АНОТАЦІЯ / ABSTRACT

Атестаційна робота магістра містить: с. – 76, рис. – 31, презентація.

ВНЗ, КРОС-ПЛАТФОРМНА РОЗРОБКА, МОБІЛЬНИЙ ДОДАТОК, РОЗКЛАД, ANDROID, IOS, HSL, JSON, MVVM, REST, XAMARIN.

Метою роботи є аналіз ринку мобільних платформ та розробка мобільного додатку для роботи з розкладом ВНЗ.

Наукові методи, використані в роботі, базуються на аналізі статистичних та прогнозованих даних.

Практичні методи полягають у використанні технологій .NET, Mono та Xamarin, мови програмування C#, середовища програмування Microsoft Visual Studio з метою проектування та створення додатку.

В результаті роботи виявлено актуальні мобільні платформи, здійснено аналіз засобів розробки, спроектовано та розроблено мобільний додаток із зазначеним функціоналом.

The certification work of the master contains: p. - 76, fig. – 31, presenation.

ANDROID, CROSS-PLATFORM DEVELOPMENT, IOS, HIGH SCHOOL, HSL, JSON, MOBILE APP, MVVM, REST, SCHEDULE, XAMARIN

The purpose of the work is to analyze the market of mobile platforms and develop a mobile application for working with university schedules.

The scientific methods used in the work are based on the analysis of statistical and forecast data.

Practical methods are to use .NET, Mono and Xamarin technologies, C# programming languages, Microsoft Visual Studio programming environment to design and create an application.

As a result of work the actual mobile platforms are revealed, the analysis of means of development is carried out, the mobile application with the specified functionality is designed and developed.

ЗМІСТ

ВСТУП

Темою даної роботи є проектування та реалізація додатку для студентів ТНТУ, що дозволить переглядати поточний розклад занять.

Дана тема є актуальною для галузі освіти (в даному випадку, для конкретного університету), оскільки існуюче рішення доступне лише у вигляді веб-сторінки. Розробка мобільного додатку дозволить зручно переглядати поточний розклад будь-де.

Для виконання даної роботи необхідно провести дослідження наявних на ринку мобільних платформ та особливостей програмування для них, виділити основні та дослідити способи ефективної, швидкої розробки програмного забезпечення, яке буде оптимальним для довготривалого користування, підтримки, розширення функціоналу та модифікації.

Об'єктом дослідження є розробка додатків під актуальні на сьогоднішній день мобільні платформи.

Предметом дослідження є технічні та програмні особливості смартфонів різних виробників, відмінність від аналогічних настільних систем, наявні на ринку операційні системи та причини їх популярності, прогнозування ринкової ситуації на майбутні кілька років для отримання рентабельності і актуальності.

Методами дослідження є:

- пошук в технічних публікаціях,
- дослідження апаратних особливостей,
- дослідження мікроархітектури процесора,
- аналіз статистичних даних з відкритих джерел,
- аналіз ринку IT на предмет мобільної розробки,
- аналіз даних, що підтверджуються статистичними дослідженнями та приведені на поточний та прогнозований періоди,
- порівняльна характеристика операційних систем.

Отримані результати мають важливе наукове та практичне значення.

Наукове значення полягає у особливостях статистичного аналізу та прогнозування даних перед розробкою програмного продукту. Такий підхід дозволить передбачити можливі ризики ще перед початком роботи над проектом та дасть змогу усунути їх вплив на подальший процес.

Практичне значення полягає у проведеній роботі над розробкою програмного продукту, оскільки може слугувати прикладом якісного аналізу ризиків, а також основних принципів та технологій, яких необхідно дотримуватись для швидкої та ефективної розробки схожого програмного продукту.

1 АНАЛІЗ МОБІЛЬНИХ ПЛАТФОРМ ТА СУЧАСНИХ МЕТОДІВ РОЗРОБКИ ДЛЯ МОБІЛЬНИХ ПРИСТРОЇВ

1.1 Дослідження особливостей мобільних платформ

На сьогоднішній день мобільна розробка є досить актуальною, оскільки практично кожен користується смартфоном. Особливістю даних пристроїв є те, що на відміну від звичайних ПК та ноутбуків їх зручно використовувати будь-де, а не лише за робочим столом. Інтернет-магазини, сервіси доставки, замовлення їжі онлайн, соціальні мережі масово створюють мобільні версії своїх сервісів, оскільки це забезпечує додатковий трафік користувачів і, як наслідок, додатковий прибуток. Актуальність мобільної розробки можна також довести за допомогою статистичних даних, що свідчитимуть про зміну кількості користувачів даного типу пристроїв.

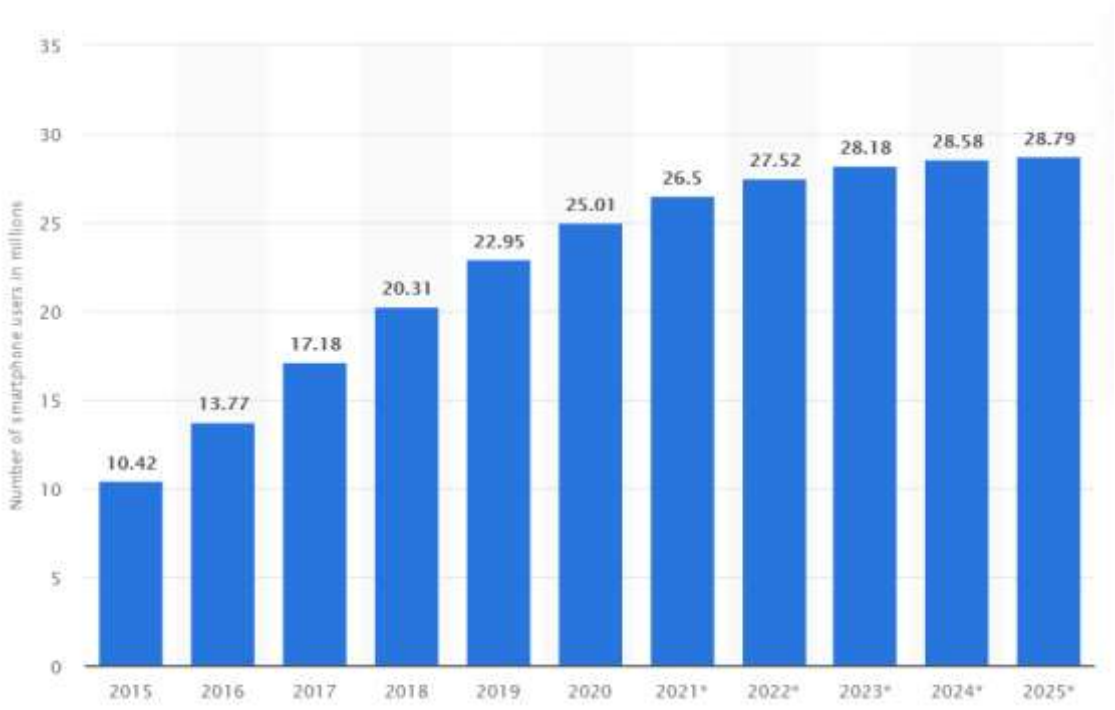

Рис. 1.1 – Фактичні дані та передбачення кількості користувачів смартфонів в Україні в період з 2015р. по 2025р.

Як можна бачити з графіку (рис. 1.1), кількість користувачів смартфонів в Україні зросла майже в 2,5 рази в період з 2015 до 2020р. Прогнозована кількість користувачів на наступні 5 років стверджує про збереження даної тенденції. Дана статистика наведена на основі відкритих даних [1].

На сьогодні існує багато мобільних операційних систем, проте багато з них не є популярними, багато застарілих а деякі так і не були випущені. Для того, щоб задіяти максимально широку аудиторію, потрібно проаналізувати частину ринку кожної мобільної ОС та визначити найбільш популярні.

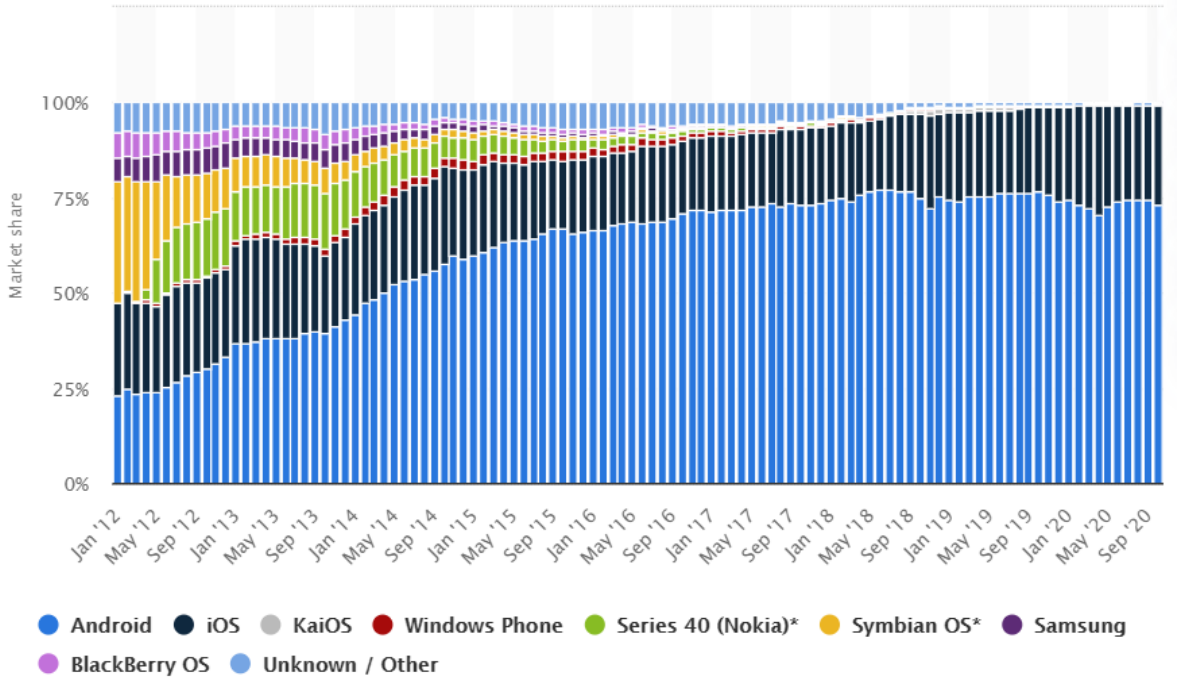

Рис. 1.2 – Співвідношення мобільних операційних систем в період 2012-2020р.

З діаграми (рис. 1.2) бачимо, що в період 2012-2016р. на ринку одночасно існувала велика кількість мобільних операційних систем, причому більшість з них – Symbian, S40 від Nokia, BlackBerry OS, Samsung та інші менш відомі ОС. Проте згодом Android та iOS стали найпопулярнішими та витіснили майже всі наявні мобільні платформи, і на сьогоднішній день сумарна Android (72.92%) та iOS (26.53%) складає близько 99.45% всіх пристроїв. Перевага Android над iOS пояснюється вартістю самих пристроїв: компанія Apple випускає лише потужні та дорогі смартфони та є монополістом iOS, Android, у свою чергу – платформа з

відкритим кодом, на основі якої працює безліч пристроїв різних виробників, що забезпечує майже 3/4 ринку [2].

Отож, можна підсумувати, що створюючи мобільний додаток потрібно орієнтуватись на дві лідируючі платформи на сьогоднішній день. Це забезпечить можливість всім користувачам завантажити мобільний додаток.

1.1.1 Операційна система Android

Android – операційна система для мобільних пристроїв, роботу над якою розпочали у 2003 році, в 2005 році компанія Google купила права розробку, а на сьогодні це найпопулярніша мобільна платформа. Її особливістю є відкритий вихідний код, що дозволяє створювати легко створювати власні оболонки різним компаніям, що випускають смартфони, а також доступність даної ОС [3].

Базується дана операційна система на ядрі Unix, тобто є схожою до ОС Linux, проте створення аплікацій мовою програмування C/C++ не є основним підходом. Android включає в себе вбудовану віртуальну машину мови Java під назвою Dalvik, що є оптимізованою для мобільних пристроїв, тому розробка нативними засобами передбачає написання додатків мовою Java. Розробка проводиться в середовищі Android Studio, що є дещо модифікованим середовищем IntelliJ IDEA Community.

Рис. 1.3 – Деякі пристрої, що підтримують Android

Android запускається не лише на смартфонах, а й на багатьох інших пристроях, до них входять як смарт-годинники, смарт-холодильники, так і телевізори та інша розумна побутова техніка (1.3). При цьому варто зауважити, що для кожного виду пристроїв додаток треба оптимізовувати або переписувати, так як різні особливості фізичних пристроїв можуть унеможливлювати запуск одного і того ж вихідного коду.

Віднедавна розроблена також нова мова програмування – Kotlin, байт-код якої також працює на віртуальній машині Java, проте сама мова має значно простіший синтаксис і спрощує роботу (рис. 1.4).

Рис. 1.4 – Однаковий за змістом код мовою Java та Kotlin

1.1.2 Операційна система iOS

На відміну від відкритого Android, iOS впродовж всього існування розроблялась лише компанією Apple, що має на неї ексклюзивні права, та встановлювалась лише на власні продукти компанії. Проте, завдяки на оптимізованості та зручності у використанні, більше четвертини ринку сьогодні – пристрої під керуванням iOS.

iOS, як і Android, запускається не лише на смартфонах. На сьогодні компанія Apple вже випускає багато продуктів, що використовують iOS як операційну систему. Сюди входять планшети та смарт-ТВ, також є смарт-годинники з дещо модифікованою watchOS (рис. 1.5).

Рис. 1.5 – Пристрої під управлінням iOS

iOS базується на ядрі Darwin, котре є теж власною розробкою компанії Apple. На відміну від Android, для якого основна мова програмування – Java, в iOS нативний спосіб створення додатків –мова програмування Objective-C, котра є першою ООП-інтерпретацією мови C (Objective-C з’явилась у 1980 році, C, в свою чергу – в 1983 році). Незважаючи на те, що мова базувалась на C, синтаксис досить

специфічний та часто незрозумілий для програмістів C/C++. Також є відносно сучасна мова програмування Swift, котра, як і Kotlin, була створена після 2010 року. Код, написаний на Swift, може використовувати фрагменти, написані мовою Objective-C, що робить перехід зручним та дає змогу інтегрувати старий код без необхідності повністю переписувати останній [4].

1.1.3 Проблема швидкого написання мобільних програм

Як бачимо з попередніх розділів, на ринку існують одночасно дві популярні мобільні платформи, проте кожна з них використовує зовсім інший підхід до написання мобільних додатків, включаючи мову програмування та середовище розробки. Проблема полягає в тому, що переважно завжди при створенні застосунку аудиторія потенційних користувачів вже користується обома платформами, тому необхідно забезпечити однаковий функціонал для обох платформ, але стандартними способами це зробити непросто з декількох причин:

- не всі елементи відображення доступні одночасно для обох платформ,
- платформи мають різні обмеження доступу до фізичних пристроїв (камера, мікрофон, Bluetooth, модуль зв’язку),
- при створенні окремих додатків для обох платформ важко забезпечити аналогічний функціонал, так як застосовуються різні мови програмування і код фізично розділений,
- підтримка таких проектів є складним завданням, оскільки потрібно змінювати одночасно обидва додатки,
- для розробки необхідні різні спеціалісти для кожної з платформ.

