Challenge

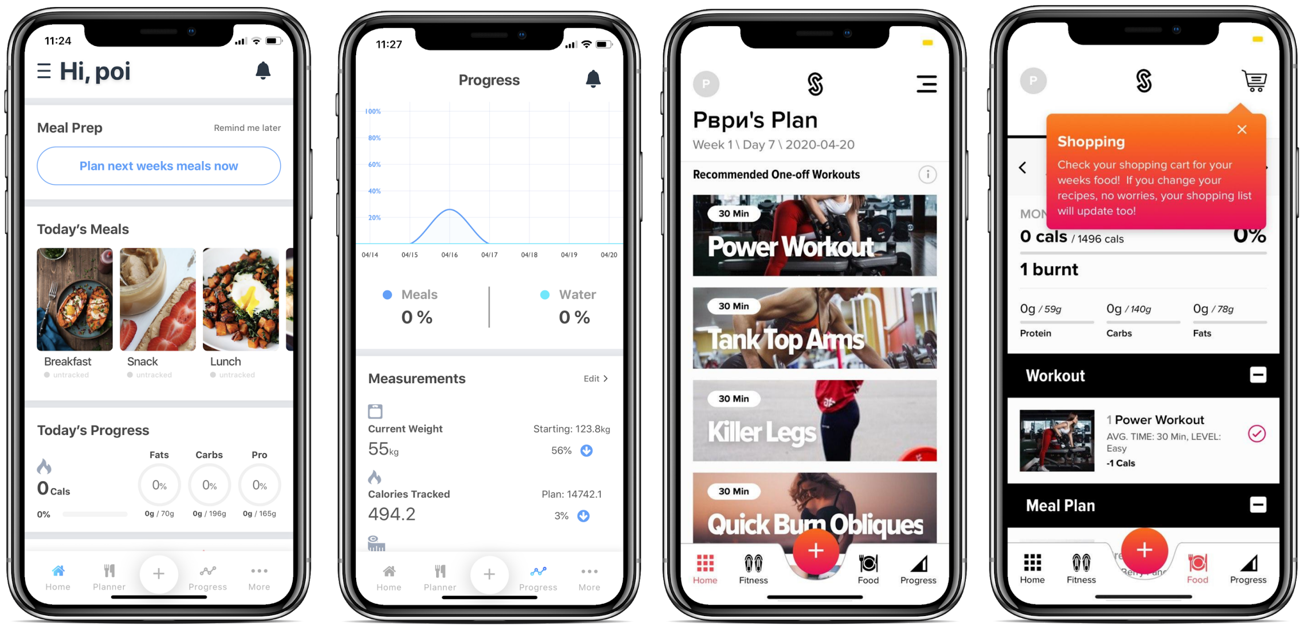
The client contacted datarockets to help their team develop existing mobile applications, Shift and G-plans, developed with React Native. Both applications have been created to promote and help people to lead a healthy lifestyle:

- Shift is an application for exercising and training that offers videos with fitness professionals, tracking food and water consumption.
- G-plans application provides a tailored meal plan and the ability to swap recipe ingredients easily according to their categories. Also, the app allows users to track water and exercises, buy supplements and appoint a call with a nutritionist to fine-tune the meal plan and get personalized recommendations.

Originally, there was a multinational development team on the project which communicated in 2 languages (Russian and English). So we had to set up clear communication processes that would allow us to be on the same page with the whole team.

Another challenge we faced while working on the project was the React Native upgrade. The old version caused the limitation in some heavily-used libraries. In order to apply the up-to-date libraries and receive bug fixes, it was crucial for the team to upgrade the current React Native version to above 0.60.x.

So our challenges list consisted of making the whole team working as one engine by setting up a transparent development process and communication, improving the existing codebase, updating libraries, and adding new functionality.



Approach

As usual, on every project, we start from exploring the codebase, doing small tasks to learn more about the project and setting up the processes for comfortable work.

