.

3.4.7 GroupsViewModel

Класс `GroupsViewModel` представляет модель представления для списка групп. Диаграмма последовательностей для представления списка групп

представлена на чертеже ГУИР.400201.019 РР.3. Свойства класса:

- predicate: определяет условие выборки групп, если значение не задано, будут выбраны все группы, @Published используется для автоматического уведомления подписчиков об изменении значения;
- searchText: текст, введённый пользователем для поиска группы. @Published используется для автоматического уведомления подписчиков об изменении значения;
- selectedSectionType: тип секции, выбранной пользователем, группировка групп по критерию;
- selectedFaculty: факультет, выбранный пользователем. nil – значение по умолчанию;
- selectedSpecialtyEducationType: тип образования специальности, выбранный пользователем. nil – значение по умолчанию;
- selectedEducationDegree: степень образования, выбранная пользователем. nil – значение по умолчанию;
- selectedCourse: курс, выбранный пользователем. nil – значение по умолчанию;
- cancellables: набор AnyCancellable для отмены подписок при удалении объекта.

Метод addSearchTextPublisher добавляет подписку на изменение свойства searchText и вызывает метод calculatePredicate, если значение изменилось.

Метод addSelectionSubscribers добавляет подписку на изменение свойств selectedFaculty, selectedSpecialtyEducationType, selectedEducationDegree и selectedCourse и вызывает метод calculatePredicate, если значение изменилось.

Метод calculatePredicate вычисляет NSPredicate на основе выбранных пользователем значений свойств и текста поиска. Если значение searchText пусто, используются только выбранные пользователем значения. Если выбранных значений нет, то отображаются все группы.

Метод update обновляет данные групп, он является асинхронным.

В инициализаторе вызываются методы addSearchTextPublisher и addSelectionSubscribers.

3.4.8 ScheduleViewModel

Класс ScheduleViewModel представляет модель представления расписания и используется для управления данными и состоянием связанными с отображением расписания. Диаграмма последовательностей для представления расписания занятий представлена на чертеже ГУИР.400201.019 РР.4. Свойства класса:

- selectedRepresentationType: тип представления расписания (ScheduleRepresentationType), определяет выбранный тип представления

расписания, такой как прокрутка или страничный;

- selectedSectionType: тип секции расписания (ScheduleSectionType). определяет выбранный тип секции расписания;
- showScrollView: флаг, определяющий видимость ScrollView;
- selectedSectionID: идентификатор выбранного раздела расписания;
- scrollWithAnimation: флаг, определяющий анимацию прокрутки;
- searchText: текст для поиска в разделах расписания;
- showSearchField: флаг, определяющий видимость поля поиска;
- selectedDate: выбранная дата;
- showDatePicker: флаг, определяющий видимость выбора даты;
- selectedSubgroup: выбранная подгруппа;
- selectedHometaskLesson: выбранное занятие для задания;
- sections: массив разделов расписания (ScheduleSection);
- filteredSections: отфильтрованный массив секций расписания;
- cancellables: набор отменяемых подписок (AnyCancellable).

Методы addSelectedRepresentationType добавляет подписчика на изменение выбранного типа представления расписания. При изменении типа представления вызывается метод updateFilteredSections для обновления отфильтрованных разделов расписания.

Метод addSelectedSubgroupSubscriber добавляет подписчика на изменение выбранной подгруппы. При изменении подгруппы вызывается метод updateFilteredSections для обновления отфильтрованных разделов расписания.

Метод addSearchTextSubscriberдобавляет подписчика на изменение текста поиска. При изменении текста поиска вызывается метод updateFilteredSections для обновления отфильтрованных разделов расписания.

Метод addSelectedDateSubscriberдобавляет подписчика на изменение выбранной даты. При изменении даты вызывается метод scrollToDate для прокрутки до соответствующей секции расписания.

Метод updateSections обновляет разделы расписания на основе переданных уроков и дат образования. Он также вызывает метод updateFilteredSections для обновления отфильтрованных разделов расписания.

Метод updateFilteredSections обновляет отфильтрованные разделы расписания на основе текущих параметров фильтрации (текст поиска и выбранная подгруппа). Если параметр returnToClosestSection установлен в true, то метод также вызывает метод initialScroll для прокрутки к ближайшему разделу расписания.

Метод initialScrollвыполняет начальную прокрутку к ближайшему разделу расписания в зависимости от выбранного типа представления. Затем устанавливает выбранный раздел и флаг showScrollView в соответствии с выбранным типом представления.

Метод scrollToDate выполняет прокрутку к разделу расписания, соответствующему переданной дате. Параметр scrollWithAnimation

указывает, следует ли использовать анимацию при прокрутке.

Метод `scrollTo` выполняет прокрутку до указанного раздела в `ScrollViewProxy`. Если идентификатор раздела не пустой, то метод прокручивает к указанному разделу. Если флаг `scrollWithAnimation` установлен в `true`, то прокрутка выполняется с анимацией.

Вычисляемое свойство `defaultRules` возвращает значение типа `Bool`, которое зависит от того, находятся ли параметры отображения в состоянии по умолчанию.

3.4.9 PrimaryScheduleViewModel

Класс `PrimaryScheduleViewModel` является моделью представления для отображения ближайшего расписания. В качестве типа для расписания может быть использован любой тип, реализующий протокол `Scheduled`.

- `scheduled` содержит объект с расписанием, которое будет отображаться в данной модели представления;
- `subgroup` содержит номер подгруппы для расписания, если он имеется.
- `title` содержит заголовок для отображаемого расписания, который будет обновляться при изменении данных;
- `section` содержит объект раздела расписания, который ближе всего по времени;
- `lesson` содержит занятие, которое ближе всего по времени;
- `state` используется для отслеживания состояния данной модели представления.

Свойство `state` может принимать следующие значения:

- `updating` отображает `ProgressView` до момента окончания обновления;
- `showLesson` отображает ближайшее занятие;
- `noClosestSection` отображает сообщение о том, что больше занятий для данного объекта в этом семестре нет.

Метод `init` инициализирует модель представления. Он получает расписание, которое будет отображаться, и опциональный номер подгруппы, который используется только для типа `Group`. При инициализации, метод `update` вызывается асинхронно для обновления данных. Если расписание реализует протокол `LessonsRefreshable`, то вызывается метод `checkForLessonsUpdates` для обновления занятий. Также создается таймер, который вызывает метод `addTimerCancellable` каждую секунду.

Метод `addTimerCancellable` создаёт таймер, который обновляет данные в модели представления. Внутри метода есть логика для обновления заголовка расписания и текущего занятия.

Метод `update` обновляет данные модели представления. Он получает объекты занятий из расписания и фильтрует их по номеру подгруппы (если имеется). Затем, используя метод `closest` из расширения массива,

содержащего `ScheduleSection`, находится ближайший объект раздела расписания к текущему времени. Если ближайшего раздела нет, то меняется состояние на `noClosestSection`. Если есть ближайший раздел, то обновляется заголовок расписания и меняется состояние на `showLesson`. Метод также вызывает метод `updateTitle` для обновления заголовка, если необходимо.

Метод `updateTitle` обновляет заголовок для отображаемого расписания. Он получает заголовок текущего раздела и обновляет заголовок расписания, добавляя его к текущему заголовку. Если заголовок не изменялся, то метод не выполняет никаких действий.

3.5 Блок извлечения

Блок извлечения реализован путём создания расширений для стандартного класса `URLSession`.

3.5.1 FetchDataType

`FetchDataType` является перечислением, которое используется в проекте для определения URL-адресов API ИИС БГУИР для получения различных данных.

В этом перечислении определены различные случаи, каждый из которых представляет собой URL-адрес для конкретного типа данных:

- `faculties` – URL-адрес для получения списка факультетов;
- `specialities` – URL-адрес для получения списка специальностей;
- `auditories` – URL-адрес для получения списка аудиторий;
- `departments` – URL-адрес для получения списка кафедр;
- `groups` – URL-адрес для получения списка групп;
- `group` – URL-адрес для получения расписания для определенной группы;
- `groupUpdateDate` – URL-адрес для получения даты последнего обновления расписания для определенной группы;
- `employees` – URL-адрес для получения списка преподавателей;
- `employee` – URL-адрес для получения расписания для определенного преподавателя;
- `employeeUpdateDate` – URL-адрес для получения даты последнего обновления расписания для определенного преподавателя.

Перечисление использует тип `String` для хранения URL-адреса. В коде он используется для создания URL-объектов, которые используются для отправки запросов на сервер. Каждый случай перечисления имеет свою собственную строку, которая представляет URL-адрес для конкретного типа данных.

3.5.2 Функции для извлечения данных их API ИИС БГУИР

Функция `data(from:) async throws -> Data` используется для получения данных с определенного URL-адреса. Она принимает один параметр: URL-адрес в виде строки.

Функция проверяет, может ли URL быть создан из заданной строки. Если URL не может быть создан, функция генерирует ошибку `URLError(.badURL)`. Затем функция использует стандартный метод `data(from:)` для получения данных из сети. При успешном получении данных функция возвращает полученные данные в формате `Data`.

Функция `data(for dataType: FetchDataType, with argument: String? = "") async throws -> Data?` принимает тип запрашиваемых данных, который описан в `enum FetchDataType`, и необязательный аргумент (название группы или идентификатор преподавателя).

Функция использует эти параметры для создания URL-адреса, по которому будет выполнен запрос на сервер. Функция выполняет запрос на сервер с использованием метода `await data(from url: URL)` и возвращает данные в виде объекта Data. Если запрос произошёл успешно, то функция возвращает запрошенные данные. Если возникла ошибка (например, не удалось создать URL-адрес или сервер вернул пустой ответ), то функция генерирует ошибку типа `URLError` и возвращает `nil`.

Обе функции используют `async` и `await` для асинхронного получения данных из сети. Это означает, что при вызове этих функций процесс выполнения не блокируется, а продолжает выполнять другие задачи, пока данные не будут получены.

3.5.3 Извлечение аудитории

Описанные выше функции являются универсальными и служат только для извлечения абстрактных данных из API. Извлечение и обработка конкретных данных производится в расширениях моделей. Ниже приведены необходимые функции для извлечения и обновления преподавателя из API.

Расширение для класса `Auditorium` в приложении, которое определяет функцию `fetchAll`, которая загружает данные о всем доступных аудиториях асинхронно.

В начале функции выполняется запрос к серверу для загрузки данных с помощью `URLSession.shared.data`. Если запрос не удалось выполнить или данные не получены, то функция возвращает управление.

Затем функция извлекает полученные данные с помощью `JSONSerialization.jsonObject` и конвертирует их в массив словарей. Если произойдёт ошибка извлечения или не удастся создать словари, функция выведет ошибку с помощью `Log.error` и завершает свою работу.

Затем функция создаёт новый фоновый контекст CoreData в режиме с

помощью и настраивает политику слияния объектов в контексте на NSMergeByPropertyStoreTrumpMergePolicy, чтобы обеспечить слияние объектов с приоритетом данных из фонового контекста при сохранении в постоянное хранилище.

Затем метод инициализирует экземпляр JSONDecoder и устанавливает свойство userInfo на контекст CoreData. Это свойство используется при декодировании JSON объектов, чтобы связать объекты с соответствующим контекстом CoreData.

Далее функция вызывает статический метод getAll, который возвращает все существующие объекты класса Auditorium из заданного контекста CoreData. Этот метод будет использоваться позже, чтобы искать объекты аудиторий в базе данных.

Затем функция итерируется по всем словарям, которые были получены из ответа на запрос к API, и конвертирует их в объекты Auditorium CoreData. Если аудитория уже существует в базе данных, функция вызовет метод update на экземпляре декодера, чтобы обновить данные объекта из словаря. Если аудитория не найдена в базе данных, то функция создаст новый объект Auditorium из словаря, используя метод decode.

В конце функция сохраняет все изменения в базе данных с помощью backgroundContext.save(). Затем функция выводит количество загруженных аудиторий с помощью Log.info и время, которое потребовалось для загрузки и обработки данных.

3.5.4 Извлечение подразделения, факультета и специальности

Расширение классов Department, Faculty и Speciliaty реализует аналогичные методы fetchAll, которые являются статическими и асинхронным, не принимает параметров и не возвращает значения.

Если данные, полученные с помощью метода URLSession.shared.data по адресу, переданному в качестве аргумента, отсутствуют, то функция завершается.

Затем получается текущее время и создаётся пара backgroundContext, JSONDecoder.

Выполняется декодирование полученных данных в массив объектов советующего типа с помощью объекта JSONDecoder. Если процесс декодирования завершается неудачно, в лог выводится сообщение об ошибке и функция завершается.

После этого изменения сохраняются в backgroundContext. Создаются логи с количеством полученных объектов и временем, затраченным на выполнение операции. Такие логи не отображаются пользователю, однако могут быть полезны для того, чтобы отслеживать медленные действия и проблемы при разработке.

3.5.5 Извлечение преподавателя

Функция `static func fetchAll() async` используется для асинхронной загрузки данных о преподавателях из внешнего источника.

Функция начинается с загрузки данных с помощью `URLSession.shared.data(from:)`, который был описан выше. Затем данные преобразуются из формата JSON в массив словарей `[[String: Any]` с помощью `JSONSerialization.jsonObject`.

Далее функция создает новый фоновый контекст CoreData с помощью `newBackgroundContext()` и задает политику слияния `NSMergeByPropertyStoreTrumpMergePolicy`, которая обеспечивает слияние объектов с приоритетом данных из контекста при сохранении в постоянное хранилище.

Затем используется `JSONDecoder`, который настроен на работу с CoreData. Информация о контексте `userInfo[.managedObjectContext]` и наличии вложенного контейнера групп `userInfo[.groupEmbeddedContainer]` используется для декодирования JSON-объектов в объекты модели CoreData.

Далее функция вызывает метод `getAll`, который возвращает все объекты модели `Employee` из указанного контекста CoreData. Результат сохраняется в массив `employees`.

Затем функция итерируется по полученному массиву словарей и декодирует каждый словарь в соответствующий объект `Employee`. Если объект с таким же `id` уже существует в массиве `employees`, то происходит обновление объекта из данных существующего объекта с помощью `JSONDecoder.update(_:from:)`. Если же объекта с таким `id` нет, то происходит добавление нового объекта в массив `employees`.

В конце функция сохраняет контекст CoreData с помощью `backgroundContext.perform(schedule: .immediate)`, чтобы сохранение происходило в фоне. В лог также выводится количество загруженных преподавателей и время выполнения функции.

3.5.6 Извлечение группы

Расширение для класса `Group` добавляет метод `fetchAll`, который получает данные из ответа API и обрабатывает их с помощью JSON-сериализации. Затем он обновляет существующие объекты `Group` или создаёт новые и сохраняет их в базе данных.

Метод начинается с проверки доступности данных с помощью `URLSession.shared.data`. Затем данные десериализуются с использованием `JSONSerialization`. Если десериализация прошла успешно, создается новый фоновый контекст CoreData и устанавливается политика слияния объектов. Далее используется декодер `JSONDecoder`, который настраивается на использование контекста и опционального контейнера для вложенных

объектов типа `Group`. Затем получаются все имеющиеся в базе данных объекты Group.

Затем каждый словарь данных в полученном массиве обрабатывается по следующим правилам: если объект Group с таким же именем уже существует в базе данных, то он обновляется с помощью метода `update`, иначе создается новый объект `Group` и сохраняется в базе данных.

После того, как все объекты обработаны и сохранены в базе данных, метод `fetchAll` заканчивается, выводя количество полученных и сохранённых групп и время выполнения операции.

Существует также расширение `Group` для обновления объектов в базе данных. Расширение добавляет два метода: `update` и `updateGroups`.

Метод `update` получает данные о группе из сети и обновляет соответствующий объект `Group` в базе данных. Первым шагом метод проверяет наличие данных о группе в API. Если данные доступны, они десериализуются, и создается новый фоновый контекст CoreData. Затем декодер `JSONDecoder` настраивается на использование контекста и опционального контейнера для вложенных объектов типа `Group`. Затем метод получает объект `Group` из базы данных, используя его `objectID`, и обновляет его с помощью метода `update`.

Метод `updateGroups` обновляет все объекты `Group` в переданном в метод массиве. Для этого он создаёт группу асинхронных задач, каждая из которых также асинхронно вызывает метод `update` для соответствующего объекта `Group`.

3.6 Блок декодирования

Блок декодирования использует стандартные функции и протоколы `JSONDecoder`, `Decodable` и `CodingKeys`, для которых написаны расширения.

3.6.1 Расширения стандартных библиотек

Расширения `JSONDecoder` и `PropertyListDecoder` расширяют протокол `DecodingFormat`. Эти расширения определяют метод decoder(for data: `Data`), который возвращает объект `Decoder` для соответствующего формата данных. Этот метод используется при декодировании объектов, чтобы создать необходимый декодер.

Далее определен протокол `DecoderUpdatable`, который имеет метод `update(from decoder: Decoder)`, который обновляет значения из переданного декодера.

Протокол `DecodingFormat`, который определяет метод `decoder(for data: Data)`, который создает и возвращает объект декодера для соответствующего формата данных, и метод `decode<T: Decodable>(_ type: T.Type, from data: Data)`, который декодирует объект типа `T` из

переданных данных.

Два класса `NestedSingleValueDecodingContainer` и `NestedDecoder` определяют контейнеры декодирования для вложенных объектов в `KeyedDecodingContainer`. `NestedSingleValueDecodingContainer` — это контейнер, который декодирует единственное значение, а `NestedDecoder` — контейнер, который декодирует вложенные объекты.

Расширение `KeyedDecodingContainer` определяет метод `update<T: DecoderUpdatable>(_ value: inout T, forKey key: Key, userInfo: [CodingUserInfoKey : Any] = [:])`, который использует NestedDecoder для обновления значений объекта `T` из ключа `key` в `KeyedDecodingContainer`.

3.6.2 Декодирование аудитории

Декодирование сущности `AuditoriumType` выполняется с использованием `JSONDecoder` и инициализации через него. Внутри метода происходит декодирование значений из контейнера с помощью ключей, заданных в перечислении `CodingKeys`. Контейнер получается из параметра `decoder`.

Значения для свойств `id`, `name` и `abbreviation` декодируются с помощью метода `decode(_:forKey:)`, который вызывается на контейнере. Типы, которые декодируются, указываются в квадратных скобках. Для свойства `id` это `Int16`, а для `name` и `abbreviation` это `String`.

После того, как все значения декодированы, создается новый экземпляр `AuditoriumType` в контексте управляемого объекта, который получается из `userInfo` параметра декодера. Значения свойств объекта заполняются декодированными значениями.

После создания объекта, вызывается метод `info` из класса `Log` для записи информации о созданном экземпляре `AuditoriumType` в логах.

3.6.3 Декодирование типа аудитории

Декодирование сущности `Auditorium` выполняется при помощи протокола `Decoder`.

Сущность `Auditorium` имеет метод `init`, который и выполняет декодирование. Метод получает контейнер со значениями, распаковывает его и сохраняет необходимые данные в свойствах сущности.

Для декодирования используются `keyedBy` контейнеры. В методе `init` создаются два таких контейнера: `container` и `buildingContainer`.

Первый контейнер хранит значения для свойств `name`, `capacity`, `note`, `type`, `department`. Второй контейнер, `buildingContainer`, используется для декодирования номера здания, в котором расположена аудитория.

Внутри метода `init` вызываются два метода декодирования: `decodeBuilding` и `decodeName`. В первом методе декодируется номер здания,

а во втором – номер и название аудитории.

При декодировании номера здания выполняется проверка на то, что здание является учебным. Если номер здания не может быть преобразован в `Int16`, то создаётся исключение типа `AuditoriumDecodingError.nonEducationalBuilding`.

Если номер здания является соответствующим одному из корпусов, то он сохраняется в свойстве `building`.

При декодировании номера и названия аудитории выполняется проверка на то, находится ли аудитория в университете. Если первый символ строки с номером и названием аудитории не является цифрой, то аудитория расположена за пределами университета.

Если аудитория расположена в университете, то номер и название аудитории разделяются. Номер преобразуется в `Int16` и сохраняется в свойстве `floor`. Название аудитории сохраняется в свойстве `name`.

После декодирования номера здания, номера и названия аудитории, а также остальных свойств, вызывается метод `formateName`, который форматирует номер и название аудитории в соответствии с определенными правилами и сохраняет полученное значение в свойстве `formattedName`.

3.6.4 Декодирование учёной степени преподавателя

Декодирование сущности `Degree` выполняется путём реализации протокола `Decodable` и переопределения метода `init`. Передаваемый в метод параметр `decoder` содержит закодированные данные, которые необходимо декодировать.

В начале метода происходит проверка наличия ключа `degree` и его значения в декодируемых данных. Если это условие не выполнено, создаётся исключение. Далее, в методе определяется контекст из передаваемых данных, создаётся объект `Degree` с помощью метода `init(context: NSManagedObjectContext)` и вызывается метод обновления свойств сущности `update`, который описан далее.

В методе `update` из контейнера данных достаются значения ключей `degree` и `degreeAbbreviation`. Если значение `degreeAbbreviation` не пустое, то свойствам `name` и `abbreviation` объекта присваиваются соответствующие значения. В противном случае, только свойству `abbreviation` объекта присваивается значение из `degree`, что связанно с особенностями, описанными в подразделах 1.1.6 и 3.1.4.

