Mappalachian: Indoor Mapping for Appalachian State University

by

Wilson Styres

Honors Thesis

Appalachian State University

Submitted to the Department of Computer Science

in partial fulfillment of the requirements for the degree of

Bachelor of Science

May 2021

APPROVED BY:

James B. Fenwick Jr., Ph.D., Thesis Project Director

Alice McRae, Ph.D., Second Reader

Raghuveer Mohan, Ph.D., Departmental Honors Director

Rahman Tashakkori, Ph.D., Chair, Computer Science

Copyright© Wilson Styres 2021
All Rights Reserved

ABSTRACT

Mappalachian: Indoor Mapping for Appalachian State University.

(May 2021)

Wilson Styres, Appalachian State University

Appalachian State University

Thesis Chairperson: James B. Fenwick Jr., Ph.D.

Online mapping applications such as Apple Maps and Google Maps might be able to help students and visitors to the Appalachian State campus find buildings, but cannot help them find specific rooms inside of buildings. Ask any freshman student how important that is as they frantically search Rankin Science or Anne Belk Hall for their first class of the semester. Mappalachian, the application that this thesis describes, is an iOS application that uses technology available in the iOS SDK to show both the exterior and the interior spaces of buildings on campus. The interface presents itself just like any other mapping application would but when zooming in on a specific building, the map changes to show the interior space on each floor level. All of this data has been gathered using real-world floor plans and coordinate data in a program called Java OpenStreetMap Editor so that they can be converted into a format readable by the application. Any user can search for a room in a building by its room number which then will focus and highlight the specified room on the map. For students, the app also can retrieve their current schedule, allowing the user to search for rooms based on their schedule rather than the room number.

Contents

List of Tables

List of Figures

List of Listings

Chapter 1

Introduction

Since 1899, Appalachian State University has accepted students who aspire to further their knowledge and obtain a higher education. Over the years the university has held many different names and many students have passed through its halls but the primary goal set by its founders remains the same: discovering educational opportunity to match the splendor of the mountains in which it lives.

As the university grew, so grew its facilities. As of 2021, Appalachian State University has over 1200 acres of land, 30 academic buildings, 20 residence halls, 3 dining facilities, and 11 recreational facilities [11]. While not as large as some other schools in the University of North Carolina system, Appalachian can still be overwhelming to new students beginning to find their way around campus.

1.1 The Problem

In this day and age it is taken for granted that one can travel to just about anywhere in the world in a matter of hours. Going back 50 years ago, this was unimaginable with the majority of the United States not even owning a car yet. Today, most households in the United States own a car and many households have multiple cars for members of their family. With the invention of cellphones and the GPS, traveling to places became easier than ever. Now, people can simply ask their phones verbally to "get me directions to Boone" and an optimal route will

be plotted that can be easily followed at every turn along the journey. However, there is one place that Google Maps can not take you: the inside of buildings.

Public, accessible indoor mapping is a relatively new concept that has grown into existence with the rise of smartphones and other handheld computers. Paper maps existed before cellphones and computers and were easily accessible for highways and towns but rarely interior buildings. A select few buildings, like malls, had accessible floor plans so that shoppers could locate stores within them. But, generally speaking, there was no central location to find interior maps of buildings in your city. Today, companies like Google and Apple have started to create indoor maps for their mapping applications, Google Maps and Apple Maps respectively. These applications are now mapping interiors of buildings like airport terminals, convention centers, libraries, and even sports stadiums. These are high traffic buildings that people may not visit often and have floor plan layouts that are relatively stable. Having indoor mapping for these buildings enables visitors to quickly find points of interest like bathrooms, food venues, or elevators.

Appalachian State University is no different from these other venues. While students and faculty may know their hallway or their floor, they likely are not familiar with the entire buildings that they reside in. This is even more common for students who might have a class in a building that may lie outside their major where they will learn the location of one particular classroom but are unaware of other features that may exist inside of the building. To add to this problem, a lot of the older buildings at Appalachian can feel like a maze with corridors to small offices in corners of the building that don't have classrooms near them so students aren't familiar with those areas. Also, App State has many visitors from prospective families to community members attending events who are unfamiliar with campus buildings.

1.2 The Solution

Apple and Google have solved this problem by creating their own indoor maps for popular venues and inserting them into their applications for use on the web or on a user's cellphone. Unfortunately, Appalachian State campus building do not yet meet their guidelines for a "popular venue." Google Maps does have an indoor map for the Belk Library and Information

Commons, but it only maps out two floors which have changed significantly since the original map was made. Google's library map also lacks important features like room numbers that could help users navigate and find classrooms or study rooms. Since indoor mapping is still such a new feature for Apple and Google Maps, it is not easy to get a building properly mapped out and added to their map. Luckily, Apple and Google provide several Application Program Interfaces (APIs) and tools for creating indoor maps of your own buildings. While their official tools are still in beta development and not available for venues like schools, this thesis explains how to construct indoor maps on our own with a little work.

Mappalachian, the application described in this thesis, aims to create maps that are accessible on a cellphone or computer so that students, faculty, and visitors can more easily navigate the university's many features. The primary goal is to make an easy to use application that behaves similarly to other mapping applications, such as Apple Maps, Google Maps, Waze, and others. Thus, the Mappalachian interface is familiar and does not obstruct any additional information provided. Another goal is to make the maps more interactive than a normal building directory on a wall. Mappalachian displays certain information about a room so that people who are unfamiliar with the building can easily see points of interest on the map. The application was designed for use on cellphones or tablets running iOS (iPhones and iPads) and computers running macOS; however, applications for other platforms could be created to use the same data format to appeal to more users with little maintenance.

Chapter 2 provides detail about the data needed and how it was collected for this project. Chapter 3 describes a standard format that is used to store data to create indoor maps for the application. The application itself and how it functions is described in Chapter 4 as well as details about the APIs used to create it. Chapter 5 discusses potential future work that can be created to expand and improve the application and data collection workflow.

Chapter 2

The Data

In order to create indoor maps, a way to represent real-world locations on a virtual map is needed. This chapter describes decimal degrees as a way to format geographic coordinates and will also go into detail about how coordinates are obtained to represent buildings and indoor rooms.

2.1 Decimal Degrees

The standard for referring to real-world locations through coordinates is called the World Geodetic System (WGS 84) [1]. Whenever talking about exact locations on the Earth, a format that is used to indicate latitude and longitude is decimal degrees. There are other formats for specifying locations, such as delimited degrees, that can be converted to decimal degrees for a more human-readable format. A coordinate in the decimal degree system can be represented using two floating point numbers which can refer to any location on the planet. Latitude indicates positions running parallel to the equator and can range from $-90°$ to $90°$ where negative values indicate locations south of the equator and positive values indicate locations north of the equator. Longitude indicates positions running parallel to the Prime Meridian and can range from $-180°$ to $180°$ where negative values indicate locations west of the Prime Meridian and positive values indicate locations east of the Prime Meridian.

Coordinates in general are not difficult to acquire. Anyone with a cellphone can get their current location represented by coordinates easily. The challenge lies in obtaining coordinates

Table 2.1: Accuracy of coordinates by number of decimal places

Decimal Places	**Degrees**	**Accuracy**	**Recognizable Object**
0	1.0	111 km	Country or state
1	0.1	11.1 km	Large city
2	0.01	1.11 km	Town or village
3	0.001	111 m	Neighborhood
4	0.0001	11.1 m	Individual street
5	0.00001	1.11 m	Individual house
6	**0.000001**	**11.1 cm**	**Individual person**
7	0.0000001	1.11 cm	Individual finger on a human hand

precise enough to refer to a certain location. Referring to things as small as corners of a room, we need to be very specific with our coordinate points so that the room or building is as close to its real life shape as possible. The more decimal places that a given coordinate has, the more specific of a location that it refers to. Table 2.1 provides levels of accuracy that can be expected when using a given amount of decimal precision in your coordinates. For example, a coordinate with no decimal places can refer to something as large as a specific country but a coordinate with six decimal places can refer to an individual human's location.

For this project, six decimal places of precision are used in order for the map to be as accurate as possible. Unfortunately, obtaining coordinates this precise often requires expensive equipment. Grabbing a measuring tape and measuring every single wall of Anne Belk Hall was immediately ruled out as infeasible unless the project only mapped a few rooms. Even if measuring a few rooms were to be enough data for this project, it is not efficient to do on the scale of an entire level let alone every building in the university. As an alternative, we needed something that had the same scale as the actual building and showed us all of the rooms. Building directories might be useful in this case but they are not accurate to scale and do not exist in every building. Fire escape maps may be size accurate and available in every building but, they are not large enough to use to make a reliable map.

