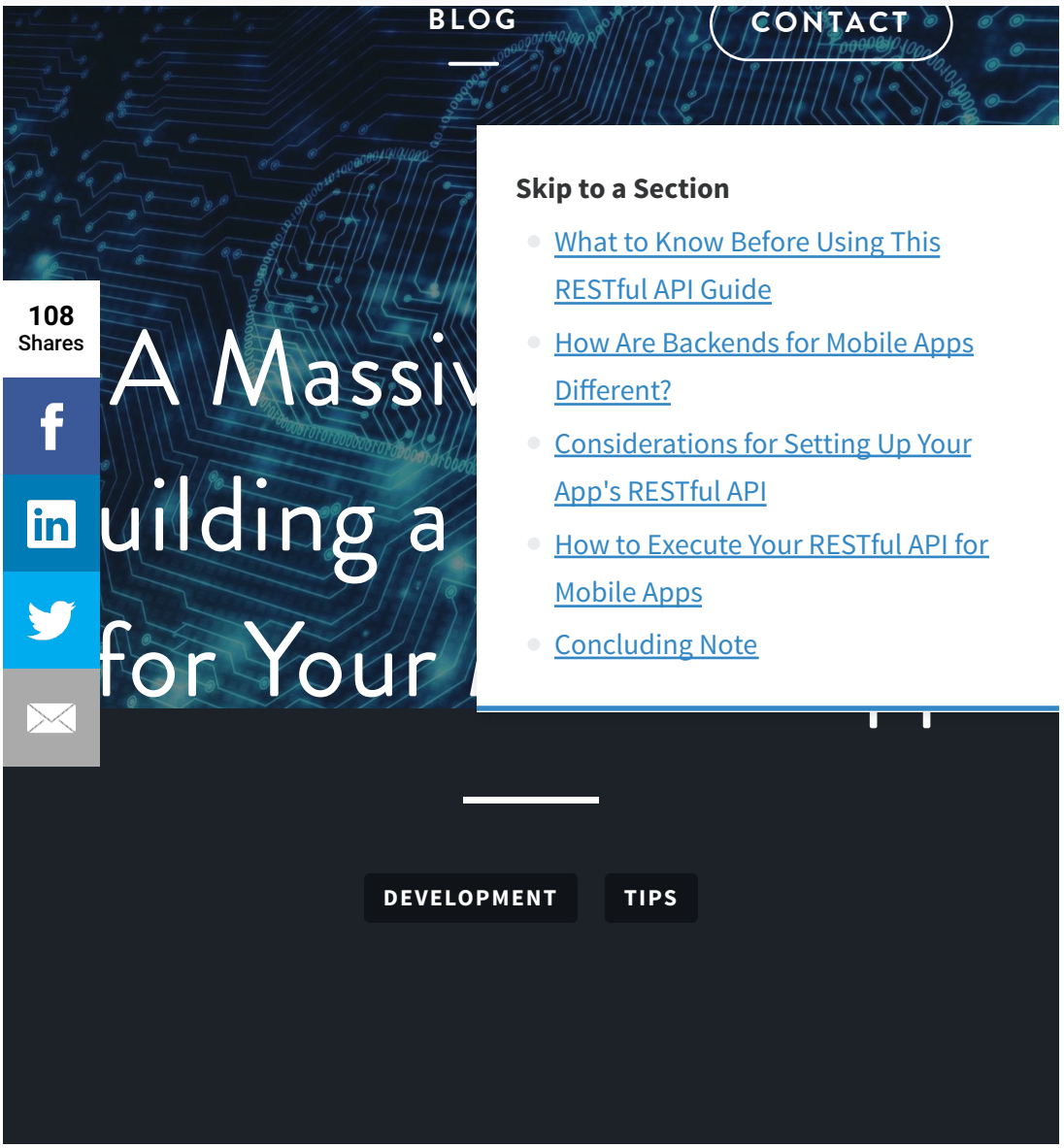


Skip to a Section

We build apps of all shapes and sizes here at Savvy Apps, l element is that they communicate with servers. Very few a operate without some sort of Internet connectivity, meanir interact with a backend, web services, or APIs. These APIs c provided by Google, Amazon, Facebook, or comparable thi also could be APIs that are developed internally.

web services or APIs for apps specifically. In our experience, we've found guidelines on how to build better APIs for mobile apps saves time and e development and reduces head

We wrote this guide to outline t bases for mobile apps and ful API specifically for mob elpful for our customers ar operly build and maintain

Skip to a Section

R

- Use a well-known archit
- Make the server do the heavy lifting so mobile clients don't have
- Version your API so it handles requests coming in from new and
- Account for offline usage and usage across devices.
- Prioritize performance and scalability when picking where to ho
- Use standard security protocols and well-vetted authentication/
- Build three backend environments: development, staging, and p
- Let your data decide the type of database you use.
- Construct API URL endpoints so that it's very clear what that res
- For requests, let the client send full objects, and the server use th
- Utilize UTC for dates/times, and let the client figure out how to d
- Remember that GET and PUT requests need to be idempotent.

