Oblikovanje i primjena animacija u dizajnu korisničkih sučelja

Ćurković, Dora

UNIVERSITY OF ZAGREB

FACULTY OF ORGANIZATION AND INFORMATICS

V A R A Ž D I N

Dora Ćurković

DESIGNING AND USING ANIMATIONS IN VISUAL INTERFACE DESIGN

MASTER THESIS

Varaždin, 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
V A R A Ž D I N

Dora Ćurković

Identification number: 0016114854

Study program: Business Systems Organization

DESIGNING AND USING ANIMATIONS IN VISUAL INTERFACE DESIGN

MASTER THESIS

Supervisor:

Assoc. Prof. Dijana
Plantak Vukovac, Ph.D.

Varaždin, September 2022.

Dora Ćurković

Izjava o izvornosti

Izjavljujem da je moj diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

---

Declaration of Authenticity

I declare that my master's thesis is the original result of my work and that I did not use other sources other than those mentioned. Ethically suitable and acceptable methods and techniques were used for the creation of this work.

The author confirms this by accepting regulations in the FOI-Thesis system

---

Summary

The topic of this thesis is the design and use of animation in visual user interfaces. The goal is to explain the thought process behind designing an animation, prototyping and visualizing, and implementing the animation. This thesis covers the principles of traditional animation and their application to visual user interface animations, as well as their use, example animations from real-life cases, microinteractions, the handoff process of animation, and the tools of implementation. The focus is on explaining the principles of UI animations and how they translate to animation properties in prototyping and implementation. The practical part of this thesis covers the design and implementation process of selected animated work from a project done during an internship.

Key words: animation, UI animation, interface, UX/UI, animation principles, prototype, handoff

Contents

1. Introduction

Animation is defined as changing some property over time. Often it is used interchangeably with motion, and often times it is. However, all motion is animation, but not all animation is motion. Motion requires movement, a change in positions, whereas animation can lead to a change in appearance, such as opacity or color change of an object.

User interface (UI) animation is an umbrella term for all of the interface animations, from screen transitions, microinteractions on input fields, to animated graphics on a 404 error screen, across multiple platforms - web, desktop, tablet, mobile and even products used in real life, such as smart home speakers. There is little that cannot be animated, as nowadays implementing animations is easier than ever, with libraries and parsing services and bridge the gap between the design and implementation. The main question is what has to be animated and how.

This topic is an extremely important element of User Experience and User Interface (UX/UI) design. We see more and more designers specializing in motion design, and I am one of them. It is a powerful and purposeful tool that can make of break an interface. It leads the way and grabs attention and it has come a long way from the beginning. It requires both design and development perspectives to be implemented efficiently, and that is what we will talk about in this thesis. What are the principles of animation, where can we use it, what is their design process and how are they implemented? [1, pp. 21-31]

2. The Case for Animation

In a long history of animation, moving graphics and images have always captivated audiences. Its purpose switched from only entertainment, so now designers often use it to give website visitors a satisfying and interesting experience.

Audience has changed its attitude toward the screens. The expectations of what using an interface should feel like have changed as a result of the rise in popularity of smartphones and tablets [1]. These tiny computers are now part of an everyday life, and thanks to their gestures, depth, and animation, their user interfaces seem more dynamic and engaging.

A sophisticated, modern, and reliable design is increasingly defined by its use of animation. A well-designed animation is one way to achieve your design goals of creating an interface that feels contemporary and sophisticated. It's a significant factor in why so many designers have begun to take animation seriously.

The animated components of a website or app are frequently seen as merely optional extras for aesthetic appeal. However, when used properly, animation can be much more than that, providing significant advantages to the website and helping to create an excellent UX design. However, it is something to thread lightly with. Some animations end up creating problems, instead of solving it. Bad animation can distract users from their goals and clutter an interface, and too much of badly implemented animation can slow down a website. [1, pp. 62-80]

2.1. Purpose of UI animation

Use of animation in interfaces has become a norm. In contrast, interfaces without it feel dull, unintuitive and dated. Interfaces with animation feel sophisticated, modern and pleasing. Animated interface is easier to understand and follow. Animating transitions between the two states reduces cognitive load by animating element's movement and making the change visible on screen, so that the users do not need to keep track [1, p. 10]. This way users can focus on the more important things, look at the content and go through the flow effortlessly.

Animation is an excellent communicator. It leads us, shows us areas of interest, tells us a story of an object. Animations will be interpreted, and it is in designer's hands to make sure they are interpreted in the right way. What message will they tell? Will our interface feel connected, even between the contexts and different platforms. Will our web and mobile interface give off the same feel, same recognized characteristics? Designing a cohesive and connected interface is the goal and animation is one of the tools to tie them together. Animation

reinforces commonalities among similar elements and hierarchy, and grab attention like nothing else. It adds an element of time into attention grabbing techniques by showing us which elements are the most important at the time. Through animation we also connect the brand with our interface. [1, p. 5]

Animation is a powerful tool. Animation tells us the personality of a brand, guides us through the interface and makes using it enjoyable. That is, if done correctly.

2.2. Caution with UI animation

Human brain is designed to notice movement, thus creating, and implementing animations into our applications is of at utmost importance. Animation is supposed to provide the feeling of an actual and real interaction creating a level of feeling and perception close to what people have when interacting with real physical object [2].

Cognitive load is a term describing the amount of mental effort required to complete a task. Users when using applications are flooded with information and giving them more information to process by adding flashy animations or overusing them will just result in users stop using our application. Designers job is to create a user interface that is easy to use, frees up mental capacity of a user to complete a task and does not affect loading speed. [3]

What happens if animation is not done correctly? Wrong implementation can be detrimental for the success of our application. If we overuse animation and put them everywhere inside our application, soon that same application will have performance issues, especially if animations are not scaled and created properly.

An article from the Nielsen Norman Group on animation attention and comprehension [4], its referenced that movement in the peripheral view can attract the user's attention quicker due to our biology of recognizing potential danger in the periphery. However, due to number of popups and advertisements which are placed on the sides of the websites we use every day, users got used to peripheral distractions and they don't pay attention to them anymore, which forces designers not to follow those patterns when creating an engaging user experience.

Animations can be a great tool to help attract users and keep existing ones using our application, they are great tool to improve overall user experience. However, today users are overloaded with the information, and we must be careful how we choose to implement them. They can add up fast across the application thus making our application slower to user and more difficult for user to navigate. [2]

3. Principles of animation

Interestingly, Disney laid out the groundwork for the animation that we know today. In 1981, Disney animators Frank Thomas and Ollie Johnston laid out the twelve principles of animation in their book "The Illusion of Life: Disney Animation" [5]. This became commonly known as the "Bible of animation," a group of teachings for any professional animators.

Over 40 years later, many of these fundamental concepts are still applied in classrooms and studios worldwide. While new and different ideas have been incorporated into animation as technology and other industries have developed, the fundamentals are still present in modern films and web design.

3.1. The twelve principles of traditional animation

So, what are the twelve principles of animation? Each is vital to the animation process and adds up to the quality of the animation. Following are the principles defined in "The Illusion of Life: Disney Animation" by Frank Thomas and Ollie Johnston. [5, pp. 48-70]

Squash and stretch is the first principle and perhaps the most fundamental one. What happens to the ball when it hits the ground? The motion's force flattens the ball, but because an item must maintain its volume, it also spreads as it hits the ground. Alternatively, a bicep when it extends and when it tightens. Or our mouths when we chew. This is referred to as squash and stretch.

Anticipation is the second principle, preparing us for the main action. When a specific move precedes main actions, it prepares the audience for what is about to happen. This can be a little physical action, like swinging back your foot before kicking a ball, or a change in expression, like a character looking off-screen to anticipate someone's arrival. Few actions in the real world come without some anticipation.

Staging, being the third principle, is extensive. It is used in many ways, but the goal is clear. Johnston and Thomas defined it as "The presentation of any idea so that it is completely and unmistakably clear" [5, p. 54], regardless of whether the idea in question is an action, a personality, an expression, or a mood. It effectively moves the plot along by focusing the audience's attention on the scene's key elements.

Straight ahead action and pose-to-pose is the fourth principle. They are essentially two approaches to animation. With the ***Straight Ahead Action***, the animation works from the first frame to the last. This is preferable for creating realistic action scenes since it gives the appearance of movement a more fluid, dynamic feel. However, maintaining proportions and

developing precise, believable stances along the way might be challenging. For dramatic or emotional scenarios, where composition and relationship to the surroundings are more crucial, ***Pose-to-Pose*** works better. In that case, the animator figures out the keyframes and then fills out the rest of the frames. ***Straight ahead action*** is widely used in computer animation, as the problems with proportions and removed.

Follow through and overlapping action is the fifth principle. In contrast to anticipation, which is the start of an action, follow through is the completion of an action. Rarely do actions come to an abrupt, complete stop; instead, they frequently continue past their intended termination point. Whereas with overlapping, the movement appears more natural when an action doesn't stop entirely before another. Continuous movement makes movement less stiff and adds life to the animation.

Slow in, slow out is the sixth principle and the one most known to those who design and implement motion on visual interfaces, albeit under a different name - ***Ease in, ease out***. It deals with the fact that objects need time to accelerate and slow down. When a person starts running, they accelerate to reach full speed. When wanting to stop running, they slow down until they come to a complete stop. In classical animation, this was created by having more frames near the animation's beginning and end and emphasizing greater speed with fewer frames.

Arcs are the seventh principle. They represent living beings' movement. Think of a rotating joint when using hand gestures—this differentiates living beings from mechanical objects that move in straight lines. Arcs often flatten out in forwarding motion and broaden in turns as an object's velocity or momentum rises. An excellent real-life example is car racing, where a fast-going car would need a much bigger arc to make a turn than a slow one.