Processes

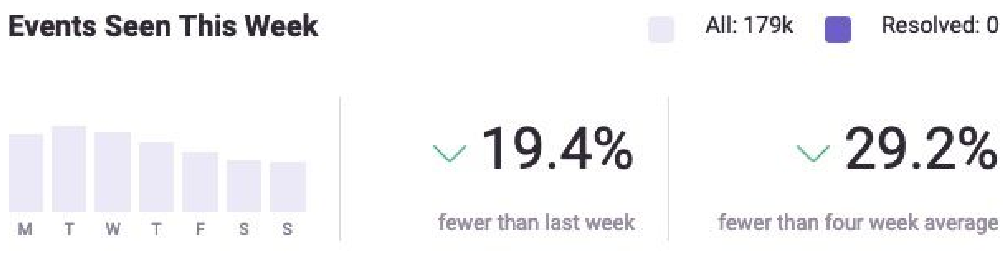
When we joined the project, there was a classic management system that separated developers from product owners. In this way, developers had communication gaps with the business owners, couldn't clarify tasks on their own and get direct feedback. At datarockets, we follow a different approach based on transparency and clear communication with the product team. Our manager started communicating more with the product management team on their end and gathering feedback and updates about the app development plans.

To set up more effective communication between the development and product teams, we began holding weekly demo calls with the client. The scheduled tasks at the current iteration would be superficially tested and ready to be demoed to the client every week. Sometimes, we would find a better way to implement a task spending less time and resources which required a minor shift from the initial task definition. Therefore, having those useful demos not only helped avoid misunderstanding between the client and datarockets team, but also gave us a chance to re-validate the results and apply changes quickly as per feedback.

To improve the quality of the product, we enforced the code review policy so that every change applied to the product had to be approved by one of the developers except for the author of the pull request.

Established crashes processing

When we started working on the project, there was no crash tracking system set up. To have a better understanding of how the app behaves on the end-users' devices, we decided to start tracking and evaluating new bugs on a weekly basis.



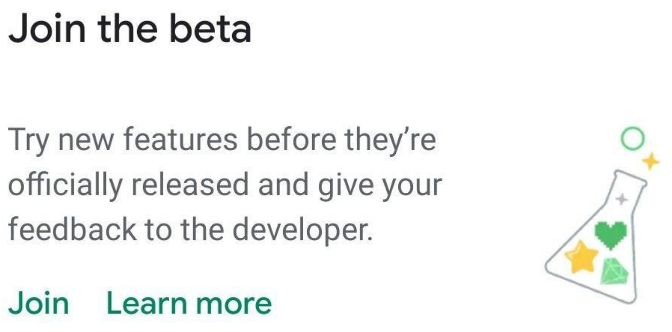
Since we had the main traffic coming on weekends, we decided to check the bug reports every Monday and sort out new bugs by severity, in order to understand how urgently we needed to fix them. For instance, when a new critical bug was uncovered, we gave it the highest priority and delivered the fix via CodePush, a cloud service that allows to deploy mobile app updates directly to their users' devices, as soon as possible. That allowed us to make the app more stable and improve its UX in overall.

QA for regression testing

Before publishing big releases, we suggested involving an external QA team to make a full regression of the app. External QA team could help uncover some unobvious places in the app's UX. Since they had no previous experience working with the project, they shared a fresh perspective on the project and discovered new UX issues and bugs.

Closed beta-testing

One of the experiments on the project was to try to involve active users in the beta testing process. We've used a previously added ability to invite users through the dedicated push notification. Once the app passed the internal QA, we handed it to the subset of real users who had signed up into the beta testing program earlier. Such an approach helped us to gather early feedback from a limited number of users and fine-tune the updates before launching them to public.


Join the beta

Try new features before they're officially released and give your feedback to the developer.

Features and improvements

Working on 2 different applications (Shift and G-plans) at the same time, we covered some of the biggest tasks that have been delivered during the 7-month work period.