3.6.5 Декодирование подразделения

Декодирование сущности `Department` выполняется при помощи `JSONDecoder` и инициализации через него.

- свойство `id` получает значение из ключа «id» контейнера, если ключ

присутствует, иначе – из ключа «idDepartment»;

- свойство `name` получает значение из ключа «name» контейнера;
- свойство `abbreviation` получает значение из ключа «abbrev» контейнера и затем передаётся в метод `formattedAbbreviationString`, который приводит его к нужному формату.

Если контейнер не содержит всех необходимых ключей, то создаётся ошибка `DepartmentDecodingError.noKeys`.

В итоге, при успешном декодировании, создаётся экземпляр класса `Department` с установленными свойствами `id`, `name`, `abbreviation` и выводится сообщение об успешном создании экземпляра в лог. Также класс `Department` имеет дополнительный конструктор `init(from string: String, in context: NSManagedObjectContext)`, который позволяет создавать экземпляры класса из строки и контекста базы данных, то есть без использования `Decoder`.

3.6.6 Декодирование типа получения образования

Декодирование сущности `EducationType` выполняется в методе `init`. В начале инициализируется новый экземпляр `EducationType` в контексте управляемого объекта CoreData, полученного из параметра `decoder.userInfo`.

Далее, с использованием контейнера, полученного из `decoder`, происходит декодирование значений свойств объекта. Используя ключи, определенные в `enum CodingKeys`, выполняется декодирование id и name из контейнера.

Значение для свойства `id` декодируется с помощью метода `decode`, вызванного на контейнере, и ожидаемого типа `Int16`. Значение для свойства `name` также декодируется с помощью метода `decode`, и ожидается тип `String`. Затем, полученное значение для свойства `name` преобразуется, используя метод `capitalized`, для того чтобы первая буква была заглавной, а остальные – строчными.

После успешного декодирования значений свойств, вызывается метод `info` класса `Log`, чтобы записать информацию о созданном экземпляре `EducationType` в логах.

3.6.7 Декодирование преподавателя

Декодирование сущности `Employee` выполняется через инициализатор `init`. В этом инициализаторе происходит получение контекста управляемого объекта из `userInfo` параметра decoder, создание нового экземпляра `Employee` полученного контекста. Затем вызывается метод `update`, в котором происходит обновление свойств объекта на основе декодированных данных.

В методе `update` сначала получается контейнер из декодера с помощью

ключей, заданных в CodingKeys, а далее вызываются остальные методы для декодирования, которые перечислены ниже.

Метод decodeEmployee используется для декодирования основной информации о преподователе. Если присутствует структура вложенного контейнера employeeNestedContainer, она извлекается, иначе используется корневой контейнер. Значения для свойств id, urlID, firstName, middleName, lastName, photoLink и rank декодируются из контейнера. Значение для departments декодируется как массив строк и затем преобразуется во множество Department, которые добавляются к свойству departments преподавателя.

Метод decodeDegree декодирует степень преподавателя. Если уже существует объект degree, вызывается его метод update, иначе создается новый объект Degree с использованием метода Degree и присваивается свойству degree.

Метод decodeLessons декодирует занятия, связанные с преподавателем. Если в контейнере есть словарь lessonsDictionary, он декодируется как [String:[Lesson]]. Уроки из словаря объединяются и добавляются в массив lessons. Затем, если есть массив exams, он также добавляется в массив lessons. Для каждого занятия в массиве lessons устанавливается массив с идентификатором переподователя. Если массив lessons не пустой, устанавливается дата обновления lessonsUpdateDate. Наконец, занятия добавляются в свойство lessons преподавателя.

3.6.8 Декодирование факультета

Декодирование сущности Faculty выполняется с использованием метода init. В этом методе происходит инициализация объекта Faculty и заполнение его свойств значениями из декодера.

Сначала получается контекст управляемого объекта из параметра decoder.userInfo. Затем вызывается инициализатор init, который создает новый экземпляр Faculty в указанном контексте.

Далее происходит проверка условия: если параметр groupContainer равен true, это означает, что инициализатор был вызван из декодера, который декодирует Group и не может найти требуемую специальность. В этом случае вызывается специальный метод updateFromGroupDecoder, который обрабатывает специфические ключи Group.

Если условие не выполняется, то вызывается метод update, который обновляет свойства Faculty из декодера. Внутри метода контейнер получается с помощью container, а затем значения свойств декодируются с помощью метода decode, вызываемого на контейнере. Значения декодируются для свойств id, name и abbreviation с соответствующими ключами из enum CodingKeys.

После успешного обновления свойств Faculty, вызывается метод info из класса Log для записи информации об обновлении Faculty в логах.

Если вызывается метод updateFromGroupDecoder, то вложенный контейнер, если он присутствует, получается из nestedContainer, иначе используется обычный контейнер. Затем значения свойств id и abbreviation декодируются из контейнера с помощью метода decode, вызываемого на контейнере. Значения декодируются с соответствующими ключами из enum GroupCodingKeys.

3.6.9 Декодирование группы

Декодирование сущности Group выполняется через метод init. Внутри этого метода создается новый экземпляр Group в контексте управляемого объекта CoreData, который получается из userInfo параметра декодера. Затем вызывается метод update, который выполняет обновление свойств объекта на основе декодированных данных.

Внутри метода update происходит следующий процесс декодирования:

- получение контейнера из декодера с помощью decoder.container;
- вызов методов decodeGroup, decodeEducationDates, decodeSpeciality и decodeLessons для декодирования различных свойств объекта Group;
- в методе decodeGroup происходит декодирование информации о группе, такой как имя, курс, степень образования и количество студентов;
- в методе decodeSpeciality происходит декодирование специальности группы. Если специальность указана, то создается новый экземпляр Speciality с помощью инициализатора Speciality и присваивается свойству speciality объекта Group;
- в методе decodeLessons происходит декодирование занятий и экзаменов группы. Если есть занятия или экзамены, то свойство lessonsUpdateDate устанавливается на текущую дату;
- после завершения декодирования вызывается метод info из класса Log для записи информации о созданном или обновлённом экземпляре Group в логах.

Декодирование сущности Group включает в себя несколько вспомогательных методов, которые выполняют декодирование конкретных свойств объекта Group из декодера. Давайте подробнее рассмотрим каждый из этих методов:

Метод decodeGroup получает контейнер из декодера, используя decoder.container. Затем метод декодирует информацию о группе из контейнера. Свойство name декодируется из строки с ключом .id и присваивается объекту Group. Если доступно значение для ключа .course, то свойство course декодируется из Int16 и также присваивается объекту Group. Свойство educationDegreeValue декодируется из Int16 с

ключом .educationDegree и присваивается объекту Group. Если доступно значение для ключа .numberOfStudents, то свойство numberOfStudents декодируется из Int16 и присваивается объекту Group.

Метод decodeSpeciality получает контейнер из декодера, используя decoder.container. Затем метод пытается декодировать значение типа Int32 из контейнера с ключом .specialityID. Если декодирование успешно, создается новый экземпляр Speciality с помощью инициализатора Speciality и присваивается свойству speciality объекта Group.

В логах записывается информация о созданной и присвоенной специальности.

Метод decodeLessons получает контейнер в качестве параметра. Используя контейнер, метод пытается декодировать массив занятий и экзаменов группы из ключей .lessons и .exams соответственно. Если есть хотя бы один из массивов занятий или экзаменов, то свойство lessonsUpdateDate объекта Group устанавливается на текущую дату и время.

3.6.10 Декодирование занятия

Декодирование сущности Lesson выполняется в методе init. Вначале из параметра decoder извлекается контекст управляемого объекта CoreData с помощью decoder.userInfo. Затем происходит инициализация нового экземпляра Lesson с использованием entity и insertInto.

Декодирование значений начинается с извлечения контейнера container из decoder с помощью decoder.container.

Затем вызывается метод decodeLesson, который выполняет декодирование свойств subject, abbreviation, note и subgroup с помощью метода decode на контейнере. Свойство subject декодируется как опциональное значение String, abbreviation не может быть опциональным, поэтому при декодировании присваивается пустая строка, в случае отсутствия значения. Свойства note и subgroup декодируются как опциональные значения String и Int16 соответственно.

Затем, если есть значение lessonTypeID в контейнере, выполняется инициализация свойства type с помощью конструктора LessonType(id:context:).

Далее вызывается метод decodeDate, который декодирует значения свойств, относящихся к дате и времени занятия. Сначала декодируется строка date, которая присваивается свойству dateString. Затем декодируются значения startLessonDate, endLessonDate, timeStart и timeEnd с помощью метода decode.

Если значения startLessonDate и endLessonDate успешно декодируются в строки, то они присваиваются соответствующим свойствам startLessonDateString, startLessonDate и endLessonDate. При этом проверяется условие, если startLessonDate больше endLessonDate, то

значения этих свойств меняются местами.

Затем, если дата `date` или `startLessonDate` имеет значение, вычисляется `weekday` с помощью свойства `weekday` у `date`.

Далее декодируется массив `weeks` с помощью метода `decode`. Если значения успешно декодируются в `[Int]`, то они присваиваются свойству `weeks` с вычитанием единицы из каждого значения. Если в массиве `weeks` содержится значение «-1», оно удаляется с помощью `removeFirst`.

Метод `decodeAuditories` отвечает за декодирование и присваивание значений свойствам, связанным с аудиториями занятия.

Внутри метода происходит попытка декодирования значения для ключа `.auditorium` из контейнера `container`. Если декодирование успешно, то полученный массив названий аудиторий преобразуется в массив объектов `Auditorium` с использованием инициализатора `Auditorium(from:in:)`, где каждое имя аудитории передаётся в инициализатор вместе с контекстом `context`. Затем полученные аудитории добавляются к свойству `auditories` занятия с помощью метода `addToAuditories`. Кроме того, имена аудиторий сохраняются в свойстве `auditoriesNames` занятия в виде отсортированного массива строк, состоящих из номера этажа, имени аудитории и номера здания.

Если декодирование значения для ключа `.auditorium` не удалось, то свойство `auditoriesNames` занятия присваивается пустой массив.

Метод `decodeAnnouncement` отвечает за декодирование и обработку значений, связанных с объявлением. Внутри метода сначала проверяется, является ли значение для ключа `.announcement` равным `true`. Если значение равно `true`, то выполняется следующая логика:

- устанавливаются значения `[0, 1, 2, 3]` для свойства `weeks`, указывающего на то, что объявлением проходит на всех неделях;

- если значение для ключа `.announcementStart` декодируется успешно с помощью, то присваивается значение свойству `timeStart`, аналогично, если значение для ключа `.announcementEnd` декодируется успешно, то присваивается значение свойству `timeEnd`.

Таким образом, в результате декодирования и обработки значений объявлений занятия, устанавливаются соответствующие свойства `weeks`, `timeStart` и `timeEnd`.

Метод `decodeEmployees` также принимает контейнер `container`, декодер `decoder` и контекст `NSManagedObjectContext`. Он проводит попытку декодировать массив преподавателей из контейнера. Если декодирование успешно, создается `NSSet`, содержащий экземпляры класса `Employee`, полученные в результате декодирования. Затем этот `NSSet` добавляется к свойству занятия `employees` с помощью метода `addToEmployees`, устанавливающего связь типа многие-ко-многим между `Lesson` и `Employee`. Таким образом, свойство `employees` будет содержать связанных с данным занятием преподавателей.

Метод decodeGroups принимает контейнер container, декодер decoder и контекст NSManagedObjectContext. Внутри метода происходит попытка декодирования массива групп студентов из контейнера. Если декодирование прошло успешно, создается NSSet, содержащий экземпляры класса Group, полученные в результате декодирования. Затем этот NSSet добавляется к свойству занятия groups с помощью метода addToGroups, который является сгенерированным CoreData методом для установки связи типа многие-ко-многим между Lesson и Group. Таким образом, свойство groups будет содержать связанные группы студентов для данного занятия.

3.6.11 Декодирование типа занятия

Декодирование сущности LessonType выполняется путём инициализации объекта с использованием соответствующих инициализаторов.

В классе LessonType определено два инициализатора. Первый инициализатор принимает только идентификатор id и контекст. Внутри инициализатора вызывается инициализатор родительского класса NSManagedObject, передавая ему сущность и контекст. Затем устанавливается значение свойства id равным переданному идентификатору.

Второй инициализатор принимает идентификатор id, название name, аббревиатуру abbreviation, цвет color и контекст управляемого объекта context. Опять же, внутри инициализатора вызывается инициализатор родительского класса с передачей сущности и контекста. Затем значения переданных свойств устанавливаются соответствующим образом.

В обоих случаях, после создания экземпляра LessonType, он будет связан с контекстом управляемого объекта и готов к сохранению или использованию в приложении.

3.6.12 Декодирование специальности

Декодирование сущности Speciality выполняется в методе init. В этом методе происходит инициализация нового экземпляра Speciality в контексте управляемого объекта, который получается из userInfo параметра декодера.

Далее проверяется условие, является ли вызов init направленным из декодера, который расшифровывает Group и не может обнаружить требуемую специальность. Если это так, вызывается специальный метод updateFromGroupDecoder, который обрабатывает ключи для группы. В противном случае вызывается метод update, который обрабатывает обычные ключи.

В методе update осуществляется декодирование значений из контейнера с помощью ключей, заданных в enum CodingKeys. Контейнер получается из параметра decoder. Значения для свойств id, name,

abbreviation, code и educationType декодируются с помощью метода decode, который вызывается на контейнере. Типы, которые декодируются, указываются в квадратных скобках. Некоторые значения могут быть опциональными, и если декодирование не удалось, то выбрасывается исключение.

Затем проверяется, можно ли декодировать facultyID и получить контекст управляемого объекта. Если это возможно, создается новый объект Faculty с помощью инициализатора, принимающего id и context, и устанавливается свойство faculty для Speciality.

После декодирования всех значений и обновления объекта Speciality вызывается метод info из класса Log для записи информации о том, что специальность была обновлена.

В методе updateFromGroupDecoder происходит декодирование значений, когда вызов init осуществляется из декодера, декодирующего Group. Если информация о специальности содержится во вложенном контейнере, он получается с помощью nestedContainer, в противном случае используется обычный контейнер. Затем значения для свойств id, name, abbreviation и faculty декодируются с помощью метода decode, а объект Faculty декодируется с помощью инициализатора, принимающего декодер.

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

В данном разделе представлено несколько ключевых алгоритмов программного средства, а также приведено описание нескольких ключевых представлений.

4.1 Представления

Так как количество представлений в программном средстве превышает три десятка, для описания выбрано только несколько представлений, при этом упор сделан на представления, которые используются как шаблоны или части включаются в другие представления.

4.1.1 Стандартизированное меню с проверкой значений по умолчанию

Во многих представлениях существует необходимость в меню с некоторыми параметрами, при этом стоит отображать пользователю, является ли выбранный в данный момент времени набор – набором по умолчанию. Для этого используется заливка значка меню цветом в момент, когда выбранные параметры не являются параметрами по умолчанию.

Исходный код представления:

```
struct MenuView<Content: View>: View {
    var defaultRules: [Bool]
    var satisfyDefaultRules: Bool {
        !defaultRules.contains(false)
    }
    var imageName: String {
        if satisfyDefaultRules {
             return "line.3.horizontal.decrease.circle"
        } else {
             return "line.3.horizontal.decrease.circle.fill"

        }
    }

    @ViewBuilder var content: () -> Content

    var body: some View {
        Menu {
            content()
        } label: {
            Image(systemName:     satisfyDefaultRules     ?
imageName")
        }

    }
```

```
}
```

Представление принимает как единственный параметр массив значений типа `Bool defaultRules`. Данный массив содержит в себе условия, которые должны выполняться для того, чтобы набор значений параметров (настроек) считался набором по умолчанию.

Вычисляемое свойство `satisfyDefaultRules` проверяет, являются ли верными все переданные в `defaultRules` значения. Если это так, то значение свойства является истинным, в ином случае – ложным.

Вычисляемое свойство `imageName` используется для того, чтобы получать названия изображения иконки меню с заливкой, либо без неё.

Этот же `imageName` передаётся в `Image`, то есть при изменении свойства `defaultRules` изменяется `satisfyDefaultRules` свойство, а из-за него изменяется уже `imageName`, а это изменяет изображение в представлении.

4.1.2 Стандартизированная квадратная ячейка

Стандартизированная квадратная ячейка является одним из базовых элементов, на которых строится иерархия представлений программного средства. Она используется как базовая ячейка для многих списков.

Исходный код представления:

```
struct SquareView<Content: View>: View {

    @ViewBuilder var content: () -> Content

    var body: some View {
        content()
        .padding()
        .roundedRectangleBackground()
        .aspectRatio(1.0, contentMode: .fit)

    }
}
```

Описание кода:

- `struct SquareView<Content: View>: View`: объявление структуры `SquareView` с обобщённым типом `Content`, который должен соответствовать протоколу `View`, сам `SquareView` также соответствует протоколу `View`;
- `@ViewBuilder var content: () -> Content`: определение свойства `content` с атрибутом `@ViewBuilder`. это свойство является замыканием, которое возвращает тип `Content`, а атрибут `@ViewBuilder`, означает, что в него может быть передано сразу несколько `View`;
- `var body: some View` объявление вычисляемого свойства `body` типа `some View`, это обязательное свойство для протокола `View;`

- `content()`: вызов замыкания `content`, которое позволяет использовать содержимое замыкания как часть представления `SquareView`;

- `.padding()`: применение модификатора `padding()` к содержимому `SquareView`, этот модификатор добавляет отступы равные 16 пунктам вокруг содержимого;

- `.roundedRectangleBackground()`: применение модификатора `roundedRectangleBackground()` к содержимому `SquareView`, этот модификатор задаёт фон в форме закруглённого прямоугольника;

- `.aspectRatio(1.0, contentMode: .fit)`: Применение модификатора `aspectRatio(_:contentMode:)` к содержимому `SquareView`. этот модификатор задаёт соотношение сторон в представлении и режим заполнения содержимого, в данном случае, соотношение сторон установлено равным единице, то есть квадрату, а режим заполнения – `.fit`, что означает, что содержимое будет подстроено под размеры представления.

4.1.3 Стандартизированная квадратная текстовая ячейка

На основании стандартизированной квадратной ячейки, которая может вмещать любое представление в себя, создана стандартизированная квадратная текстовая ячейка, которая вмещает в себя три строки текста с различными атрибутами.

Разделение стандартизированной квадратной ячейки и стандартизированной квадратная текстовой ячейки обусловлено тем, что такое разделение позволяет использовать обычную стандартизированную квадратную ячейку не только для текста. Например, она используется для кнопки выбора изображений в представлении задачи. Там вместо текста располагается изображение знака сложения.

Исходный код представления:

```
struct SquareTextView: View {
    var title: String

    var firstSubtitle: String?
    var secondSubtitle: String?

    var body: some View {
        SquareView {
            HStack {
                VStack(alignment: .leading) {
                    titleText
                    Spacer()
                    firstSubtitleText
                    secondSubtitleText
                }
                .lineLimit(2)
                .minimumScaleFactor(0.5)
```

```
                    .multilineTextAlignment(.leading)
                    .foregroundColor(.primary)

                    Spacer(minLength: 0)
                }
            }

        }
        var titleText: some View {
            Text(title)
                .font(.system(.title3,
                              design: .rounded,
                              weight: .bold))
        }
        @ViewBuilder var firstSubtitleText: some View {
            if let firstSubtitle = firstSubtitle {
                Text(firstSubtitle)
            }
        }
        @ViewBuilder var secondSubtitleText: some View {
            if let secondSubtitle = secondSubtitle {
                Text(secondSubtitle)
                    .font(.headline)
                    .foregroundColor(Color.gray)
            }
        }
    }
```

Описание кода:

- объявление структуры `SquareTextView` с одним обязательным свойством `title` типа `String;`
- объявление двух необязательных свойств `firstSubtitle` и `secondSubtitle` типа `String?`;
- создание `SquareView`, которое является контейнером с квадратной формой.
- создание `HStack` – горизонтального контейнера, в котором располагаются элементы в строку;
- создание `VStack` – вертикального контейнера, в котором располагаются элементы в столбец;
- в `VStack` добавляются тексты: `titleText`, `firstSubtitleText`, `secondSubtitleText`;
- установка ограничения на количество строк для `VStack` (`.lineLimit(2)`);
- установка минимального масштаба текста для `VStack` (`.minimumScaleFactor(0.5)`);
- выравнивание многострочного текста по левому краю для `VStack` (`.multilineTextAlignment(.leading)`);

- установка цвета текста для `VStack` (`.foregroundColor(.primary)`);

- добавление пустого пространства справа от `VStack` с использованием `Spacer(minLength: 0)`;

- определение вычисляемого свойства `titleText`, которое возвращает `Text` с заголовком;

- установка шрифта для `Text` (`.font(.system(.title3, design: .rounded, weight: .bold))`);

- определение вычисляемого свойства `firstSubtitleText` с использованием `@ViewBuilder`, которое возвращает `Text` с первым дополнительным заголовком, если он существует;

- проверка, существует ли первый дополнительный заголовок (`if let firstSubtitle = firstSubtitle`);

- определение вычисляемого свойства `secondSubtitleText` с использованием `@ViewBuilder`, которое возвращает `Text` со вторым дополнительным заголовком, если он существует.

