Sergey Ushakov

Incorporating Maps into Flutter: A Study of Mapping SDKs and Their Integration into a Cross-Platform Navigation Application

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

14 February 2024

Abstract

Author:	Sergey Ushakov
Title:	Incorporating Maps into Flutter: A Study of Mapping SDKs and Their Integration into a Cross-Platform Navigation Application
Number of Pages:	37 pages
Date:	14 February 2024
Degree:	Bachelor of Engineering
Degree Programme:	Information Technology
Professional Major:	Mobile Solutions
Supervisors:	Ilkka Kylmäniemi, Senior Lecturer

The thesis evaluates various mapping SDKs for integration into a Flutter-based navigation application, Vedu - Tallinn Transport. The study aims to identify the most suitable map SDK by comparing Google Maps, Mapbox, and flutter_map SDKs in terms of performance, feature set, cost, and developer experience. The study is critical for the application's success, given its reliance on mapping functionality. The thesis provides insights into the optimal choice of map SDK for cross-platform development in Flutter.

The SDKs were evaluated based on compatibility with the aforementioned requirements. Each SDK was studied, benchmarked, and compared on a set of quantitative and subjective key metrics. Quantitative benchmarks consisted of measurements of performance, resource consumption and pricing. SDKs were subjectively ranked from best to worst on account of documentation quality and overall developer experience.

flutter_map SDK was found to be the most suitable candidate for future application development, based on its low cost, high customizability, and Flutter-native design. Moreover, limitations were found in SDKs that augment platform-native code, such as Mapbox or Google Maps SDKs, by looking into Flutter's architecture. The main limitation was found to be Hybrid Composition, mostly affecting performance. Secondary limitations were API design and a complex mechanism to control a marker's presentation, among others.

In conclusion, this study not only identifies the optimal SDK for Flutter-based applications but also contributes to a broader understanding of integrating mapping technologies in cross-platform mobile development, underscoring the importance of tailored solutions in the evolving landscape of mobile application development.

Keywords: Flutter, maps, Google Maps, Mapbox, iOS, Android

Table of Contents

List of Abbreviations

SDK: Software Development Kit. A set of programs and tools used to develop a piece of software, usually provided by the platform vendor.

API: Application Programming Interface. A set of protocols and tools that facilitates communication between two pieces of software.

DX: Developer Experience. A term used to describe, for example, the experience of a developer using a product, its APIs, documentation, and functionality.

FPS: Frames Per Second. A metric used to measure performance of the application. Indicates how many frames per second the device can render, processing the user's input.

UI: User Interface.

UX: User Experience.

CDN: Content Delivery Network. A system of distributed servers that deliver web content to a user based on their geographic location, the origin of the webpage, and the content delivery server, ensuring fast, reliable, and secure delivery of internet content.

MAU: Monthly Active Users. This is a key performance indicator (KPI) commonly used in the digital and technology industries, particularly by websites, applications, and online platforms, to measure the number of unique users who engage with their product or service within a 30-day period.

1 Introduction

Most individuals in developed nations possess a smartphone. According to a survey conducted by Nepa in 2022, 97% of Finnish respondents confirmed their ownership of a smartphone. (DNA, 2022). The smartphone is one of the biggest software platforms in the world; therefore, it is in companies' interest to be present on this platform.

The two major software platforms on the market are iOS made by Apple and Android made by Google. In total, they capture more than 99% of the market in Europe. Android's market share is leading at 65%, while iOS's market share is at 34% (Statcounter, 2023). Since the market is divided between two platforms, a company wishing to capture the market must make its offering available on both platforms.

Two distinct approaches can be adopted when developing an application for the iOS and Android platforms. The initial approach involves creating two separate applications using platform-specific tools provided by the respective vendors. The advantage of this approach lies in the ability to tailor the application to each platform, ensuring a superior user experience that aligns with the platform's conventions. However, this approach necessitates the employment of two separate development teams, resulting in increased costs. Consequently, many companies are opting for the second approach: developing a cross-platform application.

Flutter is an open-source framework developed by Google for creating cross-platform applications using a unified codebase (Google, 2023). With Flutter, developers can build applications for various platforms such as mobile, desktop, web, and embedded systems. The applications in Flutter are written in Dart, a null and type-safe language that can be compiled to run on multiple platforms.

This thesis aims to investigate various map SDKs that can be integrated into a Flutter application. The findings of this study serve as the foundation for selecting the most suitable map SDK for a navigation application developed

using Flutter. The specific application under consideration is called Vedu - Tallinn Transport, designed to assist the residents of Estonia in navigating public transportation routes. A significant portion of the application's functionality is built around a map; hence, choosing the best map SDK is crucial for the success of the application and its development process.

The study involves a comparison of three major map SDKs available on the Flutter package repository: `google_maps_flutter`, `mapbox_maps_flutter` and `flutter_map`. `google_maps_flutter` and `mapbox_maps_flutter` are Flutter packages that encapsulate native SDKs into Flutter widgets, while `flutter_map` is a package written in Flutter from ground up. The comparison is based on the SDKs' performance, feature set and subjective developer experience. The best SDK will be used for the further development of the application.

2 Introduction to Flutter

A cross-platform application is an application that shares most of the code between both platforms. This is made possible by SDKs that facilitate sharing of the code. One such SDK available publicly is Flutter, developed by Google. Flutter applications are written in Dart and can be deployed not only on iOS and Android, but also on Web, Windows, Mac, and Linux (Google, 2023). This thesis will be focusing on Flutter for iOS and Android, since the application described in this thesis is made available on those platforms exclusively.

One of the key advantages of Flutter is its extensive ecosystem of packages. Flutter provides a wide range of packages developed by the Flutter team, while the community also contributes numerous packages. This ecosystem enables the utilization and comparison of multiple map SDK solutions, as many existing platform-native solutions have been adapted to be compatible with Flutter.

2.1 Widgets

Flutter draws significant design inspiration from React, a popular library for constructing user interfaces on both web and mobile platforms (Google, 2023). In Flutter, the user interface is defined using widgets, which declaratively describe the UI based on the current state of the application. When the state changes, the widgets are rebuilt to reflect the updated state. This reactive approach allows for efficient updates and ensures that the user interface remains synchronized with the underlying data and application’s state.

A widget in Flutter represents the smallest unit of a user interface. It has its own state and can accept inputs and have side-effects. A widget generates a specific portion of the user interface as its output. Widgets encapsulate various components, such as buttons, text fields, images, or more complex UI elements, and can be composed together to build the complete user interface of an application.

2.2 Rendering

To gain a comprehensive understanding of the rendering process in Flutter, it is beneficial to compare how Flutter applications and conventional iOS/Android applications render UI. The comparison is further enhanced by examining React Native, a cross-platform framework that shares similarities with Flutter.

Conventional native Android and iOS applications use native UI rendering engines specific to their respective platforms. Currently, in the case of Android applications, views are created using the Android framework, which utilizes the Skia graphics engine to render the user interface onto a canvas (Google, 2023). Similarly, iOS applications generate Views through the usage of UIKit and SwiftUI high-level frameworks, which rely on the Core Graphics framework for rendering the UI elements on the screen (Apple, 2012).

React Native, a cross-platform application development framework built upon React, operates by utilizing JavaScript as its primary runtime language. React Native applications generate React component trees, which are subsequently transmitted to the React Native platform renderer. Then, the React Native platform renderer interacts with UIKit and the Android framework to generate native views specific to each platform (Meta, 2023). Unlike React Native, Flutter does not perform direct calls to the platform's native rendering engine. Instead, Flutter uses Skia to render views onto a canvas on all platforms, including but not being limited to iOS, Android, Windows, and macOS.

This approach has various advantages and disadvantages. A significant benefit lies in the fact that a Flutter application, exclusively rendering Flutter widgets, has consistent appearance and behavior across both platforms. In contrast, React Native does not provide the same level of assurance due to differences in platform views used by React Native. However, the downside of Flutter's approach is the increased complexity of integrating platform-native views, as Flutter bypasses the native platform's rendering systems. Instead, Flutter attempts to combine native views with a Flutter view through a process known as Hybrid Composition, which carries major performance implications. This

consideration holds significance when selecting a suitable Map SDK, because two out of three SDKs use Hybrid Composition.

3 Requirements

This section describes the requirements to the Map SDK candidate. The requirements can be categorized into the following groups:

- Feature set and customizability
- Performance
- Cost
- Developer Experience (DX)

3.1 Feature set and customizability

The map in the target application is used to display elements that convey information to the user, as well as to display interactive elements. Considering the application's purpose of facilitating navigation with public transport, the feature set will primarily revolve around displaying markers and drawing lines on the map. The comprehensive list of elements to be displayed on the map includes:

- Markers showing locations of transport vehicles, with heading arrows
- Tappable markers representing stops
- Polylines for representing the path of the trip
- Marker showing user location and heading

The selected SDK should offer comprehensive control over the presentation of markers, allowing to utilize either a Flutter widget or an image for customization. This level of control allows the application to convey information to the user, for example, showing the position of a bus with a color-coded circle marker with the line number inside. Example use cases can be seen in Figure 1.