2.2 Floor Plans

Luckily, the university provides accurate architectural floor plans through the Office of Institutional Research, Assessment and Planning. These drawings are delivered through a vectorized PDF so that they can be scaled to any size while maintaining their scale true to the original building.

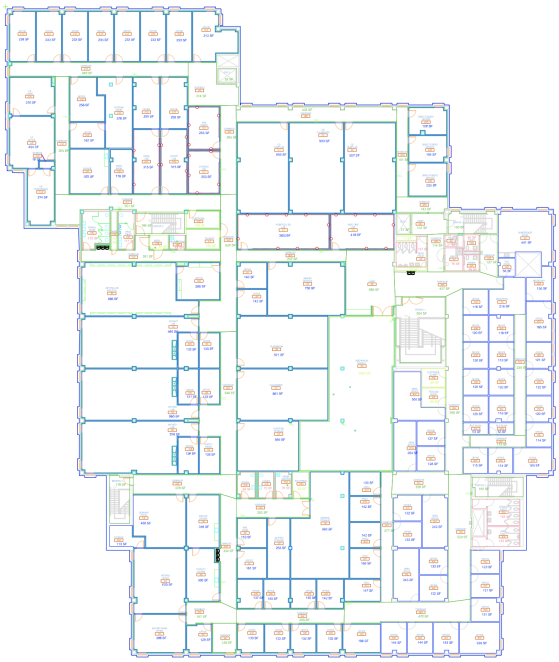

Figure 2.1: Floor plan of Anne Belk Hall as of 1/13/15, after its renovation

Each room on the provided floor plan has measurements, the room's primary purpose (at the time of the drawing), and doorways into and out of each room. At first glance, this floor plan provides all the information needed. However, these maps lack a way to anchor points on the floor plan to coordinate points in the real world. No coordinates are provided for us to be able to calculate the distance between two corners of a room. While floor plans are very helpful in visualizing the layouts of each building and floor without having to actually visit them, they need to be combined with something else in order to make a complete map.

2.3 OpenStreetMap

The next step is to situate the floor plans on to a real-world map so that coordinates can be obtained. There are several different mapping applications on the Internet for this purpose, but something easy to use and easily updated was desirable. OpenStreetMap, as its name suggests, is an open-source mapping application that relies on contributions by community members in order to build their maps [4]. The maps are completely free to use for anyone and all of the data is open source. The application does not have a controlling institution, but is supported by the OpenStreetMap Foundation and donations by the community. As a result of the maps being made and verified by members of the local community or local area, OpenStreetMap tends to be more accurate than maps that are made by larger companies such as Microsoft, Google, and Apple which do not have a permanent presence in most areas.

Figure 2.2: Exterior of Anne Belk Hall on OpenStreetMap

The map in Figure 2.2 shows Anne Belk Hall as it exists currently on OpenStreetMap. The map includes points of interest like bicycle racks, picnic tables, the Technology Support Center, one interior elevator, and parking behind the building. It also includes pathways and exterior entrances to the building. This is a great starting point because it provides accurate location data for the building. What remains is a way to edit this data and insert the data from the floor plans of each building.

Java OpenStreetMap Editor

Java OpenStreetMap Editor (JOSM) is an open-source, Java-based editor for OpenStreetMap. JOSM is compatible with Windows and macOS so that data can be created without depending on a specific platform. The program allows the user to edit the data that is available from OpenStreetMap and add new data items. By default, JOSM comes with standard map editing tools that allow for creation of streets, walkways, entrances, parking lots, and much more. After downloading the data from OpenStreetMap, JOSM displays it in the editor window as shown in Figure 2.3 and Figure 2.4. All of the map data is stored on the local computer so it can be edited offline if needed. Map data edited with JOSM can also be uploaded back to the OpenStreetMap servers if the user would like their contributions to be included in the official distribution.

JOSM also allows for the installation of user and community created plugins like "PicLayer," "BuildingTools," and "IndoorHelper" that add new features to JOSM to help with different aspects of cartography. A plugin called "PdfImport" is able to directly import PDF files into JOSM and render them as a floor plan wireframe but unfortunately due to current limitations of the plugin buildings with irregular shapes do not provide accurate coordinates. "PicLayer" is used in lieu of proper "PdfImport" support to convert the PDF into image segments and place them onto the map to use as references when creating entries on OpenStreetMap. In Mappalachian's use case, "PicLayer" allows us to import the official floor plans and overlay them onto the existing building outline that was provided by a contributor to OpenStreetMap. By aligning the floor plan with the outline of the building in JOSM as shown in Figure 2.3, we can get accurate coordinates for features of the building and begin drawing our maps.

The last step is to create a wireframe model that represents the current floor plan as shown in Figure 2.4. Because "PicLayer" only allows for rasterized images to be imported, the floor plan PDFs must first be converted into PNGs. While converting a vectorized PDF into a rasterized image results in a loss in quality, the images are still able to represent accurate locations on our real-world map. The process of creating a wireframe is done manually by tracing each floor plan and creating nodes and boundaries to represent walls that separate different areas. Plugins for JOSM could be created in the future to automatically complete this