Можливим рішенням можна було б назвати перенесення існуючих настільних програм на мобільні пристрої. До прикладу, програми на MacOS – на пристрої iOS, програми для Linux – на пристрої Android. Проте тут проблема в

сумісності на фізичному рівні – настільні та мобільні процесори відрізняються не лише розміром та потужністю, але й архітектурою.

Більшість процесорів для ПК базується на архітектурі x86, яка бере свої витоки з ранніх процесорів Intel. Набір команд, який використовують дані процесори – CISC (складний набір команд) [5] містить дуже багато вузькоспеціалізованих команд, деякі з яких сьогодні можуть і не застосовуватись, проте в часи, коли програми розроблялись мовою assembler, дані інструкції суттєво спрощували програмування. Також усі команди мають різну довжину. Мобільні пристрої створюються з процесором архітектури ARM, в якого набір команд RISC (набір простих або швидких команд). Дана архітектура передбачає лише загальні команди фіксованої довжини в невеликій кількості. Перші процесори, випущені з архітектурою RISC, були настільки простими, що не підтримували навіть звичайне множення або ж ділення як базові команди [6].

Можна підсумувати, що перенесення настільних програм для мобільних пристроїв є неможливим не лише через різні інтерфейси, а й через різну будову процесора. Код, скомпільований для CISC, буде незрозумілий для RISC, і навпаки.

Тому єдиним способом спрощення розробки було створення певного засобу, що дозволить програмувати одночасно для обох платформ. Це б здешевило, пришвидшило та популяризувало розробку.

1.2 Крос-платформні засоби розробки

В контексті даної проблеми з’являється поняття крос-платформної розробки: такої, що дозволяє запускати один код на різних пристроях. Багато компаній здійснювали спроби першими створити та запатентувати єдиний спосіб розробки для різних платформ. Для мобільних пристроїв було представлено кілька технологій крос-платформної розробки, що мали спільні і відмінні риси та використовували різні принципи використання нативних ресурсів ОС. Оскільки

завданням даної роботи є створення мобільного додатку, то доцільно використати одну з наявних технологій для швидкого створення та простої підтримки в майбутньому [7]. Для цього здійснимо огляд існуючих засобів та виберемо найбільш зручний та ефективний.

1.2.1 Apache Cordova

Перша спроба вирішити цю проблему була здійснена в 2011 році, коли компанії Adobe Systems, Apache Software Foundation випустили першу версію фреймворку Apache Cordova. Початково проектом займалась компанія PhoneGap, проте проект був переданий до Apache. Фреймворк призначений для створення «крос-платформних» додатків для різних мобільних операційних систем, суть якого полягає у створенні, фактично, веб-сторінок за допомогою HTML, CSS та JS, та відображення їх у компоненті веб-браузера [8]. Це спростило розробку додатків, проте фреймворк мав і численні недоліки, серед яких можна виділити такі:

- існуючі плагіни для Cordova швидко застарівали і потрібно писати свої;
- так як програми запускались у веб-браузері, то виникали складності в отриманні зворотного зв'язку з нативним додатком;
- потрібно було постаратися над оптимізацією додатку, щоб він працював швидко і стабільно.
- все одно додатки на Apache Cordova виглядали і працювали не так плавно, як нативні додатки.

В основному, підхід виглядав як веб-сайт, що відкривається окремим додатком і не мав доступу до більшості апаратних засобів, тому не став популярним і часто вживаним. Правильним підходом було б використовувати нативні засоби обох платформ однією мовою програмування.

1.2.2 React Native

Наступний крок був від компанії Facebook, що представила продукт, який відрізнявся від Apache Cordova принципом дії, хоч теж був побудований на JS-коді, а також був позбавлений її недоліків – React Native.

Логіка додатків у React Native пишеться мовою програмування JavaScript, як і для Apache Cordova. Але замість веб-сторінки додаток використовує нативні елементи управління та бібліотеки та виступає так званим інтерпретатором між написаним кодом та операційною системою [9]. В результаті додаток виглядає і працює повністю як нативний (підтримує жести, навігацію, доступ до системних ресурсів). Технологія, яка дозволяє реалізувати все вищеперераховане, дуже проста.

Кожен ReactNative додаток запускає два важливих потоки:

- Основний потік – обробляє все, так чи інше пов’язано з реалізацією операційної системи: представлення, дзвінки, календар, доступ до файлів, та ін.
- JavaScript потік – власне виконує код застосунку та відповідає за правильну роботу основної логіки програми, як от обчислення, збереження даних чи веб-запити.

Виникає питання як спілкуються ці потоки, та за рахунок чого такий спосіб побудови дає суттєву перевагу перед попередньою технологією крос-платформної розробки.

Для спілкування між цими потоками використовується певний спосіб обміну повідомленнями через міст (bridge), який є ядром технології React Native. Міст дозволяє спілкуватись та обмінюватись інформацією між JavaScript та основним потоком, при цьому зберігаючи асинхронну парадигму спілкування, тобто таку, що не передбачає ніяких очікувань та блокувань потоку. Асинхронний підхід дає змогу позбутись ситуацій, коли один з потоків заблокує іншого (так званий deadlock), та зробити комунікацію між логікою та представленням швидкою та ефективною. Для створення додатків за допомогою React Native потрібно спершу володіти

парадигмою розробки web-додатків React. React – модульний принцип побудови web-застосунків, що передбачає розділення представлення та логіки, створення атомарних компонентів, які згодом можна перевикористати у багатьох частинах проекту. Проте у React для представлення використовується вже звичний HTML, у той час як для ReactNative – особлива розмітка та стилі, що більше схожа до AXML для Android (рис. 1.6).

```
const DisplayAnImage = () => {
  return (
    <View style={styles.container}>
      <Image
        style={styles.tinyLogo}
        source={require('@expo/snack-static/react-native-logo.png')}
      />
      <Image
        style={styles.tinyLogo}
        source={{
          uri: 'https://reactnative.dev/img/tiny_logo.png',
        }}
      />
      <Image
        style={styles.logo}
        source={{
          uri:

'
EXRFWHRTb2Z0d2FyZQBwbmdjcnVzaEB1SfMAAABQSURBVGje7dSxCQBACARB+2/ab8QN
hFi6WSYzYLYudDQYGBgYGBgYGBgYGBgZmcvDqYGBgmhivGQYGBgYGBgYGBgYGBg
bmQw+P/eMrC5UTVAAAAABJRU5ErkJggg==',
        }}
      />
    </View>
```

Рис. 1.6 – Розмітка представлення для React Native

Можна зробити висновок, що даний підхід чудово підійде для веб-розробників, котрі раніше працювали з React, проте для опанування даної технології з нуля необхідно трохи часу. Наступною проблемою може стати мова JavaScript, котра не є строго типізована. Для розробника, що звик працювати з строго типізованою мовою програмування досить важко перевчитись використовувати динамічно типізовані об’єкти. Проблема стала досить значною, тому Microsoft випустили мову програмування TypeScript у 2012 році. TypeScript, хоч і є зворотньо сумісною з JS, проте передбачає можливість строгої типізації:

створення класів, інтерфейсів, параметризація вхідних та вихідних параметрів методів, класи-шаблони.

1.2.3 Крос-платформна технологія Xamarin

Якщо поглянути на історію настільних програм для Windows, можна виявити, що до певного часу додатки розроблялись виключно за допомогою засобів C/C++ та використовуючи нативні Windows API (user32.dll). Проте потреба в графічному представленні росла, а розробляти складні та індивідуальні програми мовою C/C++ було складно. До того ж, Microsoft давно розробляла свою мову програмування з віртуальною машиною, базуючись на історії виникнення Java. Так, у 2000 році з'явилась мова програмування C# та платформа .NET, що за своєю парадигмою дуже схожа з Java: написаний код не компілюється в машинний, а в свого роду проміжний, який при запуску насправді виконується всередині віртуальної машини. Остання й адаптує код до апаратного забезпечення комп'ютера. Проте C# мав можливість запускатись лише на ОС Windows, оскільки була прив'язка до нативних API даної операційної системи.

Після успіху C# компанія Novell (пізніше – Xamarin) створює крос-платформну версію .NET – Mono. Метою Mono було використання C# на Unix-подібних системах.

Згодом компанія Microsoft випускає революційну платформу створення користувацького інтерфейсу для ОС Windows – WPF (Windows Presentation Foundation) [10]. Даний фреймворк мав безліч переваг перед стандартним GDI+:

- Підтримку апаратного прискорення (DirectX),
- декларативний опис представлення (XAML),
- прив'язку даних безпосередньо до представлення (рис. 1.7), без посередників (MVVM) [11],
- можливість розширення,

– модульність,

– повна стилізація інтерфейсу (рис. 1.8).

```
<TextBlock Margin="10" Text="Simple Text" Name="lblSampleText" FontSize="{Binding ElementName=sliderFontSize, Path=Value}" >
</TextBlock>
```

Рис. 1.7 – Прив’язка даних у XAML

```
<Window.Resources>
<Style x:Key="BigFontButtonStyle">
<Setter Property="Control.FontFamily" Value="Times New Roman" />
<Setter Property="Control.FontSize" Value="8" />
<Setter Property="Control.FontWeight" Value="Bold" />
</Style>
</Window.Resources>
```

Рис. 1.8 – Стилізація у XAML

Цей підхід дозволив дуже швидко і просто створювати настільні додатки для будь-яких ОС Windows, починаючи з Windows Vista, що однаково виглядали і створювались нативним способом, до того ж працювали швидко та мали можливість відображати складну 2D та 3D графіку.

Компанія Xamarin використала дані досягнення для створення крос-платформного рішення, і у 2011 році розпочинає роботу над однойменним фреймворком. Для запуску додатків було використане крос-платформне ядро Mono, а от для створення – мова C# і принципи, схожі з WPF [12].

На сьогодні Xamarin передбачає два різних підходи у створенні мобільних аплікацій (рис. 1.9) [13]:

- Xamarin Native – спільним шаром є бізнес-логіка програми (як от отримання даних з сервера чи бази даних, калькуляції, збереження налаштувань, використання сповіщень). Представлення користувача (інтерфейс) виконується нативними засобами (aXML для Android та Storyboard для iOS), що дозволяє створити індивідуальний та унікальний дизайн, анімації як при створенні нативного додатку. Для доступу до нативних засобів використовуються Xamarin.iOS та Xamarin.Android бібліотеки. Незважаючи на необхідність окремого написання

представлення, основна частина більшості додатків полягає в внутрішній логіці та обробці даних, тому цей підхід є досить ефективним. Зазначимо, що застосовувати його варто лише тоді, коли необхідно створити унікальне представлення, що буде повністю відповідати бажаному дизайну та мати унікальні особливості поведінки нативної ОС. Для інших випадків існує підхід Xamarin.Forms, що значно більше спрощує створення мобільних додатків.

- Xamarin Forms – спільним шаром є не лише бізнес логіка, а й користувацький інтерфейс. Такий підхід дозволяє використати загальні засоби й елементи, що доступні для обох платформ, як от списки, поля вводу, посилання, галерея, нотифікації та описати інтерфейс мовою XAML. При цьому повністю доступні більшість засобів, що були створені для WPF, такі як стилізація, панелі, прив'язка даних, конвертування даних для представлення, команди. Даний підхід застосовується для надзвичайно швидкого створення додатків та дозволяє перевикористати код для обох платформ. При цьому за необхідності можна описати створення деяких елементів окремо для обох платформ. Це забезпечує застосовність даного підходу для більшості випадків, якщо дизайн не є на першому місці.

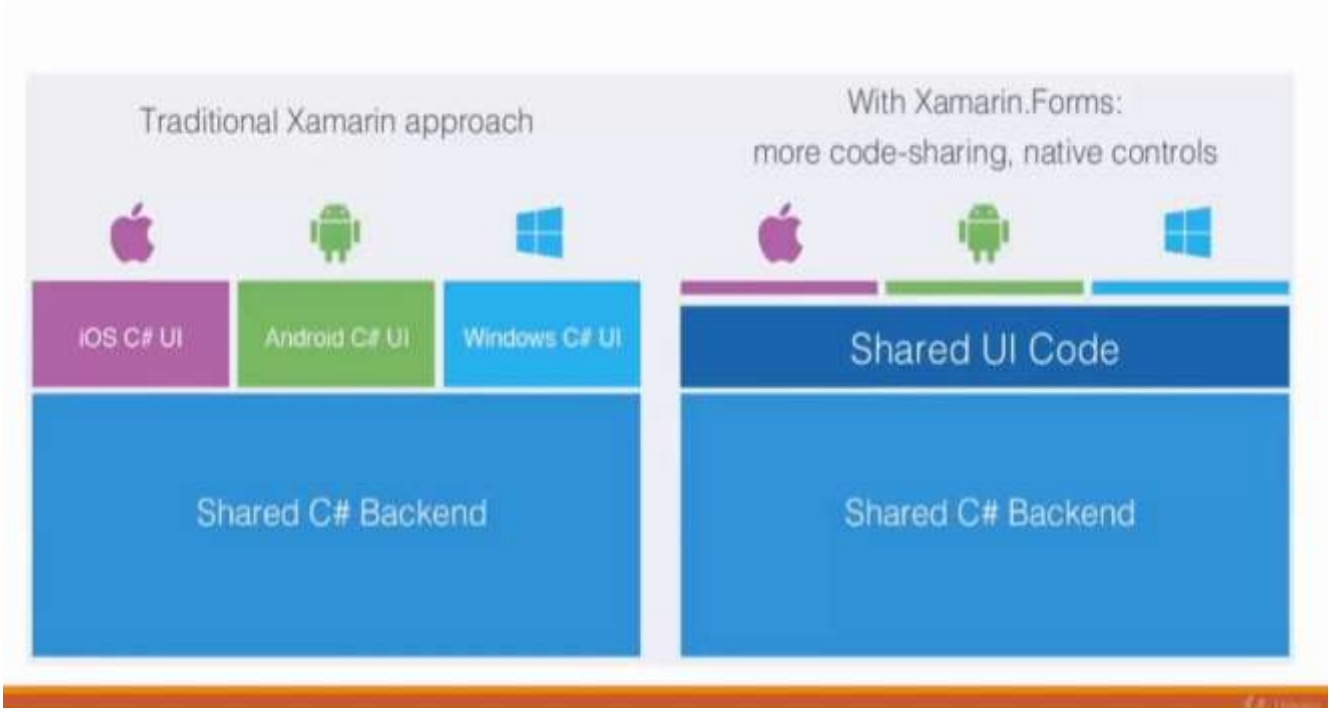

Рис. 1.9 – Архітектура Xamarin Native та Xamarin Forms

Для програмування на платформі Xamarin використовується .NET Framework або .NET Core та мова програмування C#. Це дозволяє перевикористати існуючий код настільного додатку або веб-сайту ASP.NET.

Отже, для розробки мобільних додатків з використанням Xamarin потрібно:

- мати знання платформи .NET та мови програмування C#,
- мати мінімальний досвід з розробкою настільних додатків за допомогою WPF
- знати та правильно використовувати прив'язку даних (Binding),
- розуміти та застосовувати архітектурний підхід MVVM.