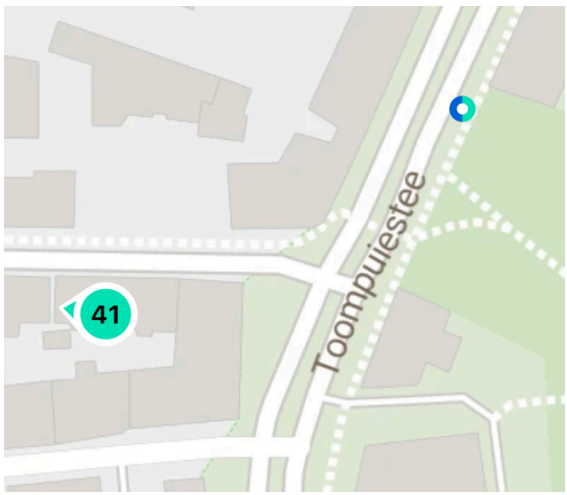

Figure 1. Markers signifying a bus and a stop positioned on a map.

As shown in Figure 1, markers with custom presentation have two purposes: to inform and to function. The green marker labeled '41' shows the position of bus number 41, and the arrow shows which way the bus is heading. The marker in the top right corner of the image represents a bus stop, and the colors show what kinds of transportation use that stop. Users can click on it to get more information about the stop.

Another important feature the SDK should provide is changing the map camera position with animations. Animations provide a natural way to guide the user around the map. When switching focus from the route overview to the user's location, the animation allows the user to understand where they are located on the route. Similar to how animations aid users in maintaining their orientation within the user interface (Daliot, 2015), they can be leveraged to assist users in retaining their spatial orientation within the map.

3.2 Performance

The map is an important interactive element of the user interface. The information about the planned route or the public transport network can be accessed via the map. Certain features can only be accessed through interactions with the map, such as tapping on stop markers to retrieve additional information about the stops.

It is important to recognize that maps are resource-intensive elements within the user interface. A poorly performing map not only hampers the overall user experience but also impacts the performance of other elements within the interface, causing slower operation. If the map feels sluggish or unresponsive during interactions, it can significantly diminish the user's perception of the product and act as a deterrent to achieving "Behavioral Delight."

Theresa Fessenden describes "Behavioral Delight" as one of the three pillars of user experience. It emphasizes that if tasks, such as finding a stop and obtaining bus line information, are exceptionally easy and enjoyable, users are more likely to recommend the product or return to it in the future (Fessenden, 2022). Therefore, ensuring high performance of the map is critical for both expanding and retaining the user base.

To evaluate and compare the performance of each map, the metric of Frames Per Second (FPS) was utilized. A higher FPS indicates that the device can render more frames in a second, resulting in smoother and more responsive visuals. Studies demonstrate that users prefer higher framerates when interacting with software (K. Debattista, 2018), with a preference for 60 FPS over 30 FPS. Thus, the target minimum rate for performance evaluation will be 60 FPS, as most smartphone screens have a maximum refresh rate of 60Hz, with many newer smartphones having a higher refresh rate of 90 or even 120Hz.

It is crucial to pay particular attention to the memory usage of the app, especially considering the complexity of Map SDKs. The intricate processes involved in rendering map tiles and performing complex calculations have the potential to consume substantial amounts of device memory. On iOS, specific memory usage limits are set for applications. These limits are designed to conserve the resources of the device and enhance the overall user experience. Exceeding those can lead to the application being terminated by the system when running in the background. (Apple, 2023). Implications of this on user experience are great; a user might lose their progress in the app or get lost along their itinerary. While exact numbers for memory limits imposed by iOS on

apps are a subject of discussion, especially with introduction of devices with higher memory capacity. Such considerations are vital for maintaining optimal app performance and ensuring a smooth user experience.

3.3 Cost

There are multiple companies that have built their business on providing Map SDKs and supplementary resources. Since providing a Map SDK and tiles for that SDK is a resource intensive process, requiring a lot of monetary and computer power investment, rarely a Map SDK is entirely free. Free alternatives are present on the market, but they usually lack customization and quality.

The expense sheet will vary based on the chosen SDK. Certain SDKs are self-contained, necessitating no additional resources for their functionality. Conversely, other SDKs may require developers to provide their own source of map tiles. Although such SDKs are typically available for free, the cost lies in acquiring the map tiles themselves. In summary, if the application demands a specific level of map customization and aims for a premium quality user experience, it is important to recognize that associated costs will likely be incurred. Free options are unlikely to offer the desired level of customization or premium quality.

3.4 Developer Experience (DX)

Developer Experience (DX) refers to the overall quality of interactions, tools, and support provided to software developers throughout the software development lifecycle. It aims to enhance productivity, satisfaction, and collaboration by minimizing barriers and optimizing workflows.

Quantitatively describing the level of developer experience proves challenging since it encompasses various factors. Several aspects collectively contribute to the perceived DX, including the development speed with the SDK, the availability of comprehensive documentation, support from the community, and the overall ease of implementing the required feature set. Although DX cannot

be precisely quantified, it will be evaluated based on developer feedback and subjective assessments.

4 Candidates

The SDKs range from plug-and-play commercial products to community-built projects that require additional resources to work. The list of considered SDKs is as follows:

- Google Maps SDK
- flutter_map SDK
- Mapbox Maps SDK

4.1 Google Maps SDK

The Google Maps SDK, provided via the `google_maps_flutter` Flutter package, is a plug-and-play SDK requiring no supplementary resources for operation. This SDK, collaboratively developed by Google and the Flutter community, is not a Flutter-native package; rather, it serves as an intermediary layer bridging the Flutter API with native platform SDKs for iOS and Android. This integration uses Hybrid Composition, which facilitates the simultaneous rendering of platform-native views in conjunction with Flutter views.

4.1.1 Feature set and customizability

Google Maps SDK fulfils all feature criteria listed in Section 3.1. It provides a declarative way to create map features, like markers, polylines, and other map features. Moreover, it provides a way to imperatively control the map's camera with animations, fulfilling that criterion as well.

4.1.2 Cost

This SDK requires an API key to function, requiring the developer to create a developer account on Google Cloud. The SDK is not free to use; hence, the developer must also provide their payment details to Google. As of July 2023, Google provides 200 USD free credit every month, allowing the developer to

implement and launch an app with the SDK for free initially, but the SDK will start generating costs the more users the app has.

Google presents the pricing of their SDK in USD per 1000 requests. The pricing is non-linear, getting cheaper the more requests are made. However, the definition of a request is not clear. Google offers a way to use the SDK for free on Android, provided that the map is not customized in any way.

Experiments with the SDK were performed on a subset of users. The usage resulted from the experiments were not considered free by Google, complicating the comprehension of their pricing strategy further.

4.1.3 Developer Experience

Given that this package leverages platform-native components, particularly the Google Maps SDK for iOS and Android, setup requires modifications to the platform-native code by the developer. Thankfully, the package comes with straightforward and easy-to-follow setup documentation, making this stage relatively simple.

Part of the application functionality is to show public transport vehicles on the map; hence, one of the requirements is the ability to render custom markers on the map. The `google_maps_flutter` package does provide this ability, but with a caveat. The package does not provide a way to use Flutter widgets to render markers' presentation.

The markers' presentation can be provided from a `BitmapDescriptor`, which is an image. It can either be loaded from assets or rendered real-time. Hence, to dynamically define a marker's visuals for use as transport markers, the developer must write code that renders the visuals using a Flutter `Canvas`, then export them as an image, and then convert it to a `BitmapDescriptor`.

This procedure is highly demanding in terms of resources, necessitating the developer to implement caching strategies to mitigate performance degradation.

Initial challenges include the complex nature of the code used for rendering, exporting, and caching marker images. Further complication is that it became impractical to render every vehicle marker with a direction arrow, a process that would require almost every vehicle marker image to be rendered individually. Eliminating the direction arrow allowed for a higher cache hit rate, consequently reducing the impact on performance.

The SDK provides an interface to imperatively control the map, `GoogleMapController`. It facilitates moving the map camera with animations, allowing not only to focus on a point, but also on a collection of points, which is useful when focusing on an itinerary, as seen in Listing 1 below:

```
final bounds = CameraUpdate.newLatLngBounds(boundsFromItinerary(locations),
10);
controller.animateCamera(bounds);
```

Listing 1. A routine to focus the map's camera on the itinerary with an animation.

