# Using layers for policy analysis: Smartphone apps

## Introduction

This module builds on Framework: Tech, layers and (un)bundling. You find this module here: https://docs.google.com/document/d/1-9hXabsoL94MeRi3D60r6CpUc62y3oe2dmkH9FSEVbI/edit#

Other parts in this series include 5G Technologies: (https://docs.google.com/document/d/1tO2HGoGjxIO6vx5hHhl_o9cIoI3C_sN6yh0zYITlibs/edit#) and Net neutrality: (https://docs.google.com/document/d/1CUr-h5WayWRWuUEIKqkqF9UMEuUFw1G9BkcUzIcoQnc/edit).

A companion glossary is available here:

https://docs.google.com/document/d/1fxsbRxBYkSzh0stmc9liXTljqYIpQ7fdAH_rC31-zJI/edit?usp=sharing

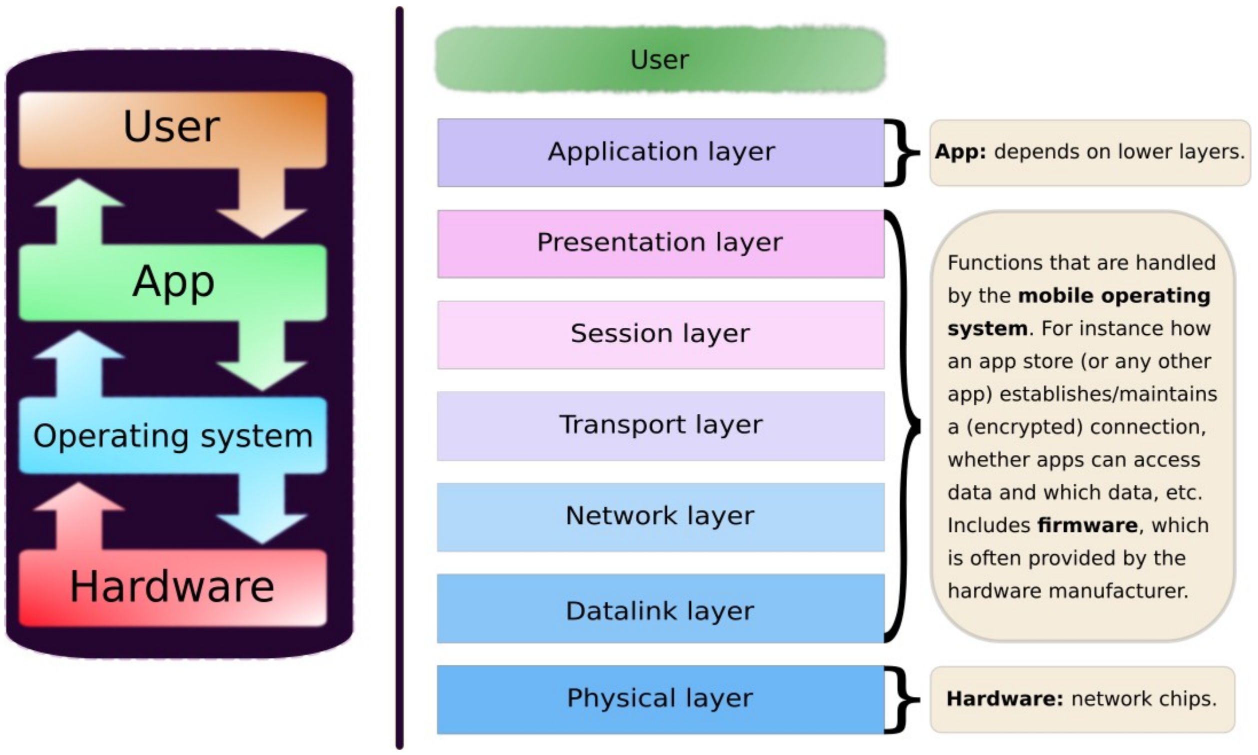
## Smartphone apps

The vast majority of apps today are bought through stores. For mobile apps, there is currently a duopoly: App Store, on devices that use iOS as the operating system, and Google Play, for devices that use the Android operating system.

Apps are developed by a myriad of independent developers, as well as by Google and Apple - Google and Apple are also developing the mobile operating systems used by our devices and own the app stores running on them.