4.1.4 Ленивая квадратная сетка

Ленивая квадратная сетка представляет из себя представление построенное на основе `LazyVGrid`. Данное представлене используется в паре с вышеупомянутыми ячейками.

`LazyVGrid` – это контейнерное представление, которое организует своих потомков в сетку с гибким количеством колонок. Он обеспечивает ленивую загрузку своих элементов, что означает, что только видимые элементы просчитываются и отображаются на экране, а остальные элементы загружаются по мере прокрутки или при изменении размеров экрана. Данное свойство позволяет значительно сократить затраты приложения на ресурсы оперативной памяти, однако, при интенсивной прокрутке может быть увеличена нагрузка на процессор.

Основные черты `LazyVGrid`:

- гибкое количество колонок: `LazyVGrid` позволяет задать количество колонок с помощью параметра `columns`, который может быть константой или динамическим;

- ленивая загрузка: `LazyVGrid` загружает и отображает только видимые элементы, что повышает производительность и улучшает использование ресурсов;

- динамическое размещение элементов: `LazyVGrid` автоматически размещает элементы в сетке, заполняя доступное пространство с учётом размеров и расположения элементов;

- поддержка разных типов представлений: `LazyVGrid` может содержать любые представления внутри себя, такие как текст, изображения, кнопки и другие пользовательские виды;

- конфигурирование размещения элементов: `LazyVGrid` предоставляет

возможность настраивать различные атрибуты размещения элементов, такие как отступы, выравнивание и прочие.

Исходный код представления:

```
struct SquareGrid<Content: View>: View {

    @ViewBuilder var content: () -> Content

    var body: some View {
        LazyVGrid(columns: [SquareTextView.gridItem],
                  alignment: .leading,
                  spacing: 8) {
            content()
        }

    }
}
```

4.1.5 Разделённая на секции ленивая квадратная сетка

Для использования с шаблонным классом `NSManagedObjectsSection`, который используется во многих представлениях для отображения списков с секциями. Используя ленивую квадратную сетку, была создана разделённая на секции ленивая квадратная сетка, которая принимает секции как параметр.

Само представление каждой секции реализуется через замыкание, которое предоставляет доступ к объекту `NSManagedObjectsSection`, внутри замыкания можно реализовать любое представление. Чаще всего там используются стандартизированные квадратные текстовые ячейки.

Исходный код представления:

```
struct SectionsSquareGrid<ItemType: NSManagedObject, Content:
View>: View {

    var sections: [NSManagedObjectsSection<ItemType>]
    @ViewBuilder var content: (ItemType) -> Content

    var body: some View {

        SquareGrid {
            ForEach(sections, id: \.id) { section in
                Section {
                    ForEach(section.items, id: \.self) { item
in
                        content(item)
                    }
                } header: {
                    HeaderView(section.title)
                }
            }
```

```
            }
        }
    }
```

4.1.6 Форма с альтернативным текстом

Форма с альтернативным текстом используется для отображения информации в основном и альтернативном видах. Например, это может быть кратное и полное название.

Исходный код представления:

```
    struct FormViewWithAlternativeText: View {
        var title: String
        var text: String?
        var alternativeText: String?

        @State var showAlternativeText = false

        init(_ name: String, _ parameter: String?, _
alternativeText: String?) {
            self.title = name
            self.text = parameter
            self.alternativeText = alternativeText
        }

        var body: some View {
            if let text = text {
                if let alternativeText = alternativeText {
                    Button {
                        withAnimation
{ showAlternativeText.toggle() }
                    } label: {
                        FormView(title, showAlternativeText ?
alternativeText : text)
                    }
                } else {
                    FormView(title, text)
                }
            }
        }
    }
```

Как и в уже описанных представлениях, тут используются проверка значений на опциональность. В случае, если `alternativeText` представлен, используется свойство `showAlternativeText` типа `Bool`. Кнопка переключает значения `showAlternativeText`, а отображение формы настроено так, что при изменении `showAlternativeText`, выбирается, какой из текстов будет отображаться.

4.1.7 Стандартизированный заголовок

Стандартизированный заголовок используется как для обозначения секций, так и для ссылок на другие представления.

Исходный код представления:

```
struct HeaderView: View {
    var title: String
    var withArrow: Bool

    init(_ title: String, withArrow: Bool = false ) {
        self.title = title
        self.withArrow = withArrow
    }

    var body: some View {
        HStack(spacing: 0) {
            Text(title)
                .multilineTextAlignment(.leading)
                .foregroundColor(.primary)
            if withArrow {
                Spacer()
                Text(Image(systemName: "chevron.right"))
                    .foregroundColor(.gray)
            }
        }
        .fontWeight(.heavy)
        .font(.title2)
        .padding(.top)
    }
}
```

Параметр `withArrow` используется для заголовка со ссылкой, он указывает представлению, что необходимо добавить справа от заголовка стрелку.

4.2 Алгоритмы

Некоторые из представленных ниже алгоритмов в исходном коде представляют сразу несколько методов или функций. Это обусловлено тем, что для улучшения читаемости исходного крупные методы разбиваются на несколько методов меньшего размера, где каждый из меньших методов выполняет конкретную небольшую задачу.

4.2.1 Алгоритм получения учебной недели

Целью данного алгоритма является вычисление номера учебной недели университета для объекта типа Date. Этот алгоритм используется в алгоритмах

получения секций расписания и при декодировании занятий.

Шаги алгоритма:

Шаг 1. Начало алгоритма.

Шаг 2. Создать константу `calendar` типа Calendar.

Шаг 3. Задать `calendar` значение `Calendar.init(identifier: .iso8601)`.

Шаг 4. Создать переменную `firstSeptemberDateComponents`.

Шаг 5. Задать переменной `firstSeptemberDateComponents` значение `calendar.dateComponents([.year], from: self)`.

Шаг 6. Задать свойству `firstSeptemberDateComponents.day` значение 1.

Шаг 7. Задать свойству `firstSeptemberDateComponents.month` значение 9.

Шаг 8. Создать переменную `firstSeptember`.

Шаг 9. Задать переменной `firstSeptember` значение `calendar.date(from: firstSeptemberDateComponents)!`.

Шаг 10. Если не выполняется условие `self < firstSeptember`, перейти к шагу 13.

Шаг 11. Отнять единицу от свойства `firstSeptemberDateComponents.year`, назначить результат свойству `firstSeptemberDateComponents.year`.

Шаг 12. Назначить переменной `firstSeptember` значение `calendar.date(from: firstSeptemberDateComponents)`.

Шаг 13. Создать константу `weeksOfSelfYear`.

Шаг 14. Присвоить константе значение `calendar.dateComponents([.weekOfYear], from: self).weekOfYear!`.

Шаг 15. Создать переменную `weeksOfDateYear`.

Шаг 16. Присвоить переменной значение `calendar.dateComponents([.weekOfYear], from: end).weekOfYear!`.

Шаг 17. Создать константу `yearOfSelf`.

Шаг 18. Присвоить константе значение `calendar.dateComponents([.year], from: self).year!`.

Шаг 19. Создать константу `yearOfDate`.

Шаг 20. Присвоить константе значение `calendar.dateComponents([.year], from: end).year!`

Шаг 21. Если не выполняется условие `yearOfSelf < yearOfDate && weeksOfDateYear != 52`, перейти к шагу 23.

Шаг 22. Прибавить к `weeksOfDateYear` 52 и назначить результат этого вычисления `weeksOfDateYear`.

Шаг 23. Вернуть значение вычитания `weeksOfSelfYear` из

weeksOfDateYear.

Шаг 24. Конец алгоритма.

4.2.2 Алгоритм создания секций расписания по датам

Целью данного алгоритма является создание из любой последовательности неповторяющихся занятий отсортированного массива секция расписания, которые основаны на дате.

Данный алгоритм получает значение educationDates типа [Date]? как параметр. Шаги алгоритма:

Шаг 1. Начало алгоритма.

Шаг 2. Создать константу startTime.

Шаг 3. Присвоить startTime значение CFAbsoluteTimeGetCurrent().

Шаг 4. Создать константу educationDates.

Шаг 5. Присвоить educationDates значение educationDates либо, в случае она равна nil, присвоить ей значение self.educationDates.

Шаг 6. Если self.educationDates не равна nil, перейти к шагу 8.

Шаг 7. Вернуть пустой массив.

Шаг 8. Перейти на шаг 25

Шаг 9. Создать переменную sections, типа [ScheduleSection].

Шаг 10. Создать переменную date.

Шаг 11. Извлечь первое значение из educationDates и присвоить его переменной date.

Шаг 12. Создать константу educationWeek.

Шаг 13. Присвоить educationWeek значение date.educationWeek.

Шаг 14. Создать константу weekday.

Шаг 15. Присвоить weekday значение date.weekday.

Шаг 16. Создать константу lessons.

Шаг 17. Присвоить lessons значение результата выполнения self.filter({ ($0.weekday == weekday && $0.weeks.contains(educationWeek) && ($0.dateRange?.contains(date) == true)) || $0.date == date }).

Шаг 18. Если количество элементов в lessons равно нулю, то перейти на шаг 23.

Шаг 19. Создать константу section.

Шаг 20. Задать section значение ScheduleSection(type: .date, date: date, educationWeek: educationWeek, weekday: weekday, lessons: lessons).

Шаг 21. Вызвать статический метод Log.info(section.description).

Шаг 22. Добавить к sections section.

Шаг 23. Если массив educationDates не является пустым, перейти к шагу 11.

Шаг 24. Вызвать статический метод Log.info("dateSections

\(sections.count) has been created, time: \((CFAbsoluteTimeGetCurrent() - startTime).roundTo(places: 5)) seconds").

Шаг 25. Вернуть sections.

Шаг 26. Конец алгоритма.

4.2.3 Алгоритм создания секций расписания по неделям

Целью данного алгоритма является создание из любой последовательности неповторяющихся занятий отсортированного массива секция расписания, которые основаны на учебных неделях и днях недели.

Шаги алгоритма:

Шаг 1. Начало алгоритма.

Шаг 2. Создать константу startTime.

Шаг 3. Присвоить startTime значение CFAbsoluteTimeGetCurrent().

Шаг 4. Создать переменную weeks.

Шаг 5. Создать переменную weekdays.

Шаг 6. Присвоить weeks статическое значение Constants.Ranges.weeks.

Шаг 7. Создать переменную week.

Шаг 8. Создать переменную weekday.

Шаг 9. Создать переменную lessonsInWeek.

Шаг 10. Создать переменную lessonsInWeekday.

Шаг 11. Извлечь первое значение из weeks и присвоить его week.

Шаг 12. Присвоить lessonsInWeek значение lessons.filter { $0.week == week }.

Шаг 13. Присвоить weekdays статическое значение Constants.Ranges.weekdays.

Шаг 14. Извлечь первое значение из weekdays и присвоить его weekday.

Шаг 15. Присвоить lessonsInWeekday значение lessonsInWeek.filter { $0.weekday == weekday }.

Шаг 16. Если количество элементов в lessonsInWeekday равно нулю, то перейти на шаг 22.

Шаг 17. Создать константу section.

Шаг 18. Задать section значение ScheduleSection(type: .week, educationWeek: week, weekday: Int16(weekday), lessons: lessonsInWeekday).

Шаг 19. Вызвать статический метод Log.info(section.description).

Шаг 20. Добавить к sections section.

Шаг 21. Если количество элементов в массиве weekdays не равно нулю, перейти к шагу 14.

Шаг 22. Если количество элементов в массиве weeks не равно нулю,

перейти к шагу 11.

Шаг 23. Вызвать статический метод `Log.info("weekSections \(sections.count) has been created, time: \((CFAbsoluteTimeGetCurrent() - startTime).roundTo(places: 5)) seconds")`.

Шаг 24. Вернуть `sections`.

Шаг 25. Конец алгоритма.

4.2.4 Алгоритм извлечения данных из API

Описание процесса извлечения данных из API и некоторые дополнительные подробности об используемых технологиях и методах приведены в подразделе 3.5. Целью алгоритма является асинхронно получить данные из API ИИС БГУИР, то есть сформировать запрос, отправить его, получить и обработать ответ. При этом необходимо обработать возможные ошибки.

Данный алгоритм получает значение `dataType` типа `FetchDataType` и `argument` типа `String?` как параметры. Шаги алгоритма:

Шаг 1. Начало алгоритма.

Шаг 2. Создать константу `url`.

Шаг 3. Если полученное как параметр значение `argument` равняется `nil`, то перейти к шагу 6.

Шаг 4. Присвоить `url` значение `URL(string: dataType.rawValue)`.

Шаг 5. Перейти к шагу 7.

Шаг 6. Присвоить `url` значение `URL(string: : dataType.rawValue + argument!)`.

Шаг 7. Если `url` не равно nil, перейти к шагу 10.

Шаг 8. Вызвать статический метод `Log.error("Can't create url for \(dataType), argument: \(String(describing: argument))")`.

Шаг 9. Вернуть ошибку `URLError(.badURL)`.

Шаг 10. Перейти к шагу 18.

Шаг 11. Создать кортеж `(data, urlResponse)`.

Шаг 12. Присвоить `(data, urlResponse)` значение `try await URLSession.shared.data(from: url)`.

Шаг 13. Если количество элементов `data` не ровняется нулю, перейти к шагу 17.

Шаг 14. Вызвать статический метод `Log.error("Can't get data from \(dataType), argument: \(String(describing: argument)), urlResponse: \(urlResponse)")`.

Шаг 15. Вернуть `nil`.

Шаг 16. Перейти к шагу 18.

Шаг 17. Вернуть `data`.

Шаг 18. Конец алгоритма.

5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

В данном разделе представлена методика испытания программного средства. Проведено некоторое тестирование функционала и графического интерфейса.

5.1 Тесты графического интерфейса

Для проверки графического интерфейса использовались встроенные в Xcode возможности для одновременного отображения представлений с разными системными настройками, такими как:

- светлая и тёмная темы;
- масштаб интерфейса;
- ориентация интерфейса.

Также необходимо провести тестирование для разных устройств, так как они имеют разные размеры дисплеев.

5.1.1 Адаптивность интерфейса для разных устройств

Для проверки адаптивности интерфейса было проведено его тестирование на устройствах с разными размерами дисплея. На рисунке пример тестирования на iPad Pro:

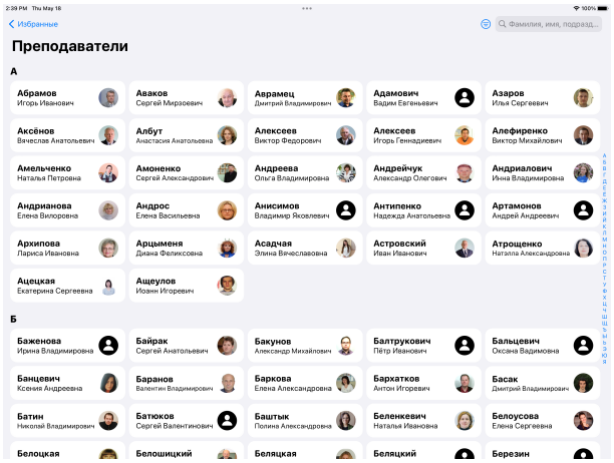


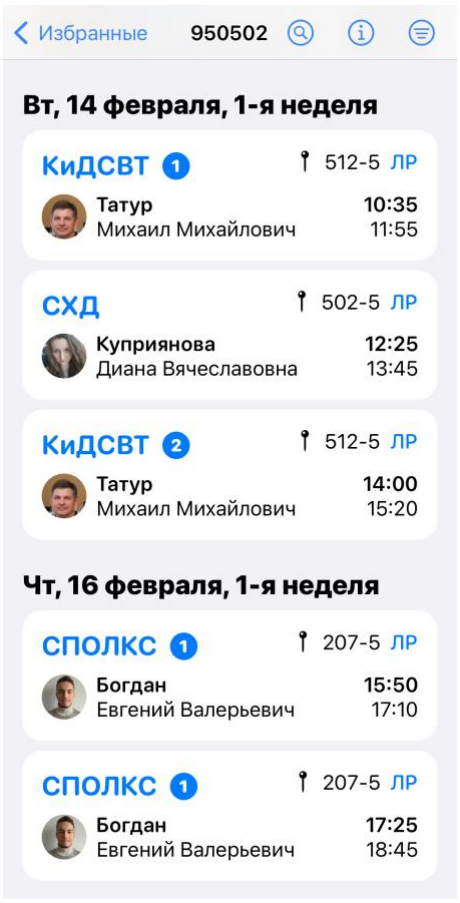
Рисунок 5.1 – представление списка преподавателей

5.1.2 Светлая и тёмная темы

Как в светлой, так и в тёмной темах, должны быть чётко видны все надписи и изображения. Тестирование проведено для всех представлений, вышеописанные критерий соблюдается для всех представлений. Стандартные цвета для типов занятий различимым как при светлой, так и при тёмной темах.

Снимки экрана представлены на рисунках:

а)



б)

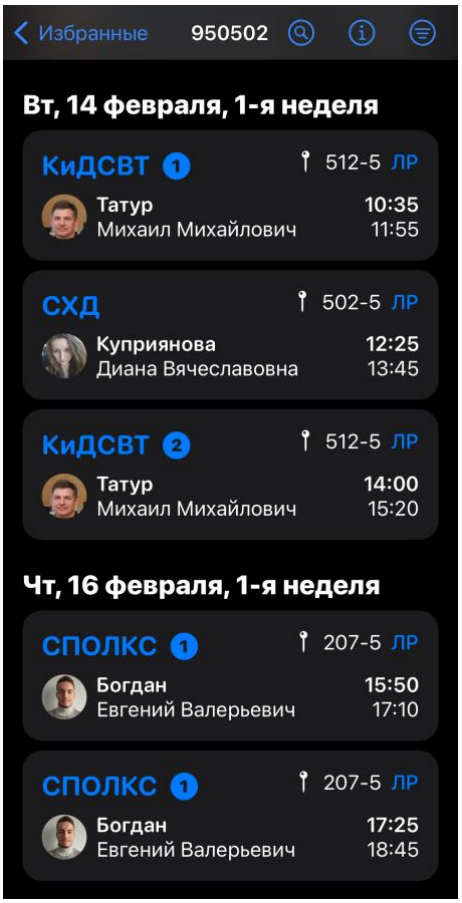


Рисунок 5.2 – представление расписания занятий:
а – светлая тема; *б* – тёмная тема

5.1.3 Масштаб интерфейса

В ОС компании Apple предусмотрено системное масштабирование элементов интерфейса, которое изменяет системные размеры шрифта и иные объекты.

Имеются следующие масштабы интерфейса в порядке возрастания:

1. X Small, Small.
2. Medium.
3. Large.
4. X Large, XX Large, XXX Large.
5. AX1, AX2, AX3, AX4, AX5.

Так как масштабы больше, чем Large используются в крайне редких случая, а их масштаб требует создания дополнительных представлений, рассматривается только первые четыре размера.

Согласно результатам теста, все представления масштабируют текст и другие элементы, на увеличенном представлении не происходит выход текста за рамки.