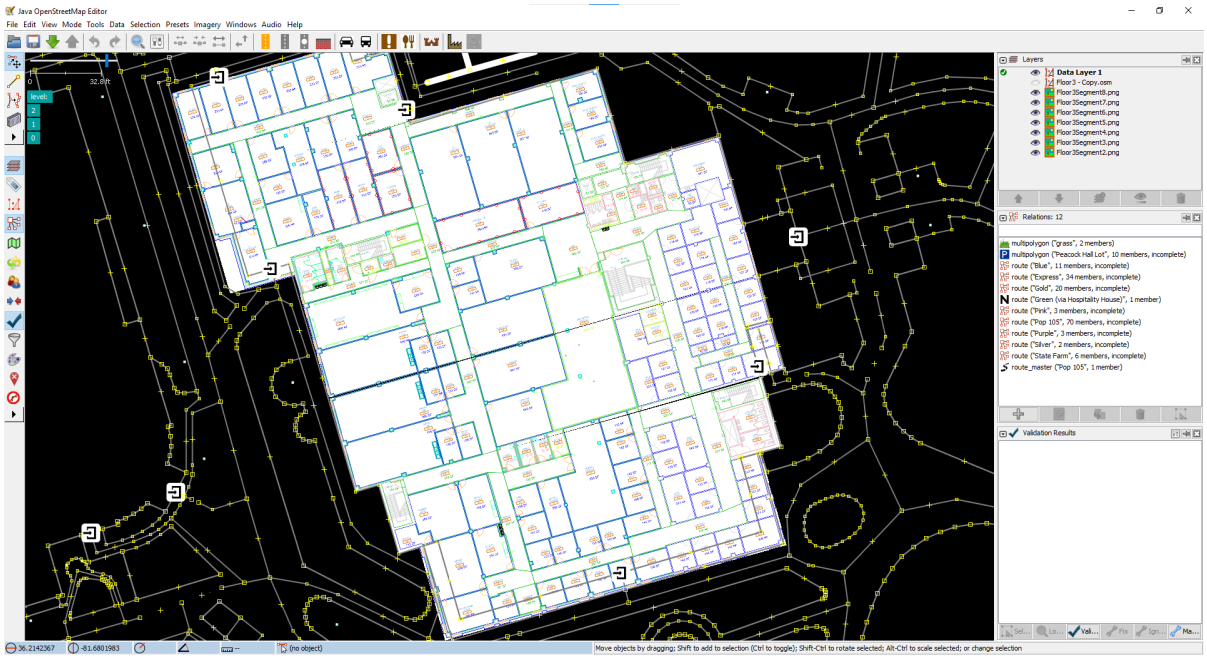

Figure 2.3: Anne Belk Hall with floor plan overlay in the JOSM Editor

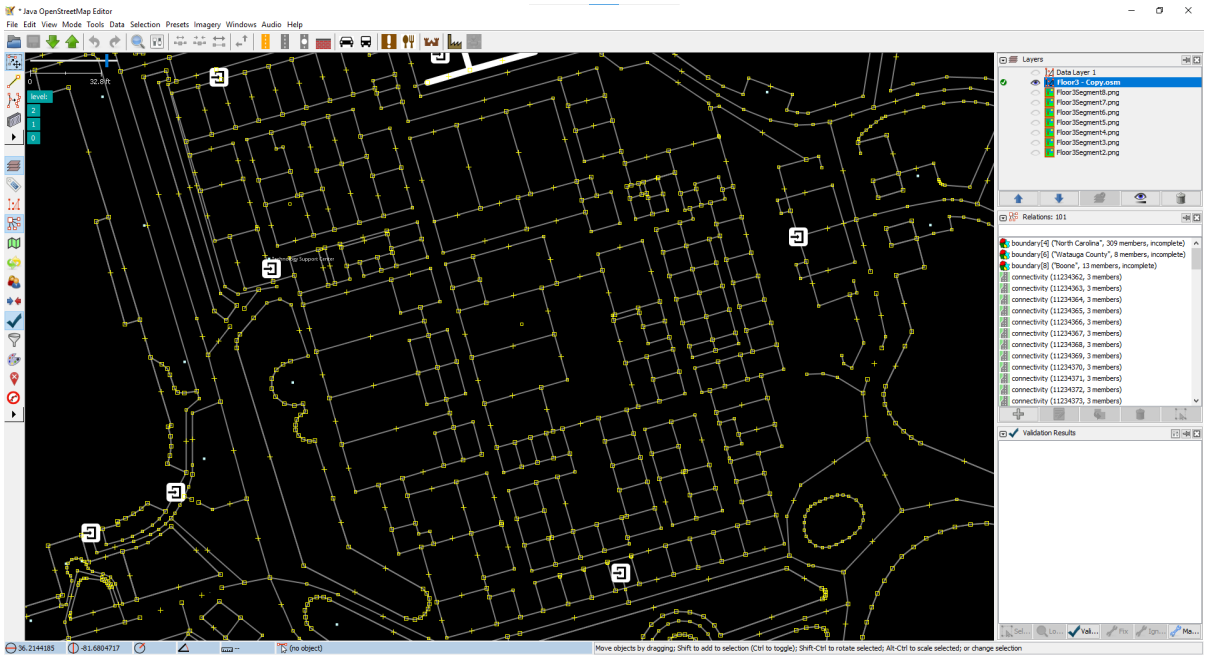

Figure 2.4: Anne Belk Hall with a completed wireframe floor plan in the JOSM Editor

process to remove the manual element which would speed up the creating of indoor maps. Once the model is complete for one level, the coordinates that represent every vertex on the level must be translated into another file format that can be exported and used in Mappalachian which will be described in Chapter 3.

Chapter 3

The Format

Chapter 2 discussed how to obtain the data needed to draw indoor maps for the Mappalachian application. A single level of Anne Belk Hall contains over 600 coordinate pairs. Multiply that by the three levels that the building contains and the total number is roughly 1800 vertices. Each vertex contains two floating point numbers representing a decimal degree which was described in Section 2.1 for a total of about 3600 decimal values needed to be stored to represent one building. The data that is extracted from JOSM is an array of coordinates with each point representing a vertex of a polygon within a room. The next thing that needs to be figured out is the best way to format and store this data so that it can be easily edited by a human if needed. For example, the layout of a floor could be changed to combine several smaller rooms into a larger or or split up a larger room into several smaller ones. In this case only a few rooms would need to be manually edited instead of re-creating the entire floor plan. Another requirement for our format was that it needed to be readable by a computer in a timely manner so that there will be no delays in the user interface while the application is in use. Any data storage option also could not be platform-specific so that it could be accessible on any computer or operating-system without the need to install an external program.

3.1 GeoJSON

GeoJSON, as defined by RFC7946, is a standardized data format based on JavaScript Object Notation (JSON) [2]. It defines several types of JSON objects to represent data about geo-

graphic features and their properties. GeoJSON uses a geographic coordinate reference system, the World Geodetic System, and units of decimal degrees as described in Section 2.1.

RFC7946 also defines eight main types of geographic shapes that can be represented by GeoJSON [2]:

Position: The basic geographic object that all other objects are composed of. Represents a location in space given as two floating point numbers indicating latitude and longitude. A third number can also be used to indicate the altitude of the position.

Point: Composed of a single Position representing one point.

MultiPoint: An array of Positions that represents a collection of unconnected points.

LineString: An array of two or more Positions that represents a line.

MultiLineString: An array of LineStrings that represent two or more connected lines.

Polygon: An array of Positions representing a complete shape.

MultiPolygon: An array of Polygon objects representing two or more distinct shapes.

GeometryCollection: A collection of Geometry objects that can be different types but stored in the same file.

Listing 3.1 shows a simple GeoJSON file defining a `Point` object that represents a location in Anne Belk Hall. When rendered on a map, Figure 3.1 shows the resulting `Point` object as a pin over the center of Anne Belk Hall.

Listing 3.1: A GeoJSON `Point` object

```
1 {
2     "type": "Feature",
3     "geometry": {
4         "type": "Point",
5         "coordinates": [-81.6804859, 36.2142697]
6     }
7 }
```

Figure 3.1: The annotation created by the `Point` object in Listing 3.1

Drawbacks

GeoJSON provides a good base for storing this data but does not concretely define certain features that could be useful to display information on a map. Additional information about a GeoJSON object *could* be added, but support for additional features depends on the implementation of the platform rendering the object and could be ignored entirely. When creating an indoor map, a way to distinguish one `Polygon` object from another would be useful to the application as they could be rendered differently. In the case of Mappalachian, Section 4.1 will describe how levels are rendered differently based on whether or not they are "active", meaning that the user is able to see all of the rooms inside. Section 4.1 will also describe how rooms and annotations within a level are rendered differently based on their primary purpose. As GeoJSON lacks standardized support for properties, it alone is not a perfect solution for creating indoor maps.

3.2 Indoor Mapping Data Format

The Indoor Mapping Data Format (IMDF) is an extension of the GeoJSON specification that aims to provide "a generalized, yet comprehensive model for any indoor location, providing a

basis for orientation, navigation and discovery" [9]. It was proposed by Apple Inc. to the Open Geospatial Consortium, an international consortium driven to make geospatial information and services findable, accessible, interoperable, and reusable, and was approved as a standard in February of 2021. The format conforms fully to the GeoJSON specification described in RFC 7946 and standardizes feature types that build upon the GeoJSON format to provide more real-world information that can be displayed on a map.

The IMDF also defines sixteen different feature types that cover a broad range of objects found inside public areas [9]:

Address: Represents a postal address associated with an element of a `Venue`.

Amenity: Models the physical presence and approximate location of a pedestrian amenity. Examples include an ATM, elevators, staircases, and bicycle parking.

Anchor: Represents a location to display a feature that either does not have an `Address` or convenient display location.

Building: Represents a physical building associated with a `Venue` (see below).

Detail: Models the physical presence, location, and extent of a physical object that is recognizable in a real-world setting. Examples include columns, walls, and medians.

Fixture: Models the presence and physical extent of a moveable or semi-permanent physical asset. Examples include desks, furniture, plants, fences, or water features.

Footprint: Models the physical extent of one or more referenced `Building` objects.

Geofence: A virtual boundary that is not displayed to the user on a map, but may activate certain features of an application if the user is within its geometry.

Kiosk: Models the presence and physical extent of furniture that facilitates the distribution of products and/or services. This type only describes the physical features of a kiosk; an `Anchor` or `Amenity` can be used to model information about the feature if desired. Examples of kiosks include vending machines, small shops, or digital signage displays.

Level: Models the presence, location, and physical extent of a floor area.

Occupant: Models the presence and location (via `Anchor`) of a business entity that trades goods and/or services that are separately represented from the larger building. Examples include restaurants in an airport or shops in a shopping mall.

Opening: Models the presence, location and physical extent of an entrance. An entrance may be a door that swings, slides or rotates, a gate/turnstile, or threshold.

Relationship: Models the physical or conceptual association between two map elements. An example of a `Relationship` object could be two `Unit` objects representing the same elevator/staircase on different floors.

Section: Models the presence and approximate extent of a theme. Sections are used for visible differences between spaces that are not separated by a wall.

Unit: Models the presence, location and approximate extent of a space separated by a wall or other physical/legal boundary.

Venue: Models the presence, location, and extent of a location. Examples of a `Venue` include Appalachian State University, Boone Mall, and Raleigh-Durham International Airport.

The objects provided by the IMDF cover just about everything that is needed to create indoor maps. A `Unit` object can represent an individual classroom, office, staircase, elevator, or restroom, a `Level` object can lay out the basic floor plan for an entire level, and a `Building` can represent the building containing the `Levels` and `Units`. `Openings` can be used to indicate entrances to `Buildings` and `Units` on a map for areas that might not have an obvious entrance. Other objects like a `Kiosk` can represent a vending machine, food truck, or service desk. Not all of these objects will be used in Mappalachian in its current state however, the potential for expansion upon the current maps is feasible.