Secondary action is the eighth principle and refers to all secondary animations that make the main one more realistic and detailed. A person swinging their arm while walking or breathing heavily while running makes those actions more realistic. Secondary actions support the main one and do not take away from it. However, sometimes they can go completely unnoticed, so it might be a good idea to place them before or after the main action.

Timing is the ninth principle and refers to the number of frames that make up an action, which translates into the speed of the action. It gives meaning to the movement. It can determine weight, size or emotions or objects, and character. The heavier the object, the slower it moves. A bigger object usually means more weight, so it also moves slower. A character that moves slowly on the screen might be tired or sad. Timing determines how audiences will perceive animated elements.

Exaggeration is the tenth principle of animation. There is a fine line between making something exaggerated and unrealistic. The idea is to emphasize the action so the audience will understand it. If a character is angry, make him furious. Balance is extremely important with the principle.

Solid Drawing is the eleventh principle and refers to the artist's ability to draw and understand the proportions, light, shadows, anatomy, etc. Nowadays, animators draw less and less, but it is still essential to understand the principles to add to computer animation.

Appeal is the twelve and last principle of animation. It refers to the character's likeability and appeal to the audience. The principle is often misinterpreted, but it actually has to do with a pleasing design, simplicity, and communication. Poor drawings, clumsy shapes, and awkward moves add to the lack of appeal of a character. It is charisma in animation. [6] [7]

It is evident that these animation principles are centered around the old-fashioned, hand-drawn animation for cartoons and animated movies. However, some might still have relevance in today's computer animation.

3.2. UI animation principles

So, do the twelve basic principles of animations apply to UI animations? They appear to be solving a different problem. "Slow in, slow out" is known as easing and is extremely important in UI, and "Anticipation" is incorporated in microinteractions and button states, but some other principles do not translate as well.

There are multiple collections of principles by multiple experts in the field. Issara Willenskomer is an expert on the subject of animation for user experiences and runs the "UX In Motion" website. He argues that, while some principles of animation from "The Illusion of Life: Disney Animation" still stand, they do not accurately describe UI animations [4]. He defined "The twelve principles of UX in motion," which is somewhat of a manifesto on creating usability in motion. He divided principles into five categories - timing, object relationship, object continuity, temporal hierarchy, and spatial continuity. Principles he defined are easing, offset & delay, parenting, transformation, value change, masking, overlay, cloning, obstruction, parallax, dimensionality, and dolly & zoom. [8]

Val Head, a senior design advocate at Adobe and a web animation expert, goes through some of the old and defines some new principles in her book "Designing Interface Animation: Meaningful Motion for User Experience." We'll take a look at how some of the principles of traditional animation translate to the UI animation.

3.2.1. Easing

"Easing" is the first principle. Disney referred to it as "Slow in, slow out." Real-world objects typically pick up speed as they move, move at their fastest midway through their movement, then slow down before stopping. In order to depict the beginning and end of each movement, which are displayed frame by frame, there is a slowing of velocity or more closely spaced frames. An illustration of this is shown in the figure below (see Figure 1).

Figure 1: Animation Principle - Easing (according to "Designing Interface Animation" by Val Head)

Despite the fact that we are animating for a visual interface, it is important to draw inspiration from the actual world so that your UI animation looks natural and makes sense. Easing plays a massive role in how users perceive animations. In UI animations, easing is usually categorized into four types - linear, ease in, ease out and ease and out.

Linear refers to the absence of easing, meaning there is no speeding up or speeding down during the course of the animation. The speed is linear and constant. It is very robotic and unusual behavior of an object, so it is not a popular choice for UI animation.

Ease in refers to the acceleration of an object at the beginning of its animation. This feels more natural. However, the end of the animation still comes abruptly as the object stops at its peak velocity. In UI animation, it can be used for elements exiting the screen, as in that case, the users see only the beginning of the animation. Material Design calls this ***Accelerated*** easing.

Ease out refers to the opposite of ease in type. The object slows down at the end of its animation but starts abruptly at its peak velocity. This makes for a great entrance animation in the UI context, as the object slows down naturally before coming to its place. Material Design calls this ***Decelerated*** easing.

Ease in, ease out combines both of the two previously mentioned types. The object accelerates into the action and slows down before stopping. It is advisable to be careful with this type of animation, as it can feel sluggish for users. Keep it short.

However, there are endless possibilities on how easing animations can look. Material Design also defines types such as ***Standard***, in which the object slows down more than it accelerates to bring the emphasis to the end of the transition, and ***Emphasized***, which puts

even more emphasis on the end by prolonging it. Custom easing curves are always an option as well. This is visualised in figure below. (see Figure 2) [9] [10]

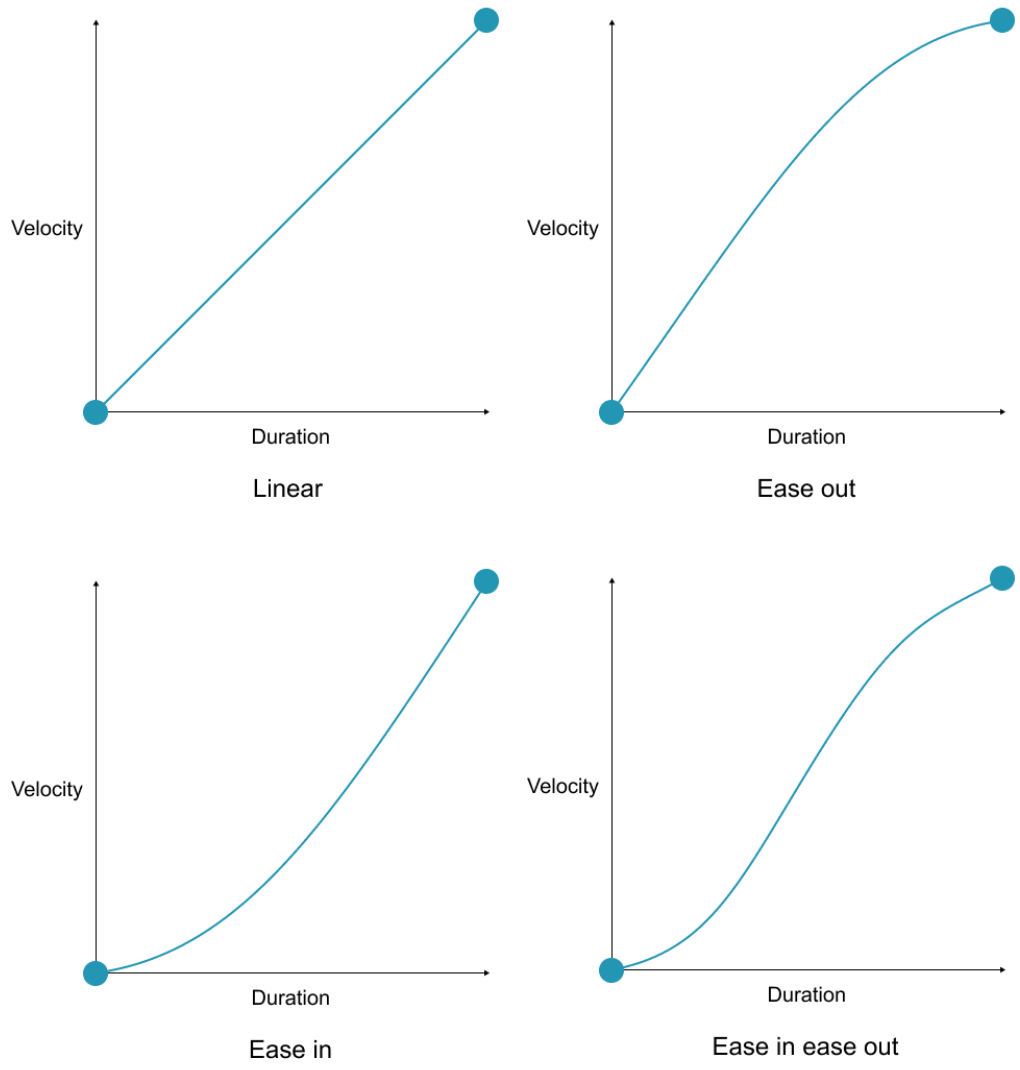

Figure 2: Animation Principle - Easing (according to "The Basics of Easing" by Paul Lewis on Web.Dev)

3.2.2. Timing

The second principle is ***Timing***. Timing is the length of a specific movement or change or the time it takes for an activity to occur. In implementing UI animations, animation timing is usually the duration of the animation. Your animation should run quickly, but not too quickly. An animation that is too quick leaves users confused and doesn't allow them enough time to take it in. On the other hand, slow animation, especially with transitions, makes users unnecessarily wait for it to finish before they can move on to the next action. Transitions should happen seamlessly. As Head says, "The best way to know you've got a good duration is to try it out and see how it feels." Prototyping and trying out animations are a way to go. Finding a good timing for animations is more of an art than a science. A good range to begin with your UI animations is 150 to 500 milliseconds. The larger the animation, the longer the timing. However, that should be a guide, not a rule. How timing can help animation feel more natural is shown in figure below (see Figure 3). [1] [11]

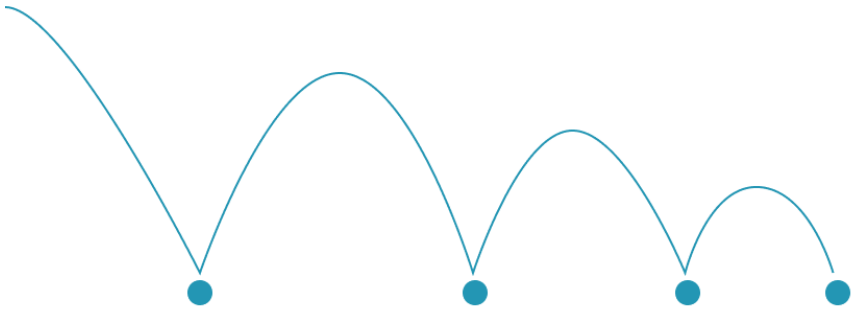

Figure 3: Animation Principle - Timing (according to "Designing Interface Animation" by Val Head)

