STX NEXT
python powerhouse

how to audit the quality of your react native code

a step-by-step guide

Table of Contents

Introduction

It's hard to imagine a digital product being made these days without a mobile application.

One of the best and most popular modern languages for app development is React Native. But whether you're already introducing your mobile app to the world, redeveloping an existing product and struggling with legacy code, or just getting started on development, there's one key step you're bound to benefit from: a code audit.

Auditing your code—meaning, thoroughly examining it in search of bugs, flaws, and other weaknesses—is essential to ensuring the quality, security, and performance of your app.

Some time ago, we wrote an accessible and easily digestible article introducing the subject in a general manner; however, because our company's been creating software since 2005 and audited literally hundreds of codebases in that time, we knew we had more to offer. That's how this document came to be.

Below you'll find a step-by-step guide to auditing the quality of your React Native code, containing lessons learned and expert insights based on real-life React Native code audits we've performed at STX Next. We divided it into three broad chapters, going from the more general aspects to the more detailed ones:

1) **Software Configuration Management:** how to manage your project technically, where we review version control, project configuration, and deployment strategies
2) **Software Architecture:** architecture details comprising technology choices, data flow, and platforms supported
3) **Programming Best Practices:** general quality checks for the codebase, including the use of tools that can help in the development process

Without further ado, let's get technical and take a deep dive into React Native code audits.

Software Configuration Management

Code repository and version control

Hosting and VCS

At the very beginning, you should check what kind of version control system (VCS) is being used in your project. If you have no version control system in place, this is reason enough for you to already be concerned.

Your code repository can be hosted on a remote server. The most popular services for this purpose include:

- GitHub
- BitBucket
- GitLab

Each of those options has pros and cons, but all of them are solid repository providers. It is also possible for you to get a self-hosted instance of GitLab or some other VCS provider.

Recommendations for your in-house or outsourced React Native development team:

- Introduce a version control system if you have none in your project
- Pick a popular and user-approved VCS provider
- Consider using a provider that integrates well with other tools used in the project; for example, BitBucket (VCS) and Jira (project management tool) both come from Atlassian, which makes integration between these two a breeze

Branching

Clean, readable, and tidy repository is one of the first signs of quality in any project. But the most important thing is to make sure every team member follows the same rules established by the whole team. There are many conventions when working with branches in Git, and the most common are Trunk-Based Development and GitFlow.

Whenever your developer is working on a task, a new branch should be created. It's worth it to introduce a naming convention for such branches, for instance by using prefixes like *feature, fix, chore:*

- `feature/push-notifications`
- `fix/facebook-auth-on-ios-15`

Some teams prefer to name branches after ticket numbers in the project management tool of their choice, for example: `137-push-notification-settings`.

In the end, the key thing is to stay consistent and make sure all team members follow the same rules.

To keep your repository clean, feature branches should be deleted after merging.

Recommendations for your in-house or outsourced React Native development team:

- Introduce a branch naming convention, for instance through the use of prefixes such as *feature, fix, chore* or by using ticket numbers
- Maintain a clean repository from all merged branches

Commit messages

You should verify if your developers follow any pattern of commit messages, such as Conventional Commits. At the very least, check whether your commits are well-structured and clearly inform you about the operations that have been done there, for example, "Add tests for datetime helpers."

There are tools like commitlint that can enforce specific rules to commit messages.

Another recommendation is to squash irrelevant commits, for instance when certain commits contain some in-progress work that later got overwritten. Such commits don't provide much value and can be merged (squashed) to make the Git history clearer and easier to interpret.

Recommendations for your in-house or outsourced React Native development team:

- Introduce a common format of commit messages while taking into account good practices or guidelines

- Introduce a common pattern of commit messages, like Conventional Commits

Git merge strategy

Two most common merging strategies are Git merge and Git rebase. Both have pros and cons, and the choice between them is usually considered a matter of personal preference.

Git merge creates a new commit where it synchronizes the feature with the main branch, but the history of the project isn't linear. Git rebase takes the history of the feature branch and puts it at the top of the main branch, creating a perfect linear history.

Recommendations for your in-house or outsourced React Native development team:

- Discuss introducing a Git merge strategy with your development team
- Keep your merging strategy consistent

Project setup and documentation

Gitignore

We recommend that your Gitignore file contain a list of all files that:

- are not necessary in a remote repository—bundle application files, temporary files, a coverage folder for test, debug, or log files;
- are too big to store them in a remote repository and at the same time can be generated fast—like a `node_modules` folder;
- contain private information—environment setup files where we store secrets, passwords, signing certificates, or keystores.

On the other hand, there are files that should definitely be stored in a remote repository, but sometimes are incorrectly placed in the Gitignore, because in other cases the work environment won't be consistent between team members and in CI/CD.

Here are examples of such files:

- Lockfiles for *npm* or *yarn*
- Settings files for Prettier and a linter (if they're used in the project)
- Settings files for TypeScript or a test environment

If your project repository doesn't contain a lockfile for *npm* or *yarn*, it can become a serious issue, as each developer will have different versions of dependencies installed. All team members should use the same package manager (either *npm* or *yarn*) and the repository should only have one lockfile (`package-lock.json` or `yarn.lock`).