Also, the controller allows setting the map style from a JSON, which can be generated in the Google Maps Platform control panel, as Listing 2 illustrates:

```
final style = rootBundle.loadString("assets/map_style.json");
controller.setMapStyle(style);
```

Listing 2. Loading the style file and using the GoogleMapController to set the style.

To summarize, the highlighted parts of the API interface fulfil the requirements completely, without necessitating any workarounds.

Adhering to platform conventions is integral for a package to be considered a good citizen within its ecosystem. As mentioned in Section 2, Flutter's widgets have a declarative API that specifies the widgets to be instantiated, rather than performing the procedural steps for their creation. The package `google_maps_flutter` aligns with this convention, which is shown in Listing 3.

```
Set<Polyline> getPolylines() {
  final state = mainMapStateController.currentState;

  return state.itinerary.legs.mapIndexed(
    (index, leg) {
      return Polyline(
        polylineId: PolylineId(index.toString()),
        points: getLegPolylinePoints(leg),
        color: leg.getDisplayColor(),
        endCap: index == state.itinerary.legs.length - 1
            ? Cap.roundCap
            : Cap.buttCap,
        startCap: index == 0 ? Cap.roundCap : Cap.buttCap,
        jointType: JointType.round,
        geodesic: true,
        patterns: leg.mode == TransportMode.walk
            ? [PatternItem.dash(10), PatternItem.gap(10)]
            : [],
        width: 4,
      );
    },
  ).toSet();
}

GoogleMap(
  initialCameraPosition: center,
  padding: mapPadding,
  myLocationButtonEnabled: false,
  zoomControlsEnabled: false,
  mapToolbarEnabled: false,
  rotateGesturesEnabled: false,
  myLocationEnabled: userLocation.locationAccessEnabled,
  onCameraMoveStarted: _handleMapCameraMoveStarted,
  onCameraMove: _handleMapCameraMove,
  onMapCreated: _handleMapCreated,
  polylines: getPolylines(),
  markers: [
    ...stopMarkers,
    ...transportMarkers,
    ...getStartStopMarkers(startStopIcons),
  ],
)
```

Listing 3. Google Map widget with polylines and markers.

The code presented in Listing 3 demonstrates that markers and polylines are represented as widgets and are subsequently provided to the `GoogleMap` widget as parameters. The package's function is to communicate with the native SDK and perform the required steps to render these elements on the screen. Notably, the code is not performing explicit invocations to the `GoogleMap` widget's methods to generate markers or polylines; it declares that the `GoogleMap` widget should display polylines returned from the `getPolylines` function.

4.2 flutter_map SDK

`flutter_map` SDK stands as the only fully Flutter-native candidate. It is open-source, it does not rely on an underlying platform-native SDK. Instead, it is implemented fully using Flutter. `Leaflet.js`, an open-source JavaScript library for implementing maps on Web, served as an inspiration for this SDK (flutter_map Authors & Maintainers, 2023). Like Leaflet, `flutter_map` provides only map rendering and control interfaces, necessitating the developer to find a source of map tiles to be displayed on the map. Without the tiles the SDK does not function.

In the context of a Map SDK, a map tile refers to a small, square piece of a map that can be combined with other tiles to create a full map. These tiles are pre-rendered and typically measure 256x256 or 512x512 pixels. Both Google Maps SDK and Mapbox SDK come with a set of tiles predefined. `flutter_map`, on the other hand, does not. An array of services, both free and paid, offer raster and vector tiles, including but not limited to OpenStreetMap, Stadia Maps, MapTiler, Google, and Mapbox. Furthermore, a suite of open-source tools is available that enables the generation and serving of the tiles using the hardware of the developer's preference.

4.2.1 Feature set and customizability

`flutter_map` fulfils the criteria listed in Section 3.2 fully. Given that the SDK necessitates the developer to supply a source of tiles, and its Flutter-native nature and a system of plugins, it emerges as the most customizable and controllable SDK among its competitors.

4.2.2 Cost

`flutter_map` SDK is entirely free to use, given that it is open-source and does not provide any tiles. The tiles, however, can generate a significant cost. If the tile presentation quality and customizability is of no importance, a free tile set like OpenStreetMap can be used. Otherwise, as previously mentioned, paid

sources of tiles can be used. For example, Mapbox provides a Static Tiles API that can be used with `flutter_map`.

The present configuration of the application, accommodating 1000 monthly active users, initiates approximately 7 million map tile requests, as determined by app analytics. As of November 2023, fulfilling these requests through Mapbox would result in expenses amounting to 2200 USD per month (Mapbox, 2023). There are more cost-efficient sources of tiles like Stadia Maps, providing a subscription service that allows 7.5 million monthly requests for 80 USD per month (Stadia Maps, 2023).

Nevertheless, at the current scale of application, self-hosting the tiles was found to be the most customizable and cost-efficient solution. Deploying a web server using open-source software to deliver the tiles together with a CloudFront Content Delivery Network (CDN) suffices to meet the current demand. Moreover, it provides capacity for scaling the userbase multiple times, at a minimal expenditure of approximately 40 USD per month.

4.2.3 Developer Experience

The `flutter_map` SDK, being Flutter-native, provides the most optimal DX in comparison to its counterparts. A substantial advantage is its ability to utilize Flutter widgets for the rendering of map markers without necessitating any intermediary steps. This facilitates the rendering of all transport markers with arrows without a significant performance degradation. The code required to render the markers is less complex and can be interpreted and modified by a Flutter developer regardless of their experience level.

While the SDK does not provide a built-in way to move the map camera with animations, a community-maintained plugin `flutter_map_animations` provides that functionality.

```
final bounds = getLatLngBoundsFromItinerary(itinerary);

controller.animatedFitBounds(bounds,
  options: FitBoundsOptions(
    padding:
     EdgeInsets.symmetric(vertical: 20, horizontal: 54)
       .copyWith(top: topPadding + 20),
)));
```

Listing 4. A routine to focus flutter_map on the selected itinerary with animations.

The code in Listing 4 shows a call made to a method provided by the `flutter_map_animations` package, which results in the map fitting the viewport to the bounds with an animation.

4.3 Mapbox SDK

Mapbox is a company offering a diverse range of mapping software as a service. Among their suite of products is the Mobile Maps SDK for both iOS and Android. Recently, Mapbox has unveiled an official Flutter package, `mapbox_maps_flutter`. This package operates similarly to the `google_maps_flutter` package, acting as an adapter layer interfacing Flutter with the native platform SDK - in this instance, the Mobile Maps SDK from Mapbox. Therefore, this plugin also employs Hybrid Composition, integrating native views concurrently with Flutter views, a process that can cause performance issues.

4.3.1 Feature set and customizability

`mapbox_maps_flutter` also covers all feature and customizability requirements. Map styling is a strength of Mapbox ecosystem, with Mapbox having created an opensource map styling standard. In fact, subjectively, Mapbox SDK is the easiest to customize, with `flutter_map` SDK being a close second. SDK supports implementing smooth camera animations. Basic features like custom markers and polylines are supported as well.

4.3.2 Cost

Commercial usage of Mapbox Mobile Maps SDK is not available free of charge. While a generous free plan is provided, the specifics of this plan lack full transparency. Mapbox implements a pricing strategy where costs are based on the number of monthly active users engaging with Mapbox services within an application. The price depends on the number of unique users interacting with the application during the month. It claims to offer service to 25,000 monthly active users at no cost (Mapbox, 2023). It should cover the application's current userbase.

4.3.3 Developer Experience

Mapbox SDK does not require developers to incorporate some platform-native code, unlike Google Maps SDK. The primary concern for developers with Mapbox is the safeguarding of the Mapbox token, ensuring it remains confidential to the machine and is not committed to the repository. The Flutter SDK offers support for environment variables, a feature that can be leveraged to use the token during development and in production.

```
resourceOptions: ResourceOptions(
  accessToken:
    const String.fromEnvironment("PUBLIC_ACCESS_TOKEN"),
)
```

Listing 5. Accessing the token value from an environment variable.

```
$ flutter build ipa --dart-define PUBLIC_ACCESS_TOKEN=<redacted>
```

Listing 6. Passing the token to Flutter SDK when building the application for production.

Listings 5 and 6 demonstrate how a token could be provided to the Mapbox SDK as an environment variable, which helps to avoid having the token captured in code.

Like the Google Maps SDK, the Mapbox SDK necessitates developers to construct procedures for marker image generation. These images are supplied to the SDK as arrays using the `Uint8List`, consisting of unsigned 8-bit values.

The image rendering process involves a `Canvas` to generate an `Image` object. Subsequently, these `Image` objects are transformed into PNG byte data and converted to `Uint8List`s.

```
final images = await Future.wait(positions.map((e) =>
    getTransportMarkerImage(e, const Size(48, 48))
        .then((value) => value.toByteData(format: ImageByteFormat.png))
        .then((value) => value!.buffer.asUint8List())));
```

Listing 7. Processing marker images into a format accepted by the Mapbox SDK.