На основі даних критеріїв Xamarin було вибрано як крос-платформний засіб для розробки мобільного додатку розкладу ТНТУ. Створений додаток можна буде скомпілювати одночасно як для Android, так і для iOS. Варто зазначити, що для компіляції додатку під iOS необхідно мати пристрій під управлінням macOS, при цьому для розробки даний критерій не є обов'язковим.

При виборі середовища розробки враховувати варто лише операційну систему комп'ютера, за допомогою якого буде проводитись розробка:

- Для Windows найзручнішим інструментом розробки є Visual Studio – інтегроване середовище розробки, що підтримує багато мов програмування, у тому числі C#, C++, JavaScript, Python, та ін. Для роботи з Xamarin у Visual Studio потрібно переконатись, що Xamarin встановлено, або ж довстановити його, використовуючи Visual Studio Installer (рис. 1.10)
- Для macOS є спеціально розроблене середовище Visual Studio for Mac, що є дещо спрощеною реалізацією Visual Studio, проте забезпечує аналогічний функціонал при розробці мобільних додатків з використанням Xamarin.

Важливим аспектом є також те, що для розробки та налагодження можна використовувати як фізичні пристрої (смартфони), так і емулятори – мобільні ОС, запущені в контейнері, які можна відкрити на ПК та перевірити в них роботу додатку.

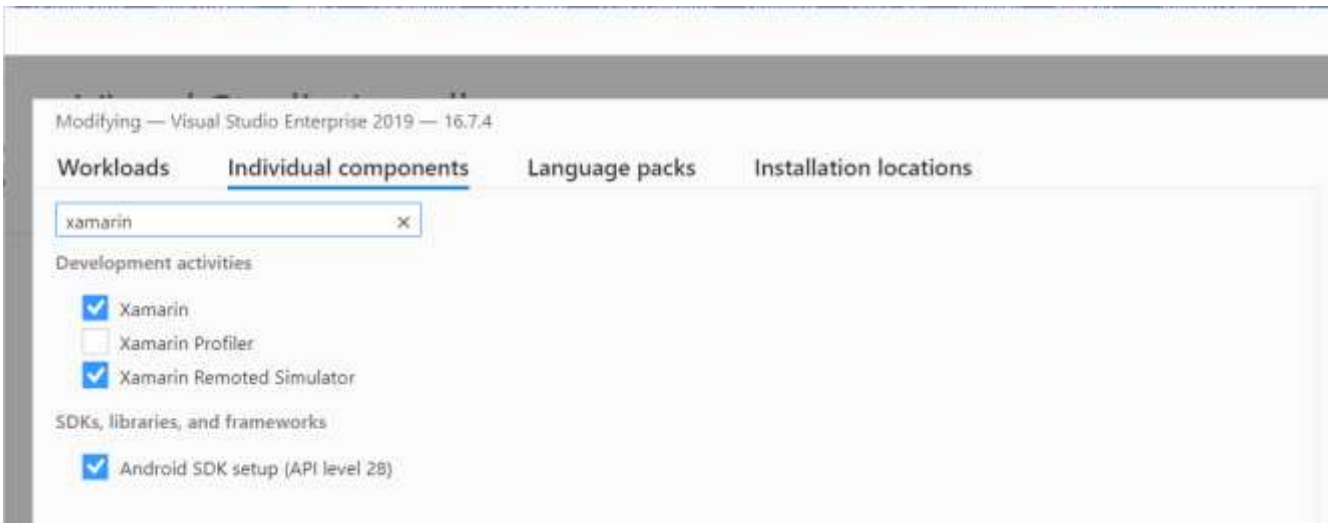

Рис. 1.10 – Додавання Xamarin до пакетів Visual Studio 2019

Після додавання пакетів у середовищі Visual Studio з'являться нові шаблони для проектів – Xamarin Android, Xamarin iOS, Xamarin Native та Xamarin Forms. В даній роботі використано підхід Xamarin.Forms.

2 ПІДГОТОВКА ТА ОБРОБКА ДАНИХ ДЛЯ ДОДАТКУ

2.1 Отримання даних за допомогою RESTful API

Для отримання даних через мережу використовуються різні засоби та протоколи [14], зокрема:

- для отримання текстових даних та веб-сторінок – HTTP,
- для захищеної передачі даних – HTTPS,
- для комунікації в реальному часі та сповіщень – WebSockets,
- для забезпечення цілісності передавання даних – TCP,
- для потокової передачі великих обсягів даних – UDP,
- для віддаленого виклику процедур та передачі електронних листів– SOAP.

Найпопулярнішим способом комунікації з веб-сервером для отримання, збереження, зміни або видалення даних найчастіше використовується RESTful API.

Варто зазначити, що REST – це не мережевий протокол, а швидше певний стандарт, домовленість, оскільки передача даних здійснюється переважно за допомогою HTTP або HTTPS. REST перекладається як передача стану і суть його полягає в передачі серіалізованих об’єктів через мережевий протокол HTTP/HTTPS, при цьому використовуючи лише обмежений набір із всіх наявних HTTP-методів:

- GET – застосовується для отримання певних об’єктів,
- DELETE – застосовується для видалення об’єктів,
- POST – застосовується для створення об’єктів,
- PUT – застосовується для оновлення вже існуючих об’єктів.

Також REST може використовувати деякі допоміжні методи для дослідження існуючого API:

- OPTIONS – отримати список доступних методів,
- HEAD – отримати лише заголовки.

Дані в RESTful API називаються репрезентативним станом або представленням, про що свідчить і назва підходу (Representative State Transfer). При цьому, на відміну від RPC, де дані передаються у бінарному форматі, REST використовує текстовий формат передачі даних, а тому при передачі об’єктів їх спочатку необхідно перевести у текстове представлення [15].

При виборі формату представлення варто враховувати, які завдання можна буде виконати з відповіддю на стороні отримувача. XML є старішим варіантом серіалізації і містить дещо надлишкові дані, оскільки кожен елемент має бути описаний закриваючим та відкриваючим тегом (рис. 2.1).

XML

```
<empinfo>
   <employees>
      <employee>
         <name>James Kirk</name>
         <age>40></age>
      </employee>
      <employee>
         <name>Jean-Luc Picard</name>
         <age>45</age>
      </employee>
      <employee>
         <name>Wesley Crusher</name>
         <age>27</age>
      </employee>
   </employees>
</empinfo>
```

JSON

```
{  “empinfo” :
      {
            “employees” :  [
            {
                  “name” : “James Kirk”,
                  “age” : 40,
            },
            {
                  “name” : “Jean-Luc Picard”,
                  “age” : 45,
            },
            {
                  “name” : “Wesley Crusher”,
                  “age” : 27,
            }
                              ]
      }
}
```

Рис. 2.1 – Порівняння XML та JSON форматів серіалізації при однаковому обсязі та структурі серіалізованих даних

XML є зручним способом представлення, коли з результатом треба одразу ж виконати деякі обчислення (додати властивості на основі існуючих, порахувати агрегуючі значення, перевести у інший формат, перевести в HTML). Такі конвертації здійснюються за допомогою XSLT-перетворень (рис. 2.2).

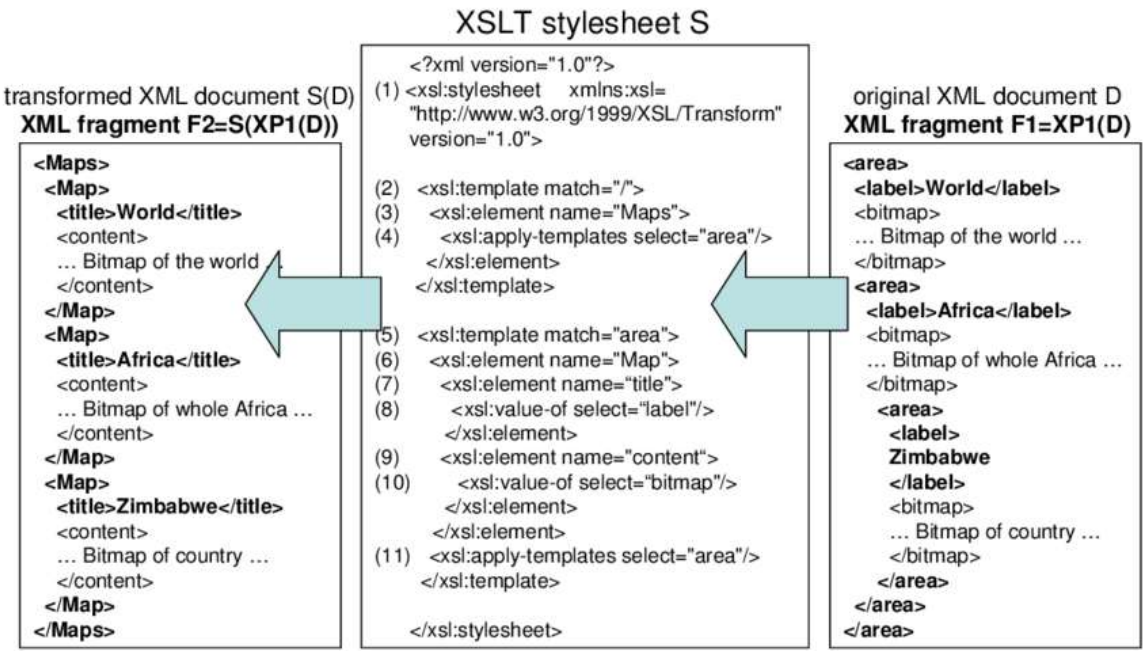

Рис. 2.2 – Приклад XSLT-перетворення.

Проте в сучасних завданнях перетворення в XML певного типу дуже рідко є важливим завданням, тому JSON на сьогодні – оптимальний спосіб серіалізації даних для передачі через REST API, оскільки розмір пакету є меншим в порівнянні з XML, а десеріалізація мовою програмування C# проводиться засобами рефлексії, що забезпечує можливість серіалізувати/десеріалізувати будь-який об’єкт без необхідності написання для цього додаткового коду [16]. Такий підхід можливий завдяки метаданим, що містяться в кожній програмі, написаній з використанням .NET. Серіалізатор отримує наявний список властивостей об’єкта та використовує його для серіалізації/десеріалізації об’єктів. JSON також є дуже зручним для читання та містить кілька базових типів, таких як bool, object, array, string, number, decimal, null.

Для даного додатку буде створено REST API для отримання актуального розкладу занять. Даний ресурс буде відкрито для публічного доступу на сервері ТНТУ, що дасть змогу студентам завантажувати та зручно переглядати розклад з мобільного пристрою, при цьому розклад буде повністю відповідати розміщеному на веб-сторінці університету.

Щоб пришвидшити та автоматизувати документацію API, оптимальним підходом є автоматична генерація. Здійснити такий підхід можна з використанням розширення Swagger.

Swagger – безкоштовний набір інструментів, котрі дозволяють швидко й ефективно здійснювати перевірку серверних медотів REST API на правильність обробки результатів, а також можливість переглянути наявні методи, які можна використати (рис. 2.3).

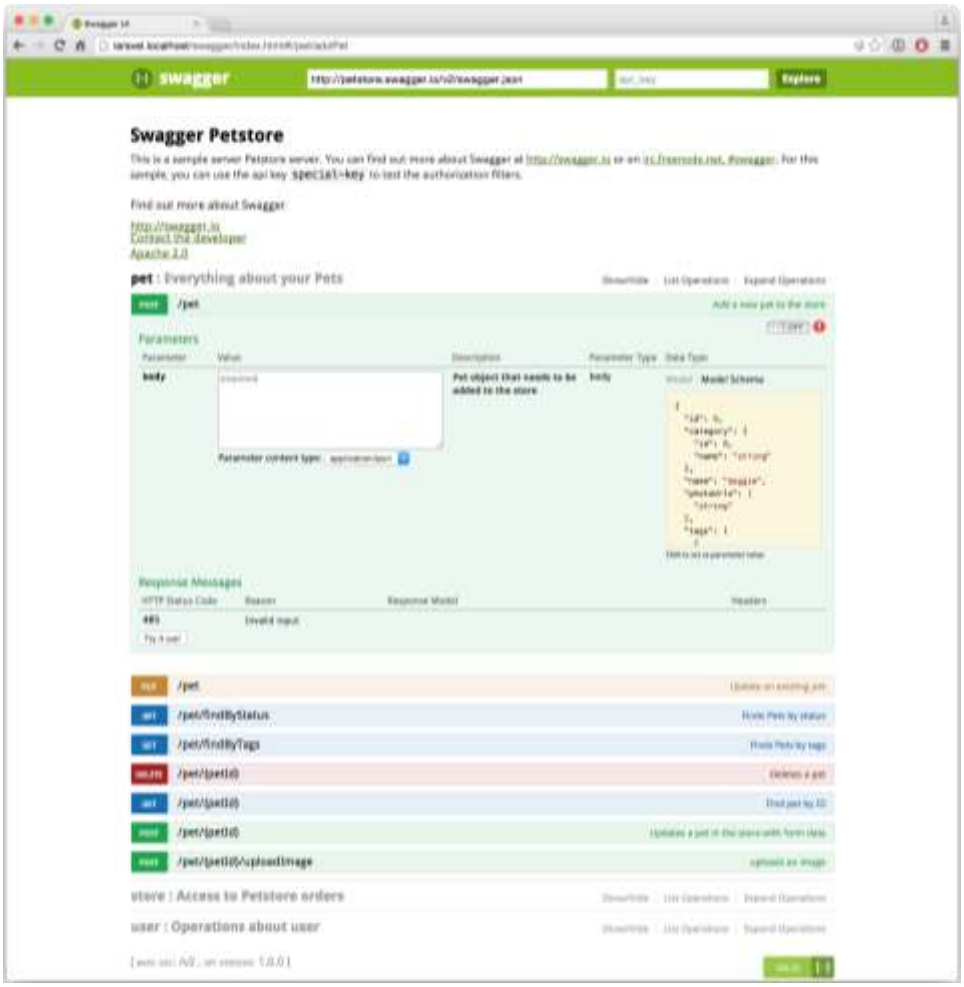

Рис. 2.3 – Представлення API за допомогою Swagger

Після обробки наявного API swagger генерує файл swagger.json з повним описом наявних методів, а для представлення генерується окрема веб-сторінка.

2.2 Парсинг веб-сторінки розкладу

На сьогоднішній день багато ресурсів вже мають наявні REST API для доступу до відкритих даних або роботи з даними (додавання, видалення, модифікація) з можливістю авторизації. Проте існують випадки, коли дані наявні лише на веб-сторінці, або ж потрібно агрегувати дані з багатьох ресурсів і деякі з них розміщують дані лише на веб-сторінці.

В таких випадках єдиним можливим варіантом отримання таких даних є парсинг веб сторінок [17].

В межах даної роботи окрім інтеграції API розкладу від університету було також реалізовано парсинг існуючої веб-версії розкладу. Розташування веб-версії розкладу: http://tntu.edu.ua/?p=uk/schedule.

Для парсингу даних спочатку необхідно виділити модель даних, яку ми повинні отримати в результаті обробки веб-сторінок. В даному випадку можна виділити три основні сутності: «Факультет», «Група», «Заняття». Для об’єднання даних в один об’єкт створено сутність «Розклад». Також, після аналізу розкладу на веб-сторінці було виявлено, що заняття може розбиватись на підгрупи та тижні (рис. 2.4).