Recommendations for your in-house or outsourced React Native development team:

- Remove lockfiles from the Gitignore and make sure all of your developers use the same package manager to install dependencies
- Remove unnecessary output files from the repository and add them to .gitignore

ReadMe

Why should your project have a ReadMe file? Because you probably won't be working on this project alone, and you probably won't be working on it forever.

A good ReadMe should contain:

- Precise steps needed to start work on the project—by "precise," we mean that after following those steps, every developer (also a junior) should be able to start work on the project with the same exact configuration like the other team members
- A list of technologies used in the project and a way of using them
- Explanations of unclear situations, for instance when you use a deprecated library on purpose

Recommendations for your in-house or outsourced React Native development team:

- Get familiar with guidelines on how a great ReadMe should look
- Create and/or modify your ReadMe file using best practices
- Check out this list of very well-written ReadMe files for inspiration

Project dependencies

The development of the software and libraries used in building mobile apps is happening at a rapid pace, with new solutions and updates being released daily. Updating libraries to the newest versions is especially important in mobile development, since new versions of iOS and Android are released annually, so your developers need to be up-to-date.

Also, Apple and Google are enforcing certain policies, like having to compile apps with the newest major version of Xcode. This means your app needs to be buildable with the newest Xcode, which in turn means that all the project dependencies need to support it.

Therefore, in the event of a code audit, check whether everything is up-to-date or not too outdated. In most cases, it is okay to use libraries that are outdated by a few months.

However, keeping the old versions for a longer period of time can cause you problems—especially when some libraries are dependent on the others.

Recommendations for your in-house or outsourced React Native development team:

- Update libraries that show deprecation warnings
- Upgrade React Native to the latest version (if possible)
- Keep track of all dependency versions and update them systematically

Automation and deployment strategy

In contrast to web apps, mobile apps have a much longer and resource-consuming build process. Sometimes creating an app for both iOS and Android can take over an hour.

This process can and should be automated to a point, and in a perfect scenario the build should not be running at a developer's local machine, as it requires a lot of computing power, which might make other work impossible while the build is running.

Introducing Continuous Delivery (CD) can greatly improve the development process and save a lot of time by letting developers continue their work instead of building the app manually. Both Apple and Google offer tools that help test the release (non-debug) build of the app before it is released to the public.

Having fully automated CD pipelines in the cloud also reduces the bottleneck problem when someone is designated to build the app. This additionally enables faster feedback, since building the app and handing it over to testers or stakeholders doesn't require any effort from the development team.

Continuous Integration (CI), on the other hand, allows you to introduce tests and other validation tools that are run before a code pull request is merged. This helps you make sure that low-quality code changes are not merged to the repository target branch.

It is crucial to set up deployment and testing strategies and document them. Having a standardized approach to versioning, testing, and releases in a project can improve team delivery velocity and reduce the cost of introducing new people to the project.

Recommendations for your in-house or outsourced React Native development team:

- Introduce automation and documentation for the release process of the app. Local automation using either Fastlane or Bitrise CLI can be used to keep the costs low. Alternatively, cloud-based solutions like Bitrise could be introduced for the best developer experience.

- Use Google and Apple services like Google Play and Testflight release channels (alpha, beta, internal testing) to distribute the app among testers, stakeholders, and early adopters. Services such as Firebase App Distribution can help you overcome issues with Google Play when the app update isn't displaying for some reason.

Security

Application security is an extremely important element that is often neglected, overlooked, or left for later. Remember that Android APK files can always be reverse-engineered, so even if you use obscurify tools, it is technically always possible to extract app secrets from the executable.

All sensitive data like certificates, tokens, and passwords should be stored in a manner dedicated to each platform—separately for iOS and separately for Android.

For iOS, you should use iOS Keychain Services, while for Android, you should use Android Keystore with the rules collected in Encrypted Shared Preferences. Fortunately, there is a library in React Native called `react-native-keychain` that allows you to unify secure storage APIs for iOS and Android.

Remember that the default AsyncStorage from React Native is not encrypted at all and should not be used for sensitive data.

Recommendations for your in-house or outsourced React Native development team:

- Remove all keystore files except debug.keystore from the repository and keep them in a secure location. Do not store signing certificates, private keys, and passwords in the code repository.
- Move all secret API keys to a dedicated environment configuration file and take that file out of the code repository.

Software Architecture

Technology choices

Language

When it comes to languages, you basically have two options: JavaScript or TypeScript.

Although React Native project initialization scripts generate projects in JavaScript, the current community standard is to use TypeScript for better maintainability, fewer bugs in general, and improved developer experience overall that leads to increased productivity.

You can read more about the pros and cons of JavaScript vs. TypeScript on our blog.