Результаты теста представлены на рисунке:

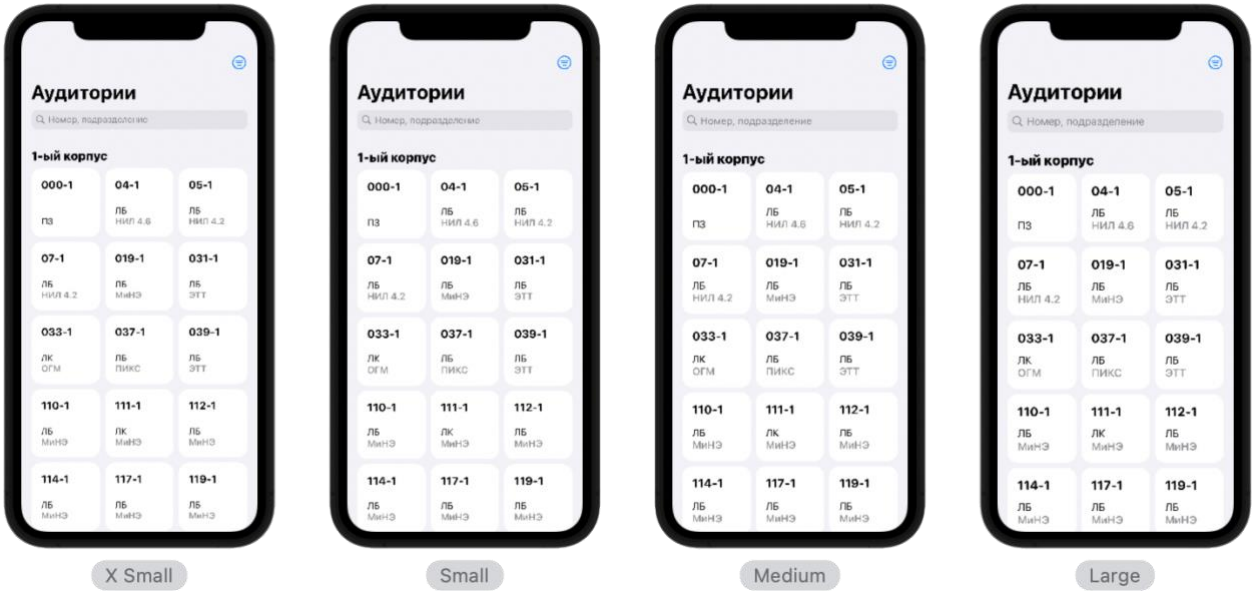


Рисунок 5.3 – представление списка аудиторий с различными масштабами интерфейса

5.1.4 Ориентация интерфейса

Интерфейс приложения должен адаптироваться к разным ориентациям. Например, в строку должно вмещаться больше ячеек. Для проверки этого средствами Xcode и в ручном режиме были проведено тестирование всех представлений. Результат теста приведён на рисунке:

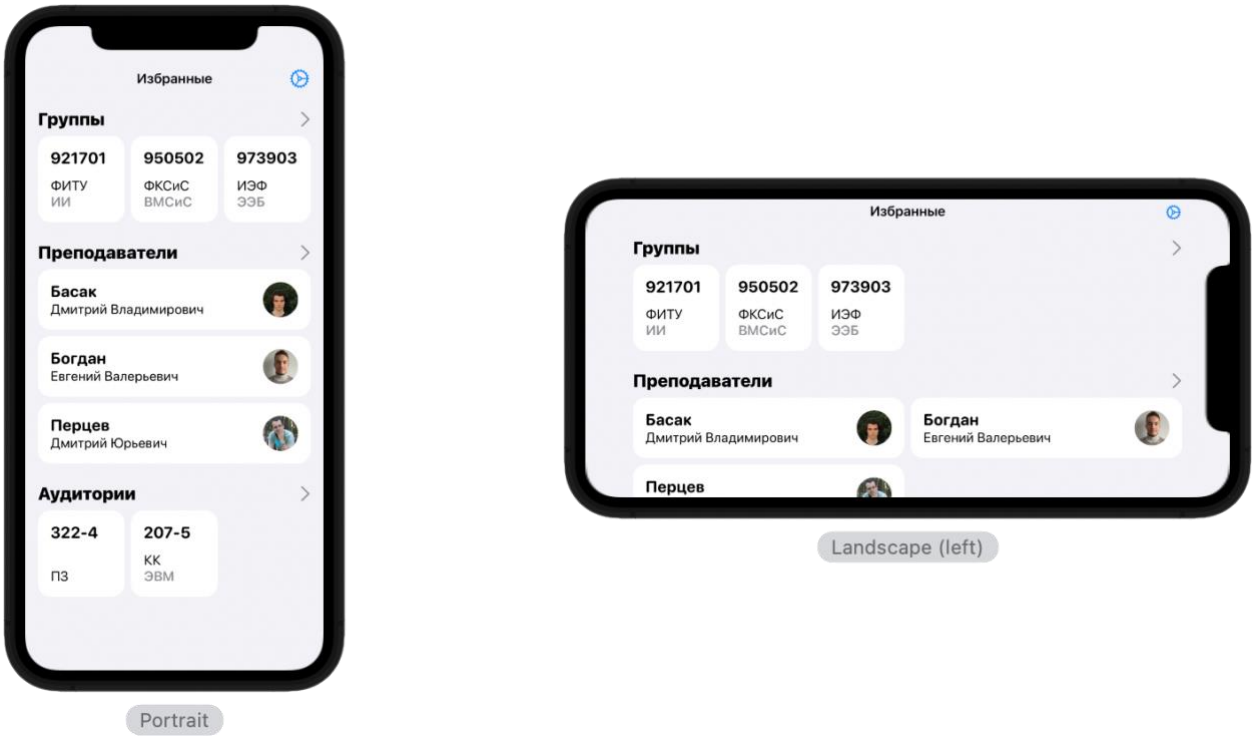


Рисунок 5.4 – представление избранного при различных ориентациях

5.2 Тесты функциональности

Тестирование функциональности необходимо для устранения возможных ошибок в исходном коде, которые вызывают некорректную обработку информации, либо критические ошибки.

5.2.1 Использование без сетевого соединения

Так как программное средство имеет локальную базу данных, большая часть его функционала должна быть доступна без сетевого соединения, а следовательно, и доступа к API ИИС. При этом связанный с обновлением функционал будет недоступен до возобновления сетевого соединения.

Результаты тестирования приведены в таблице:

Таблица 5.1 – Результаты тестирования без сетевого соединения

Тест	Ожидаемый результат	Полученный результат
Просмотр ранее загруженного расписания занятий	Расписание отображается, отсутствует возможность обновления	Расписание отображается, отсутствует возможность обновления
Просмотр ранее не загруженного расписания занятий	Расписание не отображается, появляется сообщение об отсутствии сетевого соединения	Расписание не отображается, появляется сообщение об отсутствии сетевого соединения
Просмотр даты последнего обновления расписания	Дата последнего обновления не отображается	Дата последнего обновления не отображается
Просмотр списка аудиторий	Доступны все аудитории, которые существовали на момент последнего обновления	Доступны все аудитории, которые существовали на момент последнего обновления
Просмотр списка групп	Доступны все группы, которые существовали на момент последнего обновления	Доступны все группы, которые существовали на момент последнего обновления
Просмотр списка преподавателей	Доступны все преподаватели, которые существовали на момент последнего обновления	Доступны все преподаватели, которые существовали на момент последнего обновления
Появление сети	Восстанавливается полноценный функционал программного средства	Восстанавливается полноценный функционал программного средства

5.2.2 Тестирование выхода и повторного входа приложение

Так как программное средство имеет постоянно хранилище, при выходе из приложения с выгрузкой из оперативной памяти и без.

В случаях, когда приложение остаётся в оперативной памяти, оно должно быть

Результаты тестирования без выгрузки приложения из оперативной памяти приведены в таблице:

Таблица 5.2 – Результаты тестирования без выгрузки из оперативной памяти

Тест	Ожидаемый результат	Полученный результат
Просмотр расписания	После возвращения в приложение расписание остаётся на той же дате, происходит обновление зависящих от времени параметров, таких, как затемнение прошедших занятий и строки с относительными датами	После возвращения в приложение расписание остаётся на той же дате, происходит обновление зависящих от времени параметров, таких, как затемнение прошедших занятий и строки с относительными датами
Просмотр избранных	После возвращения в приложение обновляется относительное главное расписание, расписание соответствует моменту времени	После возвращения в приложение обновляется относительное главное расписание, расписание соответствует моменту времени

В случае же с выгрузкой приложения из оперативной памяти, после повторного входа в приложения, открытые представления удаляются, а пользователю должен отобразиться представление избранных.

Таблица 5.3 – Результаты тестирования с выгрузкой из оперативной памяти

Тест	Ожидаемый результат	Полученный результат
Повторный вход	Отображается представление избранных	Отображается представление избранных
Сохранение избранных	После повторного входа отображается все сохранённые в предыдущей сессии избранные	После повторного входа отображается все сохранённые в предыдущей сессии избранные
Сохранение заданий	После повторного входа отображается все сохранённые в предыдущей сессии задания	После повторного входа отображается все сохранённые в предыдущей сессии задания

6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

6.1 Системные требования

Для работы программного средства необходимо любое устройство с установленной на нём версией iOS или iPad OS версии 16,0, либо выше.

Полный список поддерживаемых iPhone: iPhone 14 Pro Max, iPhone 14 Pro, iPhone 14 Plus, iPhone 14, iPhone 13 Pro Max, iPhone 13 Pro, iPhone 13 mini, iPhone 13, iPhone 12 Pro Max, iPhone 12 Pro, iPhone 12 mini, iPhone 12, iPhone SE (2-го поколения), iPhone 11 Pro Max, iPhone 11 Pro, iPhone 11, iPhone XS Max, iPhone XS, iPhone XR, iPhone X, iPhone 8 Plus, iPhone 8, iPhone 7 Plus, iPhone 7, iPhone 6s Plus, iPhone 6s, iPhone SE (1-го поколения).

Полный список поддерживаемых iPad: iPad Pro (12,9-дюймовый, 5-го поколения), iPad Pro (11-дюймовый, 3-го поколения), iPad Pro (12,9-дюймовый, 4-го поколения), iPad Pro (11-дюймовый, 2-го поколения), iPad Pro (12,9-дюймовый, 3-го поколения), iPad Pro (11-дюймовый, 1-го поколения), iPad Pro (12,9-дюймовый, 2-го поколения), iPad Pro (10,5-дюймовый), iPad Pro (12,9-дюймовый, 1-го поколения), iPad Air (4-го поколения), iPad Air (3-го поколения), iPad Air (2-го поколения), iPad (9-го поколения), iPad (8-го поколения), iPad (7-го поколения), iPad mini (6-го поколения), iPad mini (5-го поколения).

Таже приложение доступно на macOS с использованием технологии Mac Catalyst. Полный список поддерживаемых Mac: MacBook Air (с чипом M1), MacBook Pro 13" (с чипом M1), Mac mini (с чипом M1), iMac 24" (с чипом M1), .MacBook Pro 14" (с чипом M1 Pro или M1 Max), MacBook Pro 16" (с чипом M1 Pro или M1 Max).

При первом запуске приложения и для обновлений данных необходимо сетевое соединение.

6.2 Установка из магазина приложений Apple App Store

После выхода приложения в Apple App Store ссылка на страницу приложения будет содержаться в репозитории проекта на GitHub [8].

Для установки приложения необходимо перейти по ссылке из репозитория, войти в Apple ID, и нажать на кнопку `установить`.

6.3 Установка в ручном режиме

Так как iOS и iPadOS являются закрытыми ОС, а политика Apple предусматривает полную закрытость для установки сторонних приложений через установочный файлы без корпоративной сертификации. Альтернативным вариантом является добавление устройства, на которое предстоит установка, в список устройств в аккаунте разработчика Apple, однако этот вариант также является неприемлемым, так как не позволяет

пользователю установить программное средство самостоятельно.

Поэтому для самостоятельной установки требуется использовать устройство с установленной macOS и выполнять следующие шаги:

Шаг 1. Установить Xcode.

Шаг 2. Загрузить исходные файлы из репозитория проекта, либо использовать исходные файлы, которые приложены на компакт-диске.

Шаг 3. Открыть файл bsuirSchedule.xcodeproj.

Шаг 4. Подключить устройство к компьютеру.

Шаг 5. Выбрать меню Product, в нём нажать на Run.

6.4 Краткое руководство пользователя

При первом запуске приложения отображается представление с приветствием, которое помогает пользователю провести первоначальную конфигурацию и описывает функционал приложения.

Некоторая функциональность, которая не описана в данном представлении, представлена в данном подразделе.

6.4.1 Изменение параметров типа занятия

Для изменения параметров типа занятия, таких, как название, аббревиатура и цвет необходимо:

Шаг 1. В представлении избранных воспользоваться кнопкой перехода на представление `настроек`, которая расположена в правом верхнем углу.

Шаг 2. Выбрать `типы занятий`.

Шаг 3. Выбрать тип занятия, параметры которого необходимо изменить.

Шаг 4. Изменить параметры. Если они являются текстовыми, необходимо нажать кнопку `ввод`.

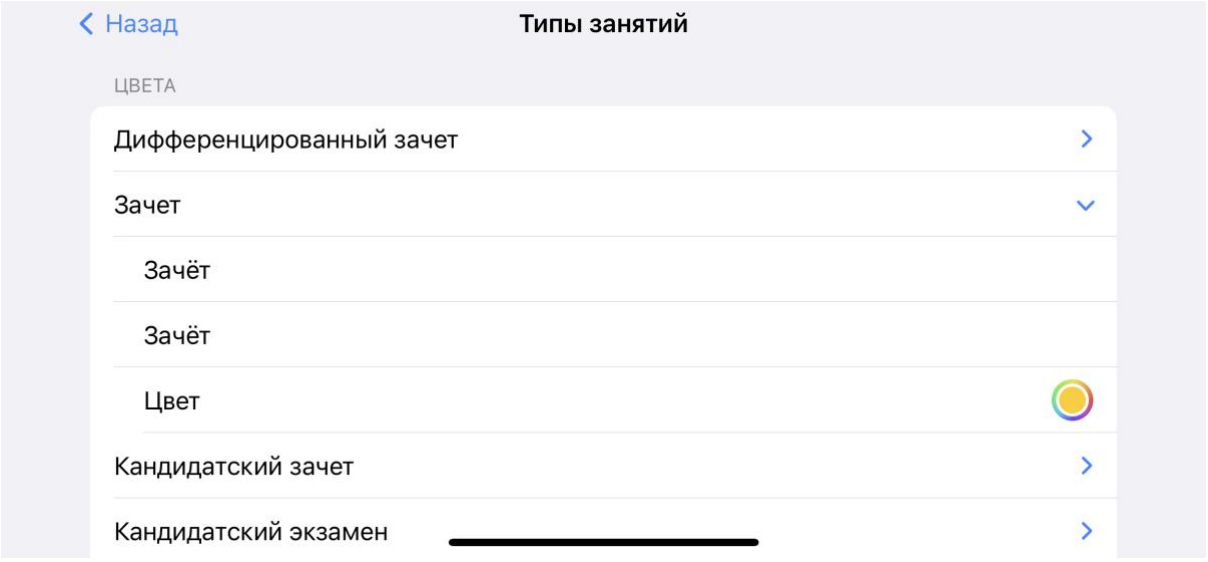


Рисунок 6.1 – выбор основного цвета приложения

6.4.2 Выбор основного цвета приложения

Для выбора основного цвета приложения, которым выделяются все кнопки и некоторые иные системные компоненты необходимо:

Шаг 1. В представлении избранных воспользоваться кнопкой перехода на представление `настроек`, которая расположена в правом верхнем углу.

Шаг 2. Выбрать пункт `основной цвет`.

Шаг 3. Изменить цвет одним из трёх предложенных методов.

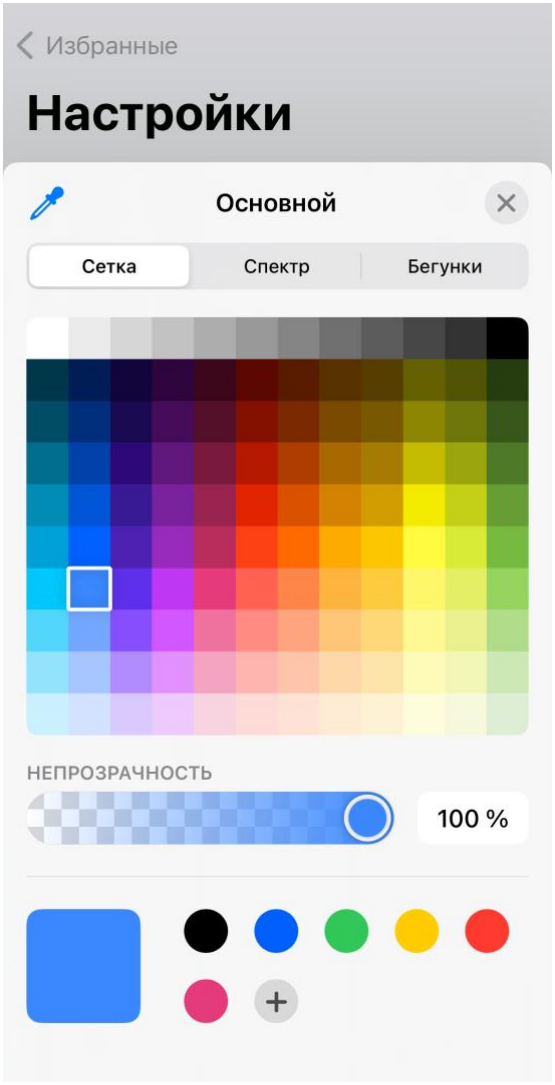


Рисунок 6.2 – выбор основного цвета приложения

7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И РЕАЛИЗАЦИИ НА РЫНКЕ ПРОГРАММНОГО СРЕДСТВА ДЛЯ ПРОСМОТРА И ОБРАБОТКИ ИНФОРМАЦИИ О РАСПИСАНИИ ЗАНЯТИЙ

7.1 Характеристика программного средства, разрабатываемого для реализации на рынке

Созданный дипломный проект представляет собой нативное приложение для ОС iOS и iPadOS, которое позволяет студентам и сотрудникам БГУИР просматривать расписание занятий и экзаменов для групп, преподавателей и кабинетов, а также общее расписание для нескольких сущностей; просматривать подробную информацию о группах, преподавателях, кабинетах, специальностях; добавлять собственные занятия; добавлять и отслеживать задания.

Целью разработки данного проекта является упрощение получения и обработки данных о занятиях и другой информации. Приложение предназначено для использования в учебных целях и помогает студентам и сотрудникам эффективнее планировать своё время, управлять заданиями и получать своевременную информацию об изменениях в учебном расписании.

Целевой аудиторией данного приложения являются студенты, и сотрудники БГУИР, которым необходим удобный инструмент для получения информации из API ИИС. Также потенциальными покупателями могут быть иные учебные заведения, которые могут быть заинтересованы в использовании данного приложения.

На iOS в момент разработки существует два приложения со схожим функционалом: BSUIR Schedule, BSUIR Timetable. Существующие решения используют данные ИИС БГУИР не в полном объёме. Функционал этих решений ограничивается отображением расписания преподавателей и групп, однако API предоставляет больше информации: кабинеты и их типы, название специальностей, факультетов и кафедры, отношение преподавателей к кафедрам и другое. На iPadOS существует только BSUIR Schedule.

Планируется распространение приложения через Apple App Store с использованием модели монетизации с полностью платной версией.

7.2 Расчёт инвестиций в разработку программного средства

7.2.1 Расчёт зарплат на основную заработную плату разработчиков

Расчёт затрат на основную заработную плату разработчиков производится исходя из количества людей, которые занимаются разработкой программного продукта, месячной зарплаты каждого участника процесса разработки и сложности выполняемой ими работы. Затраты на основную заработную плату рассчитаны по формуле:

$$З_о = К_{пр} \sum_{i=1}^{n} З_{чi} \cdot t_i, \quad (7.1)$$

где $К_{пр}$ – коэффициент премий и иных стимулирующих выплат;

n – категории исполнителей, занятых разработкой программного средства;

$З_{чi}$ – часовая заработная плата исполнителя i-й категории, р;

t_i – трудоемкость работ, выполняемых исполнителем i-й категории, ч.

Разработкой всего приложения занимается инженер-программист, Обязанности тестирования приложения лежат на инженере-тестировщике. Задачами инженера-программиста, который занимается являются создание модели данных, графического интерфейса, связи между моделью данных и графическим интерфейсом. Инженер-тестировщик занимается выявлением неработоспособных частей приложения, а также оценивает пользовательский опыт, получаемый от использования приложения.

Месячная заработная плата основана на медианных показателях для Junior инженера-программиста за 2023 год по Республике Беларусь, которая составляет 910 Долларов США в месяц, а для Junior инженера-тестировщика – 600 Долларов США [9]. По состоянию на 15 апреля 2023 года, 1 Доллар США по курсу Национального Банка Республики Беларусь составляет 2,9441 Белорусских рублей [10].

В перерасчёте на Белорусские рубли месячные оклады для инженера-программиста и инженера-тестировщика соответственно составляют составляет 2 679,13 и 1 760,46 Белорусских рублей соответственно.

Часовой оклад исполнителей высчитывается путём деления их месячного оклада на количество рабочих часов в месяце, то есть 160 часов.

За количество рабочих часов в месяце для инженера-программиста и инженера-тестировщика принято соответственно 196 и 32 часа.

Коэффициент премии приравнивается к единице, так как она входит сумму заработной платы. Затраты на основную заработную плату приведены в таблице:

Таблица 7.1 – Затраты на основную заработную плату

Категория исполнителя	Месячный оклад, р	Часовой оклад, р	Трудоёмкость работ, ч	Итого, р
Инженер-программист	2 679,13	16,74	196	3 281,04
Инженер-тестировщик	1 760,46	11,00	32	352,00
Итого				3 633,04
Премия и иные стимулирующие выплаты (0%)				0
Всего затраты на основную заработную плату разработчиков				3 633,04

7.2.2 Расчёт затрат на дополнительную заработную плату разработчиков

Расчёт затрат на дополнительную заработную плату команды разработчиков рассчитывается по формуле:

$$З_д = \frac{З_о \cdot Н_д}{100}, \quad (7.2)$$

где $Н_д$ – норматив дополнительной заработной платы.

Значение норматива дополнительной заработной платы принимает за 10 %.

7.2.3 Расчёт отчислений на социальные нужды

Размер отчислений на социальные нужды определяется согласно ставке отчислений, которая на апрель 2023 г. равняется 35%: 29% отчисляется на пенсионное страхование, 6% – на социальное страхование. Расчёт отчислений на социальные нужды вычисляется по формуле:

$$Р_{соц} = \frac{(З_о + З_д) \cdot Н_{соц}}{100}, \quad (7.3)$$

где $Н_{соц}$ – норматив отчислений в ФСЗН.

7.2.4 Расчёт прочих расходов

Расчёт затрат на прочие расходы определяется при помощи норматива прочих расчётов. Эта величина имеет значение 30%. Расчёт прочих расходов вычисляется по формуле:

$$Р_{пр} = \frac{З_о \cdot Н_{пр}}{100}, \quad (7.4)$$

где $Н_{пр}$ – норматив прочих расходов.