3.2.3. Spatial orientation

Spatial orientation is the third principle. Spatial orientation is about showing where something is in a virtual space, even if we cannot see it. Using animations to show where UI elements are, even when they are off-screen, can be especially useful in responsive design. In visual interfaces, this can be used in navigational drawers that slide into and out of the screen seamlessly on the horizontal axis. It is a simple way to let the users know where the navigation came from and to let them know it did not just disappeared but rather just moved out of the way (see Figure 4). [1] [11]

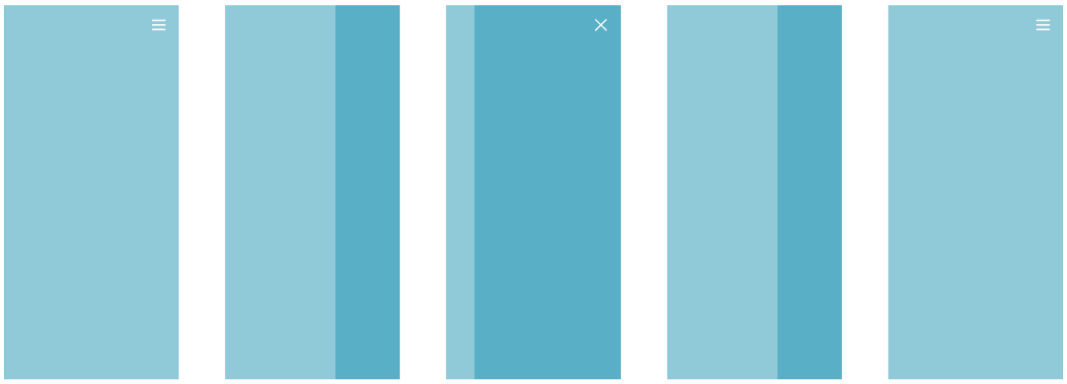

Figure 4: Animation Principle - Spatial orientation (according to "Designing Interface Animation" by Val Head)

3.2.4. Directing focus

The fourth principle is ***Directing focus/attention***. This principle refers to the animation's ability to move an object at the top of the hierarchy. It leads the eye around the interface. Animation can have much more weight than, for example, color, so it is very important to think carefully about which object we want to draw users' attention to.

This is particularly useful in microinteractions to serve the purpose of reminding the user to finish taking the test before the time runs out with a timer or to notify them their download is finished by bouncing the icon up and down. Both of these create hierarchy and draw the attention of users. [1]

3.2.5. Visual continuity

The fifth principle, ***Visual continuity***, tackles the need for continuity in screen transitions and animation of the same elements.

When talking about screen transitions, it is best to avoid harsh cuts from screen to screen. Keeping important elements that appear on both screens in view during the transition is a great way to ensure continuity and help users connect the two screens. This is often used in transitions between elements that provide information about the same subject but with different detail.

Visual continuity can also be used to animate elements of the same types. Animating elements of the same type, in the same way shows the users that they are the same thing. [1] [11]

3.2.6. Clarity

The last, sixth principle we will cover is ***Clarity***. Clarity keeps animations clear and easy to follow for the users. It is important to look at all of the UI animations holistically and think about how they work together on the interface.

Not animating too many things at once is one of the ways to achieve clarity. When animating too many things at once, we confuse and distract the users, which is the exact opposite of what we want to achieve. Asking yourself if the animation has a purpose is a good way to avoid this. If there isn't a clear purpose for the UI animation, we can probably go without it. When animating multiple things that trigger the same action, try partly overlapping their animations and using delay. Having them animate all at once might be hard to follow.

Another way to achieve clarity is to have animations follow the same direction. When two objects cross each other while moving into place, it is hard for the users to follow what is happening.

Lastly, clarity depends a lot on the context in which the UI animation appears. UX and UI principles apply in designing an interface in which the animation will appear. [1] [11]

3.3. Spotify UI animation casestudy

Spotify music player is one of the most used music players in the world. With over 200 designers on their team, they give great attention to how Spotify as a brand is recognized and perceived. After a year of building their design system, they introduced Encore - a collection of color, type styles, motion principles, spacing, and guidelines, along with design tokens.

This style is what makes Spotify recognizable. Regarding motion, Spotify has an interesting approach. Besides following basic principles, they also have their own. Their motion principles make Spotify experience feel fluid and expressive. Their objective was to incorporate rhythm and the beat of the music, which is why UI animation at Spotify is characterized by pulses, a glow of light, and color. [12] [13]

Move with purpose is one of the three main principles designers at Spotify use to guide them in creating animation for the mobile app. The animation should have a purpose; it invites interaction and provides orientation.

Next is ***Provide feedback***, which refers to the fact that animation can be an excellent tool to provide feedback on a user's action. It shows the users something is happening and that an interface is responding to their actions.

Add delight is the third principle and refers to making our interface feel alive and make users delighted about using it. Attention to detail, even with the smallest animations, can have a huge influence on a holistic experience for users.

A great example of the three of these is the heart animation for the like action (see Figure 5). They explain that they wanted to replicate a feeling of excitement when finding new music, something a simple change of state from an empty to a full heart could not do. They created storyboards to demonstrate their ideas and aim for a duration of a maximum of 500 milliseconds. Adobe After Effects was used for animating and Lottie for the implementation. We'll talk about both programs' purposes more in-depth later on.

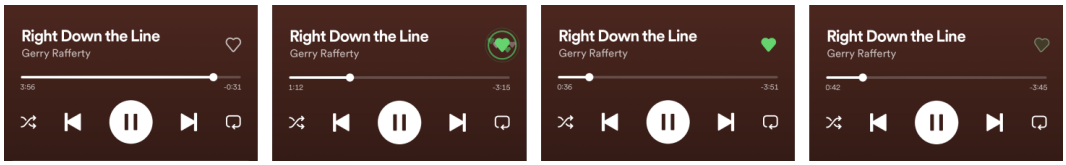

Figure 5: Spotify like interaction

Through tracking the success of the rollout, they learned that it increased users' interaction with the button and so helped them with their recommendations algorithm. Thoroughly thought through animations not only lead to a better user experience but can have a positive domino effect on metrics. [13]

Besides the ones they have talked about through their blog, Spotify also uses a lot of UI animation principles defined by the industry, some of which we have covered previously. ***Directing focus*** is one of these principles, which Spotify uses to let the users know and bring attention to which song is currently playing out of the playlist. This is done by adding an animated sound graph icon, along with a change in text color, for the song currently playing (see Figure 6).

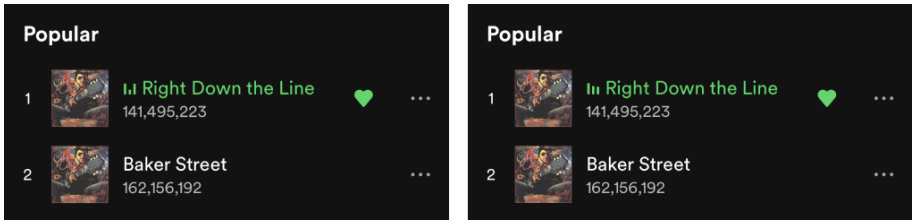

Figure 6: Spotify currently playing animation

Directing focus can also be seen on a desktop dashboard, where playlists are highlighted on hover (see Figure 7 and Figure 8). With a change of color and a Spotify green

play button, users are notified of an available clickable action that will result in playing the selected playlist.

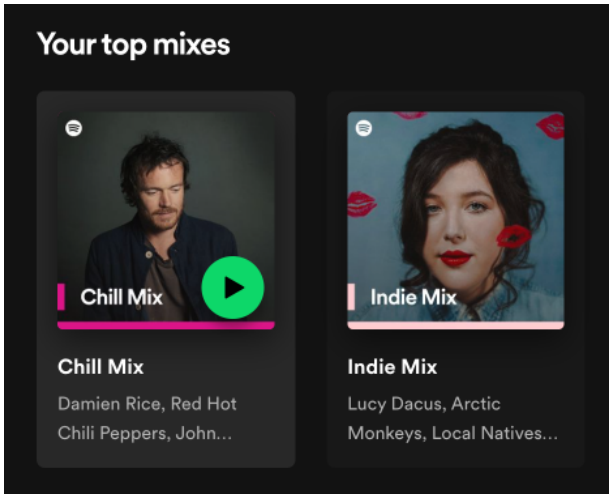

Figure 7: Spotify hover animation

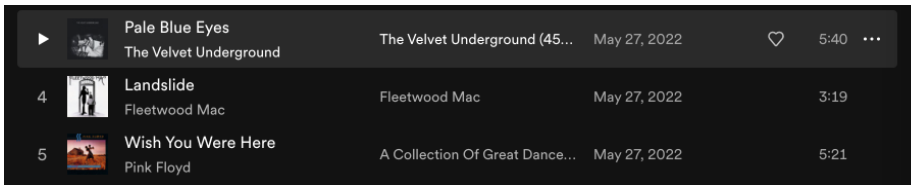

Figure 8: Spotify hover row animation

Spatial orientation is used as an alternative way of liking and disliking songs. Users can swipe the song's card to do the action, letting them know this option came through horizontal movement from the right corner of the screen (see Figure 9).

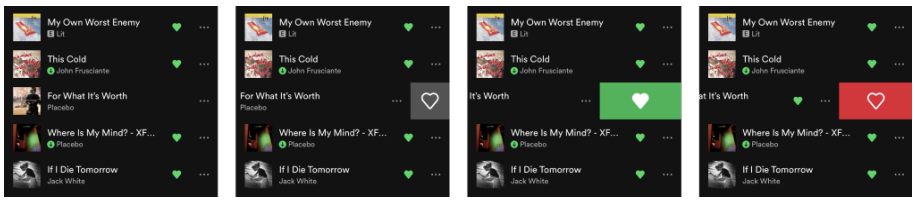

Figure 9: Spotify like animation

It is also used in screen transitioning when opening screens deeper into the hierarchy (see Figure 10). The transition is not instant and is clearly visible where the screen came from.