Apart from new functionality, the application required "housework", such as updating old libraries, because of the issues we had in the app which should have been fixed in the React Native's latest version. The upgrade progress was difficult because the project had plenty of dependencies with native code and we ought to migrate to the 0.60 version of React Native which brought significant changes in handling native dependencies.

The most crucial tasks we implemented on the project were:

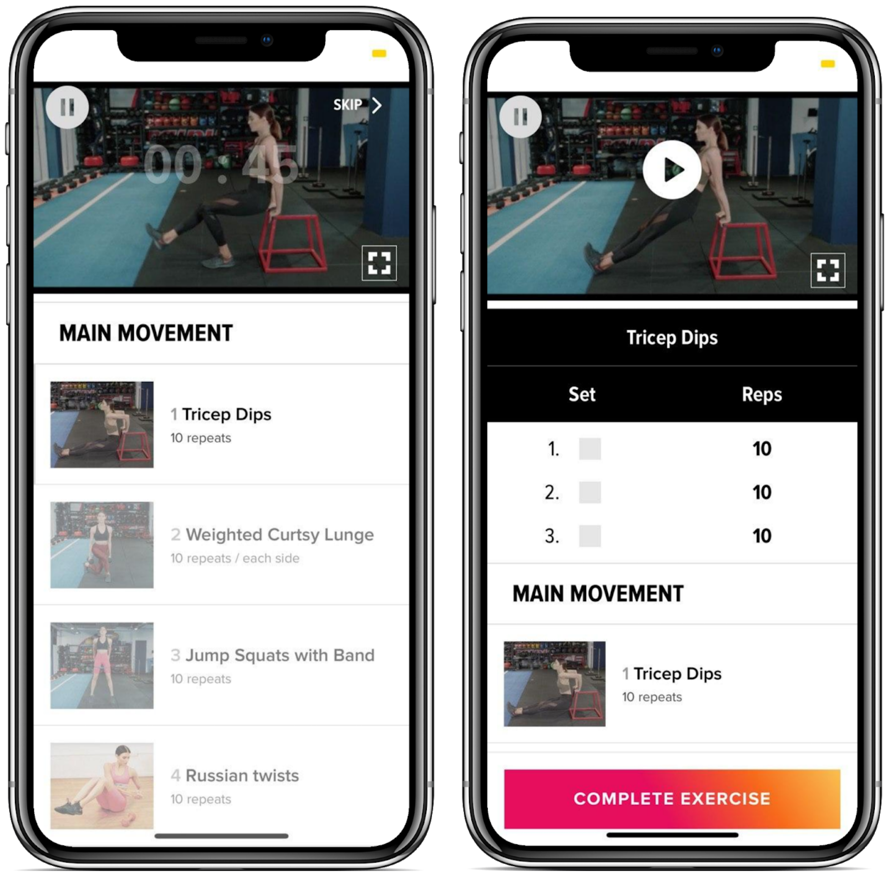
- Workouts caching that made it possible to have exercises without internet connection;
- App's global network listener that notified about a poor network connection;
- The app optimization that reduced its size ~ 3 times;
- Interactive on-boarding with a chatbot.

Workouts caching implemented

It often happens that the connection speed is much lower than expected or the user may not have internet on their device at all. That’s why the product management team decided to implement workouts caching so that the users could enjoy working out w/o internet connection.

We decided that we needed to give users the opportunity to download a set of exercises (workouts) on the phone and cache requests without internet for further synchronization of cached data with the server.

To do this, we allocated the video files for exercises to a separate service that allowed users to download them to the device (because in normal mode, we used video streaming). Next, we added the logic for saving requests to the device. Each cached request had its own unique number and timestamp, which allowed us to limit the data lifetime and optimally used the resources of the user's mobile device. Each time when the user opened the app, we checked the relevance of the data and removed the obsolete ones.