What to Know Before Using This RESTful API

REST is by far the most commonly-used style for designing APIs, especia world. There are also particular subsets of REST, like OData , that furthe should be transmitted between your apps and the server. While those su for your particular needs, we're going to keep the conversation broad er

it works and, more importantly, how new services should be built onto i

In this guide, we'll also be discu
These rules, however, will certa
your API too. In most cases, the
r handing out those resou
ages and often by differen
elow will ensure that both
the time comes for the tw
e than thinking that you've
at the mobile client can us
bers, it is critical for everyo
d blueprints to avoid unexpected miscommunications, delays, and
Adhering to a commonly-agreed set of standards and expectations will
iterate faster and more efficiently, which makes development and main
expensive in the long run.

Skip to a Section

How Are Backends for Mobile Apps Different

Before we dive into the hosting, security, architecture, and other consid
your RESTful API, let's examine what makes building an API for mobile a
other systems. These mobile-specific concerns are essential to making s
API is prepared to work efficiently with a mobile app and the expectatio

HTTPS, Not HTTP

The internet was built on HTTP, but mobile platforms enforce HTTPS rec
modern encryption and trusted signed certificates. A mobile backend n
for every endpoint. Your development, staging, and production environ
all be using the same type of signed certificates. This will save you head

Server Does Most of the Work

To save on network data costs a
doing as little work as possible.
ving and storing data remo
rful, remote machines. Plu
s your app to continue run
data quickly instead of cal
s all their devices, which is

g the server do the heavy
orms by moving and conso
both your iOS and Android apps. The server is going to have access to m
hardware than the mobile clients for number crunching, so it is wise to l

Server Issues Can Kill an App

You can also expect any errors a mobile user experiences to be broadcas
megaphone. Tech users these days have little patience when something
expected, especially mobile app users. There's no bigger stage for issues
in App Store and Google Play app reviews. If something goes wrong, the
respond with user-friendly error messages or error codes the client can
user and, hopefully, help fix the issue. Even a single error can cause a 1-s
positive reviews are critically linked to the success of an app. Too many
caused by server issues will stop new downloads for your app.

Versioning is More Important

With mobile app users updating their apps (or not) at different frequen
API becomes more important than other, more controlled environments

strategies on how to handle this later.

Plan for Push Notificati

A useful communication avenue tools that specialize in pu ss yourself. Your server ma es to users for sending pus ge device tokens and send cost effective than buildir

Skip to a Section

onciling Offline Acti

mobile users will expect the app to have some limited functionalit
Once reconnected to the server, reconciling the offline activity with the
needs to be considered. This is especially important for apps that users
multiple devices, such as their phone and tablet. Coordinating API calls
and order of operations is something that needs to be discussed by the
backend developers.

Considerations for Setting Up Your App's RES

Now that we've looked at what sets mobile apart from other systems, w
into planning your RESTful API. These tips address common concerns fo
dealing with security, creating the backend architecture, choosing datal
options, using the right tools, and supporting multiple platforms.

Hosting the Server

Choosing the location to host your server is a big decision. If you don't h
capacity to host your own bare-metal server, there are plenty of cloud-h

evaluating services for where to host your server include:

- How does the service sc
  machines) or vertically (
- How will the cost increas
- Are there any migration
  environments (developr
- What features are alread
  energy from trying to re
- What steps does the ser
  back ups, uptime monit
- How easy can the data b

Skip to a Section

Protecting the Data

Depending on your needs, you have a wide array of authentication mec
Any hosted service you choose should already include easy integration
trusted CA certificates. HTTP Basic Authentication is the easiest to imple
the least secure. OAuth2 is widely accepted as a secure, standard way o
authentication and is highly recommended. There are plenty of libraries
phone number authentication you could use as well. Do not try to write
authentication! There is no need to reinvent the wheel here when you c
protocols and libraries that have already been vetted by many others or
server side. Protecting each API endpoint behind authentication require
the norm. Don't allow free passes on a resource unless necessary for fur

Sensitive data should be protected. This is a given, but security is a spec
absolute. Encrypt your user's sensitive data. Encryption may not be nec
project, but it should always be considered. Don't store your passwords
Please. Not only should you hash passwords, but using random salts for