Listing 3.2: An IMDF `Building` object representing Rankin Science West

```
1  {
2      "features": [
3          {
4              "feature_type": "building",
5              "geometry": null,
6              "id": "RSW",
7              "properties": {
8                  "address_id": "RSW-addr",
9                  "alt_name": null,
10                 "category": "academic",
11                 "display_point": {
12                     "coordinates": [-81.6819106, 36.2142358],
13                     "type": "Point"
14                 },
15                 "name": "Rankin Science West",
16                 "restriction": null
17             },
18             "type": "Feature"
19         }
20     ],
21     "name": "building",
22     "type": "FeatureCollection"
23 }
```

Listing 3.2 shows the final GeoJSON file for Rankin Science West. On line 4 it defines the IMDF `feature_type` of `building` to indicate that this object is a `Building`. `Building` objects are one of the only objects that does not explicitly require a corresponding `geometry` The `geometry` of the building is usually left to be defined in a `Level` or `Footprint` object. One of the notable features of IMDF is the addition of standardized properties within each object. Properties like `category` and `restriction` can be used to render objects differently based on their purpose or not show them at all if the areas are restricted to the current user. For example, mechanical or electrical rooms could be restricted to only be shown to physical plant employees and other maintenance staff.

Listing 3.3 shows the final GeoJSON object for Room 310 in Anne Belk Hall. The main structure of this file is similar to that of Listing 3.2 with a few differences to indicate that this object represents one room rather than an entire building. Notably, `Unit` objects and `Building` objects use different `category` types. Mappalachian currently uses three category types for `Buildings`: `academic`, `recreational`, and `administration`. For `Units`, there is a large list of different categories possible, some examples include `classroom`, `conference`, and `restroom.female`. While many categories are defined in the IMDF specification, addi-

tional categories can be added with the caveat that support for rendering different categories will have to be added into the application separately. The listing for Room 310 also includes a `geometry` object which specifies that it is a GeoJSON `Polygon` object, discussed as a part of the GeoJSON format in Section 3.1, and includes four sequential coordinate pairs that represent corners of the room.

Listing 3.3: An IMDF `Unit` object representing Room 310 in Anne Belk Hall

```
{
    "type": "Feature",
    "feature_type": "unit",
    "id": "310",
    "properties": {
        "accessibility": null,
        "alt_name": null,
        "category": "classroom",
        "display_point": {
            "coordinates": [-81.7050517, 36.2231394],
            "type": "Point"
        },
        "level_id": "BH-level-3",
        "name": "Room 310",
        "restriction": null
    },
    "geometry": {
        "type": "Polygon",
        "coordinates": [
            [
                [-81.6805886, 36.2145217],
                [-81.6805505, 36.2144165],
                [-81.6804722, 36.2144344],
                [-81.6805102, 36.2145396]
            ]
        ]
    }
}
```

3.3 Benefits

The IMDF has several benefits that help achieve the use cases of Mappalachian. One of the main goals of GeoJSON was to be interoperable between different applications and operating systems and the IMDF also conforms to these same ideas. Because the use case of Mappalachian is not platform specific, a data format that allowed for the best interoperability between systems was desired. This meant that any platform specific code or libraries should not be used to store our data. Libraries like CoreData, a data storage solution created by Apple for their devices, may be fast and convenient, but they are only available for Apple platforms which eliminates the

possibility of using the same data to provide information to Android cellphones or use on the web. SQLite is another popular database solution for use on mobile devices however the main drawbacks to SQLite are the slight learning curve to be able to use it and that entire databases cannot be updated easily. Database solutions *could* be hosted on a remote service to make them more cross platform, but this requires the device to have constant access to high-speed internet to be able to access the map itself. Loading times downloading over 600 coordinates per-floor, in the case of Anne Belk Hall, would not be as quick as having the information stored locally. Moreover, the data does not change frequently so keeping data remote introduces an unnecessary performance degradation.

Using the IMDF, GeoJSON files can be combined together in a `FeatureCollection` where multiple `Unit` or `Level` objects can be stored in the same file. They can also be stored separately where each object has its own file to make a more human readable data structure. The ease of use of GeoJSON is an important advantage as it allows people who are not familiar with application or database development and are more familiar with collecting geographic information to not have to learn how to interface with the application or database in order to update the information. Because of the structure and overall small file size, a server-side program could be implemented to easily update individual elements a building (this is discussed further in Chapter 5).

In the case of Mappalachian, GeoJSON files are very easy to work with. Swift, the language that Mappalachian is programmed in, supports a protocol called `Codeable`. The `Codeable` protocol allows for the easy conversion of the application's data to and from an external file such as GeoJSON or JSON. An example of a `Codeable` object can be seen in Listing 3.5 and its respective JSON representation can be seen in Listing 3.4. This object is used within Mappalachian for the Schedule tab described in Chapter 4.

Listing 3.4: JSON object used to represent an authenticated user

```
{
    "status": "success",
    "userId": "123456789",
    "authId": "styreswn",
    "roles": [
        "Admissions",
        "financialaid",
        "student"
    ]
}
```

Listing 3.5: A `Codeable` swift object used to represent an authenticated user

```
struct User: Codable {
    var username: String
    var roles: [String]
    var bannerID: String
    var name: String?

    private enum CodingKeys: String, CodingKey {
        case username = "authId", roles, bannerID = "userId", name
    }
}
```

Chapter 4

The Application

Mappalachian is an application made using Xcode and the Swift 5 programming language for mobile devices running iOS. Chapter 2 described how data was collected and the specific format of this data was described in Chapter 3. This chapter will go into detail about the application's features, how it functions, and connecting the data to the application.

Mappalachian was designed to be similar to other mapping applications like Apple Maps and Google Maps that might already be installed on the user's device. For this thesis, two main features were designed that might be common use cases for users of the app. Each feature is contained in a "tab" which is a common way of displaying interfaces on iOS applications as seen in Figure 4.1. The first tab, also the initial view of the application, is the map itself which is described in Section 4.1. The second tab has the ability to view the current user's schedule and view locations of their classes and is described in Section 4.2.

Figure 4.1: Mappalachian's tab bar interface

4.1 The Map

The "Map" tab, shown in Figure 4.2, is the main feature of the app and is the first thing visible after opening the app.

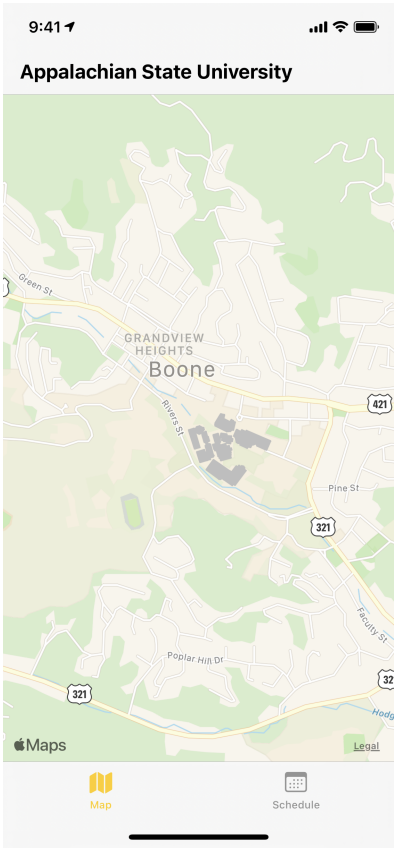

Figure 4.2: Starting view of the map tab

The map is initially set to show an overview of the university's main campus referenced by the `geometry` of the `Venue` object seen in Listing 4.1, so that the entire campus is visible when first opening the app. This is important to provide a frame of reference to users who may be unfamiliar with the campus but can recognize roads like Highway 321 and Highway 421.

Listing 4.1: An IMDF `Venue` object representing Appalachian State University

```
{
    "features": [
        {
            "feature_type": "venue",
            "geometry": {
                "coordinates": [
                    [
                        [-81.6879932, 36.2212525],
                        [-81.6914264, 36.2147959],
                        [-81.675688, 36.2093483],
                        [-81.6722547, 36.2158054]
                    ]
                ],
                "type": "Polygon"
            },
            "id": "appalachian-state-venue",
            "properties": {
                "address_id": "appalachian-state",
                "alt_name": null,
                "category": "university",
                "display_point": {
                    "coordinates": [-81.679017, 36.214034],
                    "type": "Point"
                },
                "hours": "Su-Sa 09:00-17:00",
                "name": "Appalachian State University",
                "phone": null,
                "restriction": null,
                "website": "https://appstate.edu"
            },
            "type": "Feature"
        }
    ],
    "name": "venue",
    "type": "FeatureCollection"
}
```

MapKit

MapKit is a framework created by Apple for developers to display map or satellite imagery within their apps [5]. Because the map is the main feature of Mappalachian, the application relies heavily on MapKit to handle the rendering of all of the information on screen.