### Extending the model

Can the OSI reference model give us a clue about what is going on? Only in part. Let’s look at two representations of a mobile eco-system:


User
App
Operating system
Hardware
User
Application layer
App: depends on lower layers.
Presentation layer
Session layer
Transport layer
Network layer
Datalink layer
Functions that are handled by the mobile operating system. For instance how an app store (or any other app) establishes/maintains a (encrypted) connection, whether apps can access data and which data, etc. Includes firmware, which is often provided by the hardware manufacturer.
Physical layer
Hardware: network chips.

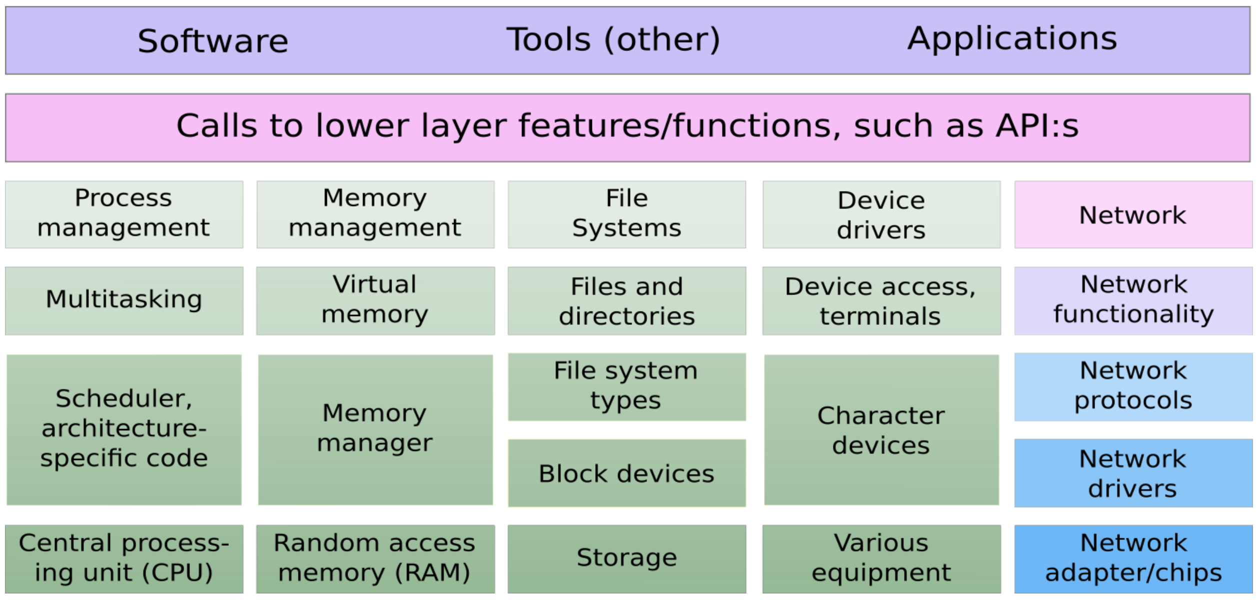
To the left, we have a value chain which broadly corresponds to the experience of a smartphone user. The hardware is easy to distinguish from the software because it is, well, hard. The mobile operating system (OS) deals with configurations, or firmwares,[1] for the different hardware parts and communication between, say, an app and the built-in camera. Apps run like stand-alone programs.

To the right, we have an OSI representation of networking functions. Most of the layers are in fact occupied by the mobile operating system - it is responsible for coordinating all the networking functions and ensuring that apps can access the networking features that they need. It is an incomplete picture, since in fact mobile operating systems do a whole lot more than just managing interconnection features.

**Example S.1:** Look back at examples O.5 and O.6 in the Framework module. If new features for communications are not standardized, it will be more difficult for apps to make use of them (Example O.5), and the downwards consolidation looks intuitive in relation to the mobile setting (Example O.6).