Planning the Architecture

As we already discussed, you're hopefully planning on building not one backend environments: develo environment is where frequent velopers. Data here can be nated scripts to populate s through all its tests in co and API endpoints) and ge onment.

taging environment is goi y, data here is an import o mation. The more realistic data used here, the better confidence yo your system will perform in production. Porting data may not be feasibl but having some sort of quasi-representative data in these first environ reducing risk and discovering bugs in the logic before becoming a real is cannot be reproduced, having at least roughly the same quantity of dat production in these environments again will pinpoint bottlenecks in the Otherwise, irreproducible slow downs, hang ups, and bugs may occur ir can be a nightmare to track down in development.

Making Database and Storage Decisions

No matter what type of database you use, it's worth noting that entity ID randomly generated UUIDs, not sequential. This helps secure resources much harder to guess. When it comes to storing your data, you might be traditional relational database like MySQL or MariaDB. Or maybe you pr of a noSQL document database like MongoDB. Or perhaps you prefer th hybrid approach that something like PostgreSQL can offer with both rel storage support. Which database your project should use is really going

MySQL/MariaDB

- Well established, stable and reliable
- Lots of libraries, framew
- Data is rigid, well structu
- Scaling requires a lot of

goDB

- No tables, no formal sch
- Easier to scale than SQL
- Easy ramp up and iterati
- Easier to shoot yourself

greSQL

- Support and popularity is growing rapidly
- Built around giving more features and tools for DB admins
- Flexible enough to mix relational data with model-independent

Cloud document storage

- A cheap solution could be something like using Amazon S3 Buck sets of whole documents

Finding the Right Tools

You're going to need the right tools to get the job done. Whether it's con between project teams, current team members, or future team member onboarding process for your project, communication is key to success o project. During development, make sure you're using a clear dev-trackir teammates have access. We're fans of Pivotal Tracker and Trello, but a work.

Skip to a Section

writing the code to log in a user in an iOS app will really want to know w
authentication API is ready to consume. They may even have a discussio
priorities of the server develope
blocking the iOS app from prog
not only what's currently being
flow of coordinating work
ble. This will also help circ
aturely, causing a delay or
t's also worth noting that
a new API is deployed, ev

Skip to a Section

done right, RESTful API e
ing both obvious use cases as well as expected edge cases for each
the core principles of REST is stateless, which makes our API endpoints
black boxes, ripe for testing. New data comes in, successful message co
newly persisted data changes are easily verifiable in the database. Requ
expected data response comes out. The testing framework to use will c
the language you're writing the backend server logic in. No matter how
they should be well-maintained and run with 100% passing rate before

During development, your testing suite may prove sufficient documenta
developers to review how the system works. Eventually you'll need to w
english, how the system you built actually works. I know this is the least
most of us, but it is critical for the success of a project in the long term. Y
is only as good as the effort you put into it. Fortunately, these days there
tools out there that can do a lot of the work for you. One of our favorite
tools is Postman. Not only is it useful for exploring or testing an API, bu
generate all the requests, responses, and handled error codes and store
reference later. You can literally build your documentation as you build
suite for your new API.

and messages. You also need to make sure this documentation is access
the team. It does no good if no one else can read it! Included below are
documentation tools you may

Skip to a Section

Swagger

- Can support bottom up
- Language neutral, but d

ry

- Built on top of open sou

e

- Supports Markdown
- Can be hosted on Github

Supporting Multiple Platforms

If you're building an API for a mobile app to consume, chances are good
support other platforms in the future. The general rule of thumb for buil
is to make the client as dumb and thin as possible, while keeping all the
filtering, number crunching, data aggregating, and consolidation on the
leverages the more powerful hardware of the server and tries to keep th
while fetching and showing the data to the user as quickly as possible.

This is important because when you're building your app on iOS, Androi
don't want to rewrite complicated filtering and parsing logic three times
For this reason, you'll want to allow robust sorting and filtering options,
fetch only what it needs. For example, don't return everything in a colle
client app to sort through the data to find what it was looking for. Use p
lists of data to avoid overwhelming both the client and the user and allo

those can be useful while reading logs and debugging in the future.