An example of an implementation is shown in Listing 7. The images are generated by the `getTransportMarkerImage`, then converted to `ByteData` which can be converted to a `Uint8List`.

The disadvantages of this method align with those of the Google Maps SDK, specifically regarding performance and code clarity. While the Mapbox SDK shows better performance than the Google Maps SDK, the image rendering impact remains evident. In terms of code structure, it follows Flutter's declarative UI building principles the least, requiring imperative calls to create markers and polylines.

The most recent version of the Mapbox SDK for Flutter, as of this writing, is version 0.4.5. This version displays a significant bug on iOS: when a marker undergoes an update, its image disappears (Jakubowski, 2023). This problem surfaced during the application development. The only known solution involves altering the library's native code to manage marker updates in an alternative manner. The existence of this bug and lack of a reaction from the package's developers to it may reflect the Mapbox developers' approach towards their user base, suggesting potential lack of future support.

In summary, while the Mapbox Mobile Maps SDK for Flutter does cover all requirements, it presents a subpar developer experience. Analogous to the Google Maps Flutter SDK, it neglects Flutter's widget-based declarative UI paradigm, choosing instead to replicate the imperative API of the native Mapbox SDK for iOS and Android which it encompasses. The documentation is lacking;

a developer must examine the examples folder to determine the API specifications.

5 Benchmarking and Comparison

For a quantitative evaluation of the three SDKs, a selection of metrics will be employed. While most of these metrics offer quantitative insights, a few are inherently subjective. The metrics will be as follows:

- Performance
  (a) FPS when moving around the UI
  (b) FPS when panning around with markers
  (c) FPS moving the bottom sheet, which results in map resizing
- App statistics
  (a) Bundle size
  (b) Startup time
  (c) Memory usage
- Pricing
- Subjective metrics
  (a) Community support
  (b) Developer Experience (DX)

The measurements will be performed using the following tools:

- Flutter SDK v3.13.7
- Android SDK v31.0.0
- Xcode v15.0
- Android Studio 2021.3
- iPhone 14 Pro (iOS 17.0.3)
- OnePlus 8 Pro (Android 13)

5.1 Performance

Performance is measured with tools provided by the Flutter SDK. When measuring performance, a real device should be used and the app should be run in profiling mode (Google, Flutter community, 2023). For all three scenarios, average FPS will be measured using the Performance page of Flutter Developer Tools.

5.1.1 FPS when moving around the UI

This metric should show how a particular map SDK will affect the performance of the whole application. All performance recordings will follow the same path through the user interface. The path involves searching and selecting trip start and end points, browsing trip results and inspecting the stops on the map.

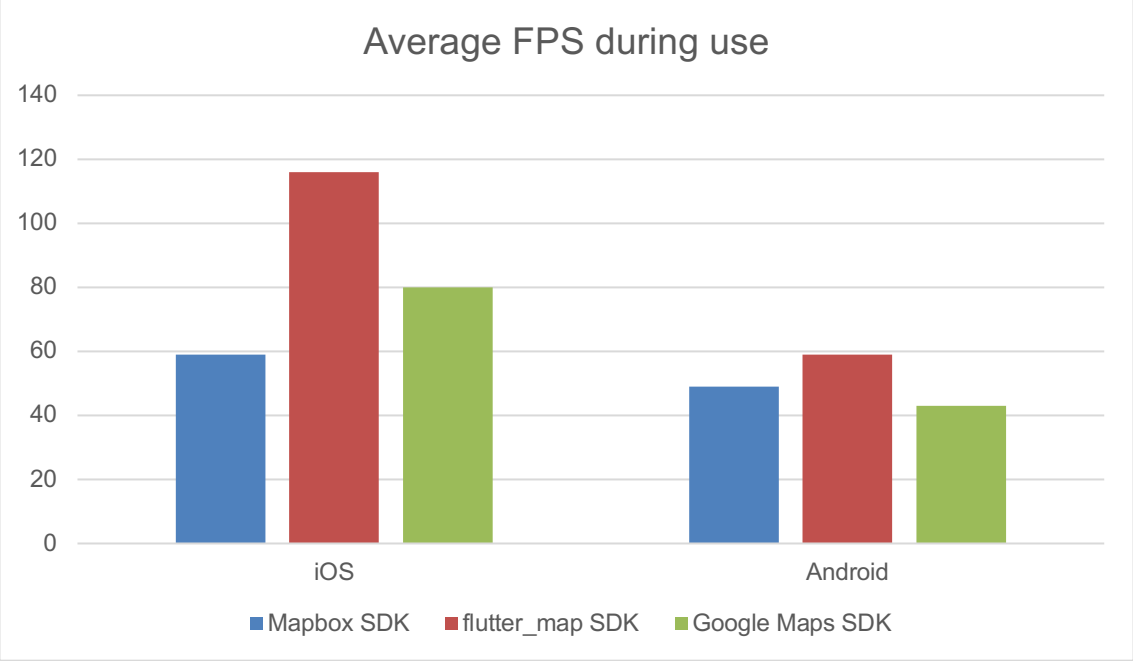

Figure 2. Average FPS of the app measured during use

Figure 2 clearly illustrates that flutter_map SDK outperforms in this specific metric, showing the smallest impact on the overall application performance. On iOS, the application operates nearly at the screen's maximum refresh rate, nearly reaching 120 frames per second (FPS). Conversely, on Android, it records a performance of 59 FPS, just marginally below the target performance level of 60 FPS, as discussed in Section 3.2.

5.1.2 FPS when panning around with markers

This measurement will be done while panning around the map with markers visible on the map. This metric will show how well a particular SDK can handle large amounts of markers displayed on the map. To minimize variables in the measurement, the application will be configured to set a specific zoom level on

the map and center on a specific coordinate on the map. The panning will be performed around the center coordinate for 10 seconds.

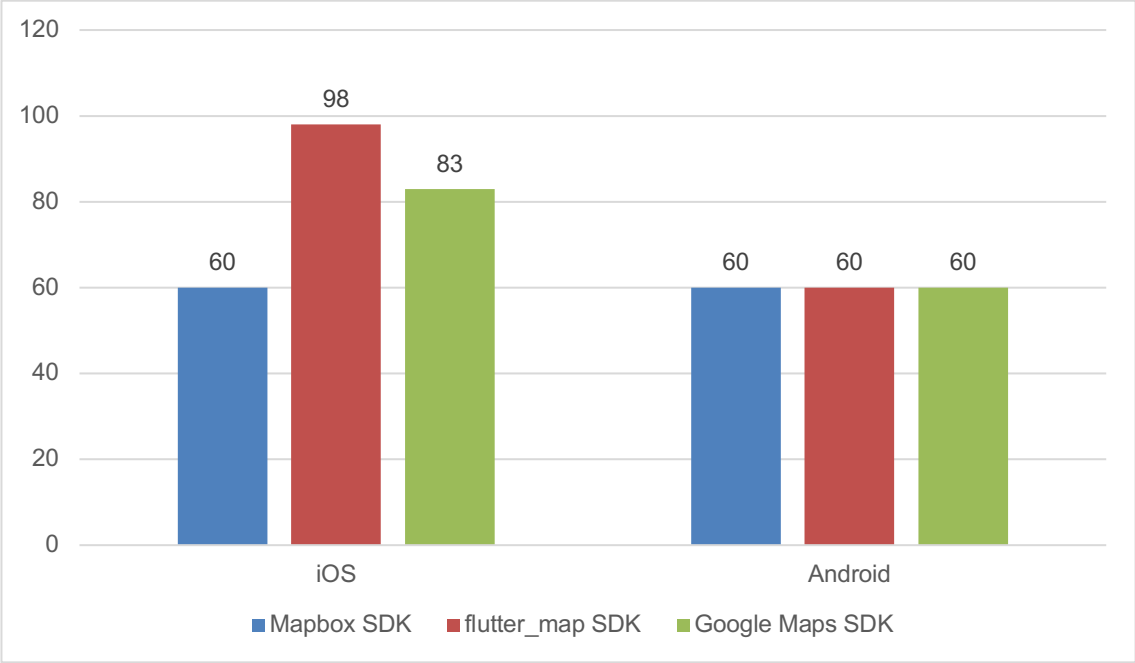

Figure 3. Average FPS measured during map interaction.

Evidenced by the comparison in Figure 3, flutter_map SDK shows the best performance on iOS and is equal to others on Android. All SDKs performed at maximum FPS on Android. Mapbox SDK does not reach above 60 FPS on iOS.