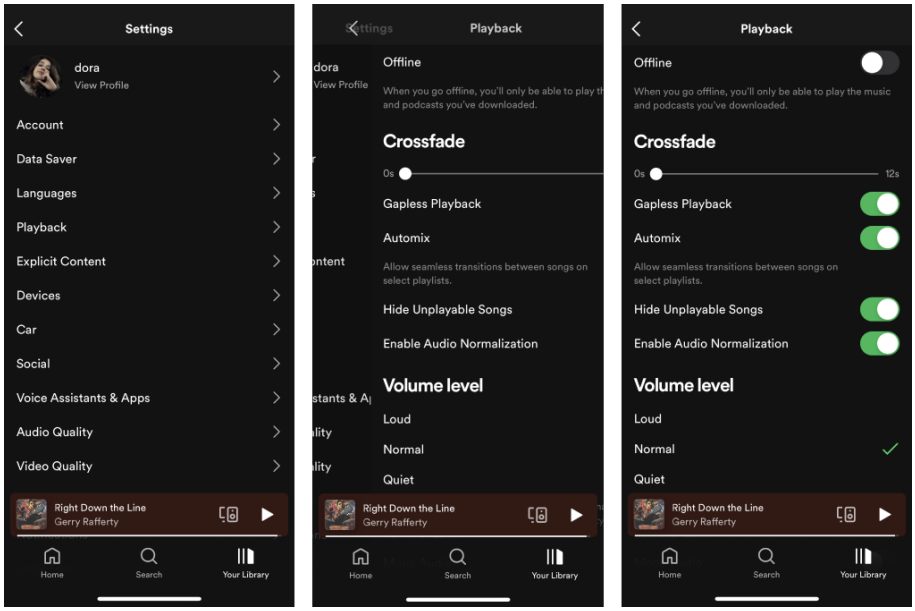

Figure 10: Spotify screen transition

Visual continuity is shown in a transition from a currently playing overview element to a fully expanded detailed element with all of the information (see Figure 11). Elements are positioned and transitioned in a way that fakes expansion from one element to the other, connecting their two contents together.

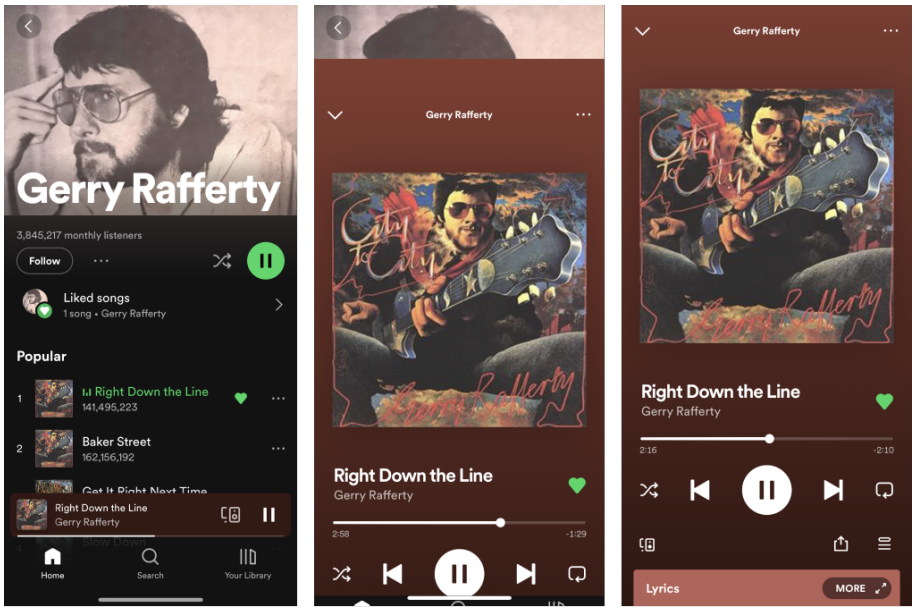

Figure 11: Spotify current song view transition

4. Microinteractions

Microinteractions are one of the most common UI animations. They are not a decoration but a functional, interactive part of an interface. They make engaging with products easier and more intuitive and have become so routine that often we don't even register them. They create a focus of attention and save time without distracting the user.

4.1. The structure

The simplest way to look at them is as a trigger-feedback pair. Microinteractions have four parts - triggers, rules, feedback, and loops/modes (see Figure 12). A trigger is a user's action or a result of a system change, and it starts an interaction. Rules determine the flow of the microinteraction. Feedback is a way to let the users know there's change. Loops and modes, in which loops determine the timing of the microinteractions, and modes deviation in the rules that should be used very sparingly.

Figure 12: Microinteraction parts (Medium, [14])

4.2. Types

Some of the most known microinteractions are:

- Input field status changes - input fields change their state depending on users' interaction with it
- Scrollbars - visual feedback to users changing position within an interface
- Buttons - if a button changes state when clicked on or hovered by the user, a microinteraction happened
- Notification - triggered by the system, this notifies the user that they have a new notification

- Volume control - visual feedback to the user changing the volume
- Progress bars - systems way of notifying users of the progress of a task

As it is evident, not all triggers are manual. In fact, depending on the nature of an interface, most triggers are system triggers, and they engage when certain conditions are met. These conditions can be errors, incoming data, other people interacting with users' profiles, task statuses, and many more.

A well-designed microinteraction can influence the holistic experience greatly. They are simple, brief, and placed highly contextually inside the interface. They can be of many uses, either by encouraging engagement, communicating progress, displaying system statuses, providing error preventions, or communicating a brand. They keep users engaged and informed, and an interface would lack without them. They are a way for interfaces to communicate with the users. [15]

Loading is one of their roles. This role fits progress bars, circular loading indicators, countdown elements, and more. These elements are communicating progress to a user, letting them know it is still working on a task, and motivating users to wait. The most common ones are loading indicators when users pull to refresh a page (see Figure 13). This shows them the system acknowledged their action, is working on delivering the result, and makes them wait for a little while longer. Users want to know the status of the system.

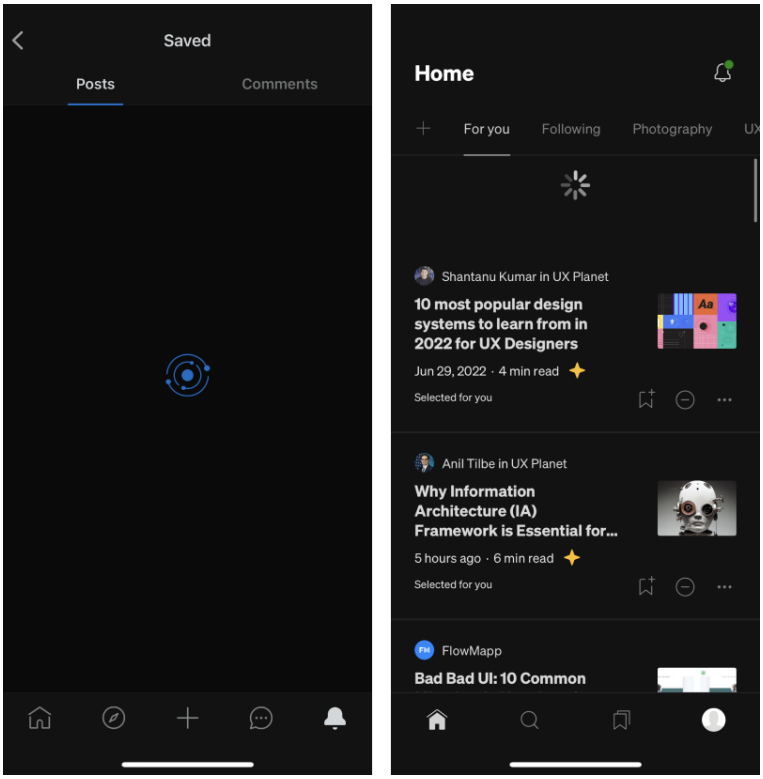

Figure 13: Reddit and Medium loading microinteractions

Standby is also one of the microinteractions, which notifies users that the system is waiting for a user to provide further information. This encourages users to continue interacting with an interface. A great example of this is Grammarly, which informs users of their mistakes, highlights the word, and offers a solution when hovered (see Figure 14). Standby can also be seen when editing a home screen on iPhones, where juggling app icons with a delete option inform the user of the ability to delete apps.