Recommendations for your in-house or outsourced React Native development team:

- If the project is written in JavaScript, consider refactoring it gradually to TypeScript
- You can start your move away from JavaScript with a rule that every new file uses TypeScript while the old files are gradually migrated when edited
- Use type aliases during refactoring that can initially point to any type; later, you can introduce a custom interface when you have more time to define a strict interface

An example here would be: `export type UserData = Record<string, any>`

You can later refactor that into:

```
export interface UserData {
  email: string;
  displayName: string;
  …
}
```

State management

State management is one of the most important aspects of your mobile app architecture.

The choice of your state manager can depend on your backend choices; for instance, if you use GraphQL on the backend, ApolloClient on the frontend will be a great choice.

The most popular state management library in React/React Native applications is Redux with data persistence ensured by `redux-persist` and asynchronous actions supported by `redux-thunk`. You can also use other asynchronous middlewares for Redux, like `redux-saga` based on generators or `redux-observable` based on rxJS.

To increase the selector performance, we recommend using libraries like `redux-reselect`. If you don't have any state management system in place, you can use Redux Toolkit as an easy-to-use template for implementing Redux.

Sometimes mobile developers use React Context as a state management tool, although it's not recommended for complex scenarios. However, for certain features like DarkMode/LightMode UI switch or simply some local state, React Context is a perfect fit.

Recommendations for your in-house or outsourced React Native development team:

- Implement a state management system
- Add features that will improve the user experience by adding data persistence or performance improvements

React navigation

Every app that has more than one screen needs its own navigation logic that will hold all the screens, managing the presentation of transition between screens, the configuration of bottom and drawer navigation, and so on.

It's very important to use a navigation library that is popular and stable at the time of writing your app. If the app uses an uncommon library, it can be difficult to maintain it in the future.

Here are two most popular navigation libraries for 2022:

- React Navigation
- React Native Navigation

The version of the library used should be updated frequently. If your current version is far behind the newest stable version, an upgrade is required.

Upgrading the navigation library can be difficult, depending on which version you're starting from.

Recommendations for your in-house or outsourced React Native development team:

- Update your navigation library to the newest version
- If your navigation library is uncommon, suggest switching to a more popular one

Unit testing

In the modern world of app development, unit tests are not optional, but simply essential. That said, achieving very high code coverage will come at a similarly high cost, so it is important to find a sweet spot between the cost and the value.

Generally speaking, unit tests should be applied in every project. During your code audit, you should check the percentage of the test coverage and the testing library that is used. For React/React Native apps, the most popular testing library is Jest, which is already delivered by the project initialization script. It's also worth considering a support library such as `react-native-testing-library`.

As you check the unit tests, you should also take a look at the codebase overall to see whether the components are testable. What that means is, large components that have multiple responsibilities can sometimes be hard or even impossible to test.

Recommendations for your in-house or outsourced React Native development team:

- Implement unit tests
- Introduce a testing support library like `react-native-testing-library`
- Apply Test-Driven Development principles

End-to-end testing

End-to-end, or E2E, testing is often done by manual testers, but there are tools on the market like the Detox framework that let you create automated end-to-end tests to cover at least some of the most common cases.

Why should your project include an E2E testing library integration and E2E tests? Because you can run them locally as well as on a CI/CD platform to perform testing for the most frequently occurring scenarios, like user login and basic content browsing.

Even if you won't be able to cover 100% of the app functionalities through test cases, you will be able to reduce the time needed to test your app by a manual tester, especially when regression testing is required. Consequently, you will also be able to reduce the amount of money spent on testing.

Recommendations for your in-house or outsourced React Native development team:

- Introduce an E2E testing tool such as Detox
- Create tests for the most common user paths

Supported platforms

Another thing you should consider during your code audit are supported system versions for Android and iOS. You can check the market share for unsupported versions and inform your stakeholders about it. What's also worth mentioning is the support for tablets, because you may be under the impression that your app will automatically work on them, which isn't necessarily the case. Just check out this report prepared by Statcounter.

In the end, it is your decision to choose which operating systems (and their versions) to support. Sometimes it might be technically impossible to support an old version of Android or iOS because your dependencies do not support them. Always check reports on the market share of the versions of the operating systems worldwide as well as in specific countries if your product is targeting a specific area.

Android and iOS support

Make sure you're using an appropriately new version of React Native that supports the latest versions of operating systems—both Android and iOS.

When designing and testing an application, it's worth getting to know your users—not only what they need and how they will use your application, but also what devices and what operating systems they use.

There may be times when some platform that is popular in one region isn't popular in another. On that note, what's commonly used in your country may not be available in another. You should also consider your target group; for instance, children and teenagers tend to have rather lower-end devices.

Make sure to check if your app supports Tablet UI and if it is explicitly defined. In general, tablets are not that popular for apps, but there are some app categories like media consumption where tablet support is quite strong.

Data flow architecture

Backend API integration

You should look into how the integration with your backend API is implemented. Check if requests are handled by a standard JS fetch method or maybe some library. The most commonly used one here is Axios. Also, pay attention to how asynchronous operations are implemented. There are a few options, though nowadays the most popular is async/await syntax with try/catch block.

If you find places with callbacks or then syntax, it's worth refactoring your code for async/await to get better readability and maintainability. There should also be some layer of abstraction for HTTP requests—a dedicated service that will separate logic requests and take care of the single-responsibility principle.