5.1.3 FPS when resizing the bottom sheet

A map displayed together with a resizable bottom sheet. Resizing the sheet will cause the map to resize, which might be a resource intensive process depending on the SDK.

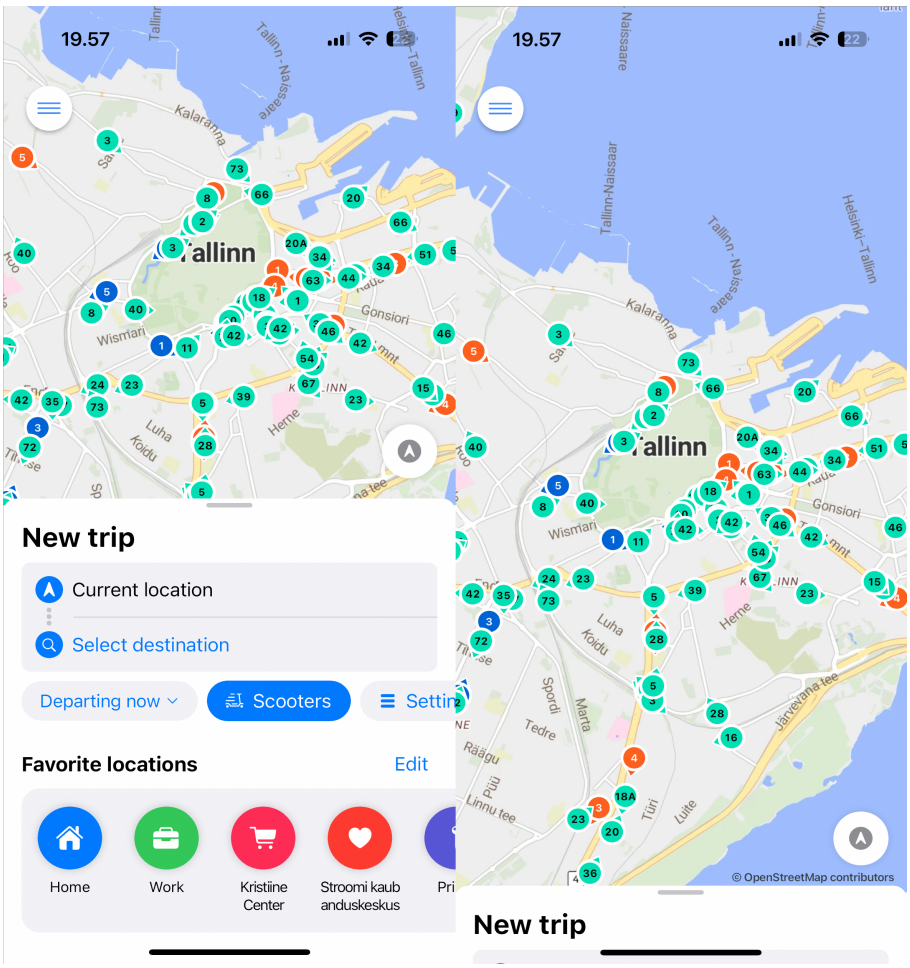

Figure 4. Bottom sheet resized to show the map

Figure 4 demonstrates that the bottom sheet can be resized to allow the map to occupy more space on the screen. It is important to note that the implementation of the resizing behavior differs depending on the SDK. flutter_map and Mapbox SDKs handle resizing of their root widget well, while Google Maps SDK causes the whole application to lag, making it unusable. A workaround is to have the root widget of Google Maps occupy the whole screen, while using the map padding API to resize the focus area of the map view. This achieves the same functional behavior as resizing the widget on other SDKs.

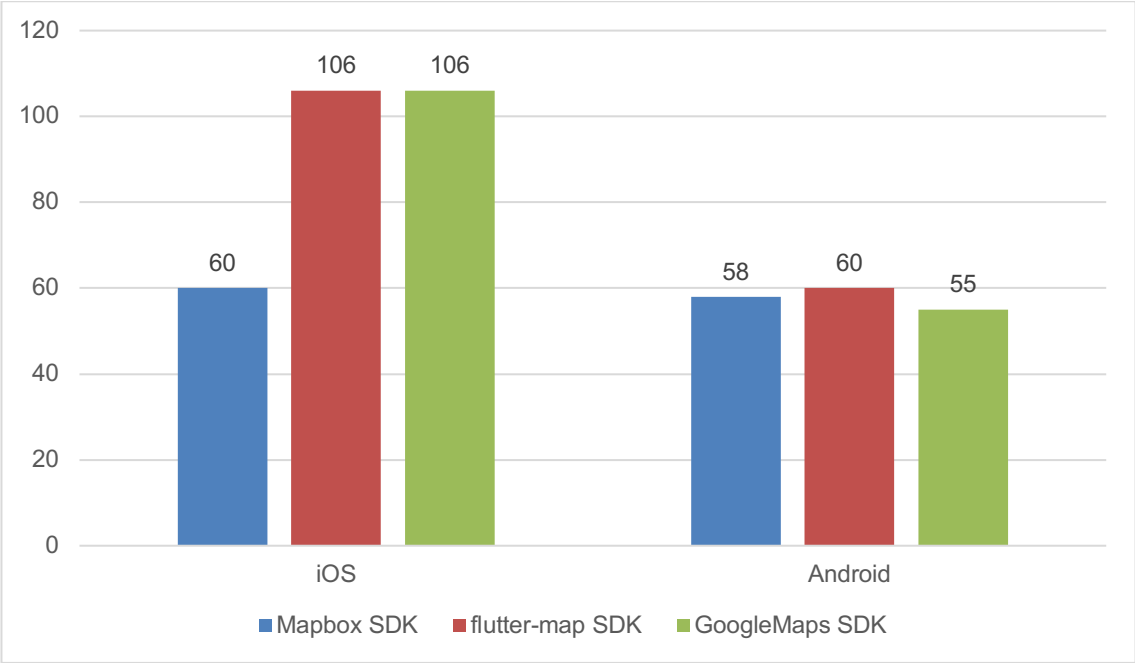

Figure 5. Average FPS when resizing the bottom sheet

Figure 5 demonstrates that Mapbox SDK on iOS is a clear outlier in terms of app performance when resizing the bottom sheet, showing worse performance on iOS compared to others. The other SDKs have demonstrated relatively equal performance.

5.2 App statistics

This section will present a comparison of quantitative metrics focused on the application parameters. The parameters selected for comparison are bundle size, startup time and memory usage.

5.2.1 Bundle size

Bundle size is measured by building an application into a container used to distribute the application on the specific platform. For iOS, the measured container will be the `.xcarchive` bundle. For Android, the compared container will be the `.apk` file.

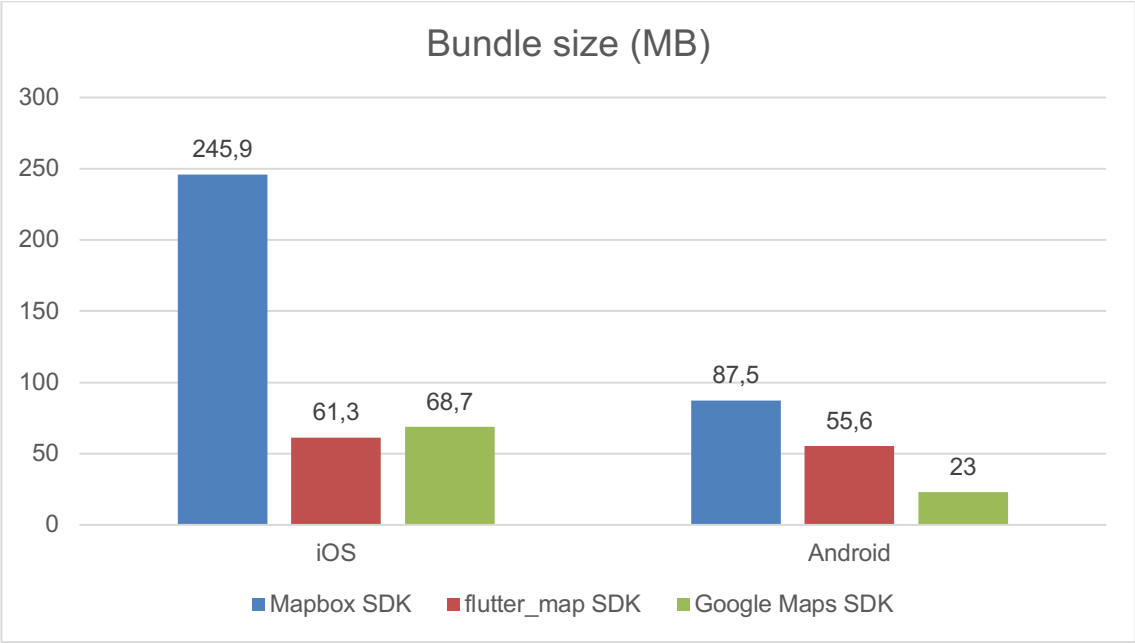

Figure 6. Bundle size comparison of the target app with the compared SDKs.