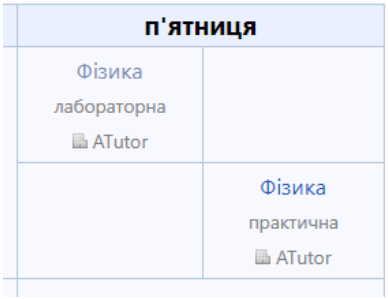

Рис. 2.4 – Розбиття заняття на підгрупи (по горизонталі) та дні тижня (по вертикалі)

Для цього було створено додаткові перерахування «День тижня» та «Підгрупа». Спільні властивості, як-от ім’я та ідентифікатор було винесено в

загальну сутність «Елемент розкладу», від якого наслідуються відповідно «Факультет», «Група» та «Заняття». Загальна структура даних зображена на діаграмі (рис. 2.5).

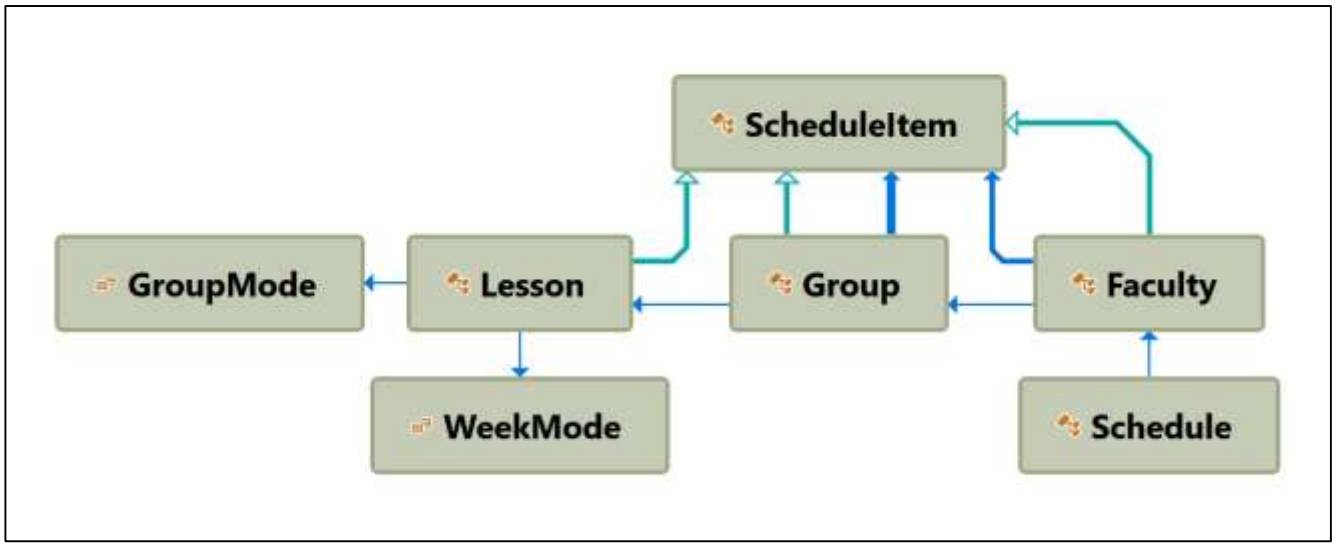

Рис. 2.5 – Структура даних розкладу занять

Найважливішим елементом даної моделі є сутність «Заняття», наведемо лістинг коду даної моделі:

```
public sealed class Lesson : ScheduleItem
{
     #region Properties
     public int GroupId { get; set; }
     public int CourseId { get; set; }
     public string Type { get; set; }
     public string Location { get; set; }
     public DayOfWeek Day { get; set; }
     public int Number { get; set; }
     public WeekMode WeekMode { get; set; }
     public GroupMode GroupMode { get; set; }
     #endregion
}
```

Як бачимо з лістингу, дана сутність міститиме таку інформацію, як назву занятя, розташування (аудиторію), день тижня, підгрупу, ID курсу, посилання на систему ATutor, тип заняття (лекція, лабораторна, практична), а також для якого тижня це заняття (перший, другий або всі тижні).

Для парсингу даних з веб-сайту необхідно завантажити сторінку та відтворити її DOM-модель. Всі веб-сторінки, створені за допомого HTML, є

своєрідним деревом об’єктів: кожен об’єкт може містити список дочірніх, наприклад, список – елементи списку, параграф – стрічки з текстом, заголовок – список посилань. Для отримання списку факультетів необхідно зорієнтуватись, які саме елементи веб-сторінки нам потрібно витягнути. Для цього скористаємось засобами розробника в веб-браузері (в даному випадку – Google Chrome) та знайдемо необхідні елементи (рис. 2.6).

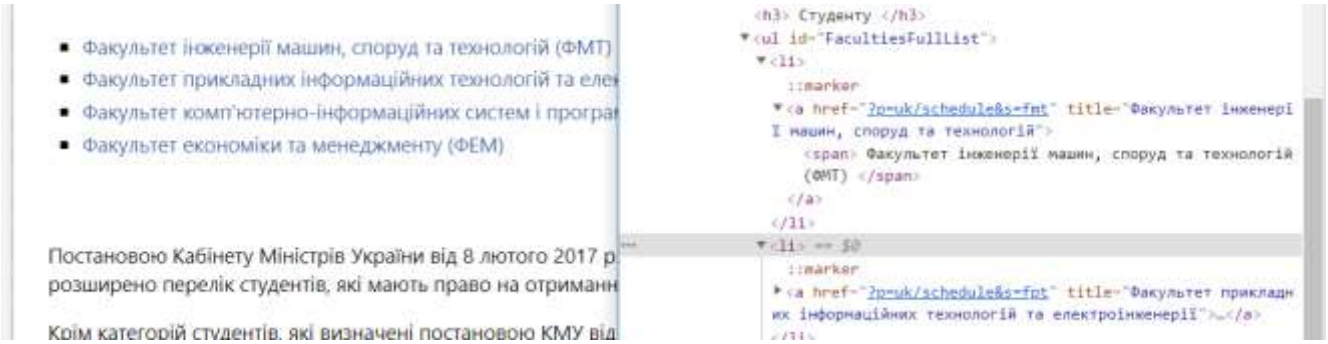

Рис. 2.6 – Список факультетів – дерево об’єктів

Як бачимо зі структури, для отримання списку факультетів та відповідних посилань необхідно отримати всі посилання (a) всередині невпорядкованого списку (ul) з ідентифікатором FacultiesFullList.

Для парсингу сторінок скористаємось бібліотекою AngleSharp. Вона дозволяє використовувати не лише XPath запити, що потребують точного абсолютного шляху, а й звичайні CSS-селектори, які значно простіші і з більшою ймовірністю продовжать працювати при мінімальній зміні структури веб-сторінки [18].

Дану бібліотеку можна встановити за допомогою засобу «NuGet Package Manager», що є основним способом встановлення додаткових бібліотек коду для платформи .NET (рис. 2.7).

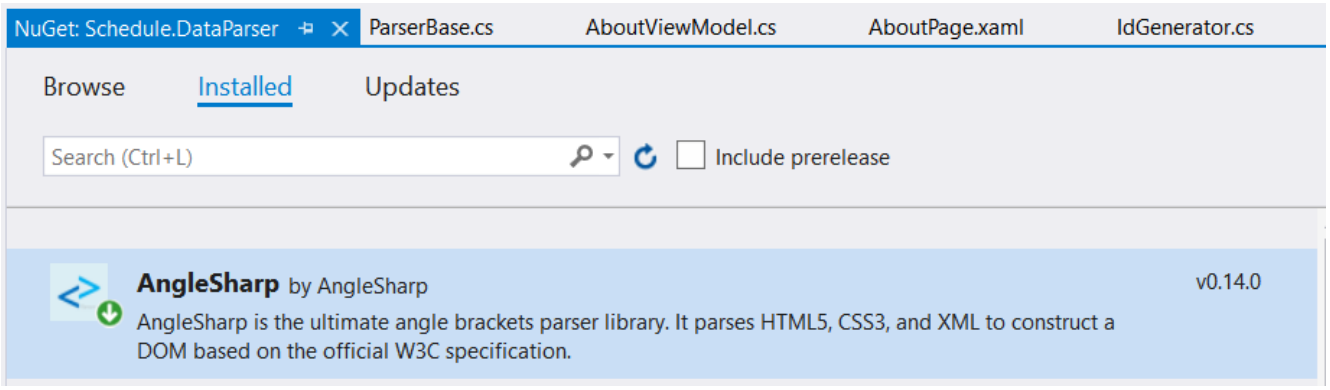

Рис. 2.7 – Встановлення AngleSharp

Наведемо код, необхідний для парсингу наявних факультетів з даної веб-сторінки:

```
    public async Task<List<Faculty>> ParseFacultiesAsync()
    {
        var document = await
GetDocumentAsync(ScheduleUrl).ConfigureAwait(false);
        var array = await Task.WhenAll(
    document.QuerySelectorAll("#FacultiesFullList a")
    .Select(ParseFacultyAsync))
    .ConfigureAwait(false);
        return array.ToList();
    }

    private async Task<Faculty> ParseFacultyAsync(IElement element)
    {
        var groupsParser = new GroupsParser();
        var selfUrl = element.GetAttribute("href");
        var faculty = new Faculty
        {
            Name = element.FirstElementChild.InnerHtml.Trim(),
            SelfUrl = Url(selfUrl),
            Groups = await
groupsParser.ParseGroupsAsync(selfUrl).ConfigureAwait(false)
        };

        return faculty;
    }
```

В даній реалізації можна побачити, що використовується CSS-селектор «#FacultiesFullList a», за допомогою якого отримуються потрібні елементи. Наступний крок – парсинг груп, який також можна побачити для кожного факультету. Принцип тут схожий – досліджуємо елемент посилання групи, знаходимо необхідний CSS-селектор, який необхідно використати для отримання

повного списку. Єдина різниця – для витягування сторінок з групами в даному випадку всі запити запускаються одночасно (await Task.WhenAll). Це дозволяє пришвидшити парсинг, проте збільшує навантаження на сервер, що може трактуватись як DDoS атака. Враховуючи той факт, що для кожного фактультету існує 50-70 груп, а завантаження сторінки займає 3-5 секунд, потрібно обмежити навантаження на сервер веб-сайту, щоб мати змогу отримати дані і не нашкодити серверу. Для цього об'єднаємо наступні запити для отримання кожного розкладу в групи по 10 запитів:

```
    public async Task<List<Group>> ParseGroupsAsync(string url)
    {
        var document = await
GetDocumentAsync(url).ConfigureAwait(false);
        var groups = new List<Group>();
        foreach (var batches in
document.QuerySelectorAll("#GroupsList a").Batch(10))
        {
            var result = await
Task.WhenAll(batches.Select(ParseGroupAsync)).ConfigureAwait(false);
            groups.AddRange(result);
        }
        return groups;
    }
```

Найскладніше завдання – парсинг занять, оскільки на даному веб-сайті розклад представлений таблицею, що має об'єднані клітинки та колонки. Проілюструємо дану проблему візуально для кращого розуміння, перший рядок – червоний, другий – зелений (рис. 2.8).

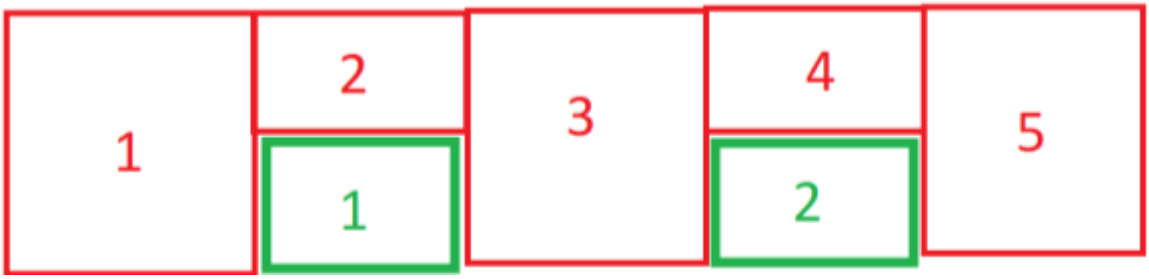

Рис. 2.8 – Розміщення вертикально об'єднаних клітинок та позиціонування наступного рядка

Як можна спостерігати з схематичного рисунка, якщо в першому рядку є об’єднані клітинки, що займають частину наступного рядка, то наступний рядок заповнює лише вільні клітинки і в даному випадку містить лише 2 клітинки, а не 5.

Для вирішення даної проблеми і аналогічної з об’єднаними колонками було вирішено створити дублюючу матрицю прапорців, розмір якої відповідає максимальному розміру таблиці з розкладом, включаючи розділені рядки та колонки. При цьому, для встановлення відповідності по дням тижня, підгрупам та порядковим парам занять створено додатково кілька словників, де ключем є поточний найменший рядок чи колонка, а значенням – відповідні параметри:

```
private Dictionary<int, DayOfWeek> _columnDictionary;
private Dictionary<int, GroupMode> _groupModeDictionary;
private int _columns;
private int _rows;
private bool[,] _matrix;
private Dictionary<int, int> _rowDictionary;
private Dictionary<int, WeekMode> _weeksDictionary;
```

Повний код парсингу занять наведено в додатку, тут наведемо лише отримання даних про конкретне заняття без прив’язки до позиції у розкладі та без деталей про підгрупу та тиждень:

```
private Lesson ParseLesson(IElement element)
{
    var divs = element.QuerySelectorAll("div");
    var firstDiv = divs[0];
    var secondDiv = divs[1];
    var reference = firstDiv.ChildNodes.Any(n => n.NodeType ==
NodeType.Element)
        ? firstDiv.QuerySelector("a")
        : firstDiv;
    var texts = secondDiv.ChildNodes.Where(n => n.NodeType ==
NodeType.Text).ToArray();
    return new Lesson
    {
        Name = reference.InnerHtml.Trim(),
        SelfUrl = reference.HasAttribute("href") ?
reference.GetAttribute("href") : null,
        Type = texts.First().TextContent.Trim(),
        Location = texts.Last().TextContent.Trim()
    };
}
```

Як можна було бачити з парсингу факультетів, даний процес є свого роду рекурсивним: парсинг факультетів запускає парсинг груп, той, у свою чергу –

занять. В кінцевому результаті отримаємо список факультетів з повними даними про групи та заняття в кожній з них. Для того, щоб мати можливість працювати з отриманими даними, потрібно, щоб кожна сутність мала унікальний ідентифікатор, який можна буде використовувати як посилання. Для цього використаємо вже існуючий ObjectIDGenerator:

```
private ObjectIDGenerator _generator;
public void FillGeneratedIds(Data.Models.Schedule schedule)
{
    _generator = new ObjectIDGenerator();
    FillFacultiesIds(schedule.Faculties);
}
private void FillFacultiesIds(IEnumerable<Faculty> faculties)
{
    foreach (var faculty in faculties) {
        faculty.Id = (int)_generator.GetId(faculty, out _);
        FillGroupIds(faculty.Groups);
    }
}
private void FillGroupIds(IEnumerable<Group> groups)
{
    foreach (var group in groups) {
        group.Id = (int)_generator.GetId(group, out _);
        FillLessonIds(group.Lessons);
    }
}
private void FillLessonIds(IEnumerable<Lesson> lessons)
{
    var objectIdGenerator = new ObjectIDGenerator();
    foreach (var lesson in lessons) {
        lesson.Id = (int)_generator.GetId(lesson, out _);
    }
    foreach (var grouping in lessons.GroupBy(l => l.Name)) {
        var id = objectIdGenerator.GetId(grouping.Key, out _);
        foreach (var lesson in grouping) {
            lesson.CourseId = (int)id;
        }
    }
}
```

Отримані дані заздалегідь збережемо в форматі .json для подальшого використання у мобільному додатку, як тимчасові дані.