How to Execute You

Now that we've discussed how
:ually design your API in a
er url paths, requests and r
ful API.

dling URL Paths

he rest of this guide, we're
ple, some of your resources might be a book, a book club, an auth
library location. The client would want access to all of those resources v
could construct some endpoints mapping to:

```
GET    /books
POST   /books
GET    /books/{bookId}
PUT    /books/{bookId}
PATCH  /books/{bookId}
DELETE /books/{bookId}
GET    /books/{bookId}/authors
GET    /bookClubs/book
GET    /users/{userId}
GET    /users/{userId}/favoriteBooks
```

Notice the pattern emerging. We hope to construct our API URL endpoir
makes it very clear what that resource contains. Now let's talk about the
design tools: nouns and verbs. The actions GET, POST, PUT, PATCH, DELI
meaning they are the HTTP methods telling the server what action to pe
path itself tells you the nouns, which are the resources that will be acte
to make this distinction here, because otherwise you could end up with

Skip to a Section

Now take a look at these (bad) endpoints:

```
/getBooks
/createNewBook
/checkOutBook/{bookId}
/returnBook/{bookId}
ldBookToFavorites
ldNewMemberToBookClub
angeBookClubMeetingTi
angeBookClubMeetingLo
moveBookClubMember
```


Skip to a Section

ay glean more about wha
ou're going to have to ad
rce. Doing so will lead to f
to keep your API concise and modular. You're not only creating an
right now, but you also need to think about how additional features or c
into the API.

To improve your API's robustness, just let the requests drive the server's
may seem like a silly statement, because of course a client will make a r
server's job is to send a response. If the server is dictating what, how, ar
fetch or change data, then that means that every time they want to perf
feature, the client developers are going to have to wait for changes to be
The server is allowed to refuse requests. That's what error messages are
your endpoints with nouns and reacting to the data in the request, you'
API that will require less maintenance over time as you allow clients to c

Now, just because we're using nouns to drive the URLs, doesn't mean ou
be the same as our data model objects. The URL should include model c
appropriate, but we want to make these paths easy to read and intuitive
resource `/favoriteBooks` probably will just return the same objects
provide the clarification of what type of books we're going to get back. T

data and presenting it to the user.

This leads directly into how to h
four sub-actions are all related
retrieve from the server. Again,
om as possible here, witho
le filtering, sorting, pagina
our API endpoint quantity
to perform more complica
ery parameters, we open u
nt to the users as they see
betical order, but tomorro
useful to end users. Or maybe they want to let end users choose h
libraries sorted as a configurable setting. The client can do all of that wi
work on the server side, which is what we want.

Here are some examples of handling these URL parameters for our libra

```
GET    /libraries?sortBy=name&isCurrentlyOpen=true&pageCou
GET    /books?queryTitle=Sherlock+Holmes&queryAuthor=Arthu
GET    /bookClubs?genre=mystery
```

API versioning is another feature we should implement to achieve the rc
especially important for mobile apps. Mobile developers don't always h
forcing software updates for all end users, so our API is going to have to
both old and new requests. We'll do this by routing requests with a vers
will argue that this version number should go in the URL path, and some
be placed in the request header. We prefer it to be in the URL path for ea
that's up to you and your team to discuss. We treat versioned resources
resources, and that's why we think they deserve a unique path rather th

```
GET     /v1/books
GET     /v2/books
```


Another important guideline to
left to right, from broad to
oints, starting with the roc
g related resources cascac
rable and intuitive for the
se in your API is going to b
designs for how the client
ing on the backend to be a
elf questions like:

Skip to a Section

- Does a user need to log in to see this resource, or is it public?
- Does a user need to choose a library or city before viewing book not attached to a particular location?
- What actions does the UI allow when viewing the details of a par how can we design the API to allow the client to best perform th

Rules for Requests and Responses

So we've laid out how we'll construct our API endpoints, now let's talk a communicate effectively within each of those endpoints. Let's start with handling a request, don't force the client to only send one or two fields. full objects if they wish, while the server just uses the fields it needs. If a in with only a few fields, don't assume the missing fields are `null`. Field should be stated explicitly in both requests and responses. This prevent from having to guess if the data really is `null` or just not included. Take example below:

```
{
    "author": "Arthur Conan Doyle"
    "publicationYear":  "1902"
}
```


Skip to a Section

While this is not a complete boo
rtant information. Here th
, as told by the URL. The r
cation year. Only these fiel
details are not important
the title of the book shou
partial object requests re

ttention to the `Content-Type` and `Accept` headers in the requ
remain `application/json` for most requests, but if you need to sup
place to do so (not in the URL!). Other types of resources like files, imag
should have their content type set and respected here as well.

After handling the request, we need to send a response. Things might n
way the client expected and an error will need to be returned. Or maybe
processed successfully and simply doesn't require any data returned ba
everything went exactly as expected. This is where the response HTTP s
out.