As shown by the graphs in Figure 6, the application utilizing the Mapbox SDK generates the largest binary sizes. In a comparison of Android builds, the binary using the Google Maps SDK is smaller relative to the binaries of the app built with the other two SDKs. This size difference may be attributed to the absence of Google Maps SDK code within the application bundle, with a reliance instead on the Google Maps services present in the Android OS. For iOS builds, the Mapbox SDK yields a bundle size that is approximately 350% larger than that of its nearest competitor, the Google Maps SDK. The underlying reasons for this size discrepancy are unknown.

5.2.2 Startup time

Startup time is measured by running the app in profile mode. The command used to run the application is as follows:

```
flutter run --trace-startup -profile
```

Running the command while having the target device selected will produce a JSON file containing four measurements (Google and individual collaborators, 2023):

- `engineEnterTimestampMicros`
  Time to enter the Flutter engine code.
- `timeToFirstFrameMicros`
  Time to render the first frame of the app.
- `timeToFrameworkInitMicros`
  Time to initialize the Flutter framework.
- `timeAfterFrameworkInitMicros`
  Time to complete the Flutter framework initialization.

The metrics are denoted in units of microseconds, as indicated by their names. The values will be converted to milliseconds before the comparison. The first metric, `engineEnterTimestampMicros`, poses several challenges. Its value's magnitude significantly exceeds other values, with 12 digit values recorded for iOS and 9 for Android. Additionally, the interpretation of this metric lacks clear explanations on the Flutter documentation and within community discourse. Notably, the metric's title suggests it signifies a timestamp, marking the precise moment an event occurred. This characteristic makes it inherently non-comparable with figures from other SDKs. Consequently, this metric will be excluded from the comparative analysis. Measurements of the other metrics are shown on the graphs in Figure 7 and Figure 8.

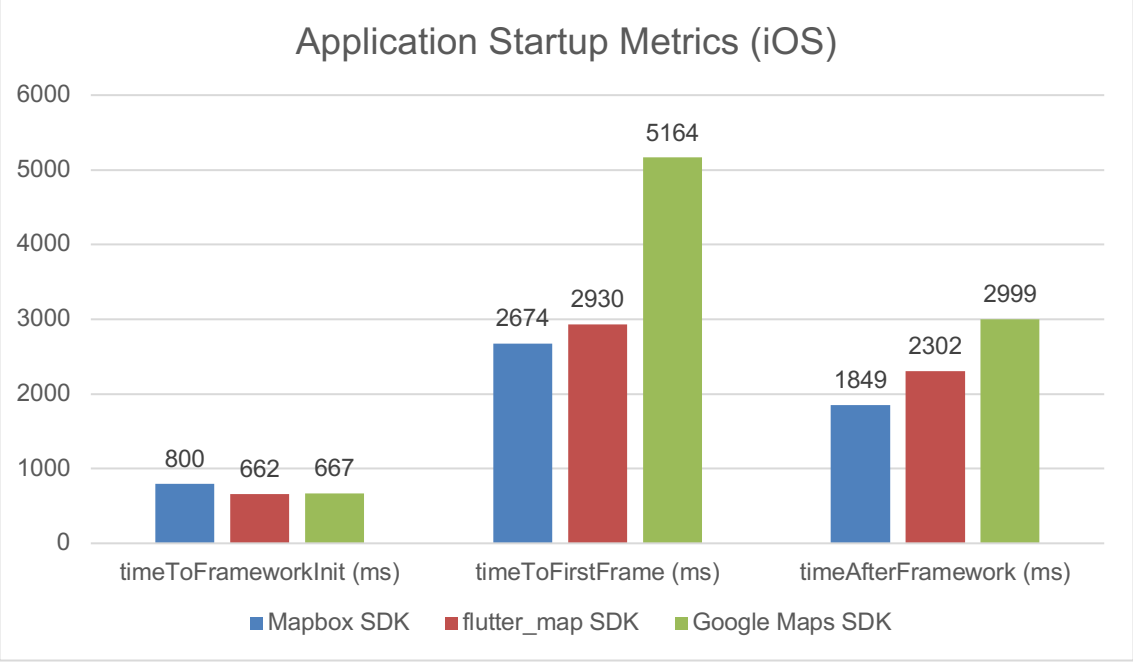

Figure 7. Comparison of three application startup metrics captured on iOS

The results measured on iOS, depicted in Figure 7, reflect the subjective experience of using the application with the SDKs. While the application with Mapbox SDK is the slowest to enter the Flutter framework, it is the fastest to render the first frame and to invoke the framework callback. The application built with Google Maps SDK is significantly slower to launch compared to others. Using flutter_map SDK results in launch times close to those of Mapbox SDK.

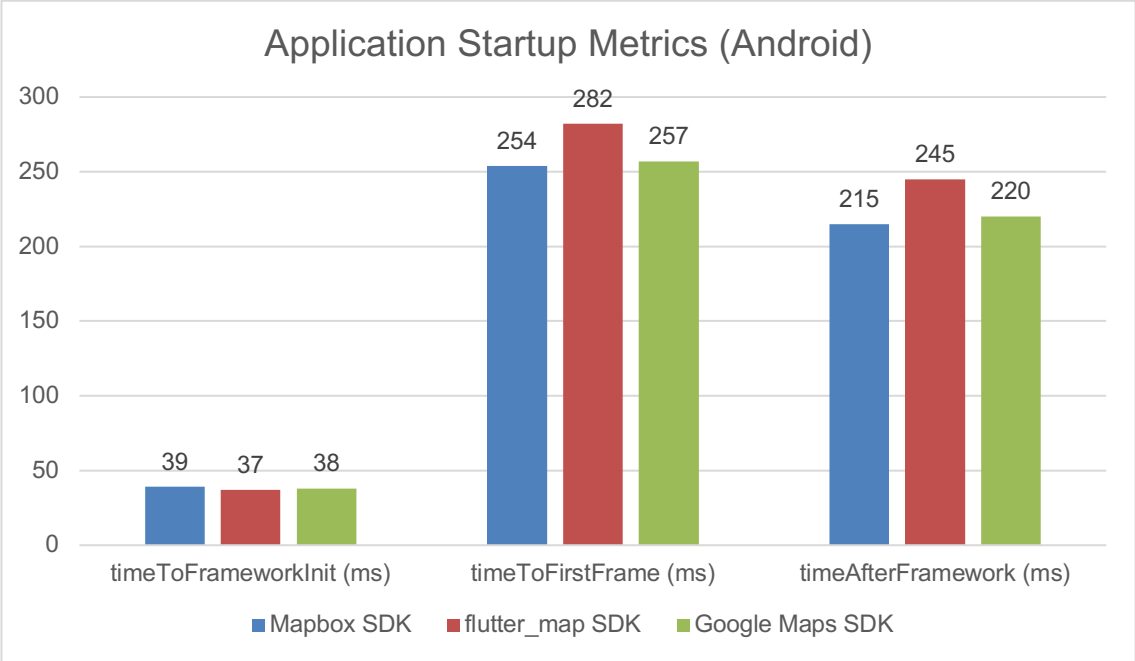

Figure 8. Comparison of three application startup metrics captured on Android

As seen in Figure 8, flutter_map SDK is the slowest to launch, while Google Maps and Mapbox SDKs are almost equal. It is important to note that the margin between the slowest and the fastest launch time is not perceptible to a human; hence, the launch times can be considered equal.

To summarize, Mapbox SDK shows the lowest startup times, flutter_map SDK coming second and Google Maps SDK coming third, due to the poor performance on iOS.

5.2.3 Memory usage

The importance of optimizing for low memory usage is discussed in Section 3.2. Memory usage can be measured using the tools provided by the platform developers, namely Xcode for iOS by Apple and Android Studio for Android by Google and Jetbrains. To simulate the worst-case scenario, memory usage levels were measured while going through the same path through the UI as when measuring FPS, described in Section 5.1.1.

On iOS, memory usage was measured using the Allocations section of the Instruments application, which is part of the Xcode package. The category named “All Heap and Anonymous VM” will be used. On Android, the application will be built for profiling with low overhead, and an average of the timeline of measurements will be taken. The results of the measurement can be seen in Figure 9.