Після запуску парсера спостерігаємо деяку кількість запитів на веб-сайт розкладу ТНТУ (рис 2.9), після чого отримуємо файл з повним розкладом занять усіх груп, який є доступ на веб-сторінці.

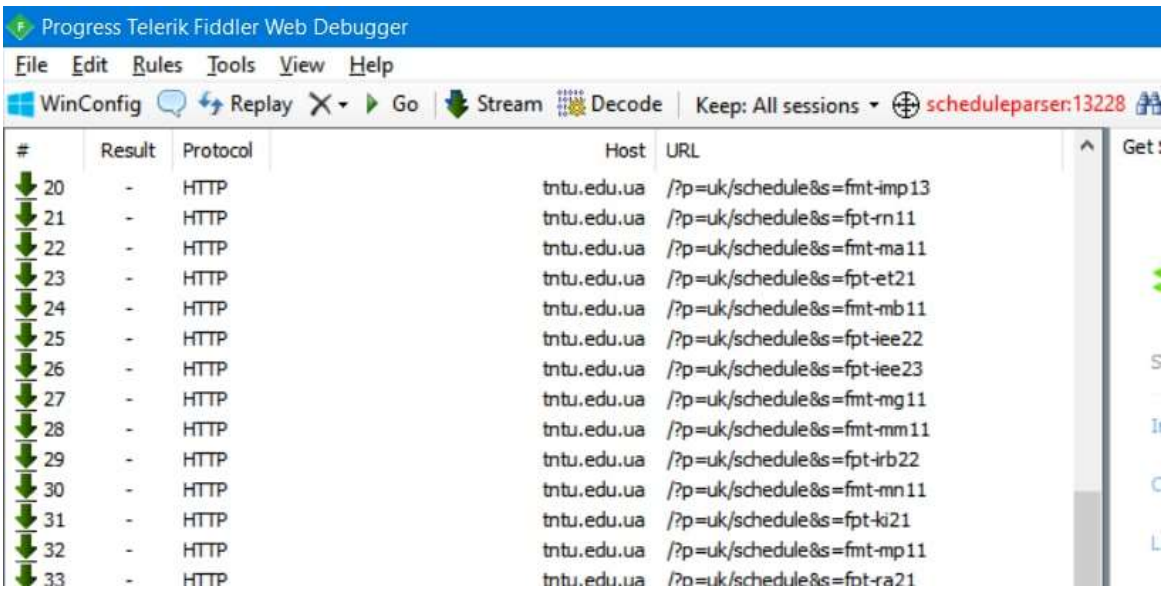

Рис. 2.9 – Вихідні запити під час парсингу.

Отримані статичні дані у форматі .json дозволять розробляти мобільний додаток, коли REST API зовсім або частково не готове. При наявності API всі дані легко інтегруються на реальні (отримані з веб-сервісу) і додаток працюватиме аналогічно тому, як працював на статичних даних.

3 РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ

3.1 Шаблон MVVM

Для розробки мобільного додатку використано технологію Xamarin.Forms, що базується на шаблоні програмування інтерфейсів MVVM (рис. 3.1) [11]. Даний шаблон передбачає розділення коду окремої сторінки на такі частини:

- View – представлення;
- Model – модель даних та доступ до них;
- ViewModel – своєрідний посередник, в якому сконцетрована вся логіка роботи з сторінкою.

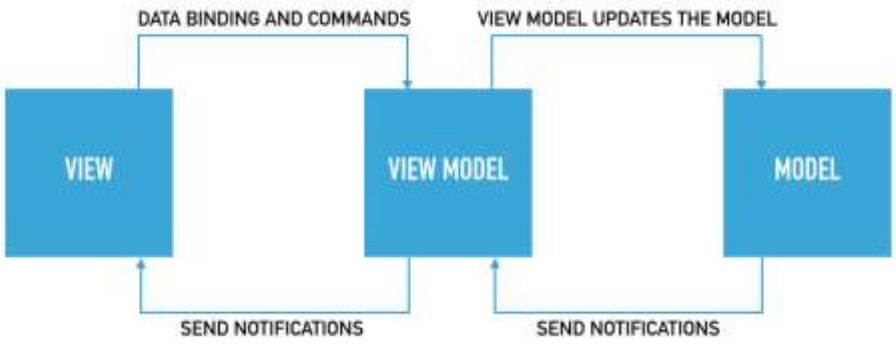

Рисунок 3.1 – Шаблон MVVM

При даному підході представлення повністю незалежне від логіки і основним завданням представлення є лише відображення даних (прив’язка) та запуск команд.

Прив’язка даних здійснюється за допомогою розширення XAML «Binding» - дає змогу однозначно вказати, які дані необхідні для конкретної властивості. При необхідності можна зазначити клас-конвертер для перетворення значення в необхідний тип чи формат (рис. 3.2).

```
<Picker Grid.Row="1"
        HorizontalOptions="Fill"
        Title="Факультет"
        ItemsSource="{Binding Faculties}"
        ItemDisplayBinding="{Binding ShortName}"
        SelectedItem="{Binding SelectedFaculty, Mode=TwoWay}"></Picker>
```

Рис. 3.2 – Прив'язка даних в MVVM

Для обробки події (таких як натискання кнопки чи вибір елементу зі списку) замість стандартних подій застосовуються команди – певна об’єктна інтерпретація подій, що дозволяють повністю розділити представлення і ViewModel та дають змогу повністю покрити ViewModel автоматичними модульними тестами. Команди можуть також мати параметри (наприклад, коли треба передати, на який конкретний елемент списку натиснули) (рис. 3.3).

```
<Grid.GestureRecognizers>
    <TapGestureRecognizer Command="{Binding NavigateToLessonCommand,
                                    Source={RelativeSource AncestorType={x:Type viewModels:HomeViewModel}}}"
                          CommandParameter="{Binding .}"></TapGestureRecognizer>
</Grid.GestureRecognizers>
```

Рис. 3.3 – Приклад прив’язки команди з параметром в MVVM.

Для спрощення роботи з MVVM та підтримку деяких розширених можливостей, таких як навігація через ViewModel та інверсія залежностей до проекту було також додано MvvmCross [19].

MvvmCross – додатковий інструмент, що забезпечує такі можливості при написанні додатків Xamarin.Forms:

– Dependency Injection, що застосовується для автоматичного проставлення всіх залежностей в ViewModel;

– Специфікація виду презентації сторінки (сторінка-контейнер для вкладок, сторінка-вкладка, звичайна сторінка, модальна сторінка);

– Автоматична навігація за допомогою ViewModel (для навігації не потрібно вказувати сторінку, на яку здійснюється перехід, достатньо лише вказати тип ViewModel, при цьому автоматично створиться і сторінка, і її модель представлення);

– Базові імплементації конвертерів;

– Імплементація нотифікацій ViewModel – представлення;

– Сервіс для пересилання сповіщень між різними сторінками.

Для розгортання MvvmCross було використано плагін MvxScaffolding (рис. 3.4).

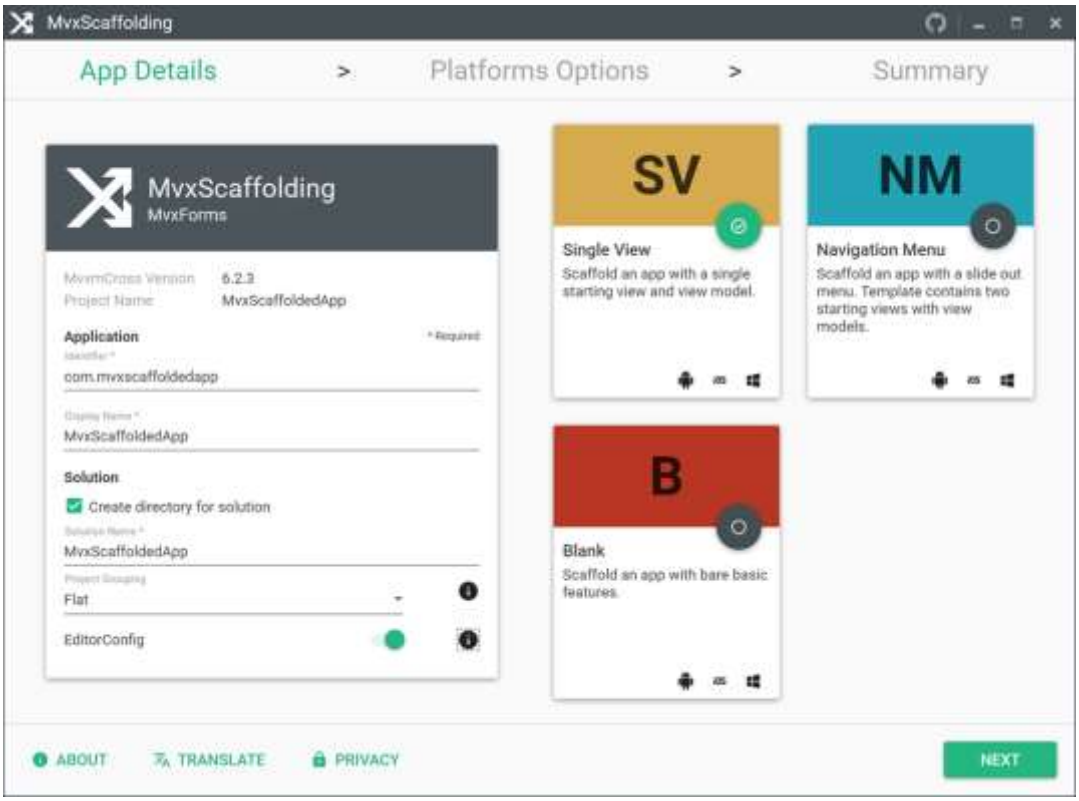

Рис. 3.4 – Створення проекту з використанням MvvmCross для Xamarin.Forms

Даний плагін дозволяє не лише створити порожній проект, а й одразу додати кілька сторінок, здійснити базові налаштування і деякі спеціальні налаштування для кожної платформи. Кінцевий результат – чотири проекти (рис. 3.5).

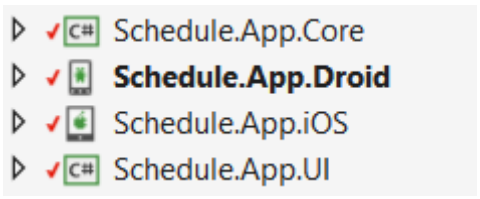

Рис. 3.5 – Автоматично створені проекти для MvvmCross.Forms

Перший проект – Schedule.App.Core містить ViewModel-і та сервіси, а також всю бізнес-логіку роботи програми. Останній проект – Schedule.App.UI – містить розмітку всіх сторінок, а також сюди варто додавати конвертери і модифіковані елементи керування, ресурси, блоки, що можна використати повторно.

Два інших проекти – Droid та iOS – створені для запуску кожної з платформ, а також тут варто розміщати код, специфічний для кожної з платформ.

3.2 Сервіс отримання розкладу

Оскільки, поточні дані наявні у форматі .json, то поточна імплементація передбачатиме лише вичитування з статичного файлу, проте для розширення в подальшому створимо окремий інтерфейс IScheduleService, для якого створимо файлову імплементацію:

```
public interface IScheduleService
{
    Task<List<Faculty>> GetFacultiesAsync();
    Task<List<Group>> GetGroupsByFacultyAsync(int facultyId);
    Task<List<Lesson>> GetLessonsByGroupIdAsync(int groupId);
    Task<Group> GetGroupByIdAsync(int groupId);
    Task<Faculty> GetFacultyByIdAsync(int facultyId);
}
```

Наведемо також частину коду, що використовується для вичитування статичного файлу з розкладом:

```
private void ReadSchedule()
{
    Assembly assembly = GetType().Assembly;
    using Stream stream =
assembly.GetManifestResourceStream($"{assembly.GetName().Name}.{Json
FileName}");
    if (stream == null) {
        return;
    }
    using var reader = new StreamReader(stream);
    var jsonString = reader.ReadToEnd();_schedule =
JsonConvert.DeserializeObject<Data.Models.Schedule>(jsonString);
    FillCollections(_schedule);
}
```

В подальшому, для заміни статичних даних на REST API запити потрібно буде просто замінити залежність, яка реєструється для IScheduleService іншою імплементацією, інших змін в додатку робити не доведеться.

3.3 Сторінка вибору групи

При першому запуску додатку потрібно буде вибрати факультет, курс та групу. Ці дані потрібно десь зберегти, оскільки незручно буде кожного разу при запуску обирати ті самі дані. У свою чергу, потрібно передбачити можливість зміни групи, якщо така вже була задана (до прикладу, розпочався новий навчальний рік). Для цього сторінка вибору групи повинна зберігати налаштовані дані. Для цього в Xamarin.Essentials є статичний клас Preferences, який за призначенням нагадує колекцію ключ-значення. Наведемо код, призначений для збереження та вичитування вибраних параметрів:

```
public sealed class SettingsService : ISettingsService
{
    public int SelectedFacultyId
    {
        get => Preferences.Get("faculty_id", 0);
        set => Preferences.Set("faculty_id", value);
    }
    public int SelectedCourse
    {
        get => Preferences.Get("course", 0);
        set => Preferences.Set("course", value);
    }
    public int SelectedGroupId
    {
        get => Preferences.Get("group_id", 0);
        set => Preferences.Set("group_id", value);
    }
}
```

Даний сервіс необхідно буде використати на сторінці вибору групи. Проаналізувавши отримані дані, було вирішено розбити вибір на три частини: спочатку факультет, потім курс, потім група. Групи відповідно будуть

фільтруватись за обраним факультетом та курсом, відображатись будуть лише ті, що відповідають попереднім критеріям.

Для вибору одного запису зі списку скористаємось елементом Picker:

```
<Picker Grid.Row="3"
        Title="Група"
        HorizontalOptions="Fill"
        ItemsSource="{Binding FilteredGroups}"
        ItemDisplayBinding="{Binding Name}"
        SelectedItem="{Binding SelectedGroup, Mode=TwoWay}">
</Picker>
```

В даному випадку бачимо, що SelectedItem має прив'язку даних в обидві сторони. Це означає, що якщо ми змінимо обраний елемент клацнувши на екран, він зміниться і в моделі представлення. Це дуже зручно для команд, наприклад визначивши, що команда «Зберегти» може бути виконана лише тоді, коли SelectedGroup має значення, таким чином кнопка автоматично активується, коли всі дані буде введено:

```
SaveCommand = new MvxAsyncCommand(OnSaveCommandAsync, CanSave);
///…///
private bool CanSave()
{
    return SelectedGroup != null;
}
private Task OnSaveCommandAsync()
{
    _settingsService.SelectedFacultyId = SelectedFaculty.Id;
    _settingsService.SelectedCourse =
SelectedCourse.CourseNumber;
    _settingsService.SelectedGroupId = SelectedGroup.Id;
    return _navigationService.Navigate<RootViewModel>();
}
```

Даний кодовий взірець також показує приклад навігації: для навігації використовується навігаційний сервіс та тип наступної моделі представлення, все інше MvvmCross робить під капотом за програміста. При збереженні всі параметри, такі як обраний факультет, курс та група зберігаються в сервісі налаштувань, який відновить ті ж значення після перезапуску додатку, що дасть можливість наступного разу одразу відкрити розкад, минувши сторінку вибору факультету, курсу та групи.