A mobile operating system is *not fully described by the OSI model*. OSI was developed to describe interconnected systems, not computing technologies in general. Instead, an operating system might look like this:[2]

1See Annex: Glossary.
2Adapted from the following representation of a Linux operating system: http://www.linux-india.org/characteristics-and-architecture-of-linux-oprating-system/



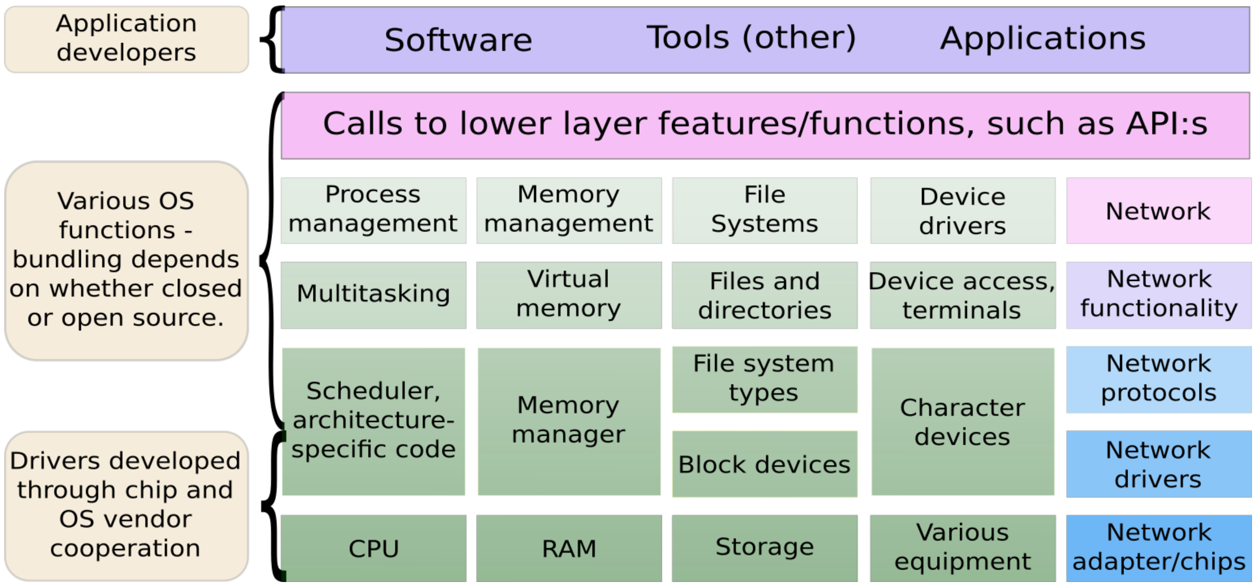
Networking features (blue-scale) are only one of many functions that a mobile operating system needs to perform (green-scale). In an operating system, the application layer (purple) is additionally full of web browsers, document viewers, games, and other software that may or may not be related to connectivity. For apps, the OSI model inspires a method of *reasoning about layers of technical infrastructure*, which is easy enough to support heuristically through already existing pictures available just an internet search away.

**A perspective of layers lets us ask** whether all functions that are currently performed by the mobile operating system have to be coded and controlled by the mobile operating system as such, or if they could be performed by a  third-party. It gives us a way of describing relationships between operating system vendors and chip vendors, and even between different software vendors. In short, it gives us a way of asking questions about barriers to entry, vertical and horizontal integration and the possibility of technical (un)bundling.
These elements are not human rights neutral; on the contrary, they can strongly affect users' rights. Barriers to entry impede competition, eliminate incentives for operators to innovate and/or to provide better quality services, as well as dramatically reducing users' choice. Vertical and horizontal integration can also reduce users' choice, and renders competition, innovation and market entry at different layers more difficult. By contrast, technical unbundling can empower users allowing them to select the manufacturer or provider they prefer (for example, technical unbundling allows users to decide which browser to use irrespective of the operating system they use).

Unlike in the OSI model, where most of these questions arise vertically, in a mobile operating system space they arise *horizontally* too. An integrated chipset consisting of network adapter and graphical and central process units (layer 1) spreads out horizontally, for instance. Various OS functions of an open source Linux operating system (layer 2-4) consist of a features developed by different people and companies. These features are enabled by *software libraries*, and they be combined into more complex applications or tools.

**Example S.2:** Mobile operating systems such as Android[3] and SailfishOS[4] are based on Linux, but contain additional closed-source features developed by the companies Google and Jolla. Apple iOS[5] and Blackberry 10[6] are examples of mobile operating systems that are not based on Linux. They contain a larger set of closed-set features on layers 2-4.