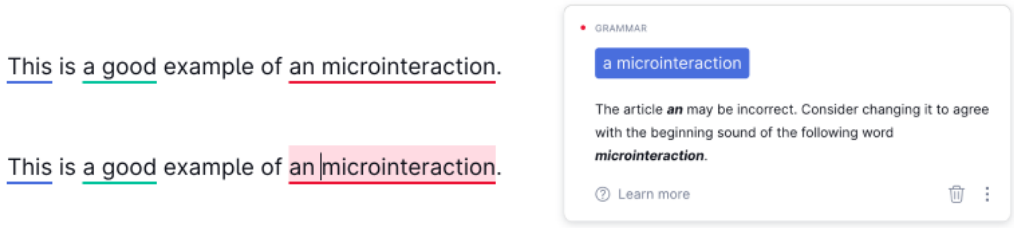

Figure 14: Grammarly standby microinteraction

Error prevention is one of the most useful microinteractions. It notifies users of the consequences of doing specific tasks or guides them to eliminate further error messages. It warns before errors are made. An example of this is counting down input characters on Twitter (see Figure 15). With the limit of 280 characters per tweet, Twitter indicates the remaining 20 characters with a yellow indicator and a number of characters left. When that number is surpassed, the indicator turns red, which lets the user know that they have exceeded the number of characters and their tweet won't go through. This is even more emphasized with a red highlight of excessive characters. [16]

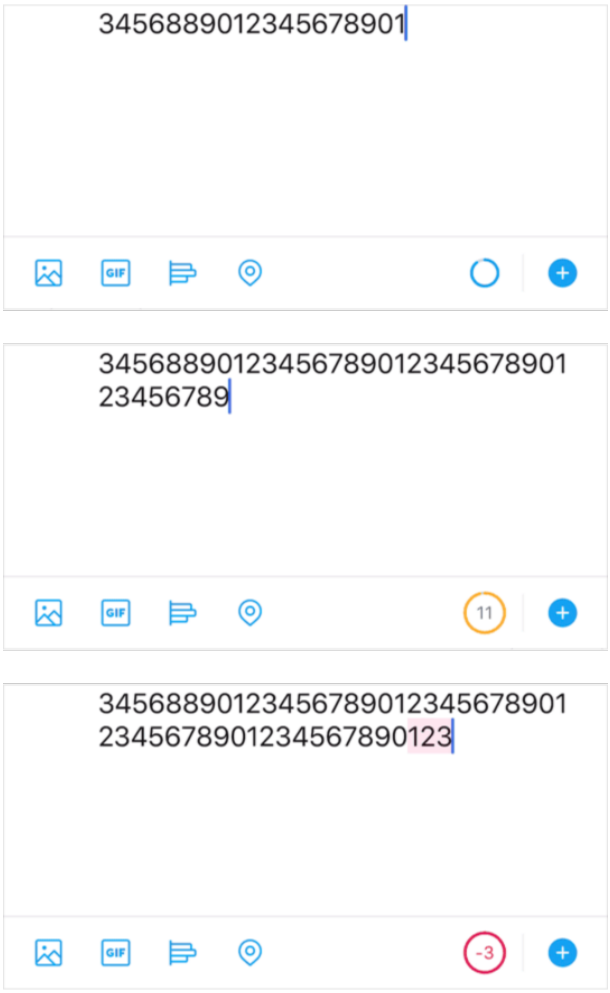

Figure 15: Twitter count down input characters (Medium, [16])

Input microinteractions are one of the most common ones. Depending on how the user interacts with an input field, it will change its state. Most common, but not the only states, are enabled, hovered, and error. When the user clicks on an input field, it changes its state to focused (see Figure 16). In this case, the bottom line gets more weight to it and changes color, and so does the label element. When hovering over an input field, the background color becomes darker, letting users know where their focus currently is. Error state lets the user know their input is invalid and is characterized by the red label, bottom line, warning icon, and error message text below the field. Depending on how the validation is set up, this can happen immediately in the focus state or when the user finished up typing and exits the focus state. [15] [17]

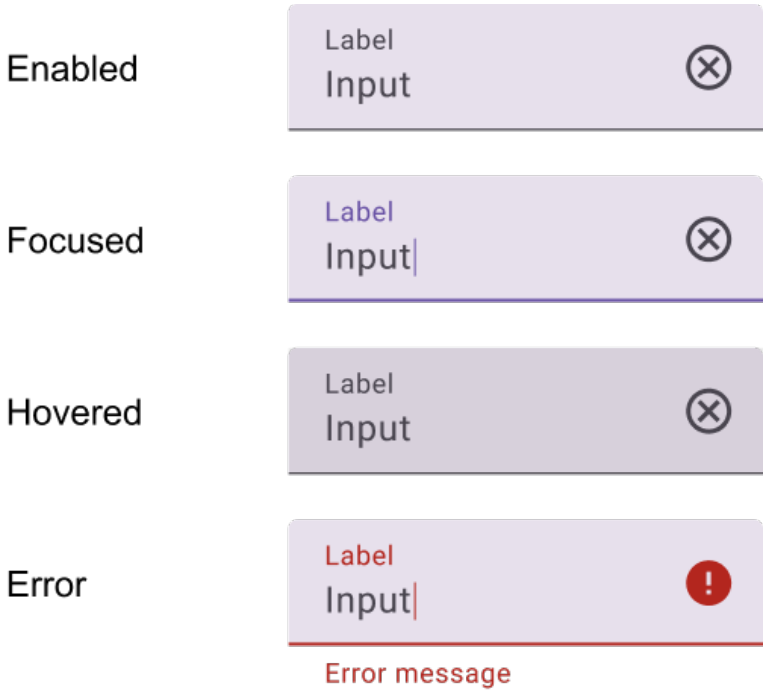

Figure 16: Material Design input states (Material Design, [18])

There are many microinteractions to be talked about. If done well, they are user-friendly, enrich the user experience and improve usability.

5. Animation design and handoff

How do animation design and handoff to developers work? If it is a branding animation, it is usually designers who will design the animation in Adobe After Effect and export it via Lottie for developers to embed it into an interface. Microinteractions, depending on their type, will either be handled by designers as well or have to be thought through by designers and handed off to developers, with the latter being the most common case.

What to do in those cases, when design handoff is needed, where animations have to be implemented by hand in code? There is a constant discussion about whether designers should code to some extent or not. UI animations are a good case for learning how to code stuff that cannot be replicated to detail by someone else with just the design. However, in most cases, that is not the way work is distributed on a project, and designers already have a lot of workload in their place. That is why, in most cases, developers take over the implementation part. And that is where the problem arises. Design is one thing; implementation is another. Designers spend a lot of them nitpicking on animations, adjusting and making subtle changes to transitions and timing. It is hard for anyone to differentiate those subtle changes in numbers, but a subtle change can disrupt the entire flow. Developers cannot replicate that without a good handoff.

UI animation handoff is, in most cases, either done by prototyping the animation in one of the interface design tools or by specificating animation properties and describing it in person, text, picture, example or a combination of those options.

5.1. Prototyping

Designers used to use Adobe After Effects to show of a prototype of their interface, but this route is timeconsuming, not interactive and often cannot be replicated in development. Figma, Sketch and Adobe XD are three of the most used UX/UI tools for interface design. All of them also come with a basic prototyping tool, which also includes some animation possibilites, transitions, microinteraction, although with limited customisation. Framer and Principle are both tools specialised in prototyping and developer handoff, with more animation options than the tools previously mentioned. Both let you build your own design and layout, but also let you import designs from Figma, Sketch and other tools, which is how most designers use them - as a high fidelity prototyping tool.

Nonetheless, proper animation specification is often needed to go with it, to specify animation properties used.

5.1.1. Figma

Figma (figma.com) is an interface design tool used widely by designers for anything from wireframing, to high fidelity screens, design systems, prototyping and lots more. It also offers a prototype option, which allows for simple interactive animations and transitions between the screen and components.

Interaction triggers start the transition from one screen to another by interacting, or having a delay, with a component. Interactions include clicking, hovering, pressing, entering a specific key, mouse leaving and more.

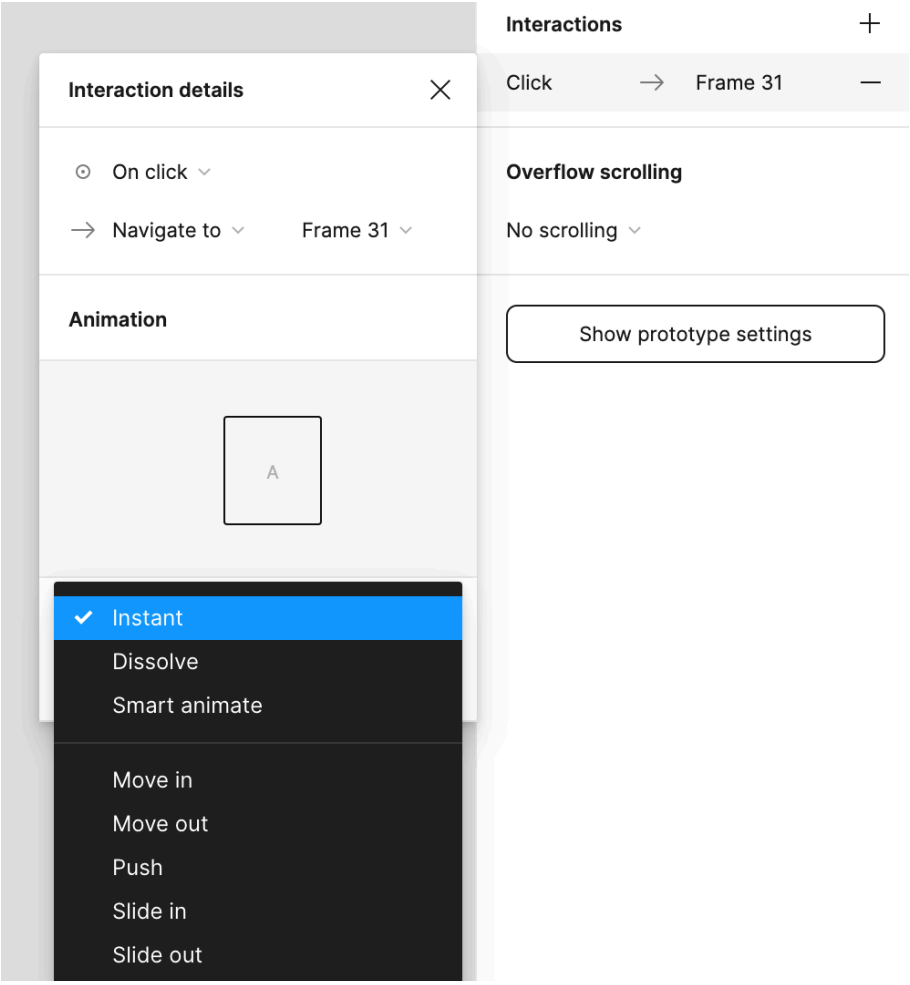

Figure 17: Figma animation properties

Then come the animation properties. Besides the instant, dissolve, move in, push, slide in a similar animation options, Figma also features a smart animate option. Smart animate connects two screens or components fluidly, recognising motion between them depending on the component differences.