Для умовного старту різних сторінок залежно від налаштувань потрібно підставити різний шлях до першої сторінки:

```
    private void RegisterAppStart()
    {
        ISettingsService settingsService =
MvxIoCProvider.Instance.Resolve<ISettingsService>();
        if (settingsService != null &&
settingsService.SelectedGroupId != 0)
        {
            RegisterAppStart<RootViewModel>();
        }
        else
        {
            RegisterAppStart<HelloViewModel>();
        }
    }
```

Отримана сторінка виглядатиме наступним чином (рис. 3.6)

Рис. 3.6 – Сторінка налаштування

Після збереження налаштувань додаток повинен підвантажувати розклад для конкретної групи та переходити на сторінку з розкладом, де повинен відображатись актуальний розклад.

3.4 Сторінка відображення розкладу

Для відображення розкладу скористаємось CollectionView, а для перемикання днів – CarouselView. Остання дозволяє використовувати жести свайпу для горизонтальної зміни сторінок. Для відображення поточної сторінки додамо IndicatorView.

Рис. 3.7 – сторінка відображення розкладу

Для створення шаблону відображення елементу в CarouselView та CollectionView скористаємось DataTemplate, при цьому для CarouselView шаблоном буде CollectionView – для кожної сторінки список, а для CollectionView – шаблон одного елементу списку.

Щоб урізноманітнити інтерфейс і явно розділити між собою предмети, а також створити візуальну прив’язку було вирішено згенерувати кольори для

кожного заняття. Як видно з рисунку, таке представлення має не лише гарний вигляд, а й візуально поєднує однакові предмети й розділяє різні. Проте стандартна кольорова схема RGB не підходить для генерації випадкового кольору, оскільки при генерації випадкових чисел для R, G та B компонентів відповідно кожен колір матиме іншу начисеність, можна згенерувати одночасно яскраві і тьмяні, насичені та сірі кольори. Для правильної генерації було проаналізовано деякі існуючі кольорові схеми та обрано HSL [20].

HSL – кольорова схема, що кардинально відрізняється від RGB. RGB складається з трьох кольорових компонент – червоного, синього та зеленого. Змішуючи дані кольори, можна отримати повну палітру. HSL ж має лише один кольоровий компонент – відтінок (hue). Інші відповідають за насиченість та яскравість (saturation і luminosity) (рис. 3.8).

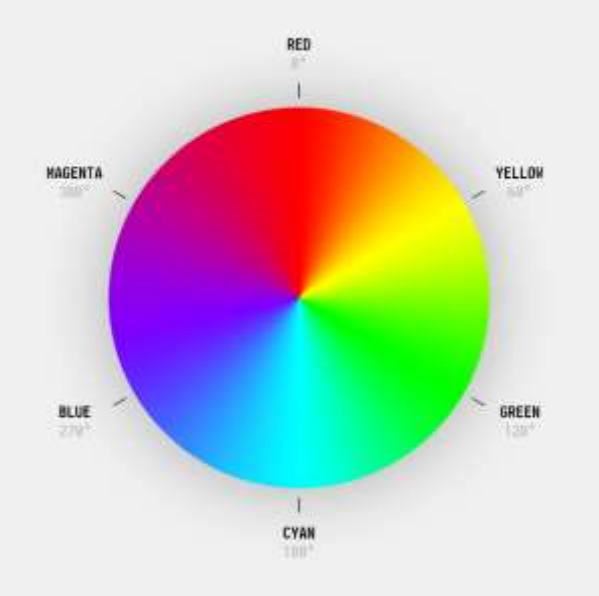

Рис. 3.8 – Компонент Hue схеми HSL

Як бачимо з рисунка, компонент Hue приймає значення в градусах 0-360 та повертає один, конкретний колір. Далі, за допомогою двох інших параметрів можна налаштувати яскравість та насиченість кольору. До прикладу, його можна зробити світлішим, темнішим, в залежності від фонового кольору додатку. Такий підхід дозволяє динамічно змінювати теми, при чому зберігаючи кольоровий контекст.

Для генерації кольору було прийнято рішення використовувати хеш-код назви предмета. Такий підхід забезпечує не лише унікальність кольору, а й те, що навіть після перезапуску додатку колір залишиться аналогічний (хеш-функція завжди повертає те ж саме значення для однакових даних). Для рандомізації кольору можна домножити на велике число та взяти Random з початковим значенням. Особливість класу Random також в тому, що для кожного початкового числа послідовність випадкових чисел завжди однакова, що дозволяє отримати випадкове число, яке щоразу буде те саме для аналогічних даних:

```
public Color Generate(double opacity)
{
    var colorHue = GenerateColorHue(_seed);
    return ApplyColorStyle(colorHue, opacity);
}
private Color ApplyColorStyle(int hue, double opacity)
{
    var h = hue / 360.0;
    return Color.FromHsla(h, 0.7, 0.7, opacity);
}
private int GenerateColorHue(int seed)
{
    seed *= 23522316;
    var rnd = new Random(seed);
    return rnd.Next(0, 20) * 18;
}
```

Xamarin.Forms підтримує HSLA одразу, тому не потрібно писати перерахунок кольору в палітру RGBA чи встановлювати додаткові залежності. Тут A – додатковий компонент, що відповідає за прозорість кольору (так званий альфа-канал). На відміну від Hue, Saturation, Luminosity та Alpha задаються в проміжку [0, 1]. В даному випадку генеруються два кольори, один повністю непрозорий для лівого краю елементу, другий – частково прозорий, для іконок перед групою, типом та розташуванням. Для відображення дня тижня поточною мовою потрібно конвертувати його для поточного регіону:

```
    protected override string Convert(DayOfWeek value, Type
targetType, object parameter, CultureInfo culture)
    {
        return
culture.DateTimeFormat.DayNames[(int)value].FirstCharToUpper();
    }
```

Даний код розміщується в конвертері, останній задається при прив’язці даних:

```
    <StackLayout HorizontalOptions="Center"
                 Margin="10,0,20,0"
                 Spacing="2">
        <Label Text="{Binding Day, Converter={StaticResource
DayOfWeekConverter}}"
               FontSize="16"
               HorizontalOptions="Center"></Label>
        <Label Text="{Binding WeekMode, Converter={StaticResource
WeekModeConverter}}"
               FontSize="12"
               TextColor="DarkGray"></Label>
    </StackLayout>
```

При використанні конвертерів для прив’язки кожного разу після оновлення поля моделі представлення значення буде автоматично проходити через конвертер і результат використовуватиметься для представлення. Також можливо задати метод ConvertBack, який буде перетворювати значення в оберненому напрямку, до прикладу, якщо текст був введений за допомогою екранної клавіатури, він пройде через метод ConvertBack конвертера, перш ніж потрапити до моделі представлення (ViewModel). При цьому важливо стежити за тим, щоб конвертер працював аналогічно-обернено у двох методах, в іншому випадку можна отримати зациклення і додаток перестане відповідати.

Для відображення графіки в додатку використані шрифти з вбудованою графікою FontAwesome. Такий підхід є оптимальний з декількох причин:

– Такі шрифти завжди містять найчастіше використовувані зображення, тому простіше їх знайти в одному шрифті;

– Шрифт в векторному форматі, це означає що незалежно від розміру тексту він не буде перетворюватись в пікселі та масштабування можна робити до будь-якого розміру;

– Для шрифта можна використовувати колір тексту, тому одне і те ж саме зображення різними кольорами залишається одним шрифтом, в випадку ж графічних зображень кожне треба малювати і додавати окремо, що збільшує розмір додатку.

Графіку в зображеннях варто використати, якщо немає схожої графіки в шрифті або графіка містить більш ніж один колір. Xamarin.Forms добре оптимізований для графічних шрифтів і має підтримку FontImageSource – даний об’єкт дозволяє сконвертувати графічний текст в ImageSource для таких об’єктів, як вкладка чи кнопка.

3.5 Сторінка деталей заняття

При натисканні на конкретне заняття додаток перенаправляє на нову сторінку. На ній текстом відображаються деталі, такі як початок, кінець заняття, підгрупа, тип, аудиторія. Також тут наявні: посилання на ATutor (якщо є відповідний курс) і список завдань (рис. 3.9).

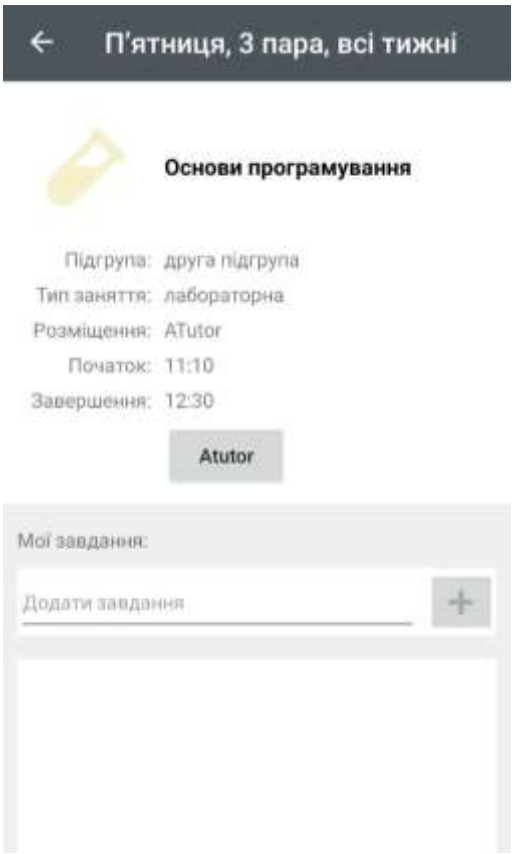

Рис. 3.9 – Сторінка деталей заняття

Завдання – спосіб для студента зберегти завдання, отримане на лекції, практичній чи лабораторній (до прикладу, «здати лабораторну №5», «підготуватись до екзамену», і т.д.). При наступному відкритті додатку невиконані та виконані завдання будуть відображатись у правому нижньому кутку заняття, що нагадає про невиконане завдання.

Для збереження завдань використаємо SQLite. Ця реалізація локальної бази даних для Xamarin є дуже зручною, оскільки підтримує базовий ORM.

ORM – проектування залежностей об'єктів, технологія, що дозволяє працювати з базою, використовуючи лише колекції та об'єкти, без використання запитів та команд.

Для роботи з локальною базою даних треба задати файл локальної бази та створити репозиторій:

```
private void RegisterDatabaseAccess()
{
    var basePath =
Environment.GetFolderPath(Environment.SpecialFolder.LocalApplication
Data);
    var dbPath = Path.Combine(basePath, DatabaseFilename);


MvxIoCProvider.Instance.RegisterSingleton<IGenericRepository<LessonT
ask>>(new GenericRepository<LessonTask>(dbPath));
}
```

GenericRepository<T> – клас, що містить реалізацію необхідних команд для роботи з базою (вставка, видалення, читання, модифікація), проте не прив'язаний до конкретної моделі даних, а лише обмежений до базової, що має ідентифікатор як єдине поле. Такий підхід дозволяє згодом додавати нові класи-моделі та перевикористовувати поточну реалізацію репозиторія без необхідності створення нового.

Наведемо для прикладу кілька методів класу GenericRepository:

```
public int DeleteItem(int id)
{
    return _sqliteDatabase.Delete<TEntity>(id);
}
public TEntity GetItem(int id)
{
    return _sqliteDatabase.Get<TEntity>(id);
}
```

```
    public List<TEntity> GetItems()
    {
        return _sqliteDatabase.Table<TEntity>().ToList();
    }
    public List<TEntity> GetItems(Expression<Func<TEntity, bool>>
condition)
    {
        return
_sqliteDatabase.Table<TEntity>().Where(condition).ToList();
    }

    public int SaveItem(TEntity item)
    {
        if (item.Id != 0)
        {
            _sqliteDatabase.Update(item);
            return item.Id;
        }
        return _sqliteDatabase.Insert(item);
    }
```

Як можна бачити з коду, ORM-підхід дозволяє уникнути написання запитів до бази даних мовою SQL і повністю використовувати об’єктну модель. Це дозволяє підставити будь який тип для TEntity, в цьому випадку автоматично створиться нова табличка для нової сутності і репозиторій працюватиме аналогічним чином. Також дана бібліотека підтримує перетворення предикатів в SQL-запити, як можна побачити з методу GetItems(). Параметр, який потрібно при цьому передати – Expression<Func<TEntity, bool>>, що являє собою динамічно генерований вираз, який при виконанні парситься і перетворюється в звичайний SQL запит, що дозволяє витягнути частину даних з таблички за певною умовою, використовуючи при цьому лише мову C#.

3.6 Сторінка налаштувань

Для можливості зміни обраної групи та для додавання деякий інших параметрів у майбутньому додамо окрему сторінку для відображення та зміни налаштувань. Для групування відображень було обрано TabbedView (рис. 3.10),

оскільки такий підхід дозволяє дати доступ до кількох сторінок одночасно, що робить навігацію простішою. TabbedView передбачає дві частини:

- Основна частина, або ж контент,
- Навігаційна панель, що дозволяє перемикати контент.