The main interface is provided by `MKMapView` which displays a generic map for users to interact with and the application to manipulate. The interface inherits all of the primary features of Apple's own "Maps" application, which can be seen in Figure 4.3, that comes standard with all iOS devices. Features such as pinching to zoom and scrolling to pan across the map are available with no extra code required. There are extra features visible in the

Figure 4.3: Standard Apple Maps interface as seen in iOS 14

Figure 4.4: Zoomed in view of the Mappalachian interface

Apple Maps interface, such as building outlines and points of interest, that are removed from the Mappalachian interface. Building outlines provided by Apple are removed due to some inaccuracies about their precise location which would overlap when buildings are added by Mappalachian. Points of Interest provided by Apple are also removed from Mappalachian's interface as they are again not as accurate as ones that Mappalachian provides. In larger cities, Apple Maps can be extremely accurate with several annotations and precise building locations. The inaccuracies that exist in Apple Maps are mainly due to Boone being a smaller town where frequent map updates are not performed. These features are removed to ensure that Mappalachian has the most control over what is displayed on the map and add extra features without relying on updates from Apple.

`MKMapView` has two main features that allow for application customization: annotations and overlays. Annotations are objects that allow for displaying points of interest like restaurants, parking, and bus stops while overlays control physical objects rendered on the map. Mappalachian makes use of both overlays and annotations to build a map of the campus and the interiors of buildings which is explored in the following sections.

Buildings

In Figures 4.2 and 4.4, buildings of the Appalachian State University campus that are rendered by Mappalachian are shown as gray overlays. Buildings such as Rankin Science West are stored in GeoJSON files, seen in Listing 3.2, and loaded into a `Codeable` object, described in Section 3.3, when the application launches. These objects are rendered directly onto to the map using `MKOverlayRenderer`.

By default, all of the buildings rendered by Mappalachian are shown without their interior rooms rendered. This is done to reduce the overall clutter of having indoor rooms being shown for every building on screen and to reduce CPU and memory usage by only rendering interiors when buildings "become active". To make a building active and view the interior, as seen in Figure 4.5, the user can zoom in using a pinch-to-zoom gesture and the building that is currently closest to the center of the screen have its the top layer removed and render all of the rooms on the topmost floor. Mappalachian renders visible `Levels` with a white fill color and a gray stroke width while inactive levels are rendered with a gray fill color and stroke color.

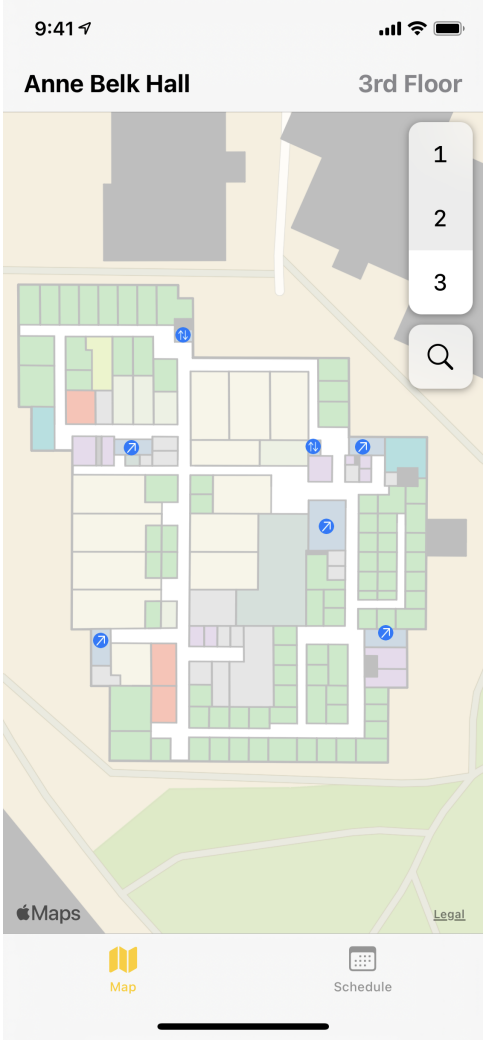

Figure 4.5: The 3rd floor of Anne Belk Hall as shown by Mappalachian

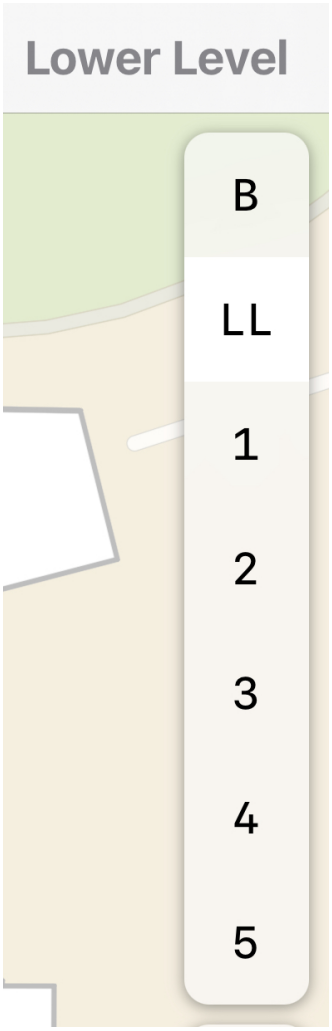

Figure 4.6: Level Picker

`Levels` that are below the currently selected level are rendered as inactive to give context to how big a given level is compared to the ones below it.

Level Picker

When viewing a building, the topmost floor is rendered by default. Having an easy way to switch floors is necessary to be able to properly view the entire building. In Figure 4.5 the level picker is visible on the right side of the map interface when viewing a building. The level picker is built dynamically based on the data provided by the `levels` array of the current

`Building` object. The `levels` array of each `Building` is filled with `Level` objects that contain information about the rendered level. Figure 4.6 shows the floors of the Belk Library and Information Commons. The level picker shows the `shortName` property in the picker itself, such as B, LL, 1, 2, 3, 4, and 5, so that it does not take up space. It also shows the currently selected level's `name` property in the navigation bar, such as Lower Level or 3rd Floor, as well as highlighting the `shortName` in the level picker itself. This is useful for representing levels that do not have numbers such as the Lower Level and Basement of the Belk Library. To use the level picker, a simple tap is required on the desired level and Mappalachian will remove the currently rendered level and show the selected one.

Rooms

On top of each `Level` overlay, `Units` are rendered. As described in Section 3.2, `Units` are enclosed areas within a given `Level`. Mappalachian uses `Unit` objects to represent rooms. Depending on the `category` property of the `Unit`, each room can be rendered differently. When an object is ready to be rendered, the application calls a method to configure the renderer based on the object's current category. For example, Mappalachian renders `Units` with a category type of `restroom` with a purple color while `elevator` and `stairs` categories are rendered with a blue color. The stroke width of each room is also thinner than the stroke width of a `Level` object to differentiate between interior and exterior walls. This is done to make rooms of the same type easily identifiable to the user.

The `category` of a room is not the only property that can change the render type of an object. Any value in the `properties` of an IMDF object can be used to configure the renderer though Mappalachian only uses the `category` value currently. As an example, other properties like `outdoor` can be used to differentiate an indoor section of a level from an outdoor patio. Listing 4.2 shows an example of the GeoJSON file representing room 312B in Anne Belk Hall.

Listing 4.2: An IMDF `Unit` object representing Room 312B in Anne Belk Hall

```
{
    "type": "Feature",
    "feature_type": "unit",
    "id": "312B",
    "properties": {
        "accessibility": null,
        "alt_name": null,
        "category": "office",
        "display_point": {
            "coordinates": [-81.6808103, 36.2146139],
            "type": "Point"
        },
        "level_id": "BH-level-3",
        "name": "Room 312B",
        "restriction": null
    },
    "geometry": {
        "type": "Polygon",
        "coordinates": [
            [
                [-81.6808103, 36.2146139],
                [-81.6807883, 36.2145528],
                [-81.6807513, 36.2145613],
                [-81.6807734, 36.2146222]
            ]
        ]
    }
}
```

Amenities

An `Amenity`, described in Section 3.2, is a way to indicate points of interest. In Figures 4.3 and 4.5, points of interest are locations on the map represented by a circular icon so that the purpose of a designated area can be viewed without fully zooming in. `Amenitiy` objects do not have polygon representations like `Levels` and `Units` do. Instead, they have a single coordinate representing where the annotation should be placed on the map.