Recommendations for your in-house or outsourced React Native development team:

- Consider refactoring the ES5 then/catch syntax to ES6 async/await for better code readability and maintainability
- Extract HTTP requests to a dedicated API integration service and use it in the state management asynchronous actions (using, for example, Redux Thunk or Saga) instead of calling HTTP actions there directly

Asynchronous state manager actions

In most cases, the state manager needs some additional middleware to handle asynchronous operations. For example, the software most often used for Redux are `redux-thunk` and `redux-saga`. It's a good idea to check out naming conventions for such actions, apply patterns like Request/Success/Failure, and stay consistent.

Recommendations for your in-house or outsourced React Native development team:

- Implement a naming convention for state manager actions
- Utilize tried-and-tested patterns like Request/Success/Failure

Data flow in components

Data flow in React should have a single direction—preferably from the state management system state to components. The state is changed by the actions dispatched from the components or middleware of the state management system.

Recommendations for your in-house or outsourced React Native development team:

- Move all API requests to the state management system
- Use asynchronous action handlers

Data persistence

Some of the data used in your app can and should be stored persistently. This will improve the user experience and overall performance of your application.

You won't have to call to the endpoint with some rare changing data each time, your users won't see loading spinners, and your screens will load faster as well as give you the ability to re-render them again when updated data is fetched from the backend API.

For React Native applications, data can be stored in AsyncStorage. Just remember that this type of storage is not encrypted, so you shouldn't use it to store sensitive information like user authentication tokens.

Recommendations for your in-house or outsourced React Native development team:

- If your data isn't stored persistently, start doing so
- If you're storing unnecessary data, try to remove it
- If sensitive data is stored in AsyncStorage, move it to an encrypted storage solution

Data security

Sensitive data should be stored in a safe place. But what kind of data is sensitive, exactly? Essentially, it's everything that can be used to:

- cause damage to your application,
- expose you to additional costs,
- acquire your users' data.

Such data can be different kinds of tokens: for API requests, for payments authentication, for connecting to external services, and so on.

Recommendations for your in-house or outsourced React Native development team:

- Use an encrypted keychain via `react-native-keychain` to store sensitive data securely
- Remove all unnecessary stored and persisted data of a sensitive nature
- Move as much sensitive data as possible to your application's backend

Programming Best Practices

General code quality

Code formatting

One of the first things you will notice as soon as you start checking the codebase will be formatting.

For readability and comfort of day-to-day work, your code must be formatted in a consistent way. The most popular code formatting library is Prettier, which is shipped together with a React Native project initialization script.

Recommendations for your in-house or outsourced React Native development team:

- Format the whole codebase consistently
- Start using the prettier tool and integrate it in a Continuous Integration pipeline, either in the cloud at the Git Repository level or locally using the Husky tool
- Don't let any developer merge unformatted code

Code linting

Your project should have some linting tool. The most commonly used one is ESLint.

You should check if the linting tool is integrated and configured properly, the number of errors equals 0, and the number of warnings is as close as possible to 0. In general, having more strict linter rules leads to better and more unified code.

Pay attention to ESLint rules that have an auto-fix feature—using them doesn't add any costs to the development process, but can bring a lot of value.

Recommendations for your in-house or outsourced React Native development team:

- Integrate ESLint into your project
- Fix ESLint issues
- Integrate ESLint check in a continuous integration pipeline, either in the cloud at the Git Repository level or locally using the Husky tool
- Don't let any developer merge code that doesn't pass the linter check

Static type checking

TypeScript in general leads to better productivity, fewer bugs, and better maintainability of the code.

Even if at first it can be problematic for developers who haven't used it before, it will pay off later on with more readable code, increased developer velocity, and a codebase that is easier to maintain.

Again, we recommend that you head over here to learn all about the advantages of TypeScript and read our comparison of JavaScript and TypeScript.

Recommendations for your in-house or outsourced React Native development team:

- Introduce TypeScript to the codebase and start refactoring the project to it
- Define all data structures used in the application

Code syntax

Both JavaScript and TypeScript are languages that are still evolving. As a result, you might encounter some code that uses old ES5 or pre-ES5 syntax. This must be checked carefully and repaired as soon as possible.

If a project was written a while ago and still contains legacy code, it should be refactored to the newest standards. For example, a change of ES5 then/catch to ES6 async/await.

Recommendations for your in-house or outsourced React Native development team:

- Eliminate all syntax errors in the codebase
- Refactor the codebase to modern ES6+ syntax and make use of improvements it introduces

Application performance

React-level optimizations

Performance is one of the crucial things in the development of any application. Great design and user experience won't be enough. Let's take a look at the possibilities React gives us to achieve better performance. One key concept here is **memoization.**

Memoization is one of the core concepts in application performance optimization. It relies on storing the result of an expensive function, later returning a cached version of the result when using this function again if the inputs of the function are the same.