7.2.5 Расчёт расходов на реализацию

Для того, чтобы рассчитать расходы на реализацию, необходимо знать норматив расходов на неё. Принимаем значение норматива равным 3%. Формула, которая использована для расчёта расходов на реализацию:

$$Р_р = \frac{З_о \cdot Н_р}{100}, \quad (7.5)$$

где $Н_р$ – норматив расходов на реализацию.

7.2.6 Расчёт общей суммы затрат на разработку и реализацию

Определяем общую сумму затрат на разработку и реализацию как сумму ранее вычисленных расходов: на основную заработную плату разработчиков, дополнительную заработную плату разработчиков, отчислений на социальные нужды, расходы на реализацию и прочие расходы. Значение определяется по формуле:

$$З_р = З_о + З_д + Р_{соц} + Р_{пр} + Р_р \quad (7.6)$$

Таким образом, величина затрат на разработку программного средства высчитывается по указанной выше формуле и указана в таблице:

Таблица 7.2 – Затраты на разработку

Название статьи затрат	Формула/таблица для расчёта	Значение, р.
1. Основная заработная плата разработчиков	См. таблицу 7.1	3 633,04
2. Дополнительная заработная плата разработчиков	$З_д = \frac{3\,633{,}04 \cdot 10}{100}$	363,30
3. Отчисление на социальные нужды	$Р_{соц} = \frac{(3\,633{,}04 + 363{,}3) \cdot 35}{100}$	1 398,71
4. Прочие расходы	$Р_{пр} = \frac{3\,633{,}04 \cdot 30}{100}$	1 089,91
5. Расходы на реализацию	$Р_р = \frac{3\,633{,}04 \cdot 3}{100}$	108,99
6. Общая сумма затрат на разработку и реализацию	$З_р = 3\,633{,}04 + 363{,}3 + +1\,398{,}71 + 1\,089{,}91 + 108{,}99$	6 593,95

7.3 Расчёт экономического эффекта от реализации программного средства на рынке

Для расчёта экономического эффекта организации-разработчика программного средства, а именно чистой прибыли, необходимо знать такие параметры как объем продаж, цену реализации и затраты на разработку.

Соответственно необходимо создать обоснование возможного объёма продаж, количества проданных лицензий расширенной версии программного средства, купленного пользователями. В БГУИР обучается примерно 16 000

студентов, а работают в университете более 2 200 человек, что в сумме составляет примерно 18 000 человек [11]. Процент пользователей в Республике Беларусь, которые используют iOS среди других мобильных ОС [12], iPadOS среди других планшетных ОС [13] и macOS среди других настольных ОС [14] на март 2023 года соответственно составляют 36,97%, 40,08% и 14,13%.

Учитывая высокую для студентов стоимость устройств компании Apple, отсутствие информации о количестве пользователей, распределённом по возрастным и социальным группам, примем за процент перспективных пользователей приложения 30% от общего числа связанных с БГУИР людей, то есть 5 400 человек. Допустим, что из них 4 000 человек установят приложение, а из них 2 000 человек приобретут расширенную версию.

Стоимость в App Store задаётся по конкретным заранее определённым компанией уровням, то есть установить любую стоимость нельзя. Расчёты в белорусском регионе App Store ведутся в Долларах США. Так как функции расширенной версии не представлены ни в одном другом приложении и не имеют аналогов, наиболее оптимальной ценой данной версии предполагается 7,99 Долларов США, что с вычетом комиссии Apple в 30% составляет 5,6 Доллара США. Таким образом, отпускная цена копии программного средства составляет 16,48 Белорусских рубля.

Для расчёта прироста чистой прибыли необходимо учесть налог на добавленную стоимость, который высчитывается по следующей формуле:

$$НДС = \frac{Ц_{отп} \cdot N \cdot Н_{д.с}}{100\% + Н_{д.с}}, \quad (7.7)$$

где N – количество копий(лицензий) программного продукта, реализуемое за год, шт.;

$Ц_{отп}$ – отпускная цена копии программного средства, р. ;

N – количество приобретённых лицензий;

$Н_{д.с}$ – ставка налога на добавленную стоимость, %.

Ставка налога на добавленную стоимость по состоянию на 15 апреля 2023 года, в соответствии с действующим законодательством Республики Беларусь, составляет 20%. Используя данное значение, посчитаем НДС:

$$НДС = \frac{16{,}48 \cdot 2\,000 \cdot 20\%}{100\% + 20\%}, = 5\,493{,}33 \text{ р.} \quad (7.8)$$

Посчитав налог на добавленную стоимость, можно рассчитать прирост чистой прибыли, которую получит разработчик от продажи программного продукта. Для этого используется формула:

$$\Delta П_{ч}^{р} = (Ц_{отп} \cdot N - НДС) \cdot Р_{пр} \cdot \left(1 - \frac{Н_{п}}{100}\right), \quad (7.9)$$

где N – количество копий(лицензий) программного продукта, реализуемое за год, шт.;

$Ц_{отп}$ – отпускная цена копии программного средства, р.;

НДС – сумма налога на добавленную стоимость, р.; $Н_{п}$ – ставка налога на прибыль, %;

$Р_{пр}$ – рентабельность продаж копий;

$Р_{пр}$ – рентабельность продаж копий.

Ставка налога на прибыль, согласно действующему законодательству, по состоянию на 14.04.2023 равна 20%. Рентабельность продаж копий взята в размере 40%. Зная ставку налога и рентабельность продаж копий (лицензий), рассчитывается прирост чистой прибыли для разработчика:

$$\Delta П_{ч}^{р} = (16{,}48 \cdot 2\,000 - 5\,493{,}33) \cdot 40\% \cdot \left(1 - \frac{20}{100}\right) = 8\,789{,}33 \quad (7.10)$$

7.4 Расчёт показателей экономической эффективности разработки и реализации программного средства на рынке

Для того, чтобы оценить экономическую эффективность разработки и реализации программного средства на рынке, необходимо рассмотреть результат сравнения затрат на разработку данного программного продукта, а также полученный прирост чистой прибыли за год.

Сумма затрат на разработку меньше суммы годового экономического эффекта, поэтому можно сделать вывод, что такие инвестиции окупятся менее, чем за один год.

Таким образом, оценка экономической эффективности инвестиций производится при помощи расчёта рентабельности инвестиций (Return on Investment, ROI). Формула для расчёта ROI:

$$ROI = \frac{\Delta П_{ч}^{р} - З_{р}}{З_{р}} \cdot 100\% \quad (7.11)$$

где $\Delta П_{ч}^{р}$ – прирост чистой прибыли, полученной от реализации программного средства на рынке информационных технологий, р.;

$З_{р}$ – затраты на разработку и реализацию программного средства, р.

$$ROI = \frac{8\,789{,}33 - 6\,593{,}95}{6\,593{,}95} \cdot 100\% = 33{,}29\% \quad (7.12)$$

7.5 Вывод об экономической целесообразности реализации проектного решения

Проведённые расчёты технико-экономического обоснования позволяют сделать предварительный вывод о целесообразности разработки данного программного продукта. Общая сумма затрат на разработку и реализацию составила 6 593,95 Белорусских рублей, а отпускная цена была установлена на уровне 16,48 Белорусских рублей.

Прирост чистой прибыли за год, исходя из предполагаемого объёма продаж в размере 2000 расширенных версий в год, составляет 8 789,33 Белорусских рублей. Рентабельность инвестиций за год составляет 33,29%.

Это означает, что разработка данного программного продукта является целесообразной и реализация программного средства по установленной цене имеет смысл.

Однако, следует учитывать возможные риски, связанные с конкуренцией со стороны аналогов, что может привести к незамеченности продукта на рынке. Кроме того, высокая рентабельность связана с рисками, и расчётные результаты были получены при предполагаемом объёме продаж в 2000 копий в год.

Тем не менее, при поддержке проект может получить долгосрочное и успешное развитие, и количество проданных копий может превысить предполагаемое количество.

Масштабирование проекта на другие учебные заведения может способствовать его успешному развитию. В целом, инвестирование в предложенный проект также оправдано.

ЗАКЛЮЧЕНИЕ

Дипломный проект полностью завершён и подготовлен для выхода на рынок. За время дипломного проектирования разработано программное средство со всем предполагаемым с периода проектирования функционалом. Приложение стабильно функционирует на всех поддерживаемых устройствах.

Созданная реляционная база данных функционирует и позволяет хранить все необходимые для работы приложения данные на устройстве. Для получения доступа к основному функционалу приложения не требуется активное сетевое соединение.

Алгоритмы для обработки информации оптимизированы и разделены для эффективного использования. Это позволяет приложению быстро обрабатывать данные. Полная загрузка и обработка всех занятий университета, которая необходима для создания расписания занятий по аудиториям, занимает менее 45 секунд.

Разработан адаптивный пользовательский интерфейс, который эффективно использует разные размеры дисплеев поддерживаемых устройств, что позволяет просматривать большее количество информации на дисплеях с большей диагональю. Поддерживается тёмная тема.

Созданы широкие возможности по кастомизации пользовательского интерфейса:

- выбор типа отображения расписания занятий;
- сортировки на секции разного типа для аудиторий, групп и преподаватели;
- выбор основного цвета приложения;
- выбор названий, аббревиатур и цветов для типов занятий;
- выбор отображаемых для занятия параметров, таких, как: группы, преподаватели, учебные недели, период проведения и дата проведения занятия.

Отличительной особенностью приложения является главная страница, на которой отображаются избранные аудитории, группы, преподаватели и подразделения, а также задачи. Данное представление позволяет пользователю быстро перейти к необходимому расписанию, либо получить иную информацию. Также на данном экране отображается основное расписание пользователя, которое всегда отображает ближайший день и занятие сразу при входе в приложение. При этом время начала занятия отображается в относительном формате, то есть, например, в день занятия, будет показано, сколько часов и минут осталось до его начала.

Для удобства пользователей создана система заданий, которая позволяет пользователю добавлять и изменять собственные задания. Отслеживать сроки их выполнения. При создании задания существует возможность выбрать срок выполнения не по дате, а назначить его на конкретную предложенную пару.

Среди достоинств разработанного программного средства можно выделить следующее:

- открытый исходный код;

- возможность добавлять и отслеживать собственные задания;
- возможность просматривать расписание кабинетов;
- возможность перехода между расписаниями через занятия, например, от занятия группы перейти напрямую в расписание ведущего её преподавателя;
- автоматическое обновление расписания;
- возможность просмотра статистической информации;
- широкие возможности кастомизации;
- отсутствие встроенной рекламы.

Среди недостатков разработанного программного средства можно выделить следующее:

- отсутствие виджетов;
- высокие системные требования, выраженные в поддержке устройств только на iOS 16,0 и выше;
- отсутствие полноценной версии для macOS;
- отсутствие системы уведомлений о приближающихся занятиях и сроках выполнения заданий.

Дальнейшему развитию приложения может поспособствовать открытый исходный код, который доступен на web-сервисе GitHub, где другие пользователи могут сообщать об ошибках, предлагать и реализовывать улучшения программного средства.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Интегрированная информационная система «БГУИР: Университет» [Электронный ресурс]. – Документация – Режим доступа: https://iis.bsuir.by/api – Дата доступа: 18.04.2023

[2] Apple App Store [Электронный ресурс]. – BSUIR Timetable – Режим доступа: https://apps.apple.com/by/app/bsuir-timetable/id1443305662 – Дата доступа: 19.04.2023

[3] Apple App Store [Электронный ресурс]. – Bsuir Schedule – Режим доступа: https://apps.apple.com/by/app/bsuir-schedule/id944151090 – Дата доступа: 19.04.2023

[4] GitHub [Электронный ресурс]. – Bsuir Schedule – Режим доступа: https://github.com/asiliuk/BsuirScheduleApp – Дата доступа: 19.04.2023

[5] Apple Developer [Электронный ресурс]. – UIKit – Режим доступа: https://developer.apple.com/documentation/uikit – Дата доступа: 17.04.2023

[6] Apple Developer [Электронный ресурс]. – SwiftUI – Режим доступа: https://developer.apple.com/xcode/swiftui/ – Дата доступа: 17.04.2023

[7] Apple Developer [Электронный ресурс]. – Interfacing with UIKit – Режим доступа: https://developer.apple.com/tutorials/swiftui/interfacing-with-uikit – Дата доступа: 17.04.2023

[8] Web-сервис «GitHub» [Электронный ресурс]. – bsuirSchedule – Режим доступа: https://github.com/andrejHurynovic/bsuirSchedule – Дата доступа: 18.05.2023

[9] Интернет-издание «Dev.by» [Электронный ресурс]. – Зарплата в ИТ – Режим доступа: https://salaries.devby.io – Дата доступа: 15.04.2023

[10] Национальный банк Республики Беларусь [Электронный ресурс]. – Официальные курсы белорусского рубля по отношению к иностранным валютам, устанавливаемые Национальным банком Республики Беларусь ежедневно, на 15.04.2023 – Режим доступа: https://www.nbrb.by/statistics/rates/ratesdaily.asp – Дата доступа: 15.04.2023

[11] Белорусский государственный университет информатики и радиоэлектроники [Электронный ресурс]. – БГУИР сегодня – Режим доступа: https://www.bsuir.by/ru/bguir-segodnya – Дата доступа: 15.04.2023

[12] Statcounter GlobalStats [Электронный ресурс]. – Mobile Operating System Market Share Belarus – Режим доступа: https://gs.statcounter.com/os-market-share/mobile/belarus – Дата доступа: 15.04.2023

[13] Statcounter GlobalStats [Электронный ресурс] Tablet Operating System Market Share Belarus – Режим доступа: https://gs.statcounter.com/os-market-share/tablet/belarus – Дата доступа: 15.04.2023

[14] Statcounter GlobalStats [Электронный ресурс]. – Desktop Operating System Market Share Belarus – Режим доступа: https://gs.statcounter.com/os-market-share/desktop/belarus – Дата доступа: 15.04.2023

ПРИЛОЖЕНИЕ А

(обязательное)