Annotations are not statically rendered objects like overlays are. They can be configured to provide information when they are interacted with or hidden when the map's camera is too far away from them. Mappalachian currently provides annotations for stairs and elevators within buildings, as seen in Figure 4.5, to show ways to travel between levels.

Other types of annotations, like restrooms, vending machines, and water fountains, can easily be added using the same configuration system used with `Levels` and `Units`. Listing 4.3 shows an example of an annotation representing the entrance to an Elevator on the third

floor of Anne Belk Hall. Line 3 indicates that the IMDF feature type is an `Amenity` object. In the `properties` list, Line 9 indicates that the category of this object is an elevator which is used to select the color and image of the annotation when rendered on the map. There are also several properties that can be used to indicate more information about the amenity such as hours, name, phone number, and website however these are not used for the elevator but could be used for other amenities such as restaurants and stores.

Listing 4.3: An IMDF `Amenity` object representing an elevator in Anne Belk Hall

```
1  {
2      "type": "Feature",
3      "feature_type": "amenity",
4      "id": "Elevator1Floor3Point",
5      "properties": {
6        "accessibility": null,
7        "address_id": "BH-addr",
8        "alt_name": null,
9        "category": "elevator",
10       "correlation_id": null,
11       "hours": null,
12       "name": null,
13       "phone": null,
14       "units": ["Elevator1Floor3"],
15       "website": null
16     },
17     "geometry": {
18       "type": "Point",
19       "coordinates": [-81.6803869, 36.2144431]
20     }
21 }
```

Room Search

Being able to view rooms on a map is certainly more useful than just knowing the exterior shape of a building, but something is still missing. A user is able to view rooms, but cannot see specifically which room number it is. A useful feature would be the ability to search for a room by its room number. Room numbers are commonly used in places like BannerWeb when registering for classes or a professor's syllabus when looking for office hours.

To make this process easier for users who may be unfamiliar with the layout of a building, the room search feature allows a user to search through an entire building to find a specific room number. When looking at a building a search button appears below the level picker. This is shown as the magnifying glass icon in Figure 4.5. Tapping on the search button brings up the search interface, shown in Figure 4.7, which displays every room in the building sorter

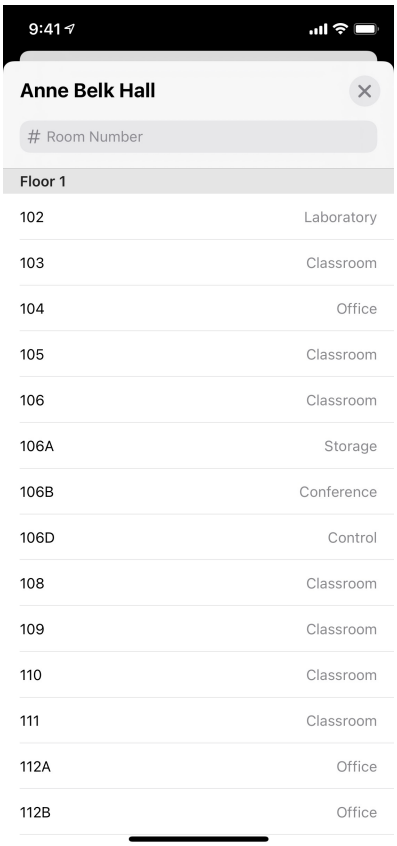

Figure 4.7: Standard search interface for Anne Belk Hall

numerically and by floor. The search interface also provides a search bar at the top for text to be entered. Search results are updated automatically to help find the room that the user is looking for, shown in Figure 4.8. Categories are also displayed on the search page to help a user narrow down the room they are looking for whether it be an office or a classroom. Categories can also help with broader searches such as finding the closest restroom or study room. Once the user has found their desired room, a tap on the room number will dismiss the search view and focus the map over the selected room, switching levels if necessary. The selected room will also be highlighted so that users can easily distinguish it from other rooms that might be clustered in the same area, as shown in Figure 4.9.

The room search feature was designed so that the functionality of the feature itself does not have to be updated when new rooms or buildings are added to Mappalachian's dataset. It is dynamically populated based on the `Unit` objects of the current `Building`. Once a room

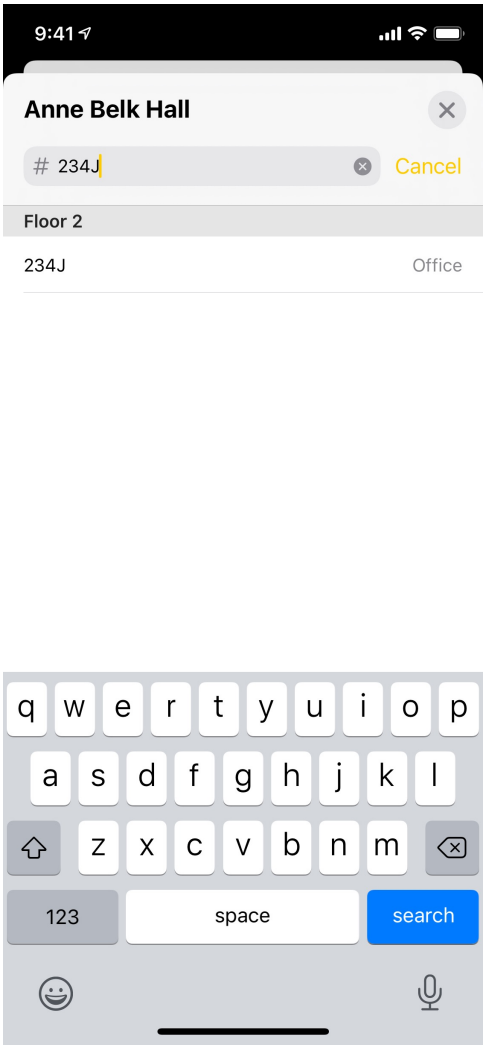

Figure 4.8: Filtered search interface after searching for a room

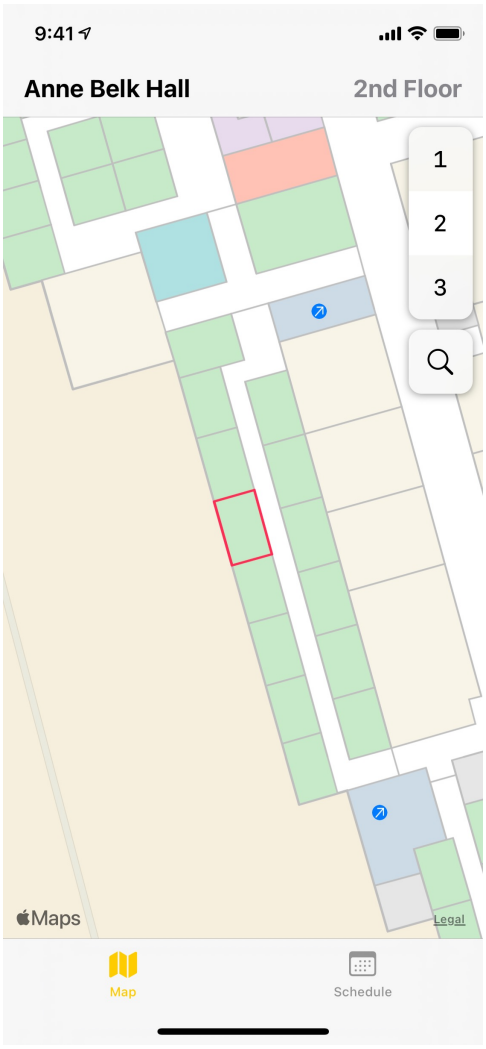

Figure 4.9: Room 234J highlighted in Anne Belk Hall

is added to the map via a corresponding GeoJSON file, it is also added to the list of available rooms without needing any updates to the room search feature or its interface.

4.2 User Schedule

The second main feature of Mappalachian is the ability to list the user's current schedule. Many students at Appalachian will walk their schedule before the first day of classes in order to know the location of their classes. But not all students do this and buildings with complicated layouts may have more than one path to a room depending on where it is entered.