Smart animate allows for the recreation of:

- Loading screens
- Horizontal scrolling
- Toggle and switches
- Expanding and collapsing content
- Bouncing effects

Smart animate keeps track of multiple component properties and how they change from one to another. These include scale, position, opacity, rotation and fill. This means smart animate will animate components growing, changing locations, appearing, rotating and changing colors. Utalizing auto layout and grouping with smart animate can replicate more complex animations. [19] [20]

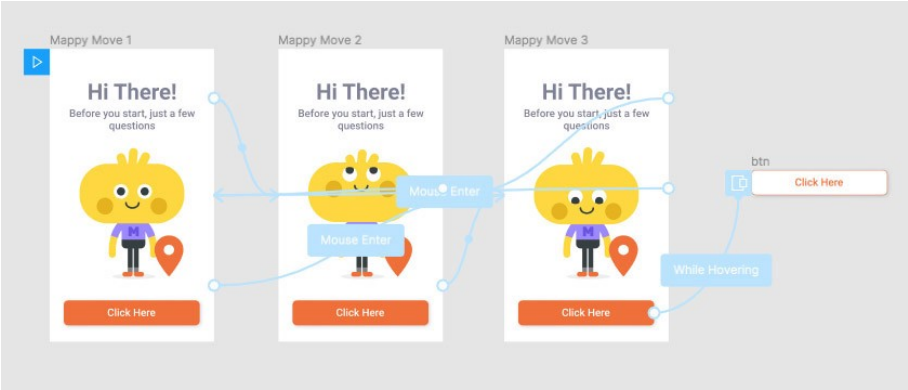

Figure 18: Figma smart animate example ([19])

Figma is a great option for a simple animated prototype that might need additional animation specification to go with it. Nonetheless, its prototype feature is a great addition to a powerful tool that allows designers to use one tool from the beginning to the end of interface design.

5.1.2. Framer

Framer (framer.com), a motion library for React, is often a handoff tool of choice for designers looking to add interactive prototypes to their output. Framer lets developers export animations in code to use directly. Animation tool gives a lot of control over motion properties without setting up an production environment.

It's intuitive, with premade transitions and animations. Set of transitions allow designers to prototype screen transitions in design view, with push and overlay being one of them.

Direction of the transition, backdrop color and opacity are also customisable. As in other prototype tools, scrollable area is also settable. There is also an option to create horizontal carousel by connecting multiple frames together. This option can also be used to reveal actions, as row deletion.

For custom animations, there is an option of code override. Code override lets you, through JavaScript function, edit properties with code. This includes changing colors, change positions, dimensions and opacity. Edited property can be treated as a style and be applied throughout the project. For more complex custom animations, code override has its limitations and it is best to use an entirely autonomous code component.

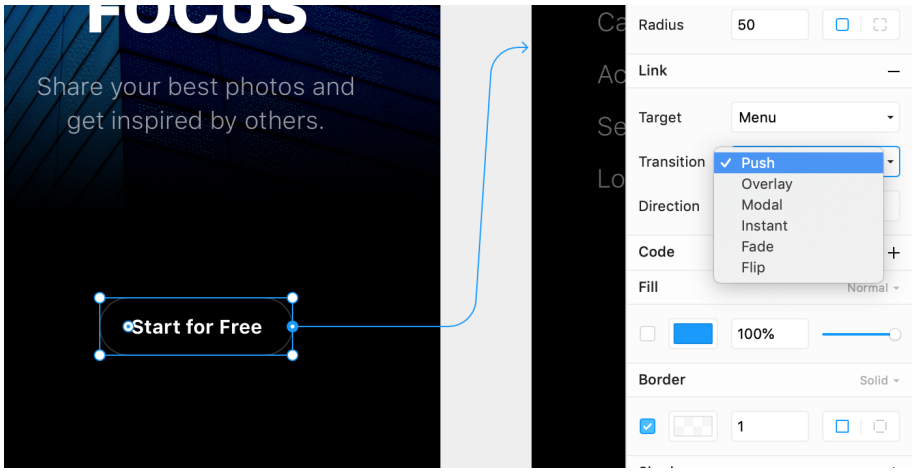

Figure 19: Framer transition settings ([21])

Framer is an excellent tool for transitions and simple animations, however, it might not be the best tool for designers looking to create something more complex but don't want to or don't have the skills to code. [22] [21]

5.1.3. Principle

Principle (principleformac.com) has a similar feel to Sketch. As with Framer, its core feature is creating transitions between screens and building a prototype. The interactivity of Principle prototypes is highly realistic and mobile suitable, as every interaction has multiple trigger options, such as tap, long press, scroll begin, release and end.

Animation panel is highly customisable and features predefined and custom effect, allows you to adjust animation curve, set priorities and durations for animations on a simplified version of Adobe After Effect animation panel, and work with a driver panel that allows multiple animation properties on one component and linking components together. Prototypes can be exported as a macOS app, to be tested and experienced like a user would the final app. This

can be of great use for presenting an interface, or flows within it, to clients, testing usability and more. Another way of testing out the prototype is by downloading the Principle Mirror iOS app or using a "Share to Web" option and testing it out directly. For presentations, there are .mov and .gif exportable files to use.

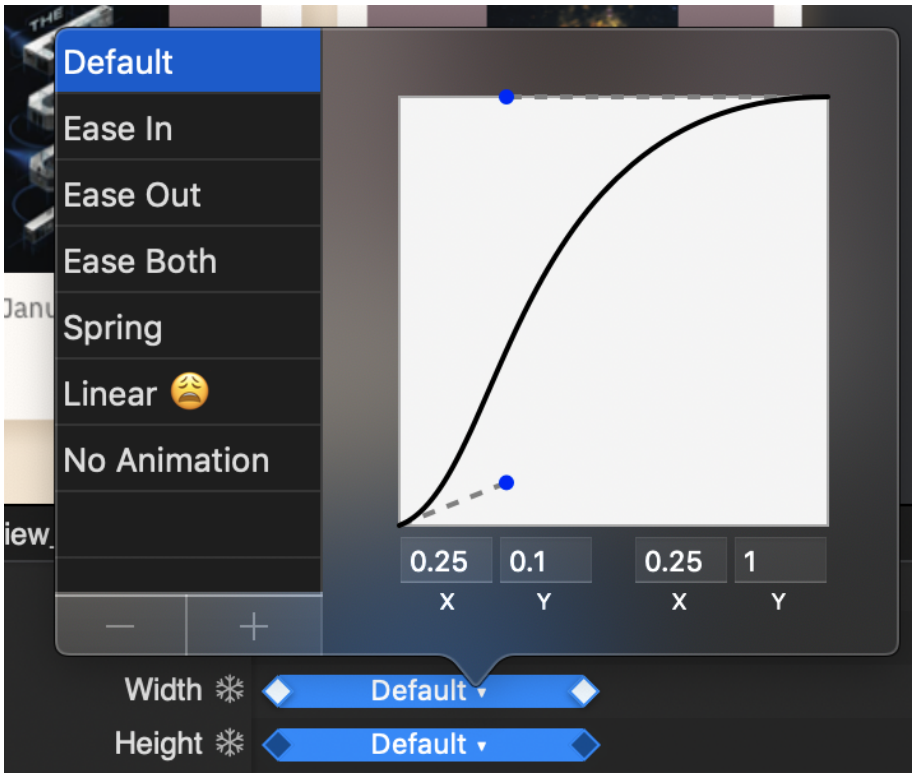

Figure 20: Principle animation curve ([23])

Overall, Principle is for designers looking to create highly interactive prototypes, efficiently, with lots of control over detailed animation properties, while having a fairly fast and simple workflow. [24] [25] [26]

5.2. Specification

Animation specification can include a mix of formats. Many times, textual specifications come with reference photos, or a frame by frame guide, an inspiration or even a functional prototype or animation video. Whatever it might contain, specification is extreamly important for developers.

Every year, Spotify designers work on redesigning the Wrapped experience customisable for all of the users. The goal is to create a standout campaign by exploring new design ideas while keeping the consistency within the Spotify brand. They created a sample animation in Adobe After Effect. They knew how the animation should look, but the problem

was the implementation - it needed to have dynamic content for each user. Thus the animation needed to be implemented in code.

They used Google Sheet to create UI animation specifications for their developers, frame by frame. Below is an screenshot from one of the animation specifications sheets. They defined durations of specific transitions within the animation, start and end face opacity values, positions, delay values and content for specific frames. [27] [28]

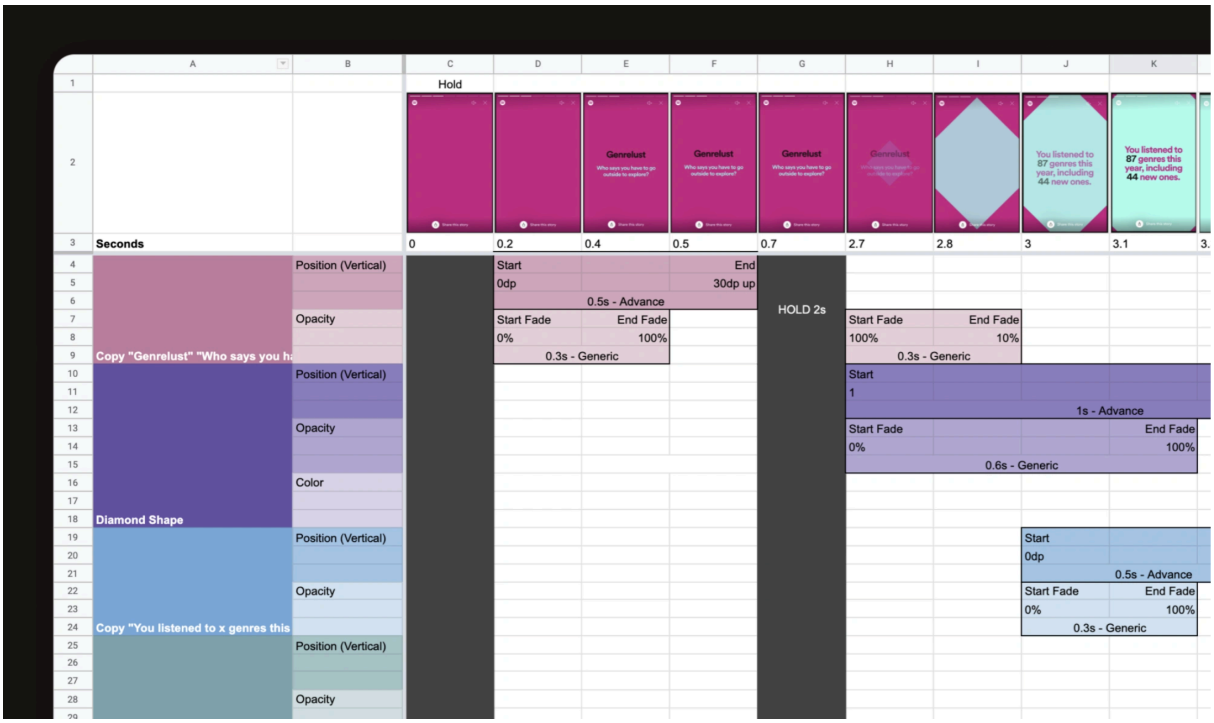

Figure 21: Spotify Wrapped UI animation handoff ([27])