Листинг ключевых фрагментов программы

```
//
//  Auditorium.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 25.09.21.
//

import CoreData

extension Auditorium {
    @nonobjc public class func fetchRequest() -> NSFetchRequest<Auditorium> {
        let request = NSFetchRequest<Auditorium>(entityName: "Auditorium")
        request.sortDescriptors = [
            NSSortDescriptor(keyPath: \Auditorium.outsideUniversity, ascending: false),
            NSSortDescriptor(keyPath: \Auditorium.building, ascending: true),
            NSSortDescriptor(keyPath: \Auditorium.formattedName, ascending: true)]
        return request
    }

    @NSManaged public var floor: Int16
    @NSManaged public var name: String
    @NSManaged public var formattedName: String
    ///Used for constraints and effective search when decoding Lessons.
    @NSManaged public var note: String?
    @NSManaged public var favorite: Bool

    @NSManaged public var outsideUniversity: Bool
    @NSManaged public var building: Int16
    @NSManaged public var capacity: Int16

    @NSManaged public var type: AuditoriumType?
    @NSManaged public var department: Department?

    @NSManaged public var lessons: NSSet?
}

//MARK: - Generated accessors
extension Auditorium {
    @objc(addLessonsObject:)
    @NSManaged public func addToLessons(_ value: Lesson)
    @objc(removeLessonsObject:)
    @NSManaged public func removeFromLessons(_ value: Lesson)
    @objc(addLessons:)
    @NSManaged public func addToLessons(_ values: NSSet)
    @objc(removeLessons:)
    @NSManaged public func removeFromLessons(_ values: NSSet)
}

//
//  AuditoriumDecoder.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 25.09.21.
//
//

import CoreData

@objc(Auditorium)
public class Auditorium: NSManagedObject {

    required convenience public init(from decoder: Decoder) throws {
        let container = try! decoder.container(keyedBy: CodingKeys.self)

        let buildingContainer = try! container.nestedContainer(keyedBy: BuildingCodingKeys.self,
forKey: .building)
        let buildingString = try! buildingContainer.decode(String.self, forKey: .name)

        //This code does not allow the creation of Auditories in non-educational buildings. The
```

```
description of the Algorithm is provided in decodeBuilding() method.
        guard let _ = Int16(buildingString.trimmingCharacters(in: CharacterSet.init([" ", "к",
"."]))) else {
            Log.warning("Fetched non-educational building, (\((try? container.decode(String.self,
forKey: .name)) ?? "No name")-\(buildingString)).")
            throw AuditoriumDecodingError.nonEducationalBuilding
        }

        self.init(context: decoder.userInfo[.managedObjectContext] as! NSManagedObjectContext)

        try self.update(from: decoder)
    }

    ///This function creates a new Auditorium object from a string in the format ["000-1 к.", "604-
5 к.", "epam 104 к1-2 к."].
    convenience public init(from string: String, in context: NSManagedObjectContext) throws {
        self.init(entity: Auditorium.entity(), insertInto: context)

        //"604-5 к." -> "5 к.".
        let buildingString = String(string.suffix(4))
        //"604-5 к." -> "604".
        let nameString = String(string.dropLast(5))
        try decodeBuilding(string: buildingString)
        try decodeName(string: nameString)
        self.formattedName = formateName()

        Log.info("Auditorium \(string) is created from string.")
    }

}

//MARK: - Update
extension Auditorium: DecoderUpdatable {
    func update(from decoder: Decoder) throws {
        let container = try! decoder.container(keyedBy: CodingKeys.self)

        //MARK: - Building container
        let buildingContainer = try! container.nestedContainer(keyedBy: BuildingCodingKeys.self,
forKey: .building)
        let buildingString = try! buildingContainer.decode(String.self, forKey: .name)
        let nameString = try! container.decode(String.self, forKey: .name)

        try decodeBuilding(string: buildingString)
        try decodeName(string: nameString)
        self.formattedName = formateName()

        self.capacity = (try? container.decode(Int16.self, forKey: .capacity)) ?? 0
        self.note = try? container.decode(String.self, forKey: .note)
        self.type = try! container.decode(AuditoriumType.self, forKey: .type)
        self.department = try? container.decode(Department.self, forKey: .department)
        Log.info("Auditorium \(String(self.formattedName)) fetched.")
    }

    ///Decodes string to Int16 building number or, in the case of non-educational building, throws
an error.
    ///
    ///Generally a string is ["1 к.", "2 к.", "3 к.", "4 к.", "5 к.", "6 к.", "7 к.", "8 к."], but
in some cases it may be ["Общежитие №4", "Филиал «Минский радиотехнический колледж»"].
    ///In order not to create unnecessary Auditories, the " к." part is removed from a string,
making string able to cast into Int16. If string is not casted to Int16, then building is non-
educational.
    ///Building container have "id" field, however there is no pattern between an id and building
number, the more universal solution is to use the "name" filed.
    private func decodeBuilding(string: String) throws {
        guard let building = Int16(string.trimmingCharacters(in: CharacterSet.init([" ", "к",
"."]))) else {
            Log.warning("Non-educational building \(string).")
            throw AuditoriumDecodingError.nonEducationalBuilding
        }
        self.building = building
    }

    ///Decodes string to floor and name, checks if is auditorium is outside of university.
    ///
    ///Generally a string is ["000", "010", "019", "04", "05", "06", "07", "08", "1 - 2", "102"],
but in some cases is may be ["epam 103 к1",, "epam 401-1", "ОАО \"Планар\""].
    private func decodeName(string: String) throws {
        //If the first character is not a digit, the Auditorium is located outside the university,
```

```
in this case there is no need to specify floor number.
        //Example names: ["epam 103 к1",, "epam 401-1", "ОАО \"Планар\""].
        if string.first!.isNumber == false {
            self.name = string
            self.outsideUniversity = true
            return
        }
        //If the Auditorium is located at the university, the name is separated into the floor
number (Int16) and the name (String).
        //Example:
        //"000"     -> floor == 0, name == "00";
        //"04"      -> floor == 0, name == "04";
        //"04a"     -> floor == 0, name == "04a";
        //"39"      -> floor == 0, name == "39";
        //"111(3)"  -> floor == 1, name == "11(3)";
        //"213б"    -> floor == 2, name == "13б";
        //"302-1"   -> floor == 3, name == "02-1".

        //
        guard let number = Int(string.prefix(3)) ?? Int(string.prefix(2)) else {
            Log.warning("Can't create name for Auditorium \(string)")
            throw AuditoriumDecodingError.incorrectName(name: string)
        }

        //If number is less then 100 and first character is not "0", then these are Auditories with
names such as ["34", "69"].
        //Floor is specified as 0. Name is specified as full string (with first character).
        //Example: "34" -> floor == 0, name == "34".
        if number < 100, string.first != "0" {
            self.floor = 0
            self.name = string
            return
        }

        //If number is greater then 100, then these are Auditories on higher floors with names such
as [100, 234, 325].
        //First character of string is floor number, it is casted to Int16.
        let floor = Int16(String(string.first!))!
        //Name is assigned as string without first character (which represents floor).
        let name = String(string.dropFirst(1))
        self.floor = floor
        self.name = name
    }

    private func formateName() -> String {
        if self.outsideUniversity {
            return self.name
        }
        return "\(self.floor)\(self.name)-\(self.building)"
    }

}

extension Auditorium: Decodable {
    private enum CodingKeys: String, CodingKey {
        case id
        case name
        case note
        case capacity

        case building = "buildingNumber"
        case type = "auditoryType"
        case department = "department"
    }

    private enum BuildingCodingKeys: String, CodingKey {
        case name
    }
    private enum AuditoriumCodingKeys: String, CodingKey {
        case id
    }
}

//
//  AuditoriumDecodingError.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 1.05.23.
```

```
//

import Foundation

enum AuditoriumDecodingError: Error {
    case incorrectName(name: String)
    case nonEducationalBuilding
}

//
//  AuditoriumFetch.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 30.04.23.
//

import CoreData

extension Auditorium: AbleToFetchAll {
    static func fetchAll() async {
        guard let data = try? await URLSession.shared.data(for: .auditories) else { return }
        let startTime = CFAbsoluteTimeGetCurrent()

        guard let dictionaries = try! JSONSerialization.jsonObject(with: data) as? [[String: Any]]
else {
            Log.error("Can't create auditories dictionaries.")
            return
        }

        let backgroundContext = PersistenceController.shared.container.newBackgroundContext()
        backgroundContext.mergePolicy = NSMergeByPropertyStoreTrumpMergePolicy
        let decoder = JSONDecoder()
        decoder.userInfo[.managedObjectContext] = backgroundContext

        let auditories: [Auditorium] = getAll(from: backgroundContext)
        //This is required because decoder actually can throw an error here, so we can't decode
whole array instantly.
        for dictionary in dictionaries {
            let data = try! JSONSerialization.data(withJSONObject: dictionary)

            let name = dictionary["name"] as! String
            let buildingDictionary = dictionary["buildingNumber"] as! [String: Any]
            let buildingString = buildingDictionary["name"] as! String


            if var auditorium = auditories.first (where: {
                if $0.outsideUniversity {
                    return "\($0.name)-\($0.building)" == "\(name)-\(buildingString.first!)"
                } else {
                    return  $0.formattedName == "\(name)-\(buildingString.first!)"
                }
            }) {
                try? decoder.update(&auditorium, from: data)
            } else {
                let _ = try? decoder.decode(Auditorium.self, from: data)
            }
        }
        await backgroundContext.perform(schedule: .immediate, {
            try! backgroundContext.save()
            Log.info("\(String((self.getAll(from:  backgroundContext)  as  [Auditorium]).count))
Auditories  fetched,  time:  \((CFAbsoluteTimeGetCurrent()  -  startTime).roundTo(places:  3))
seconds.\n")
        })
    }

}

//
//  AuditoriumExtensions.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 30.04.23.
//

import CoreData

extension Auditorium: Identifiable {}
extension Auditorium: Favored {}
```

```
extension Auditorium: EducationRanged { }

extension Auditorium: Comparable {
    public static func < (lhs: Auditorium, rhs: Auditorium) -> Bool {
        if lhs.building < rhs.building {
            return true
        }
        if lhs.formattedName < rhs.formattedName {
            return true
        }
        return false
    }
}

extension Auditorium: Scheduled {
    var title: String { self.formattedName }
}

extension Auditorium {
    var groups: [Group]? {
        guard let lessons = lessons?.allObjects as? [Lesson], !lessons.isEmpty else { return nil }
        let groups = lessons.compactMap { $0.groups?.allObjects as? [Group] }.flatMap { $0 }
        return Set(groups).sorted { $0.id < $1.id }
    }
}

//
//  AuditoriumExtensions.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 30.04.23.
//

import CoreData

extension Auditorium: Identifiable {}
extension Auditorium: Favored {}

extension Auditorium: EducationRanged { }

extension Auditorium: Comparable {
    public static func < (lhs: Auditorium, rhs: Auditorium) -> Bool {
        if lhs.building < rhs.building {
            return true
        }
        if lhs.formattedName < rhs.formattedName {
            return true
        }
        return false
    }
}

extension Auditorium: Scheduled {
    var title: String { self.formattedName }
}

extension Auditorium {
    var groups: [Group]? {
        guard let lessons = lessons?.allObjects as? [Lesson], !lessons.isEmpty else { return nil }
        let groups = lessons.compactMap { $0.groups?.allObjects as? [Group] }.flatMap { $0 }
        return Set(groups).sorted { $0.id < $1.id }
    }
}

//
//  AuditoriumSectionType.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 4.04.23.
//

enum AuditoriumSectionType: SectionType {
    case building
    case buildingAndFloor
    case department

    var description: String {
```

```
        switch self {
            case .building:
                return "Корпус"
            case .buildingAndFloor:
                return "Корпус и этаж"
            case .department:
                return "Подразделение"
        }
    }
}

//
//  AuditoriesSection.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 7.10.22.
//

extension Sequence where Element == Auditorium {
    ///Returns array of auditories sections grouped by building and floor
    func sections(_ type: AuditoriumSectionType) -> [NSManagedObjectsSection<Auditorium>] {
        switch type {
            case .building:
                return buildingBasedSections(floorSectioned: false)
            case .buildingAndFloor:
                return buildingBasedSections(floorSectioned: true)
            case .department:
                return departmentBasedSections()
        }
    }


    ///This function creates building sections for the array of Auditories.
    ///If floorSectioned is true, the sections will be grouped by building and floor.
    ///If floorSectioned is false, the sections will be grouped by building only.
    private func buildingBasedSections(floorSectioned: Bool) ->
[NSManagedObjectsSection<Auditorium>] {
        var sections: [NSManagedObjectsSection<Auditorium>] = []
        let outsideUniversitySections = self.sectioned(by: \.outsideUniversity)

        if let insideUniversityAuditories = outsideUniversitySections[false] {
            let buildingAuditoriesDictionaries = insideUniversityAuditories.sectioned(by:
\.building)
                .sorted(by: { $0.key < $1.key })
            if floorSectioned {
                sections.append(contentsOf: buildingAuditoriesDictionaries.map
{ buildingDictionary in
                    buildingDictionary.value.sectioned(by: \.floor)
                        .sorted { $0.key < $1.key }
                        .map { NSManagedObjectsSection(title: "\(buildingDictionary.key)-ый корпус,
\($0.key == 0 ? "0-ой этаж" : "\($0.key)-ый этаж")",
                                                       items: $0.value) }
                }
                    .flatMap { $0 }
                )
            } else {
                sections.append(contentsOf:
buildingAuditoriesDictionaries.map( { NSManagedObjectsSection(title: "\($0.key)-ый корпус",

items: $0.value) }))
            }
        }
        if let outsideUniversityAuditories = outsideUniversitySections[true] {
            sections.append(NSManagedObjectsSection(title: "Без корпуса",
                                                    items: outsideUniversityAuditories))
        }
        return sections
    }

    private func departmentBasedSections() -> [NSManagedObjectsSection<Auditorium>] {
        return self.sectioned(by: \.department)
            .sorted(by: {
                if $1.key == nil && $0.key != nil { return true }
                if $0.key == nil && $1.key != nil { return false }
                return $0.key!.abbreviation < $1.key!.abbreviation
            })
            .map { NSManagedObjectsSection(title: $0.key?.abbreviation ?? "Без подразделения",
                                           items: $0.value) }
```

```
    }

}


//
//  AuditoriumType.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 1.04.23.
//

import CoreData

extension AuditoriumType {
    @nonobjc public class func fetchRequest() -> NSFetchRequest<AuditoriumType> {
        let request = NSFetchRequest<AuditoriumType>(entityName: "AuditoriumType")
        request.sortDescriptors = [NSSortDescriptor(keyPath: \AuditoriumType.id, ascending: true)]
        return request
    }

    @NSManaged public var id: Int16
    @NSManaged public var name: String
    @NSManaged public var abbreviation: String

    @NSManaged public var auditories: NSSet?
}

//MARK: - Generated accessors
extension AuditoriumType {
    @objc(addAuditoriesObject:)
    @NSManaged public func addToAuditories(_ value: Auditorium)
    @objc(removeAuditoriesObject:)
    @NSManaged public func removeFromAuditories(_ value: Auditorium)
    @objc(addAuditories:)
    @NSManaged public func addToAuditories(_ values: NSSet)
    @objc(removeAuditories:)
    @NSManaged public func removeFromAuditories(_ values: NSSet)
}

extension AuditoriumType : Identifiable {

}

//
//  AuditoriumTypeDecoder.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 1.04.23.
//

import CoreData

@objc(AuditoriumType)
public class AuditoriumType: NSManagedObject {

    required public convenience init(from decoder: Decoder) throws {
        let container = try! decoder.container(keyedBy: CodingKeys.self)
        self.init(context: decoder.userInfo[.managedObjectContext] as! NSManagedObjectContext)

        self.id = try! container.decode(Int16.self, forKey: .id)
        self.name = try! container.decode(String.self, forKey: .name).capitalizingFirstLetter()
        self.abbreviation = try! container.decode(String.self, forKey: .abbreviation).uppercased()

        Log.info("AuditoriumType \(self.abbreviation) (\(String(self.id))) has been created.")
    }

}

extension AuditoriumType: Decodable {
    private enum CodingKeys: String, CodingKey {
        case id = "id"
        case name = "name"
        case abbreviation = "abbrev"
    }
}

//
```

```
//  Degree.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 10.05.23.
//

import CoreData

extension Degree {
    @nonobjc public class func fetchRequest() -> NSFetchRequest<Degree> {
        let request = NSFetchRequest<Degree>(entityName: "Degree")
        request.sortDescriptors  =  [NSSortDescriptor(keyPath:  \Degree.abbreviation,  ascending:
true)]
        return request
    }

    @NSManaged public var abbreviation: String
    @NSManaged public var name: String?

    @NSManaged public var employees: NSSet?
}

// MARK: Generated accessors
extension Degree {
    @objc(addEmployeesObject:)
    @NSManaged public func addToEmployees(_ value: Employee)
    @objc(removeEmployeesObject:)
    @NSManaged public func removeFromEmployees(_ value: Employee)
    @objc(addEmployees:)
    @NSManaged public func addToEmployees(_ values: NSSet)
    @objc(removeEmployees:)
    @NSManaged public func removeFromEmployees(_ values: NSSet)
}

//
//  DegreeDecoder.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 10.05.23.
//

import CoreData

@objc(Degree)
public class Degree: NSManagedObject {
    required convenience public init(from decoder: Decoder) throws {
        let container = (try? decoder.container(keyedBy: CodingKeys.self)
            .nestedContainer(keyedBy: CodingKeys.self, forKey: .employeeNestedContainer))
        ?? (try! decoder.container(keyedBy: CodingKeys.self))

        guard let degreeString = try? container.decode(String.self, forKey: .degree),
        degreeString.isEmpty == false else {
            throw URLError(.cannotDecodeRawData)
        }
        let context = decoder.userInfo[.managedObjectContext] as! NSManagedObjectContext
        self.init(context: context)

        Log.info("Degree \(self.abbreviation) is created.")
        try! self.update(from: decoder)
    }

    func update(from decoder: Decoder) throws {
        let container = (try? decoder.container(keyedBy: CodingKeys.self)
            .nestedContainer(keyedBy: CodingKeys.self, forKey: .employeeNestedContainer))
        ?? (try! decoder.container(keyedBy: CodingKeys.self))

        let degreeString = try! container.decode(String.self, forKey: .degree)
        if     let     degreeAbbreviationString     =     try?     container.decode(String.self,
forKey: .degreeAbbreviation),
           degreeAbbreviationString.isEmpty == false {
            self.name = degreeString
            self.abbreviation = degreeAbbreviationString
        } else {
            self.abbreviation = degreeString
        }

        Log.info("Degree \(self.abbreviation) is updated.")
    }
```

```
}

extension Degree: Decodable {
    private enum CodingKeys: String, CodingKey {
        case degree = "degree"
        case degreeAbbreviation = "degreeAbbrev"

        case employeeNestedContainer = "employeeDto"
    }
}

//
//  DegreeExtensions.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 10.05.23.
//

import Foundation

extension Degree: Identifiable {  }

extension Degree {
    var formattedName: String {
        name ?? abbreviation
    }
}

//
//  Department.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 1.04.23.
//

import CoreData

extension Department {
    @nonobjc public class func fetchRequest() -> NSFetchRequest<Department> {
        let request = NSFetchRequest<Department>(entityName: "Department")
        request.sortDescriptors = [NSSortDescriptor(keyPath: \Department.abbreviation, ascending:
true)]
        return request
    }

    @NSManaged public var id: Int16
    @NSManaged public var name: String?
    @NSManaged public var abbreviation: String

    @NSManaged public var favorite: Bool

    @NSManaged public var auditories: NSSet?
    @NSManaged public var employees: NSSet?
}

//MARK: - Generated accessors
extension Department {
    @objc(addAuditoriesObject:)
    @NSManaged public func addToAuditories(_ value: Auditorium)
    @objc(removeAuditoriesObject:)
    @NSManaged public func removeFromAuditories(_ value: Auditorium)
    @objc(addAuditories:)
    @NSManaged public func addToAuditories(_ values: NSSet)
    @objc(removeAuditories:)
    @NSManaged public func removeFromAuditories(_ values: NSSet)
    @objc(addEmployeesObject:)
    @NSManaged public func addToEmployees(_ value: Department)
    @objc(removeEmployeesObject:)
    @NSManaged public func removeFromEmployees(_ value: Department)
    @objc(addEmployees:)
    @NSManaged public func addToEmployees(_ values: NSSet)
    @objc(removeEmployees:)
    @NSManaged public func removeFromEmployees(_ values: NSSet)
}

//
//  DepartmentDecoder.swift
//  bsuirSchedule
```

```
//
//  Created by Andrej Hurynovič on 1.04.23.
//
//

import CoreData

@objc(Department)
public class Department: NSManagedObject {

    lazy var formattedName: String = self.name ?? self.abbreviation

    required public convenience init(from decoder: Decoder) throws {
        guard let container = try? decoder.container(keyedBy: CodingKeys.self) else {
            throw DepartmentDecodingError.noKeys
        }
        self.init(context: decoder.userInfo[.managedObjectContext] as! NSManagedObjectContext)

        self.id    =    (try?    container.decode(Int16.self,    forKey:    .id))    ??    (try?
container.decode(Int16.self, forKey: .idInAuditoriumContainer))!
        self.name = try! container.decode(String.self, forKey: .name)
        self.abbreviation = formattedAbbreviationString(from: try! container.decode(String.self,
forKey: .abbreviation))

        Log.info("Department \(self.abbreviation) (\(String(self.id))) has been created.")
    }

    convenience public init(from string: String, in context: NSManagedObjectContext) throws {
        self.init(entity: Department.entity(), insertInto: context)

        self.abbreviation = formattedAbbreviationString(from: string)
    }

    private func formattedAbbreviationString(from string: String) -> String {
        var string = string
        if let range = string.range(of: "Каф.") {
            string.removeSubrange(range)
        }

        return string.capitalizingFirstLetter()
    }

}

extension Department: Decodable {
    private enum CodingKeys: String, CodingKey {
        case id = "id"
        case idInAuditoriumContainer = "idDepartment"
        case name = "name"
        case abbreviation = "abbrev"
    }
}

//
//  DepartmentDecodingError.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 1.05.23.
//

import Foundation

enum DepartmentDecodingError: Error {
    case noKeys
}

//
//  DepartmentFetch.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 1.05.23.
//

import CoreData

extension Department: AbleToFetchAll {
    static func fetchAll() async {
        guard let data = try? await URLSession.shared.data(for: .departments) else { return }
```

```
        let startTime = CFAbsoluteTimeGetCurrent()

        let (backgroundContext, decoder) = newBackgroundContextWithDecoder()

        guard let departments = try? decoder.decode([Department].self, from: data) else {
            Log.error("Can't decode departments.")
            return
        }

        await backgroundContext.perform(schedule: .immediate, {
            try! backgroundContext.save()
        })
        Log.info("\(String(departments.count))         Departments         fetched,         time:
\((CFAbsoluteTimeGetCurrent() - startTime).roundTo(places: 3)) seconds.\n")
    }
}

//
//  DepartmentExtensions.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 1.05.23.
//

import Foundation

extension Department: Identifiable {}

extension Department: Favored {}

//
//  EducationTask.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 11.05.23.
//

import CoreData

extension EducationTask {
    @nonobjc public class func fetchRequest() -> NSFetchRequest<EducationTask> {
        let request = NSFetchRequest<EducationTask>(entityName: "Hometask")
        request.sortDescriptors = [NSSortDescriptor(keyPath: \EducationTask.deadline, ascending:
true)]
        return request