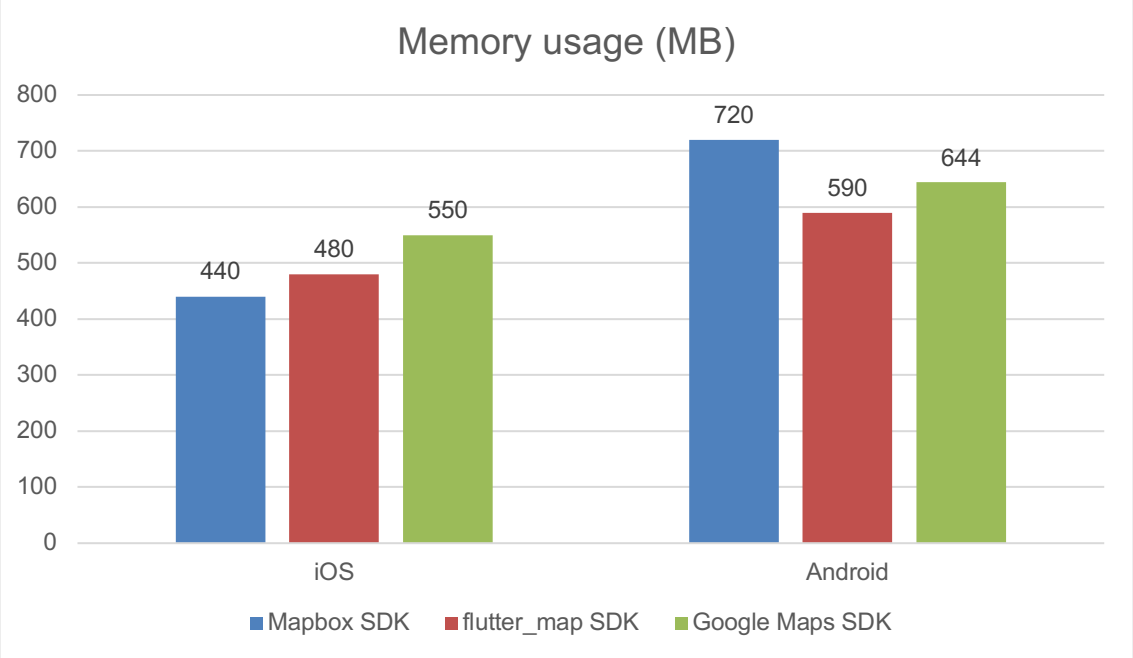

Figure 9. Memory usage of the app for both Android and iOS

The results of the measurement are mixed, not showing an obvious leader among the compared. Figure 9 shows that the app built with Mapbox SDK will have drastically different memory consumption depending on the platform, while

other SDKs behave comparably. flutter_map SDK is shown to be second on iOS and the first on Android, making it the best choice when it comes to minimizing memory consumption.

5.3 Pricing

Comparing SDK pricing is made complicated by Google's vague pricing structure for Google Maps SDK, described in Section 4.1.2. To perform the comparison, a common ground in pricing had to be found. According to the application's internal analytics, the amount of monthly active users of the current application is approximately 3000. These users generate an approximate amount of 7 million tile requests. These requests are served by a single machine costing 40 USD per month. Cloudfront CDN is serving as a cache for free at the current rate of usage. These values can be used to approximate the monthly cost of the usage of the SDKs.

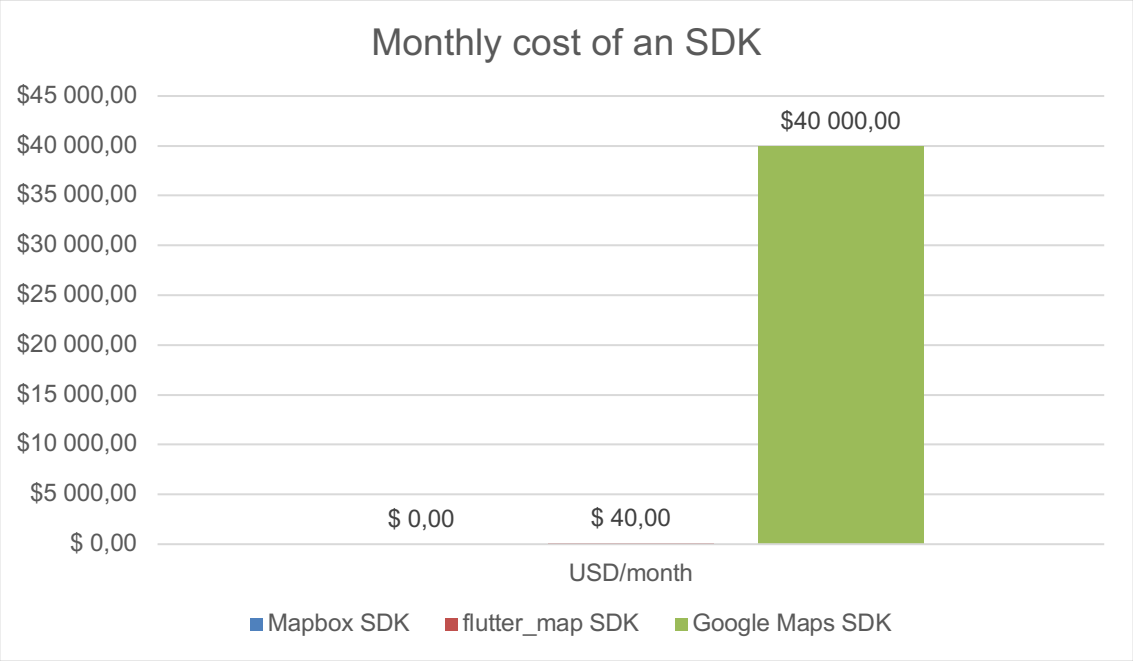

Figure 10. Approximate monthly cost of the SDK at 3000 monthly active users

A very rough approximation of the projected costs of every framework can be seen in Figure 10. As noted before, these figures are highly speculative.

Moreover, Google Maps SDK can be free to use on Android, provided that the requirements for that are followed. Nevertheless, the graph reflects the general situation: Google Maps SDK can be too expensive for a small enterprise to afford, while flutter_map SDK and Mapbox SDKs are affordable at the current usage rate.

Projecting the costs further, an assumption can be made that the flutter_map SDK will be cheaper to use if the user count grows exponentially. With the tested setup, due to the caching setup with Cloudfront, increasing user count drastically should not result in comparable increase in load and therefore cost. Most of the requests will be handled by Cloudfront CDN, reducing the impact of increase in usage.

5.4 Subjective metrics

This section will include metrics that are derived not from empirical measurements, but from subjective assessments. The SDKs will be ranked in a hierarchical order of first, second, or third place based on these evaluations.

5.4.1 Documentation quality

Quantifying documentation quality is nearly impossible. The SDKs are ranked from best to wo rst, 1 being the best and 3 being the worst. The ranking is based on the subjective experience of solving integration issues with the help of the documentation. The SDKs ranked on documentation quality:

1. flutter_map SDK
2. Mapbox SDK
3. Google Maps SDK

flutter_map SDK was found to have the best documentation of the three. The documentation consists of a website, as well as some code examples that are stored alongside with the source code in Github. The website delves deep into the available API, as well as available plugins for the SDK. Moreover, the website recommends solutions to the tiling issue described in Section 4.2.

Overall, it was found that the website and the examples were enough to find solutions to most of the issues, not necessitating the developer to seek for answers online.

The Mapbox SDK documentation is not as exemplary as the documentation of flutter_map SDK. It provides documentation in form of a README.md file, as well as code examples, stored alongside the source code. These sources explain the API of the package. It proved to be enough to get the integration started, but it also proved of little use when integration issues arose. Therefore, it is placed second in the comparison.

Google Map SDK for Flutter is poorly documented. The documentation consists of a set up guide, a step-by-step lesson on how to set the SDK up in Google's Codelabs, and finally some code examples. The examples are very sparse, consisting of only two files. This resulted in the developer using community-powered resources like StackOverflow, as well as guessing the API. The condition of the documentation is surprising, considering that Google Maps SDK for Flutter is an official Google package, and Flutter is an app development platform developed by Google. These factors result in the third place in the comparison.

5.4.2 Developer Experience (DX)

Developer experience cannot be measured but is useful in the comparison. The judgement is made from the experience of integrating the SDK into the application. It consists of multiple aspects mentioned in Section 3.4 on the topic. The ranking is as follows:

1. flutter_map SDK
2. Google Maps SDK
3. Mapbox SDK

flutter_map provides the best development experience of the listed SDKs. The specifics of the developer experience were listed in Section 4.2.3. The biggest contributor to the level of DX is ability to define marker visuals with Flutter

widgets, avoiding the image conversion complexity. The availability of documentation and package popularity contribute to the positive ranking. The package gets consistent updates that alleviate issues and add new features. All the listed factors result in the SDK ranking first.

6 Conclusion

The aim of this study was to pick out the best map SDK for a navigation application, where most of the functionality is centered around a map. The application in question is a Flutter application, released on two major mobile platforms: iOS and Android. Three SDKs were selected for the comparison, namely Google Maps SDK, Mapbox SDK and flutter_map.