App’s global network listener

Because every part of Shift was designed to be always in-sync with the server, we had a lot of network calls to the API. If the device did not have an active internet connection, our network requests obviously failed.

To eliminate such cases, we wrote our own module to listen to network changes and react accordingly. So, if a user was watching a workout video and the internet disappeared, we notified the user about his bad network connection and asked him to retry later. They also had an option to download all the data to the device and use the app even in airplane mode.

Reduced the size of Android app 3 times

When we started working on G-Plans, one of our concerns was that the Android version of the app was too heavy. The app’s download size was 100+ Mb that influenced the time to download it from the store and install on the device. This could be the reason for losing new users and revenue. The smaller app, the higher chance a user would eventually download it.

To address this issue, we decided to upload the builds to Google Play using AAB format instead of APK. By using the AAB format, we delegated part of the build process to Google and allowed them to apply certain optimizations on their side. In addition, we started optimizing the images we were using in the app and eliminating unused code and resources more strictly to reduce their impact on the resulting app bundle size.

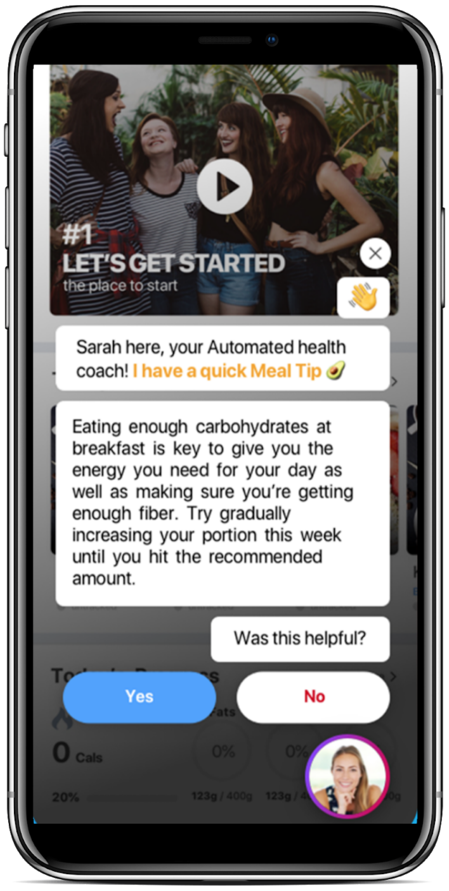
In result, the app’s download size reduced from 100+ Mb to 30-40 Mb depending on the device’s architecture and screen size.

Chatbot

Our on-boarding process was one of the most important parts in the application. More user-friendly process drives better first impressions and a higher chance to convert a user into a customer.

On G-plans, some users had problems understanding how to use the app that might have led to abandoning the app at the early stage. To address this issue, we decided to implement in-app tips shown via the chatbot and push notifications. The goal of the in-app tips was to guide the user through the parts of the app by showing them videos with appropriate guidances. Meanwhile, the purpose of notifications was to re-engage users to use the app if they had abandoned it for some time.

To implement the in-app tips, we re-used the already existing but unused chatbot with minor UI and UX tweaks. Users were able to understand and discover more features of the app that resulted in a smooth onboarding process.



Sarah here, your Automated health coach! I have a quick Meal Tip 🥑

Eating enough carbohydrates at breakfast is key to give you the energy you need for your day as well as making sure you're getting enough fiber. Try gradually increasing your portion this week until you hit the recommended amount.

Was this helpful?

Result

The project has been put on hold because of COVID-19. By that time, our team managed to improve the product development processes that included code review, demo calls with the client, crashes processing and many more.

We used the best React Native development practices to implement a variety of different features and optimizations (described above) to improve the value and UX of the applications.

Technologies

React Native, React Native Navigation, React Native Config, Redux, MobX, IAP, Intercom, Facebook SDK, Firebase, Google Fit, Apple Health, Mixpanel, AppsFlyer, Amplitude, *Bitrise*