This is a great example of what specification for an animation should look like. Frame by frame is a great option for animation with multiple frames and different animation properties. Otherwise, there is always an option to shadow developers during the implementation or simply explaining a microinteraction between two components with a couple of animation properties. Some are more time consuming, some are too vague. Choosing an option for UI animation handoff depends greatly on the type and complexity of an animation, as well as the project and team restrictions.

6. Implementation

Animations can be designed perfectly and still fail at performance. Both designers and developers need to consider the implementation process. Animations that are slow negate everything that improves the usability and just cause inconvenience.

Three of the most popular implementation techniques are CSS, JavaScript, and Lottie. Generally speaking, both CSS and JavaScript are used for web interfaces, with JavaScript handling complex animations, while Lottie is mostly used in mobile interfaces. Let's discuss their specifications.

6.1. CSS and JavaScript

One of the most popular ways to implement animations is via CSS. CSS can be added to animate both HTML and CSS-defined elements, as well as SVG formats. So, what can you animate with CSS? A lot of things, including rotating loaders, any vector-based illustrations, hamburger menus, text animations, and the way that a label moves up when activating an input field. More of that, with practical examples and code snippets, will be covered in the practical part of this thesis.

SVG stands for "Scalable Vector Graphics." They are used to define vector format graphics, meaning they are scalable, do not lose quality, are defined by code, and, therefore, can be edited in any text editor. This also means every element and property of an SVG graphic can be animated. They are usually designed in a vector graphics editor like Adobe Illustrator or Affinity Designer, but there are also some predefined shapes that can be used to code it. [29]

Animated SVGs are a great way to animate icons that indicate state changes and microinteractions, as well as all the bigger vector illustrations. This can be anything from a "like" button status change to an animated 404 error page on a website. Animations added through animating SVGs with CSS have faster load time, are lightweight and scalable, and often don't require additional JavaScript libraries.

JavaScript is used for more complex animations, as there are libraries with already predefined animation files. [1]

6.1.1. Optimisation

Optimization comes into question both when preparing an SVG (if there is an SVG) for the animation and when considering the way the animation is going to be implemented.

Both CSS and JavaScript have properties that are easier on the performance than others. For example, most browsers work great with these properties:

- opacity
- scale
- rotation
- transform position

Thankfully, they are also the most commonly needed to implement UI animations.

As far as JavaScript goes, as said, it should only be used to animate more complex animations, as embedding libraries can have a negative impact on performance.

Additionally, it is essential to prepare SVGs for animation through optimization and grouping. Often, SVG has a lot of unnecessary code, including unnecessary tags and metadata, which can be removed by hand or by some optimization tools. Think about the ways you want to animate your SVG and group layers accordingly, either through a vector graphics editor or through code. [1]

6.2. Lottie

A solution to the problem of implementing animations into applications came from an unlikely source - Airbnb. Airbnb is an American hospitality company that provides a platform for vacation rentals. The platform is accessible through the website and mobile app. [30]

Their team faced the same problem other designers and developers were facing. They used PNG sequences and other unsuitable formats to add even the slightest animation to their app. Most programs avoided utilizing animations at all due to bulky image formats and difficult-to-maintain and massive code.

Lottie (lottiefiles.com) came as a response to that. Lottie is an iOS, Android, and React Native library that renders Adobe After Effect animations so that they can be used in a lightweight graphic format. It is built upon an open-source Adobe After Effect extension called Bodymovin and uses JSON files from the extension, and renders animations with a JavaScript player.

Lottie also has a strong community of contributors, both designers, and developers, as well as animators. There is GitHub documentation everyone can contribute to. One of the resources on there is this supported features checklist for Adobe After Effect, which lists all of the animation features in Adobe After Effect and lists whether they are supported on different platforms through Lottie. [31]

Shapes	Android	iOS	Windows	Web (SVG)	Web (Canvas)	Web (HTML)
Shape	👍	👍	👍	👍	👍	👍
Ellipse	👍	👍	👍	👍	👍	👍
Rectangle	👍	👍	👍	👍	👍	👍
Rounded Rectangle	👍	👍	👍	👍	👍	👍
Polystar	👍	👍	⛔	👍	👍	👍
Group	👍	👍	👍	👍	👍	👍
Repeater	👍	⛔	👍	👍	👍	👍
Trim Path (individually)	👍	👍	⛔	👍	👍	👍
Trim Path (simultaneously)	👍	👍	👍	👍	👍	👍
Fills	**Android**	**iOS**	**Windows**	**Web (SVG)**	**Web (Canvas)**	**Web (HTML)**
Color	👍	👍	👍	👍	👍	👍
Opacity	👍	👍	👍	👍	👍	👍
Fill Rule	👍	👍	👍	👍	👍	👍
Radial Gradient	👍	👍	👍	👍	👍	👍
Linear Gradient	👍	👍	👍	👍	👍	👍
Strokes	**Android**	**iOS**	**Windows**	**Web (SVG)**	**Web (Canvas)**	**Web (HTML)**
Color	👍	👍	👍	👍	👍	👍
Opacity	👍	👍	👍	👍	👍	👍

Figure 22: Lottie After Effect compatabilities([31])

6.2.1. Why use Lottie

What makes Lottie a great library of choice? As mentioned, Lottie supports every major platform from web development, major mobile platforms, and even IoTs and desktop applications. Lottie stores away animations into text files that are easily read by humans. A byproduct of that is easy customization, debugging, and maintenance without the need for specialized tools. The text file is stored in JSON format, which means it is ready for assimilation in Javascript or any other environment that supports JSON. JSON is easily parsable, and it is widely supported, thus making Lottie the most portable animated graphic format in the world.

There is also no need to sacrifice quality. Lottie graphics are drawn using vectors that define the dimensions of geometric shapes. Vector graphics contain various qualifiers, which help with determining how will shape change if they move during animations. To make sure Lottie is displayed at best possible quality, all Lottie animations are rendered at a native resolution of the screen that is playing the animation. It also re-renderers animation at zoom to keep the same quality.

For designers, the good thing is that Lottie has Adobe After Effect plugins, and can render directly from the design process, so the animation is ready for embedding into an interface.

One of the most useful features is caching frequently used animations. The animation itself can be saved locally and reused every time, thus saving time by refetching animations every time it is needed. Lottie also supports manipulating animation speeds by simply changing a value in a JSON file, which makes it flexible. One major feature that is only supported on iOS is adding native UI components to animations at runtime, providing developers with support for using complex animated transitions. The team behind Lottie is constantly updating their library and keeping it well maintained. [32]

6.2.2. Alternatives

Lottie has been hugely popular in a few recent years, and with reason. Other alternatives are either poor in quality, massive in size, or complicated to make.

Building by hand is one of them. More often than not, it is a very time-consuming and sometimes even impossible task that is not worth the effort of both designers and developers.

GIFs are one of the most popular solutions. However, they are more than double the size of Bodymovin JSON and cannot be scaled to match higher density resolutions. Their quality is also often poor, depending on the compression settings.

Animated Vector Drawable is Android only alternative that is good performance-wise since it runs on the Render Thread. However, it is still a very limited alternative. It supports only a few of the features Lottie does, progress cannot be manually set, and it does not support dynamic colors or text.

One of the worst ways to implement animation is through PNG sequences, which designers at AirBnB used before creating Lottie. Being a PNG, animation cannot be scaled up and are often 30 to 50 times the size of Bodymovin JSON files.

There are also libraries that can be imported into the project. If we wanted to implement animations into existing applications, there are a few great alternatives, such as Marcus Eckert's Squall and Keyframes from Facebook. What separates Lottie from Keyframes is the focus and broader set of After Effects it supports. Facebook created Keyframes with a focus on animation for their reactions, which resulted in a smaller set of After Effects they support. Squall, however, has an amazing app preview for After Effects, and it is used by Airbnb's developers in combination with Lottie. One major downside of using Squall is it is only supported on the iOS platform, so it is not a viable solution for Android or web usecases. [31]

6.2.3. Where can we see Lottie?

Taking into account Lottie's key features and advantages over alternatives, it’s not hard to believe that Lottie is widely used with some big names behind it.

For its Google Home application, Google uses over 40 different animations, both on Android and iOS, which include intros, loaders, and errors.