The SDKs were compared based on their performance, resource usage, feature set and DX. Google Maps SDK, released as `google_maps_flutter` package on the Flutter platform fulfils all of the feature set criteria. Upon closer inspection, key issues were found in the SDK, namely poor performance and exorbitant pricing. Mapbox SDK, packaged into the `mapbox_maps_flutter` package, also fulfils the criteria, but it proved to have poor DX and lacking documentation. Moreover, it had critical bugs, making the package unusable without having to resort to making modifications of the source code of the package. `flutter_maps` SDK, the only package that is built with Flutter exclusively, fulfils the criteria, but does not offer built-in tiles and does not offer first-party support for vector tiles. None of the choices presented are an obvious pick.

Two notable patterns emerge from the analysis. Firstly, the application developed using the Mapbox SDK on iOS did not surpass 60 FPS. Although not explicitly documented in developer resources, this behavior appears to be an artificial limitation imposed by the SDK. Secondly, the performance measurements for all SDKs on Android did not exceed 60 FPS. This limitation is particularly observed in the OnePlus testing device. The underlying cause is attributed to the Flutter SDK's lack of signaling to the operating system that it can operate in a higher performance mode (Zoeyfan, 2020).

Shown by Figure 2, Figure 3 and Figure 6, flutter_map SDK exhibits superior performance across all platforms, which aligns with the subjective user experience during application usage. During testing phases, native SDKs exhibited noticeable stuttering. While this stuttering does not significantly impact

the average frame rate, it profoundly affects the user experience. It should be noted, however, that quantitatively measuring such experienced stuttering is a complex task and falls outside the scope of this study.

Along with superior performance, flutter_map SDK provided the best experience of using the SDK. It proved to be a good citizen of the platform, following Flutter's approach to declarative UI building. It supports Flutter widgets without image conversion. Finally, it offers good interfaces to control the SDK. It is being actively developed; a few major and multiple minor versions were released during the development of the application. The documentation and code examples are superior to the other SDKs, and community support is on par with Google Maps SDK, which makes it a great candidate to be picked for further development.

There are multiple benefits that come from flutter_map SDK being an open-source package, developed by a community of developers. Firstly, this allows developers of the applications to fix any bugs they encounter via the contribution process of pull requests. Secondly, it provides transparency and the opportunity for contributors to influence the direction of the SDK's development. Thirdly, the community-driven approach fosters an ecosystem of plugins and extensions, ensuring that the SDK evolves in response to real-world use cases and developer needs. All these factors are of great importance when choosing a fundamental dependency in an application, where most of the functionality is centered around it. Since other SDKs function as a compatibility layer that bridge Flutter with proprietary, closed-source native SDKs, they are unable to offer comparable advantages.

flutter_map SDK includes first-party support for plugins and extensions. The SDK's documentation provides several examples, accompanied by guidelines on how to augment the SDK's capabilities. A number of these plugins are under active development, with one notable example being `vector_map_tiles`. This plugin is poised to introduce the advantages of vector tiles to the flutter_map SDK. Other plugins implement a user location icon / user location icons, or offer improved performance for markers, among others.

As mentioned in Section 4.2, flutter_map SDK requires developers to provide a map tile source, which stands as a significant drawback compared to other SDKs that offer built-in tile rendering capabilities. The choice of tile source, whether it is a paid service or another method, often represents the primary cost factor associated with the use of this SDK. While Google Maps SDK's projected costs make it not relevant for comparison, Mapbox SDK remains free until 25,000 monthly active users (MAU) are exceeded, after which the expenses escalate substantially. Nevertheless, opting for self-hosted tiles presents the most transparent and effective means of managing costs.

The biggest functional drawback of flutter_map SDK is that it does not support vector map tile rendering out of the box. As previously mentioned, a plugin exists offering this functionality, but it does so at a major cost to performance, reducing it by around half, when measured in FPS. Vector tiles offer many advantages, which are listed below:

1. **Scalability**: Vector tiles maintain high quality at any scale or zoom level, avoiding the pixelation issues common with raster tiles.

2. **Smaller File Size**: Vector tiles are typically more compact and have smaller file sizes, leading to faster loading times and less bandwidth usage.

3. **Dynamic Label Placement**: With vector tiles, map labels can be dynamically placed and oriented, ensuring they are always readable and correctly aligned, regardless of the map's rotation or zoom level.

Mapbox and Google Maps SDKs both render vector tiles; hence, they display all the listed benefits.

Based on the study described in this thesis, the flutter_map SDK was selected for continued application development. This decision was influenced by its transparent cost structure, Flutter-native design, and robust community support, positioning it as an optimal choice. In comparison, the Google Maps SDK

showed to be prohibitively expensive while showing most inferior performance. The Mapbox SDK, on the other hand, was characterized by a less favorable DX and an ambiguous pricing model. The flutter_map SDK, demonstrating reliability and a promising future of community-driven improvements, was thus chosen as the preferred map SDK for the application.

References

Apple, 2012. *iOS Drawing Concepts.* [Online]
Available at:
https://developer.apple.com/library/archive/documentation/2DDrawing/Conceptual/DrawingPrintingiOS/GraphicsDrawingOverview/GraphicsDrawingOverview.html
[Accessed 18 June 2023].

Apple, 2023. *Reducing Your App's Memory Use.* [Online]
Available at:
https://developer.apple.com/documentation/metrickit/improving_your_app_s_performance/reducing_your_app_s_memory_use#
[Accessed 6 December 2023].

Daliot, A., 2015. *Functional Animation In UX Design.* [Online]
Available at: https://www.smashingmagazine.com/2015/05/functional-ux-design-animations/#orientation
[Accessed 4 June 2023].

DNA, 2022. *Do You Use a Smartphone?.* [Online]
Available at: https://www.statista.com/statistics/564643/share-of-smartphone-users-in-finland-by-age-group/
[Accessed 3 June 2023].

Fessenden, T., 2022. *Three Pillars of User Delight.* [Online]
Available at: https://www.nngroup.com/articles/pillars-user-delight/
[Accessed 7 June 2023].

flutter_map Authors & Maintainers, 2023. *flutter_map Docs.* [Online]
Available at: https://docs.fleaflet.dev
[Accessed 19 July 2023].

Google and individual collaborators, 2023. *Debugging Flutter Apps.* [Online]
Available at: https://docs.flutter.dev/testing/debugging#measuring-app-startup-

time
[Accessed 17 10 2023].

Google, 2023. *Flutter - Build Apps For Any Screen.* [Online]
Available at: https://flutter.dev
[Accessed 18 June 2023].

Google, 2023. *Flutter Architectural Overwiew.* [Online]
Available at: https://docs.flutter.dev/resources/architectural-overview
[Accessed 18 June 2023].

Google, 2023. *Introduction to Widgets.* [Online]
Available at: https://docs.flutter.dev/ui/widgets-intro
[Accessed 18 June 2023].

Google, 2023. *Multi-Platform.* [Online]
Available at: https://flutter.dev/multi-platform
[Accessed 3 June 2023].

Google, Flutter community, 2023. *Flutter Performance Profiling.* [Online]
Available at: https://docs.flutter.dev/perf/ui-performance
[Accessed 28 November 2023].

Jakubowski, P., 2023. *iOS - Image Disappears When Updating Point Annotation · Issue #241 · mapbox/mapbox-maps-flutter · GitHub.* [Online]
Available at: https://github.com/mapbox/mapbox-maps-flutter/issues/241
[Accessed 10 October 2023].

K. Debattista, K. B. S. S. T. B.-R. V. H., 2018. Frame Rate vs Resolution: A Subjective Evaluation of Spatiotemporal Perceived Quality Under Varying Computational Budgets. *Computer Graphics Forum,* 37(1), pp. 363-374.

Mapbox, 2023. *Mapbox Pricing.* [Online]
Available at: https://www.mapbox.com/pricing
[Accessed 19 July 2023].

Meta, 2023. *Render, Commit and Mount.* [Online]
Available at: https://reactnative.dev/architecture/render-pipeline
[Accessed 18 June 2023].

Stadia Maps, 2023. *Pricing & Tiers.* [Online]
Available at: https://stadiamaps.com/pricing/
[Accessed 19 July 2023].

Statcounter, 2023. *Mobile Operating System Market Share Europe.* [Online]
Available at: https://gs.statcounter.com/os-market-share/mobile/europe/#yearly-2022-2023-bar
[Accessed 3 June 2023].

zoeyfan, 2020. *Frame Rate is Locked to 60FPS On Devices With Frame Rate Optimization · Issue #35162 · flutter/flutter.* [Online]
Available at: https://github.com/flutter/flutter/issues/35162
[Accessed 3 December 2023].