The function needs to be a pure function to apply memoization, so make sure to follow functional programming principles in your code. In React, there are three places where you can utilize this concept:

1) **Component memoization**

To avoid unnecessary re-rendering of components we can wrap them with `React.memo` function for functional components or use `React.PureComponent` in case when we use class based components. Both will perform the same action: memoize components and re-render them only when necessary.

2) **Function memoization**

To increase React component performance, it is advised to use hook `useCallback` to memoize whole functions. This can reduce memory leak caused by re-creating functions on every component re-render.

3) **Function output memoization**

To bring performance to an even higher level, start using `useMemo` hook to memoize pure functions output when calling them with the same arguments.

Recommendations for your in-house or outsourced React Native development team:

- Introduce `React.memo` for functional and `React.PureComponent` for class-based components
- Wrap functions in React components in `useCallback` hook to increase performance by reducing memory leaks on each component re-render
- Memoize pure function's output by using the `useMemo` hook

State manager-level optimizations (such as Redux)

To improve performance of the application, we can memoize the data layer. For instance, you can use redux-reselect in Redux, which can simplify the redux state, as selectors can be recomposed and substates can be derived from others (with some additional parametrization). Redux reselect will also help you reduce the component's render count.

Recommendations for your in-house or outsourced React Native development team:

- Add a memoizing library for state manager to the project
- Use it when getting application state in a memoized manner

User experience overview

Image caching

Graphics are in most cases the heaviest part of the interface of an application. Therefore they should be optimized from the very beginning. It's recommended to optimize them for better performance and user experience. React Native provides a standard Image component which does not support aggressive caching. Using standard Image components can be also a reason to image flickering when navigating between screens.

Recommendations for your in-house or outsourced React Native development team:

- Introduce react-native-fastimage library
- Replace all Image components with FastImage component which supports caching much better

Interface scaling

Scaling the UI of the application based on screen size should not take place in most of the mobile apps (games are an exception), because it will cause the app to look the same on small phones like iPhone SE and iPhone Max Pro which is a much larger device.

Such UI scaling is considered a bad practice as the UI components (especially touchables) will be too tiny on smaller screens and too big on large screens.

This can also lead to problems when the user has accessibility options enabled—for example, and extra interface or text scaling. They might not work correctly, but some users just need them to use their devices.

Recommendations for your in-house or outsourced React Native development team:

- Do not use UI scaling based on screen width
- Use absolute values instead

Easy app flow

For optimal user experience, you should check if navigation and general use of the application is responsive and fluid. What does that mean? Verify whether:

- Switching between screens by pressing buttons happens immediately and doesn't need require you to hit the button multiple times
- Upper status bar is visible and readable
- Bottom activity indicator on the phone (basically iPhones) isn't covered by app content
- Gesture navigation to go back to previous screen on iOS is working

Recommendations for your in-house or outsourced React Native development team:

- Use `react-native-safearea-context` instead of hardcoded values for iPhone X/11/12/13 family devices to fix issues with SafeArea in the top and bottom of the screen
- Test navigation thru the app and extend touchable areas where needed
- Test the navigation on real devices, especially when testing gesture-based navigation

Common problems with React

This section will cover the most common issues in React Native applications. They are either bugs or flaws that lead to bad UX or poor performance.

Key issue in a list

One of the most common problems in React is using array indexes as key inside lists. You should keep an eye on this during code audit because it causes performance issues connected with unnecessary rerendering components. You can find more information on this here. Luckily, this problem is fairly easy to find and fix.

Removing event listeners

Another big problem which you can find are event listeners. You can live without them but they should be active only when they are really needed and removed when the user leaves the screen where they are used. Multiple activating event listeners without removing them can cause huge memory leaks and make your app unusable on low-end devices.

Common problems with React Native

FlatList in a ScrollView

A common mistake found in React Native code is to put `FlatLists` in the `ScrollView`. The idea behind `FlatList` is to speed up UI rendering. For this purpose, React Native mechanisms render only as many elements as can currently fit on the screen with some offset (the default is 10). This is called virtualization.

`ScrollView` differs from `FlatList` because its job is to render all the children elements at the same time. A `FlatList` that is inside a `ScrollView` will lose its ability to render its children lazily if they are in the same orientation (horizontal or vertical).

Recommendations for your in-house or outsourced React Native development team:

- Note that there is no `ScrollView` on the screens that could be replaced with a `FlatList`. This will speed up the loading and performance of the application.
- Pay attention to React Native warnings about nested `FlatList` in a `ScrollView`.

Animations not in the native thread

React Native can run animations either in the JavaScript or the native thread. Running them on the native side is always better for the performance and UX, although not all animations support it, for example height or color transitions.

It is sometimes possible to achieve the same or similar visual effects when using properties that support native thread animations, like transform or opacity changes. Consider refactoring your code to enable native thread animations.

The use of native drivers for rendering the animation requires the developer to declare it in the code.

Recommendation for your in-house or outsourced React Native development team: check if the animations, where possible, are declared with `useNativeDriver: true`

Images not caching (not using the FastImage library)

Images and photos are huge amounts of data that must be loaded into memory, rendered, and displayed on the screen. The default component from React Native for displaying images does not display them until they are fully loaded. Thanks to the FastImage library, we can cache images in the memory so subsequent renders will be faster and won't flicker.