**Example S.3:** The MicroG project provides open-source alternatives to Google's closed-source Cloud to Device Messaging and Geolocation API:s[7] for accessing features in layers 1-5, with the aim of making app developers in layer 7 less dependent on Google.[8]



**Example S.4:** The strong need for compatibility between layer 1-2 and the higher layers makes it difficult for competing operating systems to arise. A competitor to a layer 2-4 vendor who does not have strong relationships with layer 1 entities to produce layer 2 firmware, in practice runs the risk of not being able to make the device work well for the end-consumer. Without strong relations between chip vendors and operating system vendors, there is a risk that hardware will not be activated, or will work more inefficiently than expected, inconsistently or without the ability to activate the full promised set of features.

**Example S.5:** The previous two diagrams contain many coloured boxes. Between each of the boxes there can be dependencies and relations. Keeping these dependencies and relations secure, both in the operational and the information security sense, can be challenging. Recall for instance the Heartbleed bug in the world's most commonly used encryption library OpenSSL, which caused calls to memory functions (second green column from the left) to work in a way that was not desirable.[9]

Broadly, we can make the following distinctions:

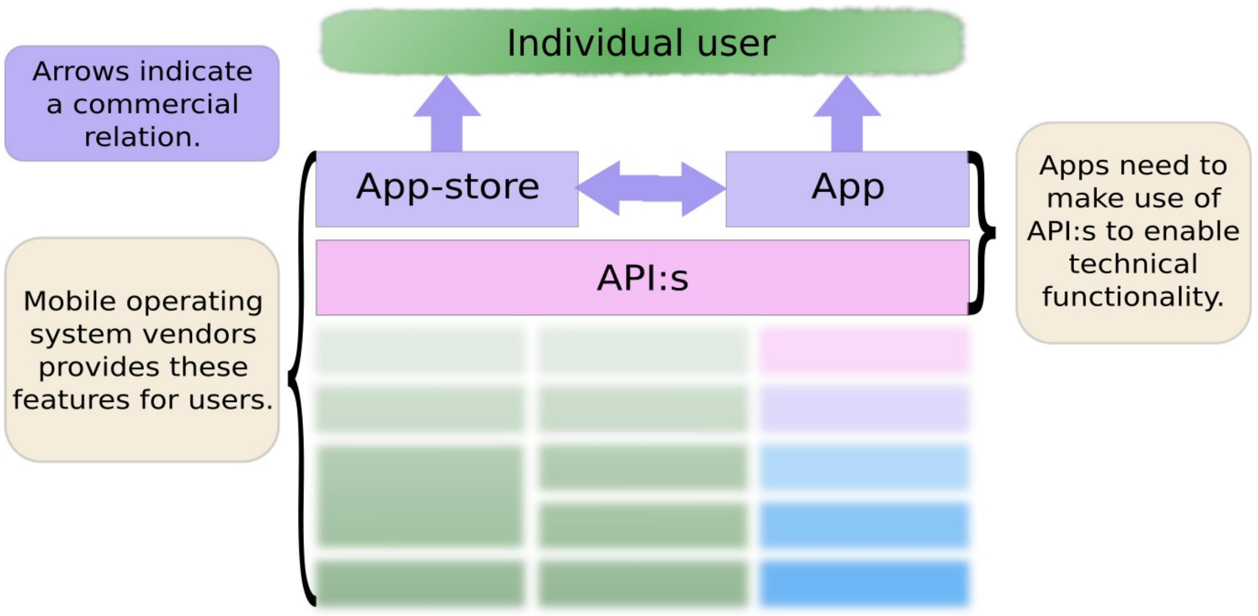
3https://developer.android.com/guide/components/fundamentals.html
4https://sailfishos.org/wiki/Architecture
5https://en.wikipedia.org/wiki/IOS
6https://en.wikipedia.org/wiki/BlackBerry_10
7See Annex: Glossary.
8https://microg.org/
9https://en.wikipedia.org/wiki/Heartbleed

| Application software (App) | Operating system |
|---|---|
| Software consisting of features that perform some task | Manages the software and the hardware parts of the system |
| Downloaded from the internet to be installed on the device or, in some cases, is pre-installed. | Almost always comes pre-installed on the device (see Example S.2) |
| Google Maps, iMessenger | Android, iOS |
| Depends on the operating system | Does not depend on the app |

**An app-store is a central part** of modern mobile operating systems, but it is in fact just an app ("layer 7"). App-stores are the primary vehicles through which consumers install more apps. It is possible to have many different app-store apps installed on the same operating system, but major mobile operating systems design their own app-stores and pre-install them on their operating system before the consumers buy the device. Most consumers will, in practice, use only the pre-installed app-store to get new apps.