    }
    @NSManaged public var subject: String
    @NSManaged public var note: String
    @NSManaged public var images: [Data]?

    @NSManaged public var creation: Date
    @NSManaged public var deadline: Date?
}

//
//  EducationTaskInitializer.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 11.05.23.
//

import CoreData

@objc(EducationTask)
public class EducationTask: NSManagedObject {
    convenience init(subject: String, context: NSManagedObjectContext) {
        self.init(entity: EducationTask.entity(), insertInto: context)

        self.subject = subject
        self.creation = .now
    }
}

//
//  EducationTaskExtensions.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 11.05.23.
```

```
//

import Foundation

extension EducationTask: Identifiable { }

//
//  Employee.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 7.09.21.
//

import CoreData

extension Employee {
    @nonobjc public class func fetchRequest() -> NSFetchRequest<Employee> {
        let request = NSFetchRequest<Employee>(entityName: "Employee")
        request.sortDescriptors  =  [NSSortDescriptor(keyPath:  (\Employee.lastName),  ascending:
true),
                                     NSSortDescriptor(keyPath:   (\Employee.firstName),   ascending:
true),
                                     NSSortDescriptor(keyPath:  (\Employee.middleName),  ascending:
true)]
        return request
    }

    @NSManaged public var id: Int32
    @NSManaged public var urlID: String?
    @NSManaged public var firstName: String
    @NSManaged public var middleName: String?
    @NSManaged public var lastName: String

    @NSManaged public var rank: String?
    @NSManaged public var degree: Degree?
    @NSManaged public var departments: NSSet?
    @NSManaged public var favorite: Bool
    @NSManaged public var lessonsUpdateDate: Date?

    @NSManaged public var educationStart: Date?
    @NSManaged public var educationEnd: Date?
    @NSManaged public var examsStart: Date?
    @NSManaged public var examsEnd: Date?

    @NSManaged public var photoLink: String?
    @NSManaged public var photo: Data?

    @NSManaged public var lessons: NSSet?
}

//MARK: - Generated accessors

extension Employee {
    @objc(addLessonsObject:)
    @NSManaged public func addToLessons(_ value: Lesson)
    @objc(removeLessonsObject:)
    @NSManaged public func removeFromLessons(_ value: Lesson)
    @objc(addLessons:)
    @NSManaged public func addToLessons(_ values: NSSet)
    @objc(removeLessons:)
    @NSManaged public func removeFromLessons(_ values: NSSet)
    @objc(addDepartmentsObject:)
    @NSManaged public func addToDepartments(_ value: Department)
    @objc(removeDepartmentsObject:)
    @NSManaged public func removeFromDepartments(_ value: Department)
    @objc(addDepartments:)
    @NSManaged public func addToDepartments(_ values: NSSet)
    @objc(removeDepartments:)
    @NSManaged public func removeFromDepartments(_ values: NSSet)
}

//
//  EmployeeDecoder.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 4.06.21.
//
```

```
import CoreData

@objc(Employee)
public class Employee: NSManagedObject {
    required public convenience init(from decoder: Decoder) throws {
        let context = decoder.userInfo[.managedObjectContext] as! NSManagedObjectContext
        self.init(entity: Employee.entity(), insertInto: context)
        try! self.update(from: decoder)
        Log.info("Employee \(self.urlID ?? "no urlID") (\(String(self.id))) has been created.")
    }
}

//MARK: - Update

extension Employee: DecoderUpdatable {
    func update(from decoder: Decoder) throws {
        let startTime = CFAbsoluteTimeGetCurrent()
        let container = try decoder.container(keyedBy: CodingKeys.self)

        decodeEmployee(decoder)
        decodeDegree(decoder)
        decodeEducationDates(decoder)
        decodeLessons(container)

        Log.info("Employee \(self.urlID ?? "no urlID") (\(String(self.id))) has been updated, time:
\((CFAbsoluteTimeGetCurrent() - startTime).roundTo(places: 3)) seconds")
    }

    private func decodeEmployee(_ decoder: Decoder) {
        //The employee information structure nested container exists only when receiving a response
to the Schedule (Group) request. This fields is also contained when fetching all groups, but located
in root.
        let container = (try? decoder.container(keyedBy: CodingKeys.self)
            .nestedContainer(keyedBy: CodingKeys.self, forKey: .employeeNestedContainer))
        ?? (try! decoder.container(keyedBy: CodingKeys.self))

        self.id = (try! container.decode(Int32.self, forKey: .id))
        if let urlID = try? container.decode(String.self, forKey: .urlID) {
            self.urlID = urlID
        }
        self.firstName = try! container.decode(String.self, forKey: .firstName)
        self.middleName = try? container.decode(String.self, forKey: .middleName)
        self.lastName = try! container.decode(String.self, forKey: .lastName)

        self.photoLink = try? container.decode(String.self, forKey: .photoLink)

        self.rank = try? container.decode(String.self, forKey: .rank)

        if let departmentsAbbreviations = try? container.decode([String].self, forKey: .departments)
{
            let context = decoder.userInfo[.managedObjectContext] as! NSManagedObjectContext
            self.addToDepartments(NSSet(array:    departmentsAbbreviations.compactMap    {    try?
Department(from: $0, in: context) }))
        }
    }

    private func decodeDegree(_ decoder: Decoder) {
        if let degree = self.degree {
            try? degree.update(from: decoder)
        } else {
            self.degree = try? Degree(from: decoder)
        }
    }

    private func decodeLessons(_ container: KeyedDecodingContainer<Employee.CodingKeys>) {
        var lessons = [Lesson]()

        if let lessonsDictionary = try? container.decode([String:[Lesson]].self, forKey: .lessons)
{
            let mappedLessons = Set(lessonsDictionary
                .map{ $1 }
                .joined())
            lessons.append(contentsOf: mappedLessons)
        }
        if let exams = try? container.decode([Lesson].self, forKey: .exams) {
            lessons.append(contentsOf: exams)
        }
```

```
        for lesson in lessons {
            lesson.employeesIDs = [self.id]
        }

        if lessons.isEmpty == false {
            lessonsUpdateDate = Date()
        }

        self.addToLessons(NSSet(array: lessons))
    }

}

//MARK: - CodingKeys

extension Employee: Decodable {
    private enum CodingKeys: String, CodingKey {
        case id
        case urlID = "urlId"
        case firstName
        case middleName
        case lastName

        case departments = "academicDepartment"
        case rank
        case degree
        case degreeAbbreviation = "degreeAbbrev"

        case photoLink

        case lessons = "schedules"
        case exams = "exams"

        case employeeNestedContainer = "employeeDto"
    }
}

//
//  EmployeeFetch.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 1.05.23.
//

import CoreData

extension Employee: AbleToFetchAll {
    static func fetchAll() async {
        guard let data = try? await URLSession.shared.data(for: .employees) else { return }
        let startTime = CFAbsoluteTimeGetCurrent()
        guard let dictionaries = try! JSONSerialization.jsonObject(with: data, options: []) as?
[[String: Any]] else {
            Log.error("Can't create employees dictionaries.")
            return
        }

        let backgroundContext = PersistenceController.shared.container.newBackgroundContext()
        backgroundContext.mergePolicy = NSMergeByPropertyStoreTrumpMergePolicy
        let decoder = JSONDecoder()
        decoder.userInfo[.managedObjectContext] = backgroundContext
        decoder.userInfo[.groupEmbeddedContainer] = true

        var employees: [Employee] = getAll(from: backgroundContext)

        for dictionary in dictionaries {
            let data = try! JSONSerialization.data(withJSONObject: dictionary)

            if var employee = employees.first (where: { $0.id == dictionary["id"] as? Int32 }) {
                try! decoder.update(&employee, from: data)
            } else {
                let employee = try! decoder.decode(Employee.self, from: data)
                employees.append(employee)
            }
        }

        await backgroundContext.perform(schedule: .immediate, {
            try! backgroundContext.save()
            Log.info("\(String(employees.count))         Employees         fetched,         time:
```

```
\((CFAbsoluteTimeGetCurrent() - startTime).roundTo(places: 3)) seconds.\n")
        })

    }

}

//MARK: - Update

extension Employee {
    func update() async -> Employee? {
        guard let urlID = self.urlID,
              let url = URL(string: FetchDataType.employee.rawValue + urlID),
              let (data, _) = try? await URLSession.shared.data(from: url) else {
            Log.error("No data for employee (\(self.urlID ?? "no urlID"))")
            return nil
        }

        guard data.count != 0 else {
            Log.warning("Empty data while updating employee \(self.urlID ?? "no urlID")
(\(String(self.id)))")
            return nil
        }

        let backgroundContext = PersistenceController.shared.container.newBackgroundContext()
        backgroundContext.mergePolicy = NSMergeByPropertyStoreTrumpMergePolicy
        let decoder = JSONDecoder()
        decoder.userInfo[.managedObjectContext] = backgroundContext
        decoder.userInfo[.groupEmbeddedContainer] = true

        var backgroundEmployee = backgroundContext.object(with: self.objectID) as! Employee
        let previousPhotoLink = backgroundEmployee.photoLink

        try! decoder.update(&backgroundEmployee, from: data)

        await backgroundContext.perform(schedule: .immediate, {
            try! backgroundContext.save()
        })

        if backgroundEmployee.photoLink != previousPhotoLink || backgroundEmployee.photoLink !=
nil, backgroundEmployee.photo == nil {
            backgroundEmployee.photo = await fetchPhoto()
        }

        await backgroundContext.perform(schedule: .immediate, {
            try! backgroundContext.save()
        })

        return self
    }

    static func updateEmployees(employees: [Employee] = Employee.getAll()) async {
        let startTime = CFAbsoluteTimeGetCurrent()
        try! await withThrowingTaskGroup(of: Employee?.self) { taskGroup in
            for employee in employees {
                taskGroup.addTask {
                    await employee.update()
                }
            }
            try await taskGroup.waitForAll()
        }
        Log.info("Employees updated in time: \((CFAbsoluteTimeGetCurrent() -
startTime).roundTo(places: 3)) seconds")
    }

}

//MARK: - Photo

extension Employee {
    func fetchPhoto() async -> Data? {
        guard let photoLink = self.photoLink,
              let url = URL(string: photoLink),
              let (data, _) = try? await URLSession.shared.data(from: url) else {
            Log.error("No data for employee photo (\(String(self.id)))")
            return nil
        }
```

```
        guard data.count != 0 else {
            Log.warning("Empty data while updating employee photo \(self.urlID ?? "no urlID")
(\(String(self.id)))")
            return nil
        }

        Log.info("Employee \(self.urlID ?? "no urlID") (\(String(self.id))) photo has been
updated.")
        return data
    }

}

//
//  EmployeeExtensions.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 1.05.23.
//

import Foundation

extension Employee: Identifiable { }
extension Employee: Favored { }

extension Employee: EducationBounded { }
extension Employee: EducationRanged { }

extension Employee: Scheduled {
    var title: String { self.lastName }
}

//MARK: - Group

extension Group {
    var employees: [Employee]? {
        guard let lessons = self.lessons?.allObjects as? [Lesson] else { return nil }

        let employees = Set(lessons.compactMap { $0.employees?.allObjects as? [Employee] }
            .flatMap { $0 })
            .sorted { $0.lastName < $1.lastName }

        guard employees.isEmpty == false else { return nil }
        return employees
    }

}

//MARK: - Other

extension Employee {
    var formattedName: String {
        var formattedName = firstName
        if let lastName = self.middleName {
            formattedName.append(" \(lastName)")
        }
        return formattedName
    }

    var departmentsArray: [Department]? {
        guard let departments = departments?.allObjects as? [Department] else {
            return nil
        }
        return departments
    }
    var departmentsAbbreviations: [String]? {
        guard let departments = departmentsArray else {
            return nil
        }
        return departments.map { $0.abbreviation }
    }
}

//
//  EmployeeSectionType.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 4.04.23.
```

```
//

enum EmployeeSectionType: SectionType {
    case firstLetter
    case department
    case rank
    case degree

    var description: String {
        switch self {
            case .firstLetter:
                return "Алфавит"
            case .department:
                return "Подразделение"
            case .rank:
                return "Ранг"
            case .degree:
                return "Степень"
        }
    }
}

//
//  EmployeesSections.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 4.04.23.
//

import CoreData

extension Sequence where Element == Employee {
    func sections(_ type: EmployeeSectionType) -> [NSManagedObjectsSection<Employee>] {
        switch type {
            case .firstLetter:
                return self.sectioned(by: \.lastName.first!)
                    .sorted { $0.key < $1.key }
                    .map { NSManagedObjectsSection(title: String($0.key),
                                                   items: $0.value) }
            case .department:
                return departmentSections()
            case .rank:
                return self.sectioned(by: \.rank)
                    .sorted { $0.key ?? "" < $1.key ?? "" }
                    .map { NSManagedObjectsSection(title: $0.key ?? "Без ранга",
                                                   items: $0.value) }
            case .degree:
                return self.sectioned(by: \.degree)
                    .sorted { $0.key?.abbreviation ?? "" < $1.key?.abbreviation ?? "" }
                    .map { NSManagedObjectsSection(title: $0.key?.formattedName ?? "Без степени",
                                                   items: $0.value) }
        }
    }

    private func departmentSections() -> [NSManagedObjectsSection<Employee>] {
        let departments: [Department] = Department.getAll()
            .sorted { $0.abbreviation < $1.abbreviation }
        var sections: [NSManagedObjectsSection<Employee>] = departments.compactMap { department in

            let employees = self.filter { employee in
                if let departments = employee.departments, departments.contains(department) {
                    return true
                } else {
                    return false
                }
            }
            guard employees.isEmpty == false else { return nil }

            return NSManagedObjectsSection(title: department.abbreviation,
                                           items: employees)
        }


        let noDepartmentsEmployees = self.filter( { $0.departmentsArray == nil } )
        if noDepartmentsEmployees.isEmpty == false {
            sections.append( NSManagedObjectsSection(title: "Без подразделения",
                                                     items: noDepartmentsEmployees))
        }
```

```
        return sections
    }

}

//
//  FacultyDecoder.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 21.09.21.
//

import CoreData

@objc(Faculty)
public class Faculty: NSManagedObject {

    required convenience public init(from decoder: Decoder) throws {
        let context = decoder.userInfo[.managedObjectContext] as! NSManagedObjectContext
        self.init(context: context)

        //If init is called from a decoder that decodes a Group and cannot find the required
specialty, a special method is called to process specific Group keys.
        if let groupContainer = decoder.userInfo[.groupEmbeddedContainer] as? Bool,
           groupContainer == true {
            try! self.updateFromGroupDecoder(decoder)
        } else {
            try! self.update(from: decoder)
        }
    }

    convenience public init(id: Int16, name: String? = nil, abbreviation: String? = nil, context:
NSManagedObjectContext) {
        self.init(entity: Faculty.entity(), insertInto: context)

        self.id = id
        self.name = name
        self.abbreviation = abbreviation
    }

}

//MARK: - Update

extension Faculty: DecoderUpdatable {
    func update(from decoder: Decoder) throws {
        let container = try! decoder.container(keyedBy: CodingKeys.self)

        self.id = try! container.decode(Int16.self, forKey: .id)
        self.name = try! container.decode(String.self, forKey: .name)
        self.abbreviation = try! container.decode(String.self, forKey: .abbreviation)
        Log.info("Faculty \(self.abbreviation!) (\(String(self.id))) has been updated")
    }

    func updateFromGroupDecoder(_ decoder: Decoder) throws {
        //If the decoder container is received from a Groups API call, the Speciality information
is contained in root, but if it is received from a Group API call, the required information is
contained in a nested container.
        let container = (try? decoder.container(keyedBy: GroupCodingKeys.self)
            .nestedContainer(keyedBy: GroupCodingKeys.self, forKey: .groupNestedContainer))
        ?? (try! decoder.container(keyedBy: GroupCodingKeys.self))

        self.id = try! container.decode(Int16.self, forKey: .id)
        self.abbreviation = try! container.decode(String.self, forKey: .abbreviation)
        Log.info("Faculty \(self.abbreviation!) (\(String(self.id))) has been created from Group
container")
    }

}

//MARK: CodingKeys

extension Faculty: Decodable {
    private enum CodingKeys: String, CodingKey {
        case id
        case name
        case abbreviation = "abbrev"
```

```
    }

    private enum GroupCodingKeys: String, CodingKey {
        case groupNestedContainer = "studentGroupDto"

        case id = "facultyId"
        case abbreviation = "facultyAbbrev"
    }
}

//
//  FacultyFetch.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 5.05.23.
//

import CoreData

extension Faculty: AbleToFetchAll {
    static func fetchAll() async {
        guard let data = try? await URLSession.shared.data(for: .faculties) else { return }
        let startTime = CFAbsoluteTimeGetCurrent()

        let (backgroundContext, decoder) = newBackgroundContextWithDecoder()

        guard let faculties = try? decoder.decode([Faculty].self, from: data) else {
            Log.error("Can't decode faculties.")
            return
        }

        await backgroundContext.perform(schedule: .immediate, {
            try! backgroundContext.save()
        })
        Log.info("\(String(faculties.count)) Faculties fetched, time: \((CFAbsoluteTimeGetCurrent()
- startTime).roundTo(places: 3)) seconds.\n")
    }

}

//
//  FacultyExtensions.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 5.05.23.
//

extension Faculty : Identifiable {}

//
//  Group.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 6.09.21.
//

import CoreData

extension Group {
    @nonobjc public class func fetchRequest() -> NSFetchRequest<Group> {
        let request = NSFetchRequest<Group>(entityName: "Group")
        request.sortDescriptors = [NSSortDescriptor(keyPath: \Group.name, ascending: true)]
        return request
    }

    @NSManaged public var name: String
    @NSManaged public var numberOfStudents: Int16
    @NSManaged public var educationDegreeValue: Int16
    @NSManaged public var course: Int16
    @NSManaged public var favorite: Bool
    @NSManaged public var lessonsUpdateDate: Date?

    @NSManaged public var nickname: String?

    @NSManaged public var speciality: Speciality?

    @NSManaged public var educationStart: Date?
    @NSManaged public var educationEnd: Date?
```

```
    @NSManaged public var examsStart: Date?
    @NSManaged public var examsEnd: Date?

    @NSManaged public var lessons: NSSet?
}

//MARK: - Generated accessors for lessons

extension Group {
    @objc(addLessonsObject:)
    @NSManaged public func addToLessons(_ value: Lesson)
    @objc(removeLessonsObject:)
    @NSManaged public func removeFromLessons(_ value: Lesson)
    @objc(addLessons:)
    @NSManaged public func addToLessons(_ values: NSSet)
    @objc(removeLessons:)
    @NSManaged public func removeFromLessons(_ values: NSSet)
}

//
//  GroupDecoder.swift
//  Group
//
//  Created by Andrej Hurynovič on 6.09.21.
//

import CoreData

@objc(Group)
public class Group: NSManagedObject {
    required public convenience init(from decoder: Decoder) throws {
        self.init(context: decoder.userInfo[.managedObjectContext] as! NSManagedObjectContext)
        try! self.update(from: decoder)
        Log.info("Group (\(String(self.name))) has been created.")
    }
}

//MARK: - Update

extension Group: DecoderUpdatable {
    func update(from decoder: Decoder) throws {
        let startTime = CFAbsoluteTimeGetCurrent()
        let container = try decoder.container(keyedBy: CodingKeys.self)

        decodeGroup(decoder)
        decodeEducationDates(decoder)
        decodeSpeciality(decoder)
        decodeLessons(container)

        Log.info("Group (\(String(self.name))) has been updated, time:
\((CFAbsoluteTimeGetCurrent() - startTime).roundTo(places: 3)) seconds")
    }

    private func decodeGroup(_ decoder: Decoder) {
        //The group information structure nested container exists only when receiving a response
to the Schedule (Group) request. This fields is also contained when fetching all groups, but located
in root.
        let container = (try? decoder.container(keyedBy: CodingKeys.self)
            .nestedContainer(keyedBy: CodingKeys.self, forKey: .groupNestedContainer))
        ?? (try! decoder.container(keyedBy: CodingKeys.self))

        self.name = try! container.decode(String.self, forKey: .id)
        if let course = try? container.decode(Int16.self, forKey: .course) {
            self.course = course
            }
        self.educationDegreeValue = try! container.decode(Int16.self, forKey: .educationDegree)

        //Number of student presented only in Lesson group data.
        if let numberOfStudents = try? container.decode(Int16.self, forKey: .numberOfStudents) {
            self.numberOfStudents = numberOfStudents
        }

    }

    private func decodeSpeciality(_ decoder: Decoder) {
        let container = (try? decoder.container(keyedBy: CodingKeys.self)
            .nestedContainer(keyedBy: CodingKeys.self.self, forKey: .groupNestedContainer))
        ?? (try! decoder.container(keyedBy: CodingKeys.self))
```

```
        guard let _ = try? container.decode(Int32.self, forKey: .specialityID) else { return }
        self.speciality = try! Speciality(from: decoder)
        Log.info("The speciality (\(String(describing: self.speciality?.id)) - \(String(describing:
self.speciality?.abbreviation))) is created and assigned to group \(String(self.name))")

    }

    private func decodeLessons(_ container: KeyedDecodingContainer<Group.CodingKeys>) {
        let lessons = try? container.decode([String:[Lesson]].self, forKey: .lessons)
        let exams = try? container.decode([Lesson].self, forKey: .exams)

        if lessons != nil || exams != nil {
            self.lessonsUpdateDate = Date()
        }
    }

}

extension Group: Decodable {
    private enum CodingKeys: String, CodingKey {
        case lessons = "schedules"
        case exams = "exams"

        case groupNestedContainer = "studentGroupDto"

        case id = "name"
        case numberOfStudents
        case educationDegree = "educationDegree"
        case course
        case specialityID = "specialityDepartmentEducationFormId"
    }
}



//MARK: - CodingUserInfoKey
extension CodingUserInfoKey {
    ///A boolean value that indicates that container should be decoded as being received from a
Group container. Required for Faculty and Speciality decoding.
    static let groupEmbeddedContainer = CodingUserInfoKey(rawValue: "groupEmbeddedContainer")!
}

//
//  GroupFetch.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 1.05.23.
//

import CoreData

//MARK: - Fetch

extension Group: AbleToFetchAll {
    static func fetchAll() async {
        guard let data = try? await URLSession.shared.data(for: .groups) else { return }
        let startTime = CFAbsoluteTimeGetCurrent()
        guard let dictionaries = try! JSONSerialization.jsonObject(with: data, options: []) as?
[[String: Any]] else {
            Log.error("Can't create group dictionaries.")
            return
        }

        let backgroundContext = PersistenceController.shared.container.newBackgroundContext()
        backgroundContext.mergePolicy = NSMergeByPropertyStoreTrumpMergePolicy
        let decoder = JSONDecoder()
        decoder.userInfo[.managedObjectContext] = backgroundContext
        decoder.userInfo[.groupEmbeddedContainer] = true

        var groups: [Group] = getAll(from: backgroundContext)

        for dictionary in dictionaries {
            let data = try! JSONSerialization.data(withJSONObject: dictionary)

            if var group = groups.first (where: { $0.name == dictionary["name"] as? String }) {
                try! decoder.update(&group, from: data)
            } else {
                let group = try! decoder.decode(Group.self, from: data)
```

```
                groups.append(group)
            }
        }

        await backgroundContext.perform(schedule: .immediate, {
            try! backgroundContext.save()
            Log.info("\(String(groups.count)) Groups fetched, time: \((CFAbsoluteTimeGetCurrent()