What's more, we can prioritize the images and the order in which they are to be loaded. As a result, we accelerate the display of the entire screen. This can greatly improve the UX of the application.

Recommendations for your in-house or outsourced React Native development team:

- Don't use a regular Image component
- Use `react-native-fastimage` instead

SafeArea handling a.k.a. notches and bottom indicator

`SafeAreaView` makes sure that the content we want to display is not obscured by other elements that are natively displayed on the screen, such as the status bar with clock, battery level, but also camera notch and the bottom home indicator.

Make sure to always test your app on different devices—both the ones that have a screen notch with a bottom screen indicator and those without them.

Recommendations for your in-house or outsourced React Native development team:

- Check if all elements of the application are displaying properly on the iPhone X/11/12/13 device family
- Make sure they don't overlap with the bottom screen indicator and the status bar.

New iOS and Android migration issues

New versions of operating systems come out every year, usually around September or October. System performance and speed are often changed, and such changes also affect how the application is compiled and even what (native) libraries are used.

Such changes also require changes in React Native. Therefore, when choosing the libraries to be used in your application, you should check whether they have a good update history and are supported frequently.

Thanks to this, you can assume that when the systems are changed, your libraries will receive an update, and you'll be able to release your application on new systems.

That is why it's prudent to be on the lookout for the releases of new systems and test your application on the latest phones and systems earlier, before the update is released to all users. To that end, make sure to check your apps every year around September.

Final Thoughts

Thank you for reading our step-by-step guide to auditing the quality of your React Native code. If it doesn't cover absolutely everything you need to know, it will at least serve as a very solid foundation to support you throughout the process.

One last thought we'd like to leave you with: whenever you're working on a codebase, all sections and subsections should contain comments and recommendations on how to improve the project's quality.

Each recommendation should have a priority assigned: low, medium, high, or critical. The priority needs to be based on the issue severity and time complexity, as well as potential risks and benefits it can bring to both your development team and end users.

Most of all, though, remember that you don't have to do it alone. If for any reason you'd rather have someone experienced, who's really seen it all when it comes to code audits, by your side to help out—we're here for you.

STX Next may be Europe's Python Powerhouse—we even have an article about Python code audits with a great checklist and sample report to boot—but mobile development in React Native has been a key focus for us as well since 2017. Software testing and quality assurance is also something we take extremely seriously.

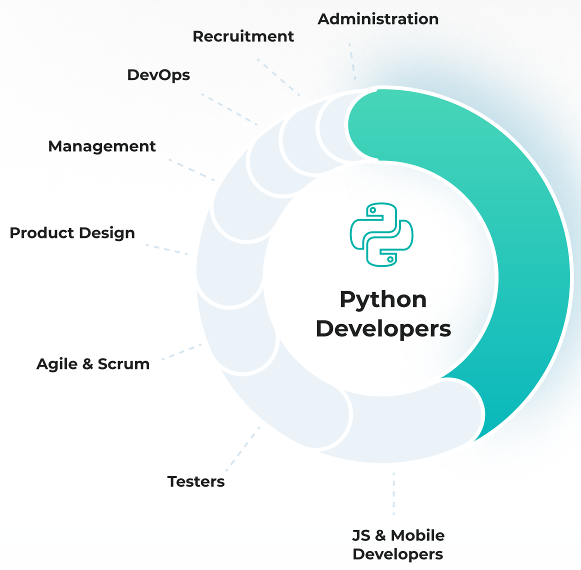
To that end, we can audit your React Native code for you—and, for a limited time only, **we're offering to do it for free!** Sign up here today and we'll get back to you before you know it.

But if there's anything else we can support you with, whether it's team extension or end-to-end product development, all you have to do is reach out to us—you'll receive an answer to any question just as quickly!

Hire an exclusive Python development team

Accelerate your software project with Europe's largest Python software house.
For companies with big projects and fast deadlines.

Team Extension	End-to-End Development	Consulting & Expertise
Additional developers or experts supporting your development efforts within 14 days	Full development team taking your project all the way from discovery to deployment	Solving your problems or improving your product with the help of subject matter experts



Over 400 developers

Ready to empower any project with well-reviewed code and a results-driven Agile process

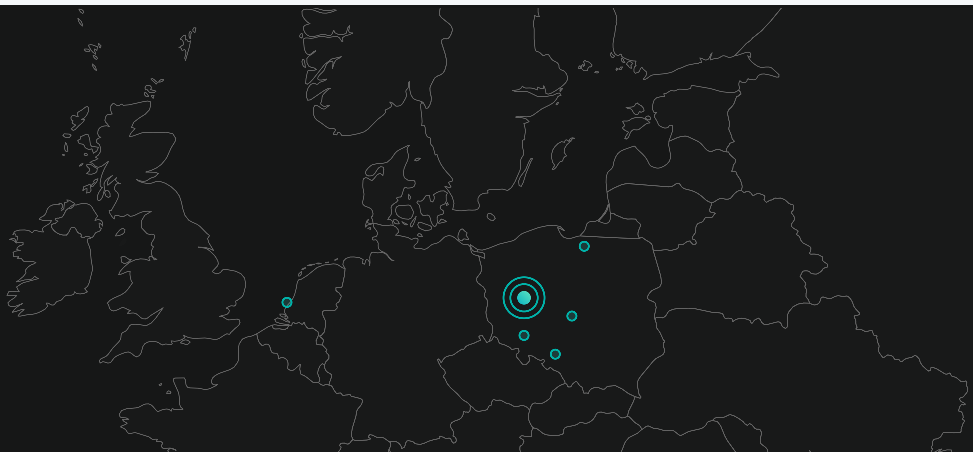
17+ years market experience	**750+** projects delivered	**3.5+ years** average partnership
300+ clients served	**550+** professionals on board	**6.5+ years** average experience of our developers