Be deliberate with what status code is returned, with 200, 201, and 204 f
responses, and 4XX codes for errors. You can bet the clients will be payi
codes, so we need to be careful with what we send back with our respon
response though, we need to remain consistent. We should not be mixi
“snake_case” for our JSON keys. The norm is camelCase, but an argume
for snake_case being more legible. Whatever you choose, use it everywh

about a book's ISBN. Pick one and use that key everywhere you need to

When sending responses that c
object versus an array. If the cli
put it in an array and return it th
ent for every programmin
g to parse special cases on

ip side of this advice is do
s it provides value. Wrapp
ime. As a follow up, don't i
y, we want the data mode
ving in extra fields that aren't part of the requested object will just
Below are some examples of what **not** to put in your responses:

Skip to a Section

Don't wrap data with useless envelopes.

```
GET /authors?genre=mystery

{
    "data": {
         [{ "authorName": "Arthur Conan Doyle",
         ...
         }]
    }
}
```

Don't return a single object when the client should expect a collection.

```
GET /authors?genre=mystery

{
    "authorName": "Arthur Conan Doyle",
    ...
```

Don't include metadata that isn't relevant to the client requested data.

```
GET /authors?genre=myster

{
    "requestInfo": {
        "genre": "mystery
    },
    [{
        "authorName": "A
        ...
    }]
```


Skip to a Section

dealing with dates and ti
in ISO 8601 format with U
ay format, or determine what precision the date-time data needs; t
apps will figure out how best to display the date and time to the user.

Aside from reading a request and creating or finding its response, the se
going to need some programmed logic to best execute each request. We
mind when writing this logic that GET and PUT requests need to be ider
that no matter how many times a client might call `GET /books` on our
remain unchanged. We should avoid doing anything "extra" or behind t
client might not expect. Whatever logic or indirect actions the server ne
response to a request needs to be communicated and documented to a

Concluding Note

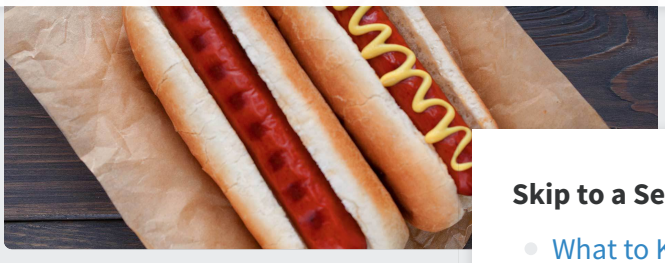
Hopefully, this guide has been insightful. When designing and creating
we focus on placing resources in a modular, explorable, and extendible
communicating our decisions and priorities effectively. Constant collab
developers working on the client apps and backend also ensures that r
avoided and continual progress is achieved. The less time we spend gu

Skip to a Section

BY: **MATT TEA**

Matthew Tea is a developer with a passion for quality, tested code. He's with a strong desire to learn new and upcoming technologie

RECOMMENDED ARTICLES

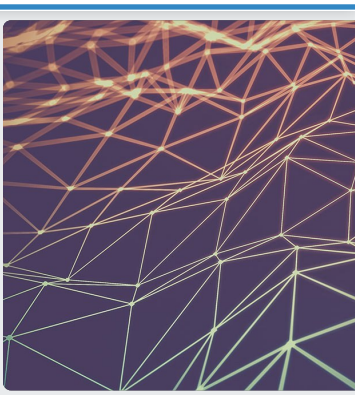


Core ML in iOS Really Do H
tion Without Server-side

ne learning has quickly becor
tant bedrock for a variety of a
bile implementation, howeve
reach for many in...

Reading

Skip to a Section



How to Start Android Development with an iOS Background

If all you've ever done in the past is iOS development, looking to build an app on Android might make you feel like you're entering...

Keep Reading

How to Use Deep Linkir App

What is deep linking? Deep creators to drive user enga the app onboarding proces tracking referrals and...

Keep Reading




RECENT ARTICLES

How to Conduct User Testing For Your App

Choosing a Firebase Database For Your App: Realtime Database vs. Cloud Firestore

ToggleButtonLayout: Easily Create Toggle Buttons for Your Android App

The Definitive Guide to Expanding Your Native App to the Web

YOU MADE IT THIS FAR SO...

Let's Talk

Skip to a Section

Want to work with us?

Savvy Apps is a Washington D.C. mobile design and mobile development company serving global brands and cutting-edge startups. We're a product team for hire that's driven by making life better, one app at a time.

CONTACT US NOW

OR

COMPLETE AN APP BRIEF

Drive Suite 100 Reston, Virginia 20191

(703) 544-9191

See It On a Map




Skip to a Section