We say that vendors *bundle* the app-store with the operating system. Occupying the roles of operating system vendor and app-store vendor gives these companies *leverage*.

App-stores manage requirements for those who want to provide apps to end-consumers, and can be called *two-sided markets*, meaning that they serve both end-consumers and other consumers. App providers need to make their apps accessible to consumers, and most consumers use the pre-installed app-store, so app providers need to fulfill the requirements set by the app-store.

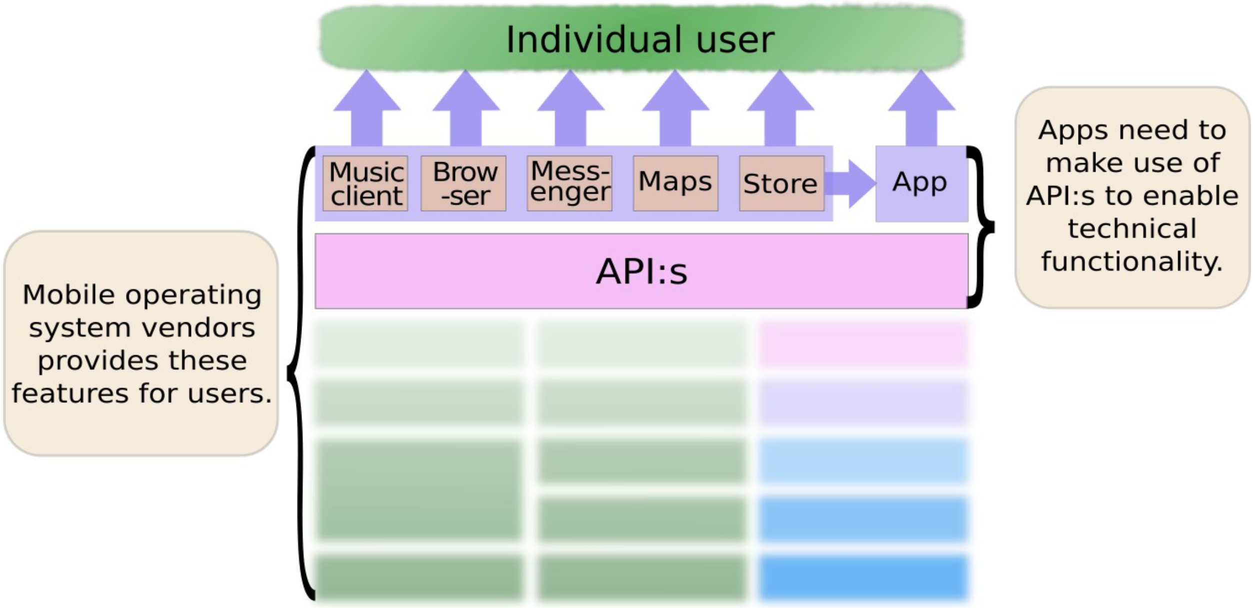

Arrows indicate a commercial relation.
Individual user
App-store
App
API:s
Apps need to make use of API:s to enable technical functionality.
Mobile operating system vendors provides these features for users.

**Requirements come in two** broad categories: technical requirements and contractual requirements. They can be upheld in two different ways: by automated review or non-automated review.

An app-store may, for instance, require that apps can only be installed using installation features provided for by the operating system. We consider this to be a technical requirement since it dictates a feature that depends on technical functions (green part of the picture). Requirements may also be that apps only contain non-discriminatory imagery. We consider this a contractual requirement since the determination of what is discriminatory does not depend on technical functions (it relates to content).