- startTime).roundTo(places: 3)) seconds.\n")
        })
    }

}

//MARK: - Update
extension Group {
    func update() async -> Group? {
        guard let url = URL(string: FetchDataType.group.rawValue + self.name),
              let (data, _) = try? await URLSession.shared.data(from: url) else {
            Log.error("No data for group (\(String(self.name)))")
            return nil
        }

        guard data.count != 0 else {
            Log.warning("Empty data while updating group (\(String(self.name)))")
            return nil
        }

        let backgroundContext = PersistenceController.shared.container.newBackgroundContext()
        backgroundContext.mergePolicy = NSMergeByPropertyStoreTrumpMergePolicy
        let decoder = JSONDecoder()
        decoder.userInfo[.managedObjectContext] = backgroundContext
        decoder.userInfo[.groupEmbeddedContainer] = true

        var backgroundGroup = backgroundContext.object(with: self.objectID) as! Group
        try! decoder.update(&backgroundGroup, from: data)

        await backgroundContext.perform(schedule: .immediate, {
            try! backgroundContext.save()
        })

        return self
    }

    static func updateGroups(groups: [Group] = Group.getAll()) async {
        let startTime = CFAbsoluteTimeGetCurrent()
        try! await withThrowingTaskGroup(of: Group?.self) { taskGroup in
            for studentGroup in groups {
                taskGroup.addTask {
                    await studentGroup.update()
                }
            }
            try await taskGroup.waitForAll()
        }
        Log.info("Groups      updated      in      time:      \((CFAbsoluteTimeGetCurrent()      -
startTime).roundTo(places: 3)) seconds")

    }

}

//
//  GroupExtensions.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 11.04.23.
//

import Foundation

extension Group: Identifiable { }
extension Group: Favored {}

extension Group: EducationBounded { }
extension Group: EducationRanged { }

extension Group: Scheduled {
    var title: String { self.name }
}
```

```
//MARK: - Employee

extension Employee {
    var groups: [Group]? {
        guard let lessons = self.lessons?.allObjects as? [Lesson] else { return nil }

        let groups = Set(lessons.compactMap { $0.groups?.allObjects as? [Group] }
            .flatMap { $0 })
            .sorted { $0.id < $1.id }

        guard groups.isEmpty == false else { return nil }
        return groups
    }
}


//MARK: - Other

extension Group {
    var flow: [Group]? {
        guard var flow = self.speciality?.groups?.allObjects as? [Group] else { return nil }
        flow.removeAll { $0.course != self.course || $0 == self }

        guard flow.isEmpty == false else {
            return nil
        }
        return flow.sorted { $0.id < $1.id }
    }

}

extension Array where Element == Group {
    func description() -> String {
        guard self.isEmpty == false else {
                return ""
            }

        guard self.count > 1 else {
            return self.first!.name
        }

        var groups = self.map { $0.name }.sorted()
            var nearGroups: [String] = []
            var finalGroups: [String] = []

            repeat {
                nearGroups.removeAll()
                nearGroups.append(groups.removeFirst())
                if groups.isEmpty == false {
                    while groups.isEmpty == false, Int(groups.first!)! - Int(nearGroups.last!)! ==
1 {
                        nearGroups.append(groups.removeFirst())
                    }
                    if nearGroups.count > 1 {
                        finalGroups.append("\(nearGroups.first!)-
\((String(nearGroups.last!)).last!)")
                    } else {
                        finalGroups.append(String(nearGroups.first!))
                    }

                } else {
                    finalGroups.append(String(nearGroups.first!))
                }

            } while (groups.isEmpty == false)
            return finalGroups.joined(separator: ", ")
    }
}

//
//  GroupSectionType.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 11.04.23.
//

enum GroupSectionType: SectionType {
```

```
    case specialityAbbreviation
    case specialityName
    case faculty
    case flow

    var description: String {
        switch self {
            case .specialityAbbreviation:
                return "Специальность сокращённая"
            case .specialityName:
                return "Специальность полная"
            case .faculty:
                return "Факультет"
            case .flow:
                return "Поток"
        }
    }
}

//
//  GroupsSections.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 11.04.23.
//

import CoreData

extension Sequence where Element == Group {
    func sections(_ type: GroupSectionType) -> [NSManagedObjectsSection<Group>] {
        switch type {
            case .specialityAbbreviation:
                return specialitySections(useSpecialityName: false)
            case .specialityName:
                return specialitySections(useSpecialityName: true)
            case .faculty:
                return facultySections()
            case .flow:
                return flowSections()
        }
    }

    private func specialitySections(useSpecialityName: Bool = false) ->
[NSManagedObjectsSection<Group>] {
        var sections: [NSManagedObjectsSection<Group>] = []
        var sectionedGroups = self.sectioned(by: \.speciality)
        if let noSpecialityGroups = sectionedGroups.removeValue(forKey: nil) {
            sections.append( NSManagedObjectsSection(title: "Без специальности",
                                                     id: "",
                                                     items: noSpecialityGroups) )
        }
        sections.insert(contentsOf:
                            sectionedGroups.sorted { $0.key!.formattedID() <
$1.key!.formattedID() }
            .map { NSManagedObjectsSection(title: $0.key!.formattedDescription(useSpecialityName:
useSpecialityName),
                                           id: $0.key!.objectID.description,
                                           items: $0.value) },
                        at: 0)
        return sections
    }

    private func facultySections() -> [NSManagedObjectsSection<Group>] {
        self.sectioned(by: \.speciality?.faculty)
            .sorted { $0.key?.abbreviation ?? "" < $1.key?.abbreviation ?? "" }
            .map { NSManagedObjectsSection(title: $0.key?.abbreviation ?? "Без факультета",
                                           items: $0.value) }
    }

    private func flowSections() -> [NSManagedObjectsSection<Group>] {
        Dictionary(grouping: self, by: { String($0.name.prefix(4)) })
            .sorted { $0.key < $1.key }
            .map { NSManagedObjectsSection(title: $0.key ,
                                           items: $0.value)}
    }

}
```

```swift
//
//  Lesson+CoreDataProperties.swift
//  Lesson
//
//  Created by Andrej Hurynovič on 6.09.21.
//
//

import CoreData

extension Lesson {
    @nonobjc public class func fetchRequest() -> NSFetchRequest<Lesson> {
        let request = NSFetchRequest<Lesson>(entityName: "Lesson")
        request.sortDescriptors = []
        return request
    }

    @NSManaged public var subject: String?
    @NSManaged public var abbreviation: String
    @NSManaged public var type: LessonType?
    @NSManaged public var note: String?

    @NSManaged public var dateString: String
    @NSManaged public var weekday: Int16
    @NSManaged public var weeks: [Int]
    @NSManaged public var startLessonDate: Date?
    @NSManaged public var startLessonDateString: String
    @NSManaged public var endLessonDate: Date?
    @NSManaged public var timeStart: String
    @NSManaged public var timeEnd: String

    @NSManaged public var groups: NSSet?
    @NSManaged public var subgroup: Int16
    @NSManaged public var auditories:  NSSet?
    @NSManaged public var auditoriesNames: [String]
    @NSManaged public var employees: NSSet?
    @NSManaged public var employeesIDs: [Int32]

}

//MARK: - Generated accessors for groups
extension Lesson {
    @objc(addGroupsObject:)
    @NSManaged public func addToGroups(_ value: Group)
    @objc(removeGroupsObject:)
    @NSManaged public func removeFromGroups(_ value: Group)
    @objc(addGroups:)
    @NSManaged public func addToGroups(_ values: NSSet)
    @objc(removeGroups:)
    @NSManaged public func removeFromGroups(_ values: NSSet)
    @objc(addEmployeesObject:)
    @NSManaged public func addToEmployees(_ value: Employee)
    @objc(removeEmployeesObject:)
    @NSManaged public func removeFromEmployees(_ value: Employee)
    @objc(addEmployees:)
    @NSManaged public func addToEmployees(_ values: NSSet)
    @objc(removeEmployees:)
    @NSManaged public func removeFromEmployees(_ values: NSSet)
    @objc(addAuditoriesObject:)
    @NSManaged public func addToAuditories(_ value: Auditorium)
    @objc(removeAuditoriesObject:)
    @NSManaged public func removeFromAuditories(_ value: Auditorium)
    @objc(addAuditories:)
    @NSManaged public func addToAuditories(_ values: NSSet)
    @objc(removeAuditories:)
    @NSManaged public func removeFromAuditories(_ values: NSSet)
}

//
//  Lesson+CoreDataClass.swift
//  Lesson
//
//  Created by Andrej Hurynovič on 6.09.21.
//
//

import CoreData
import SwiftUI
```

```
@objc(Lesson)
public class Lesson: NSManagedObject {

    required convenience public init(from decoder: Decoder) throws {
        let context = decoder.userInfo[.managedObjectContext] as! NSManagedObjectContext
        self.init(entity: Lesson.entity(), insertInto: context)

        let container = try! decoder.container(keyedBy: CodingKeys.self)

        decodeLesson(container, context)
        decodeDate(container)
        decodeAnnouncement(container)
        decodeGroups(container, decoder, context)
        decodeEmployees(container, decoder, context)
        decodeAuditories(container, decoder, context)
    }

    private func decodeLesson(_ container: KeyedDecodingContainer<Lesson.CodingKeys>, _ context:
NSManagedObjectContext) {
        self.subject = try? container.decode(String.self, forKey: .subject)
        //Abbreviation cannot be optional, because it is used as constraint
        self.abbreviation = (try? container.decode(String.self, forKey: .abbreviation)) ?? ""

        self.note = try? container.decode(String.self, forKey: .note)
        self.subgroup = Int16(try! container.decode(Int.self, forKey: .subgroup))

        if let lessonTypeID = try? container.decode(String.self, forKey: .lessonTypeValue) {
            self.type = LessonType(id: lessonTypeID, context: context)
        }
    }

    private func decodeDate(_ container: KeyedDecodingContainer<Lesson.CodingKeys>) {
        //DateString is a constraint, so it cannot be optional.
        if let date = try? container.decode(String.self, forKey: .date) {
            self.dateString = date
        } else {
            self.dateString = ""
        }

        if let startLessonDateString = try? container.decode(String.self, forKey: .startLessonDate)
{
            self.startLessonDateString = startLessonDateString
            self.startLessonDate = DateFormatters.short.date(from: startLessonDateString)
            self.endLessonDate = DateFormatters.short.date(from: try! container.decode(String.self,
forKey: .endLessonDate))

            if let startLessonDate = startLessonDate,
               let endLessonDate = endLessonDate,
               startLessonDate > endLessonDate {
                self.endLessonDate = startLessonDate
                self.startLessonDate = endLessonDate
            }
        } else {

            self.startLessonDateString = ""
        }

        self.timeStart = try! container.decode(String.self, forKey: .timeStart)
        self.timeEnd = try! container.decode(String.self, forKey: .timeEnd)

        if let date = (self.date ?? self.startLessonDate) {
            self.weekday = date.weekday
        }

        // An array of weeks can take values [0, 1, 2, 3 ,4], but it is more convenient to count
the weeks from zero, and in the API 0 means that there is an occupation for all weeks, so we
subtract one from all the values of the array.
        //[1, 2 ,4] -> [0, 1, 3]
        if let weeks = try? container.decode([Int].self, forKey: .weeks) {
            self.weeks = weeks.map{ $0 - 1 }
            //Пояснить это позже
            if self.weeks.contains(-1) {
                self.weeks.removeFirst()
            }
        } else {
            weeks = []
        }
```

```
    }

    private func decodeAnnouncement(_ container: KeyedDecodingContainer<Lesson.CodingKeys>) {
        //Перепроверить
        if try! container.decode(Bool.self, forKey: .announcement) == true {
            //Announcement can be repeated every certain day of the week, the boundaries of which
are defined by startLessonDate and endLessonDate.
            self.weeks = [0, 1, 2, 3]
            //Start and end time in announcementStart and announcementEnd fields can be different
then time in timeStart and timeEnd.
            //Depreciated?
            if let announcementStart = try? container.decode(String.self,
forKey: .announcementStart) {
                self.timeStart = announcementStart
            }
            if let announcementEnd = try? container.decode(String.self, forKey: .announcementEnd)
{
                self.timeEnd = announcementEnd
            }
        }

    }

    private func decodeAuditories(_ container: KeyedDecodingContainer<Lesson.CodingKeys>, _
decoder: Decoder, _ context: NSManagedObjectContext) {
        if let auditoriumNames = try? container.decode([String].self, forKey: .auditorium) {
            let auditories = auditoriumNames.map { (try! Auditorium(from: $0, in: context)) }
            self.addToAuditories(NSSet(array: auditories))
            self.auditoriesNames = auditories.map { "\($0.floor)\(String(describing: $0.name))-
\($0.building)" }.sorted()
        } else {
            self.auditoriesNames = []
        }

    }

    private func decodeEmployees(_ container: KeyedDecodingContainer<Lesson.CodingKeys>, _ decoder:
Decoder, _ context: NSManagedObjectContext) {
        if let employees = try? container.decode([Employee].self, forKey: .employees) {
            self.addToEmployees(NSSet(array: employees))
            self.employeesIDs = employees.map { $0.id }.sorted()
        } else {
            self.employeesIDs = []
        }
    }

    private func decodeGroups(_ container: KeyedDecodingContainer<Lesson.CodingKeys>, _ decoder:
Decoder, _ context: NSManagedObjectContext) {
        if let groups = try? container.decode([Group].self, forKey: .groups) {
            self.addToGroups(NSSet(array: groups))
        }
    }

}

extension Lesson: Decodable {
    private enum CodingKeys: String, CodingKey {
        case subject = "subjectFullName"
        case abbreviation = "subject"
        case lessonTypeValue = "lessonTypeAbbrev"
        case announcement = "announcement"
        case note = "note"

        case date = "dateLesson"
        case weeks = "weekNumber"
        case startLessonDate = "startLessonDate"
        case endLessonDate = "endLessonDate"
        case announcementStart = "announcementStart"
        case announcementEnd = "announcementEnd"
        case timeStart = "startLessonTime"
        case timeEnd = "endLessonTime"

        case groups = "studentGroups"
        case subgroup = "numSubgroup"
        case employees = "employees"
        case auditorium = "auditories"
    }
```

```
}

//
//  LessonExtensions.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 1.05.23.
//

import CoreData

extension Lesson : Identifiable {
    func id(sectionID: String? = nil) -> String {
        return  "\(sectionID  ??  "")-\(abbreviation)-\(timeStart)-\(subgroup)-\(auditoriesNames)-
\(String(describing: employeesIDs))"
    }
}

extension Lesson {
    public override var description: String {
        return "\(abbreviation)  (\(type?.abbreviation ?? type?.id ?? "no type"))  \(timeStart)-
\(timeEnd), groups: \((groups?.allObjects as? [Group])?.description() ?? "no groups") employees:
\(employeesIDs)"
    }
}

//MARK: - Filters

extension Sequence where Element == Lesson {
    func filtered(abbreviation: String) -> any Sequence<Lesson> {
        guard !abbreviation.isEmpty else { return self }
        return self.filter { $0.abbreviation.localizedStandardContains(abbreviation) }
    }

    func filtered(subgroup: Int?) -> any Sequence<Lesson> {
        guard let subgroup = subgroup else {
            return self
        }
        let allowedSubgroups: [Int16] = [0, Int16(subgroup)]
        return self.filter { allowedSubgroups.contains($0.subgroup)  }
    }
}

//MARK: - Date and time

extension Lesson {
    ///Converts dateString to Date type
    var date: Date? {
        guard dateString.isEmpty == false,
              let date = DateFormatters.short.date(from: self.dateString) else {
            return nil
        }
        return date
    }
    ///Range between start and end date
    var dateRange: ClosedRange<Date>? {
        if let startLessonDate = self.startLessonDate, let endLessonDate = self.endLessonDate {
            return startLessonDate...endLessonDate
        } else {
            return nil
        }
    }

    ///Date range in form timeStart to timeEnd
    var timeRange: ClosedRange<Date> {
        return     DateFormatters.time.date(from:     timeStart)!...DateFormatters.time.date(from:
timeEnd)!
    }
}

//MARK: - Others

extension Lesson {
    ///A string representation of the lesson's weeks.
    var weeksDescription: String? {
        guard self.weeks.isEmpty == false else { return nil }
        guard self.weeks.count != 1 else { return String(weeks.first! + 1) }
```

```
        var weeks = self.weeks.map { $0 + 1 }

        var subStrings: [String] = []
        var nearWeeks: [Int] = []

        repeat {
            nearWeeks.removeAll()
            nearWeeks.append(weeks.removeFirst())

            while(weeks.isEmpty == false) {
                if nearWeeks.last! + 1 == weeks.first {
                    nearWeeks.append(weeks.removeFirst())
                } else {
                    break
                }
            }

            if nearWeeks.count == 1 {
                subStrings.append(String(nearWeeks.first!))
            } else {
                subStrings.append("\(nearWeeks.first!)-\(nearWeeks.last!)")
            }

        } while(weeks.isEmpty == false)

       return subStrings.joined(separator: ", ")
    }
}

//
//  LessonType.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 1.05.23.
//
//

import CoreData
import SwiftUI

extension LessonType {
    @nonobjc public class func fetchRequest() -> NSFetchRequest<LessonType> {
        let request = NSFetchRequest<LessonType>(entityName: "LessonType")
        request.sortDescriptors = [NSSortDescriptor(keyPath: \LessonType.id, ascending: true)]
        return request
    }

    @NSManaged public var id: String

    @NSManaged public var name: String?
    @NSManaged public var abbreviation: String?

    @NSManaged var colorData: String?

    @NSManaged public var lessons: NSSet?
}

extension LessonType {
    var color: Color? {
        get {
            guard let data = colorData else { return nil }
            return Color(rawValue: data)
        }
        set { colorData = newValue?.rawValue }
    }

}
extension LessonType {
    @objc(addGroupsObject:)
    @NSManaged public func addToLessons(_ value: Lesson)

    @objc(removeGroupsObject:)
    @NSManaged public func removeFromLessons(_ value: Lesson)

    @objc(addGroups:)
    @NSManaged public func addToLessons(_ values: NSSet)

    @objc(removeGroups:)
```

```
    @NSManaged public func removeFromLessons(_ values: NSSet)
}

//
//  LessonTypeDecoder.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 1.05.23.
//
//

import CoreData
import SwiftUI

@objc(LessonType)
public class LessonType: NSManagedObject {
    required public convenience init(id: String, context: NSManagedObjectContext) {
        self.init(entity: LessonType.entity(), insertInto: context)

        self.id = id
    }

    required public convenience init(id: String,
                                     name: String,
                                     abbreviation: String,
                                     color: Color,
                                     context: NSManagedObjectContext) {
        self.init(entity: LessonType.entity(), insertInto: context)

        self.id = id
        self.name = name
        self.abbreviation = abbreviation
        self.color = color
    }
}

//
//  Speciality.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 15.10.21.
//

import CoreData

extension Speciality {
    @nonobjc public class func fetchRequest() -> NSFetchRequest<Speciality> {
        let request = NSFetchRequest<Speciality>(entityName: "Speciality")
        request.sortDescriptors = [NSSortDescriptor(keyPath: \Speciality.name, ascending: true),
                                   NSSortDescriptor(keyPath:       \Speciality.educationType?.id,
ascending: true)]
        return request
    }

    @NSManaged public var id: Int32
    @NSManaged public var name: String
    @NSManaged public var abbreviation: String

    @NSManaged public var educationType: EducationType?
    @NSManaged public var code: String?

    @NSManaged public var faculty: Faculty?
    @NSManaged public var groups: NSSet?

}

//MARK: - Generated accessors for groups
extension Speciality {

    @objc(addGroupsObject:)
    @NSManaged public func addToGroups(_ value: Group)

    @objc(removeGroupsObject:)
    @NSManaged public func removeFromGroups(_ value: Group)

    @objc(addGroups:)
    @NSManaged public func addToGroups(_ values: NSSet)
```

```
    @objc(removeGroups:)
    @NSManaged public func removeFromGroups(_ values: NSSet)

}

extension Speciality : Identifiable {}

//
//  SpecialityDecoder.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 15.10.21.
//

import CoreData

@objc(Speciality)
public class Speciality: NSManagedObject {

    required convenience public init(from decoder: Decoder) throws {
        self.init(context: decoder.userInfo[.managedObjectContext] as! NSManagedObjectContext)

        //If init is called from a decoder that decodes a Group and cannot find the required
specialty, a special method is called to process specific Group keys.
        if let groupContainer = decoder.userInfo[.groupEmbeddedContainer] as? Bool,
           groupContainer == true {
            try! self.updateFromGroupDecoder(decoder)
        } else {
            try! self.update(from: decoder)
        }
    }

}

//MARK: - Update
extension Speciality: DecoderUpdatable {
    func update(from decoder: Decoder) throws {
        let container = try! decoder.container(keyedBy: CodingKeys.self)

        self.id = try! container.decode(Int32.self, forKey: .id)
        self.name = try! container.decode(String.self, forKey: .name)
        self.abbreviation = try! container.decode(String.self, forKey: .abbreviation)

        self.code = try! container.decode(String.self, forKey: .code)

        self.educationType          =          try!          container.decode(EducationType.self,
forKey: .educationTypeContainer)

        if let facultyID = try? container.decode(Int16.self, forKey: .facultyID),
           let context = decoder.userInfo[.managedObjectContext] as? NSManagedObjectContext {
            self.faculty =  Faculty(id: facultyID, context: context)
        }

        Log.info("Speciality \(self.id) (\(self.abbreviation)) has been updated.")
    }

    func updateFromGroupDecoder(_ decoder: Decoder) throws {
        //If the decoder container is received from a Groups API call, the Speciality information
is contained in root, but if it is received from a Group API call, the required information is
contained in a nested container.
        let container = (try? decoder.container(keyedBy: GroupCodingKeys.self)
            .nestedContainer(keyedBy: GroupCodingKeys.self, forKey: .groupNestedContainer))
        ?? (try! decoder.container(keyedBy: GroupCodingKeys.self))

        self.id = try! container.decode(Int32.self, forKey: .id)
        self.name = try! container.decode(String.self, forKey: .name)
        self.abbreviation = try! container.decode(String.self, forKey: .abbreviation)

        self.faculty = try! Faculty(from: decoder)

    }
}

//MARK: - CodingKeys

extension Speciality: Decodable {
    private enum CodingKeys: String, CodingKey {
        case id
```

```
        case name
        case abbreviation = "abbrev"

        case code

        case facultyID = "facultyId"

        case educationTypeContainer = "educationForm"
    }

    private enum GroupCodingKeys: String, CodingKey {
        case groupNestedContainer = "studentGroupDto"

        case id = "specialityDepartmentEducationFormId"
        case name = "specialityName"
        case abbreviation = "specialityAbbrev"
    }

    private enum EducationTypeCodingKeys: String, CodingKey {
        case educationTypeId = "id"
    }
}

//
//  SpecialityFetch.swift
//  bsuirSchedule
//
//  Created by Andrej Hurynovič on 1.05.23.
//

import CoreData

extension Speciality: AbleToFetchAll {
    static func fetchAll() async {
        guard let data = try? await URLSession.shared.data(for: .specialities) else { return }
        let startTime = CFAbsoluteTimeGetCurrent()

        let (backgroundContext, decoder) = newBackgroundContextWithDecoder()

        guard let specialities = try? decoder.decode([Speciality].self, from: data) else {
            Log.error("Can't decode specialities.")
            return
        }

        await backgroundContext.perform(schedule: .immediate, {
            try! backgroundContext.save()
        })
        Log.info("\(String(specialities.count))        Specialities        fetched,        time:
\((CFAbsoluteTimeGetCurrent() - startTime).roundTo(places: 3)) seconds.\n")
    }
}
```

ПРИЛОЖЕНИЕ Б

(обязательное)
Спецификация

ПРИЛОЖЕНИЕ В

(обязательное)
Ведомость документов