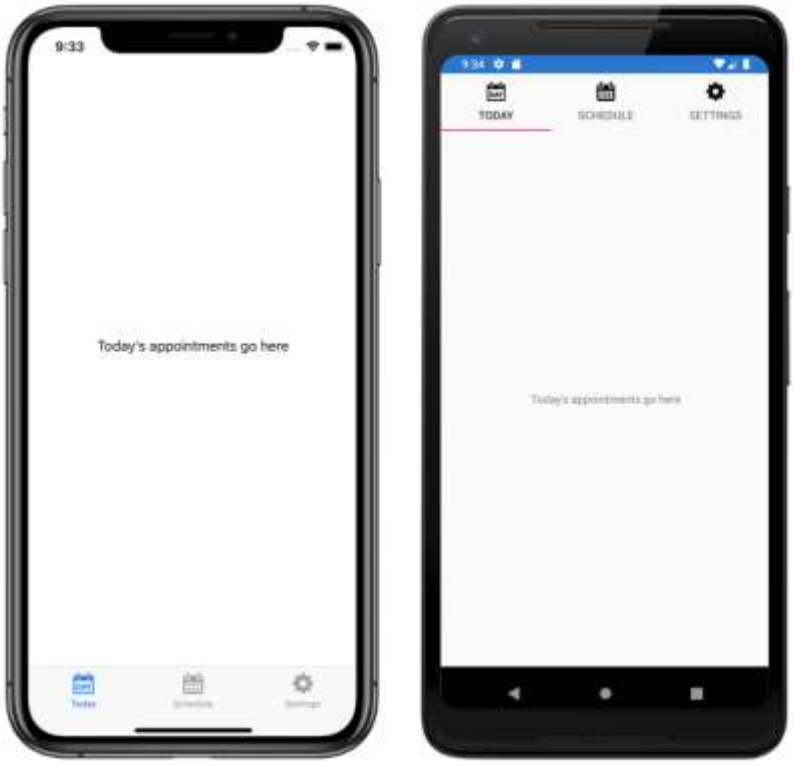

Рис. 3.10 – Відображення TabbedView для різних платформ

Наведемо реалізацію TabbedPage з використанням MvvmCross:

```
     <views:MvxTabbedPage x:TypeArguments="viewModels:RootViewModel"
     xmlns="http://xamarin.com/schemas/2014/forms"
     xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
     xmlns:views="clr-
namespace:MvvmCross.Forms.Views;assembly=MvvmCross.Forms"
     xmlns:pages="clr-
namespace:Schedule.App.UI.Pages;assembly=Schedule.App.UI"
     xmlns:fontAwesome="clr-
namespace:Schedule.App.Core.Tools.FontAwesome;assembly=Schedule.App.
Core"
     xmlns:viewModels="clr-
namespace:Schedule.App.Core.ViewModels;assembly=Schedule.App.Core"
     xmlns:android="clr-
namespace:Xamarin.Forms.PlatformConfiguration.AndroidSpecific;assemb
ly=Xamarin.Forms.Core"
     android:TabbedPage.ToolbarPlacement="Bottom"
```

```
    Title="{Binding Source={RelativeSource Self},
Path=SelectedItem.Title}"
    android:TabbedPage.IsSwipePagingEnabled="False"
    x:Class="Schedule.App.UI.Pages.MainPage">
    </views:MvxTabbedPage>
```

Розмітка такої сторінки полягає лише в створенні об’єкту MvxTabbedPage, основна ж частина, пов’язана з дочірніми вкладками та навігацією – реалізована в моделі представлення. Наведемо приклад:

```
    public sealed class RootViewModel : BaseViewModel
    {
        private readonly IMvxNavigationService _navigationService;
        private bool _firstTime = false;

        public                    RootViewModel(IMvxNavigationService
navigationService)
        {
            _navigationService = navigationService;
        }

        public override void ViewAppearing()
        {
            if (!_firstTime)
            {
                _firstTime = true;
                _navigationService.Navigate<HomeViewModel>();
                _navigationService.Navigate<SettingsViewModel>();
            }
        }

        public override Task Initialize()
        {
            return base.Initialize();
        }
    }
```

Для сторінки налаштувань було також прийнято рішення показувати попередньо вибрані значення. Для цього скористаємось вже відомим сервісом налаштувань для отримання заданих параметрів. Необхідні дані – ідентифікатор групи, ідентифікатор факультету та курс. Також скористаємось сервісом розкладу для отримання текстової інформації:

```
    public override async Task Initialize()
    {
        Faculty currentFaculty = await
_scheduleService.GetFacultyByIdAsync(_settingsService.SelectedFacult
yId);
```

```
        Group currentGroup = await
_scheduleService.GetGroupByIdAsync(_settingsService.SelectedGroupId)
;
        var currentCourse = _settingsService.SelectedCourse;

        Faculty = currentFaculty.Name;
        Group = currentGroup.Name;
        Course = $"{currentCourse} курс";
    }
```

Представлення в даному випадку повністю відображатиме поточні налаштування. Для зміни налаштувань скористаємось вже готовою сторінкою вибору початкових параметрів:

```
    private async Task OnChangeGroupAsync()
    {
        await
_navigationService.Navigate<GroupSelectionViewModel>();
        await Initialize();
    }
```

Готова сторінка налаштувань зображена на рис. 3.11.

Рис. 3.11 – Сторінка налаштувань

Після вибору нових параметрів додаток автоматично завантажить розклад та відобразить його на головній сторінці.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Охорона праці

Під час роботи над розробкою мобільного додатку для розкладу ТНТУ, основна робота виконувалась за допомогою персонального комп’ютера, тому було важливо дотримуватись правил охорони праці, техніки безпеки та пожежної безпеки, які регламентують роботу з ПК.

Для забезпечення безпечної роботи з ПК та скорочення негативного впливу на здоров’я користувача, потрібно дотримуватись норм передбачених у наступних документах:

- Закон України «Про охорону праці»;
- НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями», затверджені наказом Державного комітету України наказ Міністерства соціальної політики України від 14.02.2018 № 207;
- ДСанПІН 3.3.2.007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин »;
- НПАОП 40.1-1.21-98. «Правила безпечної експлуатації електроустановок споживачів»;
- НАПБ А.01.001-2004 «Правила пожежної безпеки в Україні».

Перед початком розробки мобільного додатку було проведено інструктажі з охорони праці, техніки безпеки та протипожежної безпеки.

При роботі над завданням з використанням ПК враховано, що площа на одне робоче місце повинна становити не менше ніж 6,0 м2, а об'єм не менше ніж 20,0 м3 [21]; дотримано вимог стосовно освітлення, оптимальних умов мікроклімату, ергономічних характеристик основних елементів робочого місця, рівнів шуму, вібрації, електромагнітного, ультрафіолетового та інфрачервоного випромінювання та електростатичного поля викладених у ДСанПіН 3.3.2-007-98 [21].

Оскільки, основне навантаження під час роботи з ПК припадало на зорову систему, тому штучне освітлення в приміщеннях з робочими місцями, обладнаними ЕОМ та ПЕОМ, здійснювалось системою загального рівномірного освітлення, а значення освітленості на поверхні робочого столу в зоні розміщення документів становила 300-500 лк. За умов коли ці значення освітленості неможливо було забезпечити системою загального освітлення, використовувалось місцеве освітлення. Також, практикувались короткі перерви кожні 20 хв., що дозволило зменшити напруженість зорового нерву і відповідно знизити імовірність його травмування. Згідно з рекомендаціями, час від часу виконувався спеціальний комплекс вправ для зниження втомлюваності очей та зменшення нервового напруження.

Щоб унеможливити ураження електричним струмом під час роботи було виконано наступні положення:

- лінія електромережі для живлення ЕОМ виконана як окрема групова трипровідна мережа шляхом прокладання фазового, нульового робочого та нульового захисного провідників. Нульовий захисний провідник використовується для заземлення (занулення) електроприймачів;
- ЕОМ підключаються до електромережі тільки за допомогою справних штепсельних з'єднань і електророзеток заводського виготовлення;
- заземлюючі конструкції, які знаходяться в робочому приміщенні було покрито ізоляцією, щоб унеможливити потрапляння працівника під напругу;
- у штепсельних з'єднаннях та електророзетках, крім контактів фазового та нульового робочого провідників, є спеціальні контакти для підключення нульового захисного провідника. Їхня конструкція така, що приєднання нульового захисного провідника відбувається раніше, ніж приєднання фазового та нульового робочого провідників[22].

Так, як проектування та розробка програмного забезпечення потребує високої концентрації уваги важливим аспектом є забезпечення шумоізоляції, для цього обладнання, яке становило джерело шуму було розміщено в окремих приміщеннях.

Згідно з правилами протипожежної безпеки приміщення в яких відбувалося дослідження було обладнано системою автоматичної протипожежної сигналізації та необхідною кількістю вогнегасників.

Для захисту працівника ЕОМ та дотримання регламентованих норм світлового випромінення використовувалися екранні світлофільтри та локальні світлофільтри.

Все обладнання, яке використовувалося під час розробки програмного забезпечення було сертифіковане для використання на території України.

Окрім усіх вище перерахованих вимог було виконано решту інструкцій описаних в законах та нормативно-правових актах, що стосуються охорони праці, техніки безпеки, протипожежної безпеки та електробезпеки при роботі з ЕОМ та ВДТ.

Враховуючи все вище описане, можна сказати що розробка мобільного додатку для розкладу ТНТУ була виконана згідно з вимогами охорони праці та техніки безпеки.

4.2 Безпека в надзвичайних ситуаціях.

Індивідуальне завдання для розділу «Безпека в надзвичайних ситуаціях»:

- Оцінка стійкості систем управління і постачання суб’єктів господарювання, підготовка до відновлення порушеного виробництва
- Організація протипожежного захисту та проведення протипожежної профілактики на промисловому підприємстві

4.2.1 Оцінка стійкості систем управління і постачання суб’єктів господарювання, підготовка до відновлення порушеного виробництва

Під стійкістю роботи ОГД розуміють його спроможність в умовах надзвичайної ситуації випускати продукцію в запланованому обсязі та номенклатурі, а при отриманні середніх руйнувань або порушенні зв'язків з кооперації та поставок відновлювати виробництво у мінімальні терміни.

Під стійкістю роботи об'єктів, які безпосередньо не виробляють матеріальні цінності розуміють їх спроможність виконувати свої функції в умовах НС.

На стійкість роботи ОГД в умовах НС впливають наступні фактори:

- надійність захисту робітників та службовців;
- спроможність інженерно-технічного комплексу об'єкта протистояти у визначеному ступеню уражаючих факторів стихійного лиха, аварій, катастроф та сучасних видів зброї;
- захищеність об'єкта від вторинних уражаючих факторів (пожеж, вибухів, зараження ОР та СДОР);
- надійність системи забезпечення об'єкта всім необхідним для виробництва (сировиною, паливом, комплектуючими вузлами і деталями, електроенергією, водою, газом та іншим);
- стійкість та безперервність управління виробництвом та ЦЗ;
- підготовленість об'єкта до ведення РІНР та робіт щодо порушеного виробництва.

Дослідження стійкості роботи ОГД — це всебічне вивчення обстановки, яка може скластися під час надзвичайної ситуації та визначення її впливу на виробничу діяльність підприємства. Мета дослідження полягає в тому, щоб виявити слабкі місця в роботі об'єкта та виробити найбільш ефективні пропозиції, спрямовані на підвищення його стійкості.

Дослідження стійкості роботи ОГД проводиться силами інженерно-технічного персоналу із залученням спеціалістів науково-дослідних та проектних організацій. Організатором та керівником досліджень є керівник підприємства.

Увесь процес планування і проведення досліджень поділяється на три етапи: підготовчий; оцінка стійкості роботи ОГД; розроблення заходів, які підвищують стійкість роботи ОГД в умовах НС.

На першому етапі розробляються керівні документи, які визначають склад учасників досліджень та організовується їх підготовка.

Термін дослідження встановлюється залежно від обсягу роботи та підготовки учасників і може тривати 2—3 місяці. Залежно від складу основних виробничо-технічних служб на об'єкті створюються дослідницькі групи. їх кількість і чисельність залежить від обсягу вирішуваних завдань, специфіки виробництва.

На підготовчому етапі з керівниками груп проводиться спеціальне заняття, на якому керівник підприємства доводить до виконавців план роботи, завдання кожної групи та визначає терміни дослідження.

На другому етапі проводиться безпосереднє дослідження стійкості роботи об'єкта. В ході дослідження визначаються умови захисту робітників та службовців від уражаючих факторів, проводиться оцінка уразливості виробничого комплексу від різних уражаючих факторів, оцінюється характер можливих пошкоджень від вторинних уражаючих факторів, вивчається стійкість роботи системи забезпечення та кооперативних зв'язків з іншими об'єктами, з'ясовуються вразливі місця в системі управління, виробництвом.

Кожна група оцінює стійкість відповідних елементів виробничого комплексу та робить необхідні розрахунки. До складу групи комплексних досліджень (керівник — головний інженер) входять керівники всіх груп.

Група досліджень стійкості будівель та споруд (керівник — начальник відділу " капітального будівництва) на основі аналізу характеристик і стану виробничих будинків та споруд: визначає ступінь їх стійкості до дії уражаючих факторів ЛВ; оцінює розміри можливої шкоди від дії вторинних уражаючих факторів; проводить розрахунки сил і засобів, необхідних для встановлення

виробничих споруд при різних ступенях руйнування. Крім того, група досліджує та оцінює захисні властивості захисних споруд, визначає необхідну кількість ЗС на території об'єкта та в заміській зоні.

Група досліджень стійкості виробничого обладнання (керівник — головний механік) оцінює стійкість технологічних ліній, верстатів та механізмів і визначає: можливі втрати станків, приладів і систем автоматичного управління при різних ступенях пошкодження; способи збереження і захисту особливо цінного обладнання; потребу в силах і засобах, терміни та обсяги відновлюваних робіт; можливості створення резерву обладнання та порядок його використання

Група досліджень стійкості систем енергозабезпечення (керівник — головний енергетик) оцінює: стійкість системи електро-, газо- та водопостачання, каналізації та переведення котелень на інші види палива; нормативно-необхідні та мінімальні потреби з кожного виду енергії; основні та додаткові джерела енергопостачання, можливості внутрішніх та зовнішніх джерел; стійкість заводських комунікацій; наявність та можливості автономних джерел енергопостачання.

Штаб Цивільного захисту ОГД в цей період оцінює загальний стан ІДО і визначає заходи для забезпечення захисту робітників і службовців. Для цієї роботи притягається ряд начальників служб, які виконують відповідні функції.

Служба оповіщення і зв'язку вивчає і оцінює стійкість зв'язку з органами ЦЗ, виробничими підрозділами і формуваннями ЦЗ. Оцінює надійність системи зв'язку і оповіщення, повноту обладнання пунктів управління.

Служба сховищ і укрить оцінює інженерний захист робітників і службовців, правильність експлуатації сховищ і укрить, готовність їх до використання за прямим призначенням. Розраховує час на оповіщення робітників і службовців, збір і укриття їх в захисних спорудах.

Служба радіаційного та хімічного захисту оцінює можливості роботи об'єкта в умовах радіації і дає пропозиції щодо захисту робітників і службовців від радіоактивного зараження, визначає типові режими радіаційного захисту людей, розробляє графік робочих змін для проведення РіНР. Аналізує забезпеченість

робітників і службовців засобами індивідуального захисту, умови зберігання і порядок їх видачі. Готує пропозиції щодо організації і ведення радіаційної і хімічної розвідки, організації санітарної обробки людей, знезараження одягу, транспорту, техніки і споруд.

Медична служба розробляє заходи щодо організації медичного обслуговування робітників та службовців на об'єкті і в заміській зоні, а також при проведенні РіНР. Визначає сили і засоби для надання першої медичної допомоги потерпілим. Виробляє рекомендації з організації дозиметричного контролю при перебуванні людей у зоні радіоактивного зараження і рекомендації з захисту продуктів харчування і джерел водопостачання

На третьому етапі підводяться підсумки проведених досліджень. Групи спеціалістів за підсумками досліджень готують підсумки і пропозиції з захисту робітників та службовців і підвищенню стійкості елементів виробництва, які досліджуються.

Група комплексних досліджень на основі доповідей інших груп складає загальний план, в якому визначаються: можливості щодо захисту робітників і службовців в ЗС; загальна оцінка стійкості об'єкта, найбільш слабкі (вразливі) ділянки виробництва; практичні заходи, терміни та обсяги робіт, які виконуються при повсякденній діяльності та при загрозі НС; порядок та приблизні строки відновлюваних робіт при різних ступенях руйнування.

За результатами досліджень розробляються плани, в яких визначаються відповідні заходи, необхідні кошти на їх проведення, терміни і відповідальні особи за їх виконання.

В зв'язку з тим, що заходи щодо підвищення стійкості роботи виконуються завчасно (в мирний час), з оголошенням загрози виникнення НС (нападу ворога) та в умовах НС, відповідні плануючі документи, для зручності користування ними, складаються на кожну можливу ситуацію.