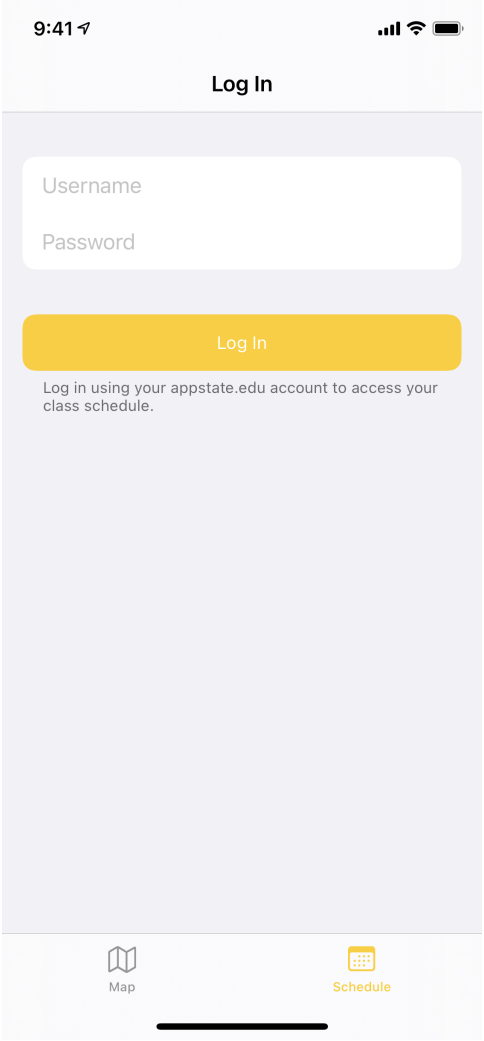

Figure 4.10: Login screen for the schedule tab

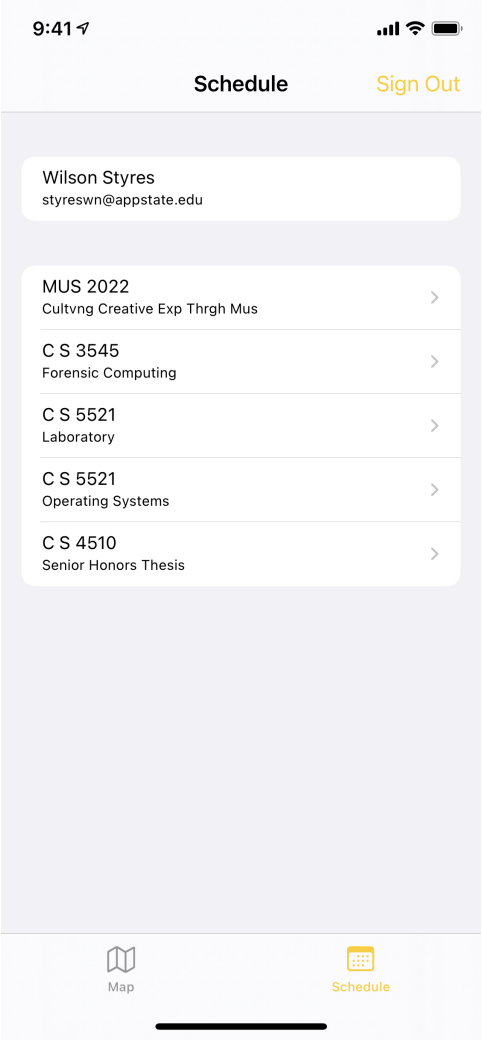

Figure 4.11: Schedule tab when the user is authenticated

Since the search feature described in Section 4.1 was implemented to locate rooms on the map, a natural extension was to create a way to list a student's schedule from within the app so that a user can more quickly locate rooms on their schedule. Students who use the app will need to authenticate with their appstate.edu credentials with the login screen, seen in Figure 4.10, in order to see their schedule. The application authenticates directly with Bannerweb, a service used by Appalachian for many student activities including financial aid, registration, and applications, to retrieve the student's schedule for the current term. The student's credentials are securely sent through the Banner Mobile API using the `NSURLRequest` class from Apple's

`CoreFoundation` library. If an authentication is successful, the user's credentials are saved securely using `NSURLCredentialStorage` so that they do not have to be re-entered every time the application is opened. Users who are not students or are not authenticated can still use the app as normal but will have no access to the schedule feature. Once the student is logged in, the schedule screen in Figure 4.11 shows the user's information as well as their schedule for the current semester.

Tapping on a course in a user's schedule behaves similarly to the search feature seen in Figure 4.9. The app switches views to the map tab and repositions the map's camera to focus on the selected room. The map also highlights the selected room to increase visibility of the room to the user. This feature re-uses some of the code used for highlighting and focusing the camera with the room search feature. All rooms that are visible on the map with the proper building and room identifier are able to be selected by a course in the user's schedule tab.

4.3 Mac Catalyst

Mac Catalyst, more commonly referred to as Catalyst, is a native solution created by Apple to share application code across their iOS and macOS platforms [8]. With Catalyst, an app built for the iPad ARM64 architecture can be recompiled to be compatible with an x86_64 Mac and display the same interface with minimal adjustments. This allows applications built for the Apple ecosystem to function on macOS without any major code modifications. Because the codebase is shared between platforms, all features or new map data that is incorporated into the iOS app are available in the macOS app. Many of Apple's own applications like News, Maps, and Music now are built using Catalyst to increase feature parity between platforms.

The macOS version of Mappalachian can be seen in Figure 4.12. The user interface is largely the same as the iOS application but the interface is scaled up to accommodate the extra screen space and also uses the macOS window manager which can be seen by the familiar "stop light" window controls in the upper left corner. While the app primarily is designed for a touch interface, it still can be used on a desktop environment without a touch screen. Instead of the user touching the screen with their fingers the mouse primarily is used to navigate the application. Keyboard support also is available for text and map manipulation shortcuts. For

example: + and - can be used to increase and decrease the zoom level respectively. The arrow keys also can be used to pan through the map without the use of the mouse.

Mappalachian's support for the Mac may not have the same mobility and convenience as having the application on your phone, but a better use case for desktop use is in a kiosk environment. As visitors to the campus and members of the community may be unfamiliar with the application, buildings that regularly have visitors like the Belk Library, Student Union, and the Schaefer Center could have interactive directories in common areas. These kiosks would allow visitors to search for rooms within the building without the need for a phone with Mappalachian installed on it.

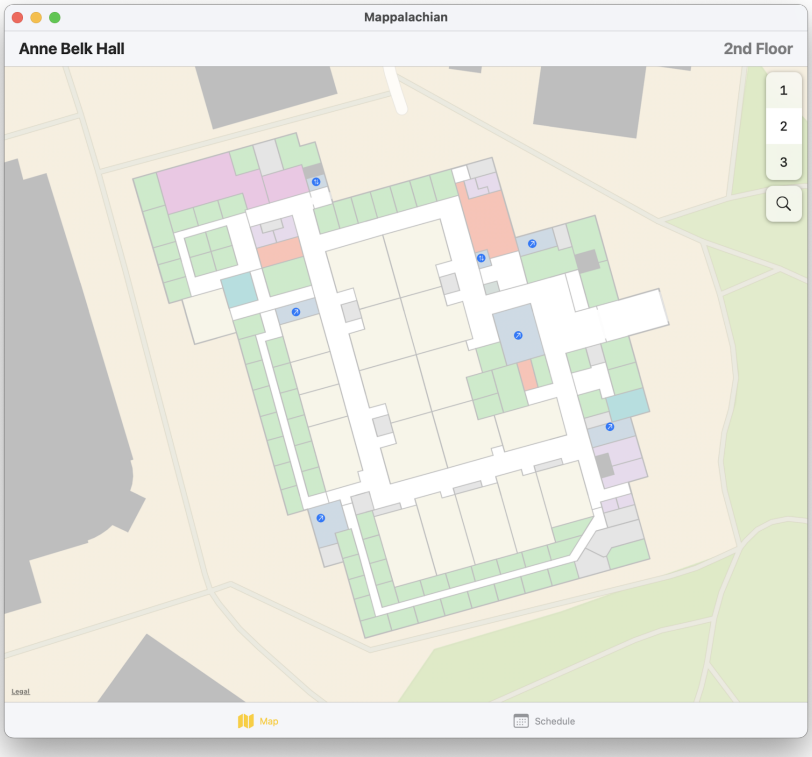

Figure 4.12: Mappalachian running on macOS using Catalyst

Chapter 5

Future Work

The potential use cases for Mappalachian extend far beyond the ones covered in this thesis. Many ideas were explored during the creation of Mappalachian, but there was insufficient time to do all of them. This chapter will briefly explore ideas for the application that were not implemented but would be useful to users of the app in the future.

5.1 Legends - The Mappalachian Backend

All the data gathered in Chapter 2 is stored locally within in the application itself. Every install of the application contains all of the information needed to draw a complete map of Appalachian State University's campus. This is done in order for the application to access this data as quickly as possible without loading time. However, not every piece of data can be feasibly contained within the application itself. As with paper maps, Mappalachian also needs a legend of its own. Named "Legends", after one of the buildings on Appalachian's campus, the service would provide a representational state transfer (REST) API that would be able to serve information that is not stored locally within the application. A few examples of what Legends could be used for and their benefits will be explored in this section.