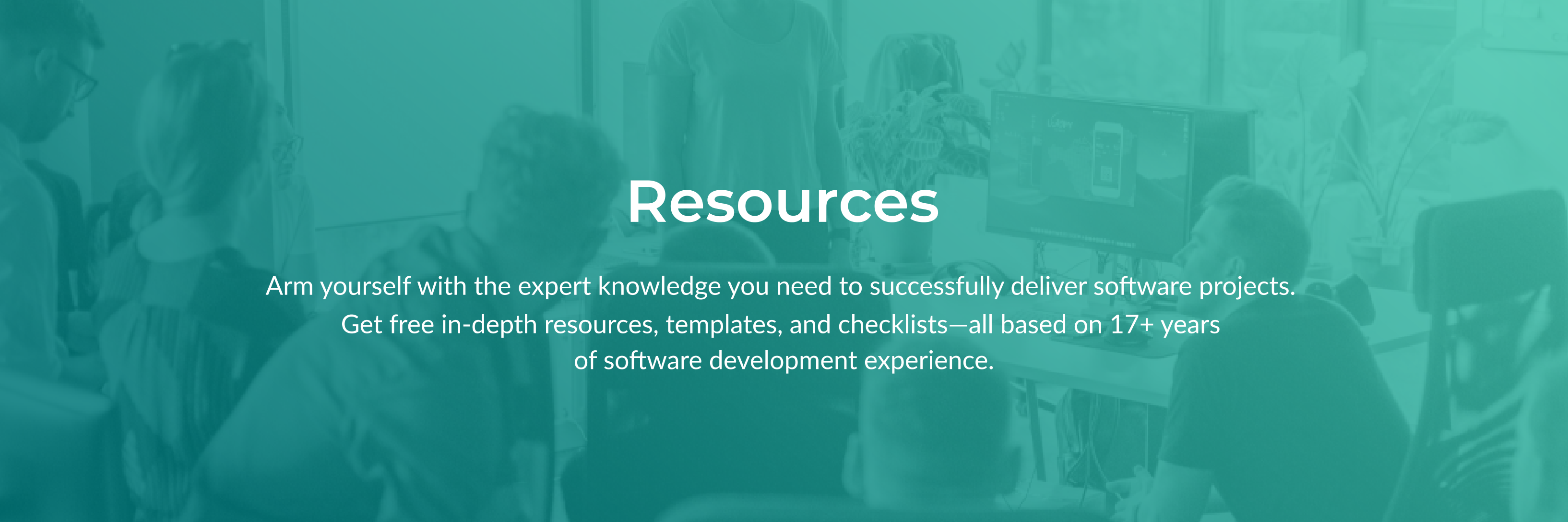
Locations

Poznań (HQ)

Mostowa 38
61-854 Poznań, Poland
+48 61 610 01 92

- Wrocław
- Olsztyn
- Katowice
- Łódź
- Hague (Netherlands)





Resources

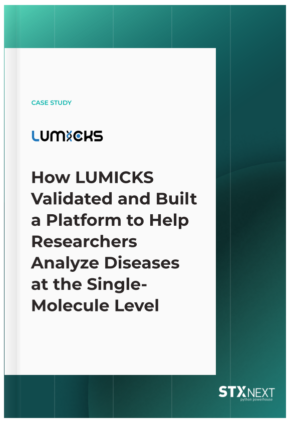
Arm yourself with the expert knowledge you need to successfully deliver software projects. Get free in-depth resources, templates, and checklists—all based on 17+ years of software development experience.