**Example S.6:** Apple decides on the requirements for apps that can be sold or found on its Apple App Store[10] and Google decides on the policies for its Google Play Store.[11] The requirements may be spread out over several legal documents: terms and conditions, policies, feature-specific guidelines and so forth. They may be technical or non-technical in nature. As app developers need their app to be on the pre-installed app-store in order to reach the majority of consumers, they are in a position of weakness towards the mobile operating system vendor.

Operating system vendors also develop other apps, such as browsers, map viewers, music players and messenger clients. Vendors pre-install them on devices running their operating system and leave it up to end-consumers to search for competing alternatives. Nevertheless, having apps pre-installed by default on a device strongly impacts end consumers' choice architecture, and discourages them from looking for alternatives[12].

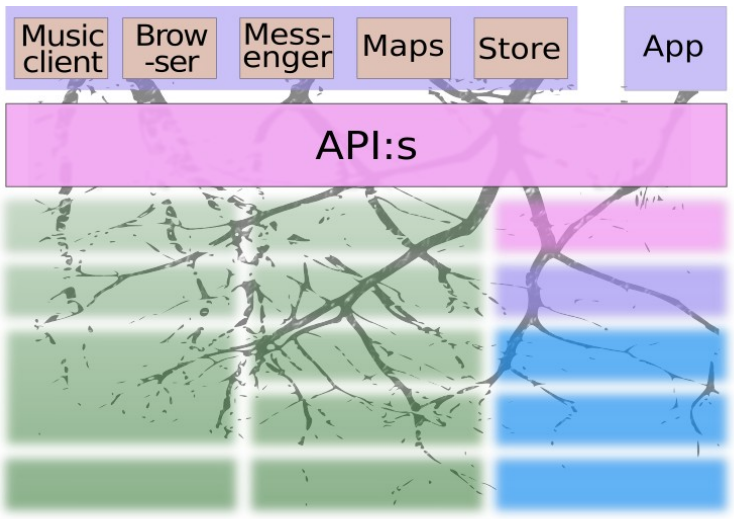

Individual user
Music client
Brow -ser
Mess- enger
Maps
Store
App
API:s
Mobile operating system vendors provides these features for users.
Apps need to make use of API:s to enable technical functionality.

10https://developer.apple.com/app-store/review/guidelines/
11https://play.google.com/intl/en-US/about/developer-content-policy/
12The negative impact of default settings on consumers has been analysed from various perspectives, including from the perspective of data protection rules. See, among others: https://fil.forbrukerradet.no/wp-content/uploads/2018/06/2018-06-27-deceived-by-design-final.pdf

## What do the API:s do?

Application Programming Interfaces (API) are interfaces to some software library, database or hardware function that are meant to make it easier for an app developer to use that library, database or function. In Example S.3, the Maps API enables an application to look up OpenStreetMap[13] data, such as data about locations or the maps containing the locations. It might contain functions such as disableMyLocation() which make it easier for an app developer to help the end-user stop location updates than if the programmer had to code the function as a direct interaction between the GPS hardware and the application.

**Example S.7:** An API is, essentially, a layer of abstraction that hides the real complexity of performing a specific task from the app developer. It removes some control from the app developer over what is actually happening when a feature is activated. This may be advantageous. In especially tricky security situations, for instance mobile payment, having one well-audited solution which can be called through an API is better than having lots of different app developers implement, perhaps with bugs and security errors, their own solutions to mobile payment.



API:s can do anything and everything: they can make it easier to interact with hardware, or make it easier to use security features, or make it easier to use any other feature. API:s are not necessarily supplied by a mobile operating system vendor (as demonstrated by Example S.3) but in practice this is almost always the case. In particular, if the mobile operating system is based on closed-source software it may not be possible for anyone other than the vendor to interface smoothly with the various operating system functions that interact with the hardware (cf. Example S.4). This is why we can say that API:s operate as gatekeepers.

13http://openstreetmap.org

**Example S.8:** Because API:s are so important for app developers, developers are also influenced a lot when API:s are updated, patched or replaced. If an app is programmed to make use of a specific API function but this function is withdrawn by the API provider, the app will no longer work.

**API:s feed relevant functionality** from lower layers into higher layers. In spite of API:s being very common and crucial to a functioning mobile operating system environment, there is not much API standardization. It is not clear that mobile operating systems compete with each other on API design and implementation - app developers need to use the API:s accessible on each platform, since end-consumers will not choose apps using API design as a selection criterion.