4.2.2 Організація протипожежного захисту та проведення протипожежної профілактики на промисловому підприємстві

Під пожежною безпекою об'єкта розуміють такий його стан, за якого з регламентованою імовірністю виключається можливість виникнення і розвитку пожежі та впливу на людей небезпечних чинників пожежі, а також забезпечується захист матеріальних цінностей.

Забезпечення пожежної безпеки об'єкта досить складне і багатоаспектне завдання, тому до його вирішення необхідно підходити комплексно. Комплекс заходів та засобів щодо забезпечення пожежної безпеки об'єкта складається із відповідних систем (рис. 4.1), кожна з яких поділяється на підсистеми, а ті, в свою чергу, на підсистеми нижчого рівня, які на рисунку не показані.

Основними системами комплексу заходів та засобів щодо забезпечення пожежної безпеки об'єкта є: система запобігання пожежі, система протипожежного захисту та система організаційно-технічних заходів.

Система запобігання пожежі - це комплекс організаційних заходів і технічних засобів, спрямованих на унеможливлення умов, необхідних для виникнення пожежі.

Система запобігання пожежі включає такі два основні напрями:

- запобігання утворенню горючого середовища;

- запобігання виникненню в горючому середовищі (чи внесенню в нього) джерела запалювання.

Рис. 4.1. Загальна схема комплексу заходів та засобів щодо забезпечення пожежної безпеки об'єкта

Горюче середовище - це середовище, здатне самостійно горіти після видалення джерела запалювання. Таке середовище утворює горюча речовина разом з окисником. Якщо в технологічних процесах застосовують горючі речовини та існує можливість їх контакту з повітрям, то небезпека пожежі та вибуху може виникнути як усередині апаратів, так і поза ними, в приміщенні та на відкритих площадках.

Однак для виникнення пожежі чи вибуху необхідно мати ще й джерело запалювання - засіб енергетичного впливу, що ініціює виникнення пожежі. Джерелом запалювання є: відкрите полум'я, розжарені предмети, електричні розряди, іскри від ударів і тертя, теплові прояви хімічних реакцій і механічних дій, сонячна радіація, електромагнітні та інші випромінювання.

Запобігання утворенню горючого середовища досягається: застосуванням герметичного виробничого устаткування; максимально можливою заміною в технологічних процесах горючих речовин та матеріалів негорючими; обмеженням кількості пожежо- та вибухонебезпечних речовин при використанні та зберіганні, а також правильним їх розміщенням; ізоляцією горючого та вибухонебезпечного

середовища; організацією контролю за складом повітря в приміщенні та контролю за станом середовища в апаратах; застосуванням робочої та аварійної вентиляції; відведенням горючого середовища в спеціальні пристрої та безпечні місця; застосуванням в установках з горючими речовинами пристроїв захисту від пошкоджень та аварій; використанням інгібувальних (хімічно активні компоненти, що сприяють припиненню пожежі) та флегматизаційних (інертні компоненти, що роблять середовище негорючим) добавок та ін.

Запобігання виникненню в горючому середовищі джерела запалювання досягається: використанням устаткування та пристроїв, при роботі яких не виникає джерел запалювання; використанням електроустаткування, що відповідає за виконанням класу пожежо- та вибухонебезпеки приміщень та зон, груп і категорії вибухонебезпечної суміші; виконанням вимог щодо сумісного зберігання речовин та матеріалів; використанням устаткування, що задовольняє вимоги електростатичної іскробезпеки; улаштуванням блискавкозахисту; організацією автоматичного контролю параметрів, що визначають джерела запалювання; використанням швидкодіючих засобів захисного вимкнення; заземленням устаткування, видовжених металоконструкцій; використанням під час роботи з ЛЗР інструментів, що не допускають іскроутворення; ліквідацією умов для самоспалахування речовин і матеріалів; усуненням контакту з повітрям пірофорних речовин; підтриманням температури нагрівання поверхні устаткування пристроїв, речовин та матеріалів, які можуть контактувати з горючим середовищем нижче гранично допустимої (80 % температури самозаймання).

ВИСНОВКИ

Під час виконання роботи було здійснено глибокий аналіз ринку мобільних пристроїв та мобільних операційних систем на основі відкритих статистичних даних. На основі аналізу було виявлено деякі застарілі платформи, під які розробка не має сенсу і є збитковою, а також було виявлено платформи, які не володіють значною аудиторією. Однак під час дослідження також було виділено дві мобільні операційні системи – Android та iOS, які є актуальними на сьогоднішній день і практично повністю займають ринок смартфонів.

Було проаналізовано засоби швидкої розробки уніфікованого коду, який матиме змогу запускатись на обох платформах. Виявлено такі технології як Apache Cordova, React Native, Xamarin.Native та Xamarin.Forms. Вибір було зроблено на користь Xamarin.Forms, зважаючи на простоту розробки, розширюваність, спільноту розробників, набір інструментів та сторонніх бібліотек, що спрощують роботу з даною технологією.

Було здійснено короткий огляд підходу REST, а також різних форматів серіалізації даних. Проаналізовано розмір файлу для різних форматів при однаковому навантажені даних. Обрано JSON формат завдяки зрозумілій структурі та меншій кількості надлишкових даних.

Для отримання даних для відображення було описано підхід парсингу веб-сторінок, здійснено детальний аналіз процесу. Створено програмний продукт, що призначений для отримання повного розкладу занять всіх груп та підгруп ТНТУ на основі веб-сторінки з розкладом. Проаналізовано навантаження на сервер під час виконання процесу розбору розкладу, оптимізовано процес на основі таких параметрів, як стійкість сервера, здатність його відповідати на запити за однаковий час, та швидкість отримання даних. Створено модель даних на основі наявних даних, додано можливість майбутньої інтеграції з сервісом RESTful API.

Зроблено короткий огляд технології MVVM, а також бібліотеки MvvmCross. Створено мобільний додаток з підходом Xamarin.Forms. Описано основні використані підходи шаблону MVVM, а також наведено деталі реалізації певних аспектів.

Досліджено наявні кольорові схеми, такі як RGB, CMYK, HSL, HSV. На основі наявних даних було створено простий та водночас ефективний алгоритм генерації кольорів, що базується на кольоровій схемі HSL.

Досліджено можливість збереження локальних даних в межах мобільної аплікації, описано ORM-підхід для SQLite баз даних.

Як результат виконання роботи було створено програмний продукт, що є стабільним, протестованим та може використовуватись студентами для роботи з розкладом та для внесення робочих записок чи завдань. Забезпечено можливості для розширення функціоналу додатку.

СПИСОК ПОСИЛАНЬ

1. J. Luty. Forecast of smartphone user numbers in Ukraine from 2015 to 2025 [Електронний ресурс] / J. Luty – Режим доступу до ресурсу: https://www.statista.com/statistics/1134645/predicted-number-of-smartphone-users-in-ukraine/.

2. O'Dea S. Mobile operating systems' market share worldwide from January 2012 to October 2020 [Електронний ресурс] / S. O'Dea. – 2020. – Режим доступу до ресурсу: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/.

3. What Everybody Ought to Know About Android : Introduction, Features & Applications [Електронний ресурс] – Режим доступу до ресурсу: https://www.elprocus.com/what-is-android-introduction-features-applications/.

4. Neuburg M. iOS 10 Programming Fundamentals with Swift: Swift, Xcode, and Cocoa Basics 1st Edition / Matt Neuburg., 2016. – 620 с.

5. Patterson D. A., Ditzel D. R. The case for the reduced instruction set computer [Електронний ресурс] / D. A. Patterson, D. R. Ditzel – Режим доступу до ресурсу: https://dl.acm.org/doi/10.1145/641914.641917.

6. Triggs R. ARM VS X86: INSTRUCTION SETS, ARCHITECTURE, AND ALL KEY DIFFERENCES EXPLAINED [Електронний ресурс] / Robert Triggs. – 2020. – Режим доступу до ресурсу: https://ipsnews.net/business/2020/06/29/arm-vs-x86-instruction-sets-architecture-and-all-key-differences-explained/.

7. Angulo E. A Case Study on Cross-Platform Development Frameworks for Mobile Applications and UX [Електронний ресурс] / E. Angulo, X. Ferre – Режим доступу до ресурсу: https://dl.acm.org/doi/abs/10.1145/2662253.2662280.

8. Camden R. Apache Cordova in Action 1st Edition / Raymond Camden., 2015. – 248 с.

9. React Native: What is it? and, Why is it used? [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://medium.com/@thinkwik/react-native-what-is-it-and-why-is-it-used-b132c3581df.

10. Yosifovich P. Windows Presentation Foundation 4.5 Cookbook / Pavel Yosifovich., 2012. – 464 с.

11. Likness J. Model-View-ViewModel (MVVM) Explained [Електронний ресурс] / Jeremy Likness – Режим доступу до ресурсу: https://www.wintellect.com/model-view-viewmodel-mvvm-explained/.

12. Versluis G. A Brief History of Xamarin / Gerald Versluis // Xamarin.Forms Essentials / Gerald Versluis. – с. 3–18.

13. Logiticks F. Xamarin Native vs. Xamarin.Forms: How to Choose [Електронний ресурс] / Finlead Logiticks. – 2020. – Режим доступу до ресурсу: https://dzone.com/articles/xamarin-native-vs-xamarinforms-how-to-choose.

14. Hartpence B. Packet Guide to Core Network Protocols / Bruce Hartpence., 2011. – 264 с.

15. Liew Z. Understanding And Using REST APIs [Електронний ресурс] / Zell Liew. – 2018. – Режим доступу до ресурсу: https://www.smashingmagazine.com/2018/01/understanding-using-rest-api/.

16. SAFRIS S. A Deep Look at JSON vs. XML, Part 1: The History of Each Standard [Електронний ресурс] / SEVA SAFRIS – Режим доступу до ресурсу: https://www.toptal.com/web/json-vs-xml-part-1.

17. Webscraping with C# [Електронний ресурс]. – 2016. – Режим доступу до ресурсу: https://www.codeproject.com/Articles/1041115/Webscraping-with-Csharp.

18. AngleSharp - Getting Started [Електронний ресурс] – Режим доступу до ресурсу: https://anglesharp.github.io/docs/Basics.html.

19. Getting Started with MvvmCross [Електронний ресурс] – Режим доступу до ресурсу: https://www.mvvmcross.com/documentation/getting-started/getting-started.

20. CMYK and RGB? HSV and HSL? Introducing the Chromatic Compendium! [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://dev.to/r4h33m/cmyk-and-rgb-hsv-and-hsl-introducing-the-chromatic-compendium-1d7.

21. ДСанПІН 3.3.2.007-98 Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин – 1998.

22. НПАОП 40.1-1.21-98 (ДНАОП 0.00-1.21-98) Правила безпечної експлуатації електроустановок споживачі. // Держнаглядохоронпраці (Державний комітет України з нагляду за охороною праці). – 1998.

23. МЕТОДИЧНІ ВКАЗІВКИ до виконання атестаційної роботи магістра за спеціальністю 121 – ІНЖЕНЕРІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ (Освітньо-професійна програма - «ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМ», Освітньо-наукова програма - «ІНЖЕНЕРІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ») для с / М. Р.Петрик, Д. М. Михалик, О. Ю. Петрик, Г. Б. Цуприк. – 2020. – С. 52.

ДОДАТКИ

ДОДАТОК А

ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ПУЛЮЯ
КАФЕДРА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

ТЕХНІЧНЕ ЗАВДАННЯ

на розробку кваліфікаційної роботи
«Система управління розкладом ТНТУ»

Розробники:
виконавець ст. гр. СПм-61
Борівець Богдан Ярославович

_____
(підпис)

керівник роботи
Дячук Степан Федорович

_____
(підпис)

Тернопіль 2020

ЗМІСТ

1 ПІДСТАВИ ДО РОЗРОБКИ

Розробка проводиться у відповідності до графіку навчального плану підготовки магістрів за спеціальністю 121 «Інженерія програмного забезпечення».

Тема проекту: «Система управління розкладом ТНТУ».

Термін виконання: до «___» _____ 2020р.

2 ПРИЗНАЧЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Інформаційна система призначена для перегляду розкладу занять ТНТУ в мобільному додатку.

Інформаційна система буде корисною для студентів ТНТУ та має освітнє призначення.

Інформаційна система дозволить здійснювати перегляд списку занять, а також надасть можливість робити нотатки та вписувати поточні завдання.

3 ВИМОГИ ДО ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1Функціональні вимоги

Система повинна передбачати одну роль:

− користувач

Для користувачів система надає доступ до повного переліку функцій:

− вибір факультету;

− вибір курсу;

− вибір групи;

− перегляд списку занять;

− зміна дня тижня;

− додавання нотаток/завдань;

− видалення нотаток/завдань.

− зміна групи

Набір даних функцій дозволяє користувачу завжди дізнаватись актуальний розклад та отримувати першим всі можливі зміни, а також не забувати та вчасно виконувати завдання, поставлені викладачем.

3.2 Технічні вимоги

Вимоги до клієнтської частини: ОС Android, не старіше 4.2 версії, інтуїтивно зрозумілий, не перевантажений інтерфейс: будь-яка бажана дія зі сторони користувача повинна досягатися в межах трьох дотиків до екрану.

3.3 Програмні вимоги

Використання СУБД: SQLite.

Розробка клієнтської частини: XAMARIN.

Додаткові вимоги: використання алгоритму генерації випадкових кольорів за допомогою палітри HSL для спрощення взаємодії з додатком.

4. ЕТАПИ РОЗРОБКИ

Розробка інформаційної системи проводиться в наступному порядку:

- аналіз мобільних платформ;
- вибір засобів розробки та архітектури системи;
- аналіз існуючого API;
- розробка програмного забезпечення системи;
- тестування інформаційної системи на реальних даних;
- оформлення супровідної документації;
- здача роботи.

Результати виконання кожного етапу проекту погоджуються з керівником роботи.

5. СУПРОВІДНА ДОКУМЕНТАЦІЯ

Для інформаційної системи повинні бути розроблені наступні документи:

- пояснювальна записка до роботи;
- презентація роботи;
- рецензія на роботу;
- диск з кодом інформаційної системи.

Пояснювальна записка до кваліфікаційної роботи оформляється згідно діючих вимог до нормоконтролю проектів.

6. ПОРЯДОК ЗДАЧІ РОБОТИ

Розроблена інформаційна системи повинна відповідати вимогами, що складаються з перерахованих у п.3.1 цього документу характеристик.

Для здачі проекту необхідно підготувати весь перелік документів зазначений у п.5 цього документу.

Приймання проекту проводиться спеціально створеною комісією в термін зазначені в п.1 цього документу.

7. ВІДМІТКИ ПРО ВИКОНАННЯ ЕТАПІВ ТА ЗМІНИ В ПРОЕКТІ

Назва етапу	Відмітка*
Аналіз мобільних платформ	
Архітектура системи	
Розробка системи	
Використання системи	
Супровідна документація	

* відмітки про виконання етапу ставляться керівником проекту

ДОДАТОК Б

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ПУЛЮЯ

М А Т Е Р І А Л И

VIII НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

9–10 грудня 2020 року

ТЕРНОПІЛЬ
2020

СЕКЦІЯ 4. **ПРОГРАМНА ІНЖЕНЕРІЯ ТА МОДЕЛЮВАННЯ СКЛАДНИХ РОЗПОДІЛЕНИХ СИСТЕМ**

ДОДАТОК В