Detailed Room Information and Search API

The map tab of the application, as described in Section 4.1, displays rooms in their real-world locations and colors them based on their primary use, but it does not provide much information

other than that. Information like tutoring events, classes, and office hours that take place in a given room would be useful information to display to users of the application to increase discoverability of events on campus. Since room numbers, the dimensions of a room, and the room's primary purpose do not change often, they are stored in the application and do not need to be fetched often. However, because other information such as class schedules and the location of a professors office could change often, it would not be kept locally so that the entire application would not need to be updated in order to update this information. Other campus events, such as voting sites, concerts, and club meetings, are other locations that could be included in this search to both students and members of the community alike.

A simple REST API could be added to Legends to provide this information, when requested, by the application. This extra information could be accessible either through the map itself or through the search feature. The extra information would enable users, such as students, to view information about rooms that are not on their schedule. As an example, a student wishes to go to a professor's office for office hours but does not know where the office is or the room number. The student would be able to search for the professor's name through the application and view their office on the map.

Remote Map Updater

As discussed in the previous section, information about a room such as room number, dimensions, and primary purpose of a room do not change often, but that does not mean that they never need to be updated. In recent years, many of Appalachian's buildings have undergone renovations or complete reconstruction. Even in buildings that have not been renovated, minor changes to the layout of a given floor could be made to make use of space or add new accommodations. However, these changes do not happen often enough to warrant having all of the layout information stored remotely. As the application would be distributed through Apple's App Store, any updates to the application itself would require going through the app review process which could delay information after buildings are officially opened.

An update mechanism could be implemented into Mappalachian with data storage located remotely in order to update layout information of a given building or add new buildings entirely. When the application launches, information about map updates would be fetched from

Legends and if an update was available it would be downloaded and stored in the application for future use. To make this process as quick as possible, the application would ask the server if an update is available using either the current version or the date that the data was last updated. A delta update mechanism and compression would also be used to reduce total file size of the updates. This would allow the application's data to be updated remotely without app review delays or relying on application development cycles.

5.2 Turn-by-Turn Directions

In most mapping applications, such as Apple Maps and Google Maps, the user can request turn-by-turn directions to navigate them to a desired location. Implementing this feature in Mappalachian would be useful for buildings that might be hard to navigate for those unfamiliar with its layout. Since the interface of Mappalachian is three dimensional, it can be confusing seeing only one floor on the screen at a time. Mappalachian mitigates this confusion by using annotations to indicate transitions, such as stairs and elevators, between floors. A full turn-by-turn interface would provide a clearer a way to navigate through a building and between floors to get to a given room.

5.3 Self-Guided Tour

One of the most important things when choosing a university to attend is the atmosphere around it. Many students tour Appalachian State's campus each year trying to decide whether or not the school is the right fit for them. Tour guides provide valuable information about on campus areas and events that might not be easily accessible for a student who is looking at the campus online. Because Mappalachian already contains information about the layout of campus, it would be a great medium to provide information about areas of campus to prospective students. Using many already discussed features such as annotations, the search feature, and turn-by-turn directions, Mappalachian can allow users to tour our campus without needing a formal in-person tour. These tours could also include pre-recorded videos by campus tour guides, either in video form or even in augmented reality using Apple's ARKit, so that the formal tour experience is provided for those who cannot attend one.

5.4 Java OpenStreetMap Plugins

As discussed in Section 2.3, the Java OpenStreetMap Editor can install third-party plugins for assistance in making maps. The process used to create Mappalachian's maps was largely manual with much of the GeoJSON and IMDF data being created by hand. Plugins for JOSM would make this process a lot more efficient and increase the ability of future maintainers to update the application's data and map future construction with ease. Potential plugins could include:

- Modifications to "PdfImport" to allow for the importing of non-standard building layouts.
- A plugin to automatically convert floorplans in a PDF to nodes and ways in JOSM.
- A plugin to export entire buildings into their corresponding IMDF representation.

5.5 Other Platforms

Mappalachian is currently only available on iOS and macOS platforms and this is extremely restrictive for usability of the application. While Mappalachian is designed in Swift, a largely iOS specific language, thought was put into other platforms that Mappalachian could be transferred to in the future. As discussed in Chapter 3, care was taken in choosing a format that is not platform specific. Though the IMDF is created by Apple, it uses the GeoJSON format which can be used on any platform depending on the ability to parse it. This section will explore other platforms that can make use of the data collected and display them on other platforms.

Android

A natural extension to Mappalachian on iOS would be Mappalachian on Android. Android is the other dominant cellphone operating system and is used by several manufacturers whereas iOS is used exclusively by Apple for Apple products. Creating an Android based application for Mappalachian would be beneficial for all of the students, faculty, visitors, and community members who do not have access to an iPhone or Mac. Similar to iOS, Google has frameworks that allow developers to expand upon their native mapping application (in this case Google

Maps as Apple Maps is not available on Android devices). Using these frameworks, a similar application could be made for Android users to view interiors of Appalachian State's buildings.

Mappalachian on the Web

The platform that could expose the largest number of users to Mappalachian would be having a web-based interface. Luckily, several services are available that could allow Mappalachian to be hosted on the web.

The first, and easiest based on the work done for the iOS application, is MapKit JS [7]. MapKit, described in Section 4.1, is Apple's framework for displaying maps in iOS. They also provide a web-based framework called MapKit JS which provides many of the same functions as MapKit but can be displayed on the web. Because our IMDF data is already compatible with MapKit on iOS, minimal work is needed to display indoor maps with MapKit JS.

Another option to bring Mappalachian to the web is to use the Google Maps APIs to upload our mapping data to Google Maps. If the native Android application was created ahead of time, data could be used from that application similar to how MapKit JS uses the data from the iOS application. Creating a web-based interface also opens the door to react native and electron applications for a variety of platforms. This would reduce overall maintenance that would be needed for the codebase of different platform but could reduce the overall functionality of the application due to lack of support for native frameworks in web-based applications.

Having a web-based interface would also remove the cost component for creating interactive kiosks that were described in 4.3. With Mappalachian on the web, any computer, not just the on-average more expensive Mac computers, could be used to display indoor maps for visitors. Even something as simple as a Raspberry Pi could be used to display a web interface that could be updated automatically for events that are happening within a given building.

Chapter 6

Conclusion

The application described in this thesis provides a working base for an application that could change the way that people at Appalachian State University discover and explore its campus. Several off-campus students and students who transferred after their first year may not even realize the many benefits that the campus has to offer.

Mappalachian provides an easy way for users to explore the campus through a medium that is easily accessible. While the application may be the most useful for students at the moment, many ideas described in this thesis also appeal to a much broader audience that either work at or visit Appalachian. In addition, the application also can be brought to other platforms to be usable by a much wider audience.

Overall, Mappalachian has the potential to redefine Appalachian State University with its ability to massively improve discoverability of locations and events on campus. In the future, it may even be a central point for all Appalachian State related activities that are happening on-campus.

Bibliography

[1] National Geospatial Intelligence Agency. World geodetic system 1984. http://web.archive.org/web/20210127210044/https://earth-info.nga.mil/GandG/publications/tr8350.2/wgs84fin.pdf, 2002.

[2] H. Butler, M. Daly, A. Doyle, S. Gillies, S. Hagen, and T. Schaub. The geojson format. RFC 7946, RFC Editor, August 2016.

[3] Open Geospatial Consortium. About ogc. https://www.ogc.org/about.

[4] OpenStreetMap Foundation. Openstreetmap. https://www.openstreetmap.org/about.

[5] Apple Inc. Mapkit. https://developer.apple.com/documentation/mapkit?language=objc, 2009.

[6] Apple Inc. Encoding and decoding custom types. https://developer.apple.com/documentation/foundation/archives_and_serialization/encoding_and_decoding_custom_types, 2014.

[7] Apple Inc. Maps on the web. https://developer.apple.com/maps/web/, 2018.

[8] Apple Inc. Mac catalyst overview. https://developer.apple.com/mac-catalyst/, 2020.

[9] Apple Inc. Indoor mapping data format. OGC 20-094, Open Geospatial Consortium, Februrary 2021.

[10] Immanuel Scholz and Dirk Stöcker. Josm. https://josm.openstreetmap.de.

[11] Appalachian State University. Appalachian state university facts. https://www.appstate.edu/about/facts/.