## Case Study (i): App fees collection by mobile OS provider

Spotify, like any other app developer, needs access to the Apple Appstore to reach the majority of potential consumers. Spotify  offers a service in direct competition with one offered by Apple itself, namely, Apple Music. Apple’s Terms of Service oblige Spotify to use Apple’s payment system for signing consumers up to subscriptions to its premium service, and asks for a 30% fee on payments made through that payment system.

Spotify complaints that this is anti-competitive, since no similar fee is leveraged on Apple Music for making use of Apple’s payment system. While Spotify has to pay for copyright licenses and for using the payment system, Apple Music only has to pay for copyright licenses.

Developing and maintaining the payment system requires time and resources. Payment systems need to be reliable and secure, they must fulfill a range of regulatory requirements originating from payment services legislation, privacy legislation and IT security legislation. The fees are presumably leveraged to cover the costs of development and maintenance, and they do not only apply to services that compete with Apple’s own services but to all apps that make use of the payment system.

Apple could charge its own music subscription service for using its own payment system the same way it does with competing music subscription apps, to ensure that having simultaneous control over the payment system and some app-mediated services does not foreclose competition in the app-mediated service market. This would be a *vertical separation*, which could provide incentives for competitors to develop and sell new and better apps on the Apple Appstore, providing more and better choices for end-consumers.

## Case Study (ii): Mobile OS provider apps

In Example S.6 we discussed how mobile operating system vendors may also develop apps and pre-install them on a mobile phone before it is delivered to an end-user.

Google develops Android and pre-installs on Android devices its browser Chrome. The European Commission decided in 2018 that Google Android had committed a breach of EU competition rules, since other competing browser vendors for the mobile operating system market were disproportionately disadvantaged by the high visibility of Chrome for end-consumers[14].

In effect, the European Commission found that Google was leveraging its market power, through the operation of its mobile operating system, to cement its position in the search engine market - a form of vertical integration of mobile operating systems.  By pre-installing Chrome on Android phones, Chrome became the *de facto* standard for browsers. The remedy to this is ensuring that consumers choose which web browser to install from the app store themselves.This follows closely the conclusions of the European Commission referenced in Example B.13, where competing media players and web browsers in a computer-based operating system were determined to need larger visibility to ensure competition.

When a consumer visits the appstore to choose a browser, the appstore can only display a limited amount of web browser applications because of the display area size limits on the mobile phone screen. The European Commission determined that Google was in a position to give a disproportionate visibility advantage to the browser which is displayed first to the consumer. This solution is an example of *unbundling*, as it decouples the web browser and search engine from the mobile operating system.

## Case Study (iii): Data portability

Article 20 of the EU General Data Protection recognises the individual’s right to data portability. It allows individuals to obtain and reuse their personal data for their own purposes across

14In 2019, the Competition Commission of India has opened a similar case, see: https://techcrunch.com/2019/07/09/google-india-android-antitrust/

different services. More precisely, it allows them to move, copy or transfer personal data easily from one IT environment to another in a safe and secure way, without affecting its usability. The right applies only to information that the individual has provided to a controller.

**Example S.9:** A consumer can transmit the data already provided to a controller, for example an electricity provider, to another controller, for example an applications and services provider that can use this data to find him/her a better deal or help him/her understand his/her spending habits.

Data portability raises a number of technical issues: which sort of data is covered? Once that is established, what format does the data need to have? The first question is not legally obvious: does measurement data on the temperature of the central processing unit of a particular end-users' smartphone belong to that end-user? If manufacturers had access to this data over millions of devices, they could conceivably make the central processing unit more energy efficient. On the other hand, the measurement data from any particular individual could reveal whether they use their smartphone for computation heavy activities, how often they use their phone, and for how long. Many of these questions will have to be resolved by regulatory authorities and courts, but industry actors are attempting to create solutions with the help of obfuscating statistics, such as differential privacy.[15] In other cases, the answer might be more straightforward, such as when it relates to the profile data on Facebook, or the email address an end-consumer provided to Apple when setting up their appstore account.