Reports and case studies

TheGlobalCTOSurvey × STX NEXT

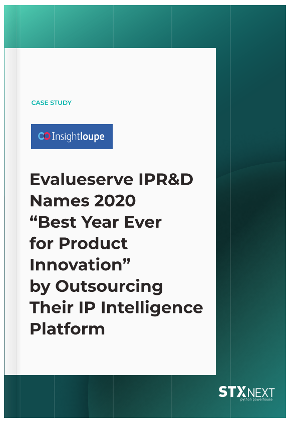
SURVEY RESULTS & INSIGHTS

The Global CTO Survey 2021 Report


CASE STUDY

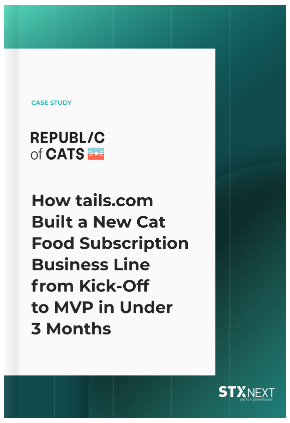
LUMICKS

How LUMICKS Validated and Built a Platform to Help Researchers Analyze Diseases at the Single-Molecule Level


CASE STUDY

Insightloupe

Evalueserve IPR&D Names 2020 "Best Year Ever for Product Innovation" by Outsourcing Their IP Intelligence Platform


CASE STUDY

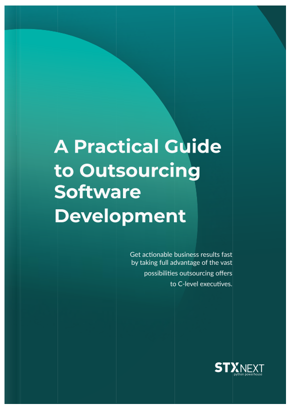
REPUBLIC of CATS

How tails.com Built a New Cat Food Subscription Business Line from Kick-Off to MVP in Under 3 Months

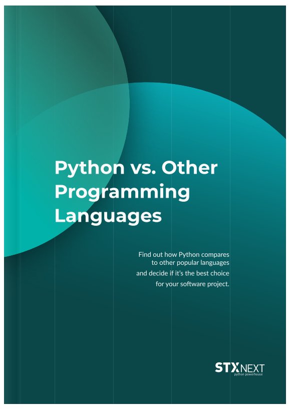
Guides

What Is Python Used for?

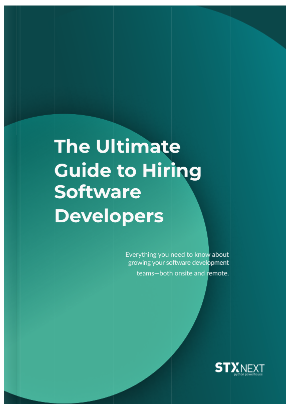
Discover why Python is such a great fit for so many industries and what areas of technology or business it benefits the most.


A Practical Guide to Outsourcing Software Development

Get actionable business results fast by taking full advantage of the vast possibilities outsourcing offers to C-level executives.

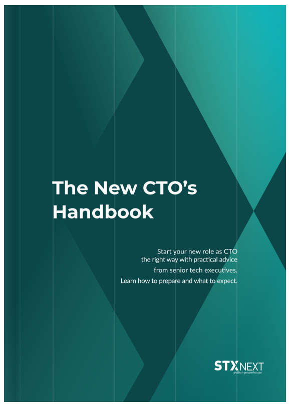

Python vs. Other Programming Languages

Find out how Python compares to other popular languages and decide if it's the best choice for your software project.

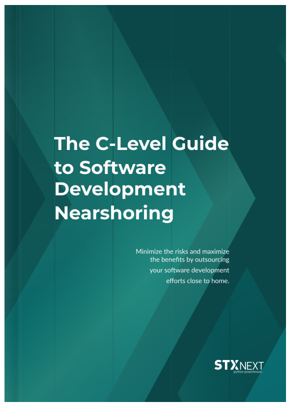

The Ultimate Guide to Hiring Software Developers

Everything you need to know about growing your software development teams—both onsite and remote.

Ebooks


The New CTO's Handbook

Start your new role as CTO the right way with practical advice from senior tech executives. Learn how to prepare and what to expect.


The C-Level Guide to Software Development Nearshoring

Minimize the risks and maximize the benefits by outsourcing your software development efforts close to home.

TECH LEADERS HUB

Tech Leaders Hub: Management & Growth

Become a great leader by leveraging the huge experience of tech leadership experts who have spent years managing and growing teams.

The True Cost of Hiring In-House Developers

From training through benefits to paid leave, the cost of adding new members to your team is never just the salary.

DISCOVER MORE →

Services

Your project is all that matters. We'll build it like it was our own. Whether it's team extension, end-to-end product development, or expert consulting you're after, we'll do everything in our power to meet your needs.

- Python Development
- JavaScript Development
- .NET Development
- Web Development
- Software Testing & QA
- Mobile Development
- Django Development
- Node.js Development
- React Native Development
- Fintech Development
- Machine Learning
- Data Engineering
- DevOps
- Product Design
- Discovery Workshops

REVIEWED ON Clutch ★★★★★ 82 REVIEWS

python SOFTWARE FOUNDATION

TOP PYTHON & DJANGO Clutch DEVELOPERS 2021

TOP DEVELOPERS Clutch POLAND 2021

TOP CUSTOM SOFTWARE DEVELOPMENT Clutch COMPANIES 2021

Tell us about your project

Speed up work on your software projects and outpace the competition.

HIRE US →

Marta Błażejewska
DIRECTOR OF SALES
marta@stxnext.com
+48 506 154 343

Sebastian Resz
HEAD OF SALES
sebastian@stxnext.com
+48 690 433 578

Follow us