Set your Goals and Milestones

Achieve the bigger goals by setting small milestones and deliverables for yourself.

For each skill that you would like to improve, determine projects and activities that measure your knowledge. Launch projects that will put your new skill into action. Start with *the why*, and *the what* will fall into its place.

Make these projects useful and practical on some level. Building an app for real users is better than creating something just for yourself. Teaching others, blogging, shooting a youtube video where you explain something are all great ways of measuring your progress.

Make sure your goals and milestones are tangible. Don't come up with a goal like "I want to learn ReactJs by understanding the example in the book I purchased." Most likely, you would spend most of your time trying to learn nitpicky details that you would forget within a week.

Many books and courses are excellent, I have learned from many of them myself too. However, if you finish a book without creating something new that goes beyond the content of the book, your knowledge of the subject will fade as time passes. If you push your boundaries beyond the theory, you will learn some skills that may stay with you for the rest of your professional career.

A reasonable goal may be the following:

Example goal: Create a single page application in React and ES6 that enables software developers to track their productivity using the Pomodoro technique and other useful productivity hacks such as (...).

Feel free to refine the exact specification later.

Your projects also keep your work focused. You won't waste time on unimportant details. You will learn what matters. When I wrote ES6 in Practice, I had two guidelines in mind:

Practice, I had two guidelines in mind:

- I wanted to tell you just enough to get started with solving exercises.
- These exercises tell you stories and challenge you with a continuously increased difficulty. You will not only feel good after solving them, but you will also know that you can solve problems on your own. Some of the exercises include concepts other books would give you in a theoretical section. By solving such an exercise, you prove yourself that you are capable of learning new ideas once you need to use them. This should give you confidence in your abilities and deepen your understanding at the same time. Worst case: you read the solution and learn as much as you would have from other tutorials.

Be specific about the time you are willing to invest in your side projects. Many people, sometimes including myself, tend to underestimate the time needed for a tech project often by a factor of 2 or 3. If you figure out that your lifestyle does not support creating a complicated side project, you have some life decisions to make. You will either change your lifestyle or career plan, or you will have to create a different learning plan for yourself.

Your goal may be quite simple. For instance, if you learn React, and you already have a blog, you may want to set yourself the following target:

1. Create a boilerplate for a React application, and upload it on GitHub
    - Skills: ES6, React fundamentals, Webpack, NPM, BabelJs
    - Deliverable: React boilerplate on GitHub
2. Use your React boilerplate to implement a Todo application
    - Skills: more in-depth React knowledge
    - Deliverable: a refined React boilerplate based on the learnings of creating an application
3. Write a blog post about your learnings
    - Skills: more structured React knowledge
    - Deliverable: blog post

You may set a big goal for yourself, and you deliver your solutions incrementally in multiple milestones. For instance, to realize a productivity

app, you might want to create:

1. an abstract productivity model to implement
    - Skills: Pomodoro technique, kanban, productivity research
    - Deliverable: productivity model, documentation, software specification
2. a responsive website with components
    - Skills: Bootstrap or pre-made web components
    - Deliverables: the skeleton of the website with static data
3. automated testing and mocking are most likely needed for development
    - Skills: Mocha, Chai, SinonJs, API endpoint design
    - Deliverables: automated testing framework connected to the application, and a way to intercept and fake API calls. Design your API endpoints.
4. a React layer
    - Skills: React, NPM, Webpack, BabelJs
    - Deliverables: none, as it makes perfect sense to do D and E together
5. you may want to manage the application state with a library like Redux
    - Skills: Redux
    - Deliverables: a fully working application with a fake server
6. you may need some server-side code and a database
    - Skills: NodeJs, MongoDB
    - Deliverable: a fully working solution

As you can see, in these six milestones, you may be learning an avalanche of technologies. Set some target dates for yourself by estimating the amount of work needed for finishing the application.

Make sure you don’t treat yourself too hard for missing your target dates. The importance of measuring time is to figure out how good you are at estimating your development velocity.

your development velocity.

A side project like this will keep you busy for a long time. However, you will reap the rewards once you can demo your application. If your application is useful enough, you may even be able to monetize it.

Preparing you to monetize your application is outside the scope of this book. To develop an application for making money, you will have to test what your users want continuously and aggressively modify your product. This process does not create an optimal learning experience for you, so let's continue focusing on your career for now.