To address the question of data format, a number of internet companies, which are currently among the major data controllers, launched an open source initiative for consumer data portability: the Data Transfer Project[16]. The idea is for an end-consumer to have their data stored in a similar way in all services, so that once the end-consumer decides to change services they can easily export and import data. If two services do not store data in the same way, the services will not be able to parse each other's datasets.

15https://datatracker.ietf.org/doc/slides-104-pearg-amelia-christoffer-differential-privacy/
16https://datatransferproject.dev/

**Example S.10:** If an end-consumer has a Microsoft Outlook email client but wants to switch to a Thunderbird email client, the end-consumer might be interested in transferring their email archives from Outlook to Thunderbird. Then it is important that emails are stored in a format that is recognisable by both clients.

**Example S.11:** If an end-consumer has a Google+ account but wants a Facebook account, they may want to transfer personal photos, contact information and status updates. Then it is important that the organization of the profile is similar between both services, so that, for instance, photos are labelled in the same way in the archive across both services, and that the labels for contact information, status updates, and private messages are the same in the data dump.

**Example S.12:** In some cases, data transfers can be facilitated with open API:s. For instance, Twitter used to have open API:s that made it possible for a competing services to collect any public tweets issued by a user and re-publish them on a different site. Competing micro-blogging tools such as Identi.ca used to leverage these open API:s to provide open source alternatives to Twitter.

Data formatting is a very real and broad challenge that also faces the public sector. The Open Knowledge Foundation has worked extensively on open data, including technical aspects such as data formats.[17]

17http://opendatahandbook.org/guide/en/how-to-open-up-data/

## Self-evaluation questions

1. What properties should an “open API” have?
2. Can you provide some examples of technical and contractual requirements imposed by an appstore on app developers?
3. Are there API:s for:
    a. Opening a file on a computer?
    b. Creating a button to close a window?
    c. Getting data from the smartphone GPS?
    d. Making your address book accessible to apps?
    e. Fetching email to your phone?
4. Why should default installations or default settings be relevant for end-consumers?

## Proposed answers

1. The concept of “openness” typically implies that the features made accessible through the API are available in open source code and/or that there is a specification for the API which is available for free online. Access to the source code would enable someone to verify that the API does exactly what it claims and nothing else (for instance), while a specification would be like a handy-guide to invoking API functionalities.

    Commercially, openness typically means that anyone can use the features specified by the API without paying an access fee.

    In this sense “openness” can be understood as both a technical (source code/specification) and commercial (no access fees) quality of the API. However, depending on who is using the word “open” it could mean any combination of the above commercial and technical qualities, or include completely different qualities also.
2. Some examples:
    **Contractual requirements >** Among others, Google Play does not allow apps that: facilitate or promote illegal activities (such as facilitating the sale or purchase of illegal drugs or prescription drugs without a prescription; epicting or encouraging the use or sale of drugs, alcohol, or tobacco by minors; instructions for growing or manufacturing illegal drugs); use another app or entity’s brand, title, logo, or name in a manner that may result in misleading users; it requires apps to disclose the collection, use, and sharing of users’ data, and to limit the use of the data to the purposes disclosed, and the consent provided by the user. In addition, it requires the apps to stick on their additional requirements about personal and sensitive information.[18]
    **Technical requirements >** Apps on Google Play must comply with the default Android system optimization requirements documented in the Core App Quality guidelines for Google Play.[19]

18Go here for more information: https://play.google.com/about/privacy-security-deception/#!#personal-sensitive
19Go here for more information:

3. Answers:
    a. Yes. If not, programming an app or a software that opened files would be very tedious.
    b. Yes. There are several, depending on which graphics library you are using to create the button.
    c. Yes. As exemplified for MicroG in the text.
    d. Yes.
    e. Sort of. There is a protocol for fetching email, sort of like a standardized API.

   All things considered, it may be more relevant to ask for those things for which there are no API:s.

4. Default installations and default settings can limit end-consumer choices by making it more difficult, or too difficult, to access alternatives, including better ones. At the same time, default installation options can be used to limit competitors' access to the market, because they make it more difficult for competing services to be visible to end-consumers. This, in turn, diminishes competitors' incentives to develop and sell new products, to the detriment of end-consumers.

http://developer.android.com/distribute/essentials/quality/core.html#listing