There are many more examples of applications that are using Lottie for their animations, such as Microsoft, Uber Eats, Target, Walgreens, PenPal, and so many more. Lottie’s flexibility, availability to be used on so many different platforms, and scalability is one of the biggest reasons why all those companies would choose Lottie. [33]

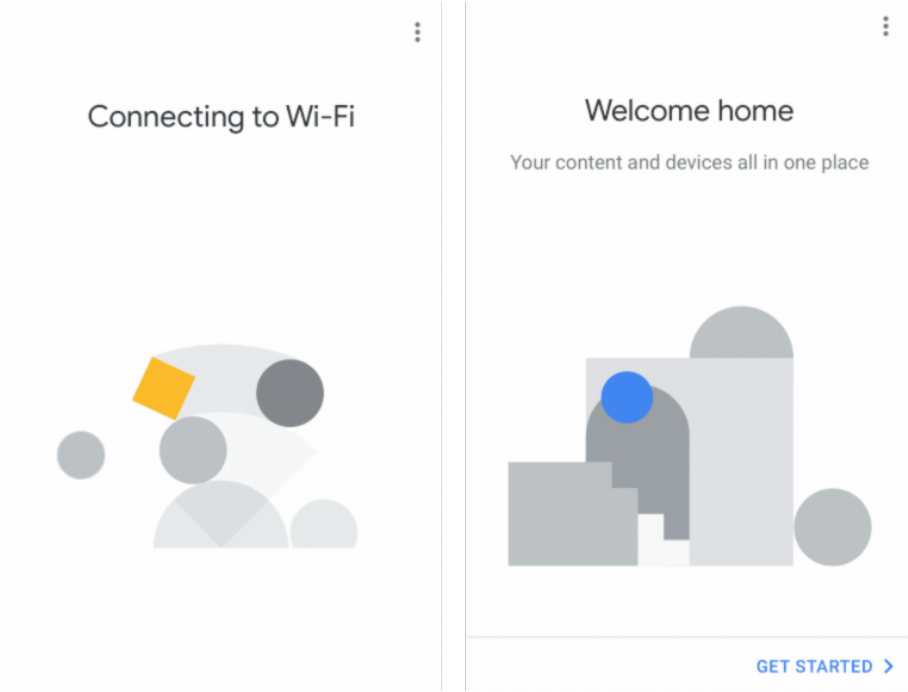

Figure 23: Google Home animations

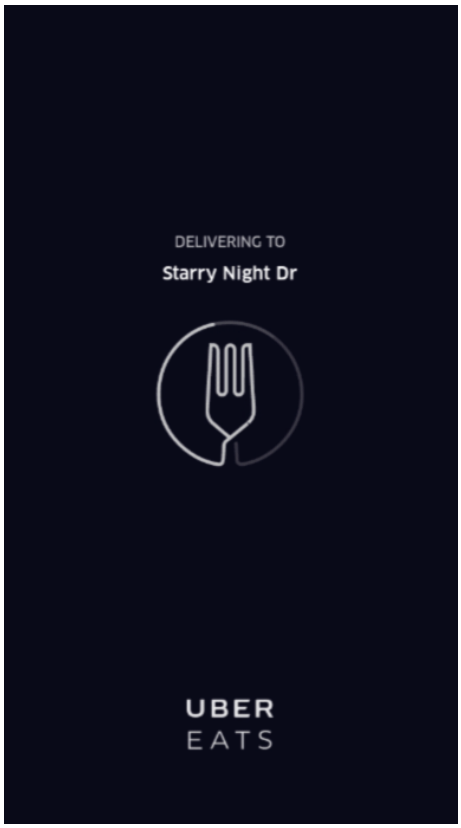

Figure 24: Uber Eats animation

7. Practical Assignment - Async Labs Web Animations

In the practical assignment, the process of designing, preparing, and implementing animations will be shown on an example of a digital agency's website. This was all done during my digital design internship at Async Labs, where one of my first assignments was to help redesign and create content for our new website.

The project was done as a medium priority throughout the period of 6 months, starting with a discovery phase, information structure, navigation architecture, to design, illustrating, and designing animations, with the addition of tracking implementation of the design and implementing some of the animations. Information structure and navigation architecture were done in cooperation with my design mentor. Design for work, contact, and team pages was done entirely by me, along with some additions to the other pages. All of the icon illustration and animation design mentioned in this practical assignment, as well as the implementation of services illustrations, were done by me. Implementation of microinteractions was done in cooperation with our frontend developer.

Unfortunately, animation is extremely difficult to showcase is this thesis, but the entire project is stored and available for viewing on a Behance casestudy [Appendix 1]. What can be shown is the illustration process, prototyping and code used to implement animation, with the addition of prototypes on Figma and demonstrations on CodePen.

7.1. Briefing

Our primary goal was to build a visual identity of the company, reflective in the design of the website, illustrations, photographs, animations and copy, and thus also increase SEO results. Navigation architecture is a simple one and lists pages interesting to both the potential clients and potential employees, with project showcases, services descriptions, contact forms, a gallery of team members and so on. All of the pages are:

- Home
- Work
- About
- Services
- Contact

- Careers
- Team
- Blog

The entire website can be seen on asynclabs.co [Appendix 2].

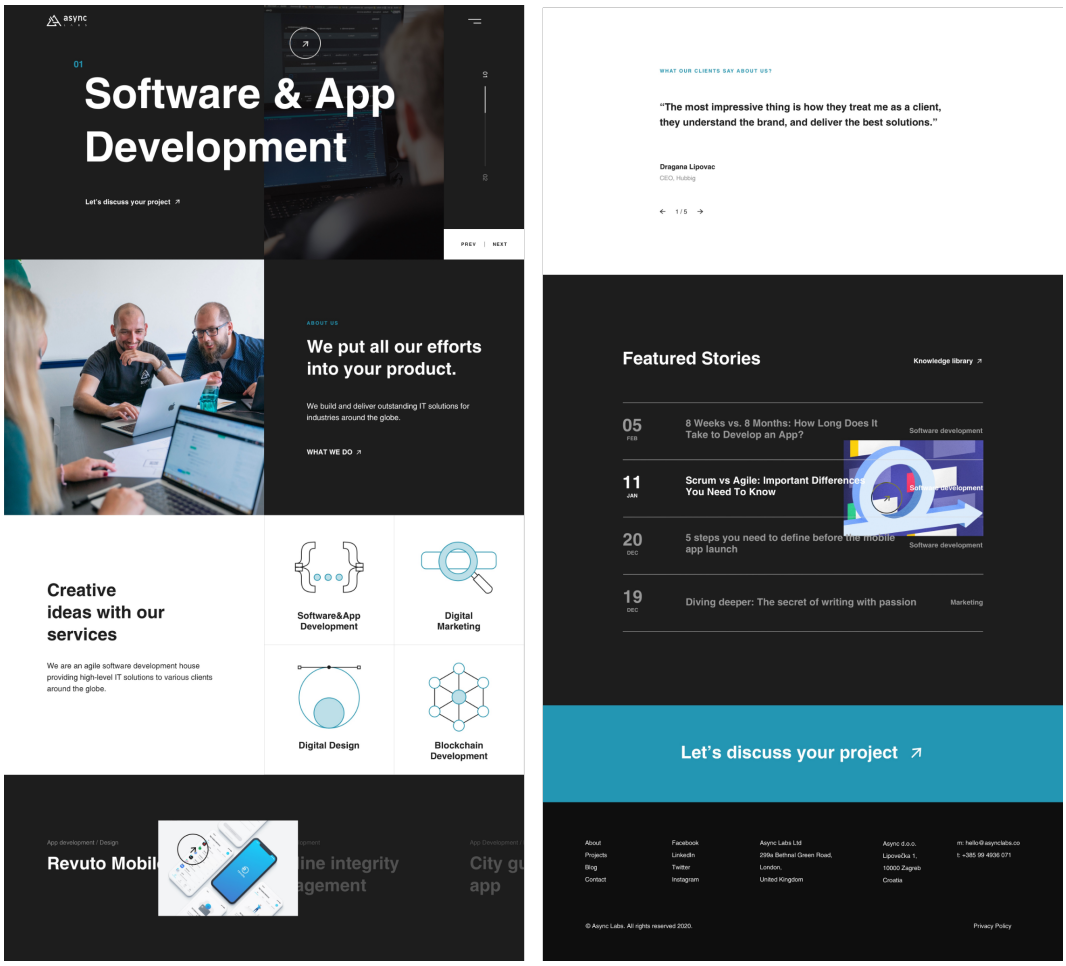

Figure 25: AsyncLabs website

Throughout the entire website there are interactive components which grab users attention, smooth horizontal transitions, overlays, appearing pictures that track the movement of the mouse, coded messages, microinteractions and GIFs of the office atmosphere, including animated portraits on Teams page.

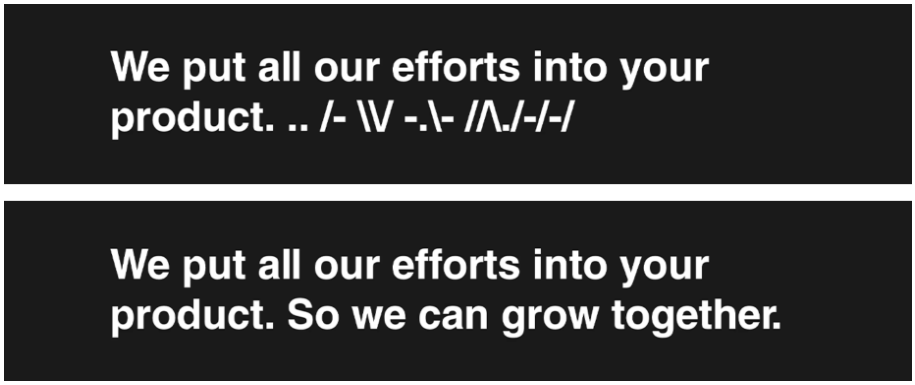

Figure 26: AsyncLabs coded message animation

7.2. Animating Vector Illustrations

When first entering the site, we are met with a landing page which gives visitors an insight into the company. One of the sections on it lists services the company provides. To indicate focus on one of the services, illustrations animate on hover.

Animation design and implementation process are a good demonstration of previously mentioned principles and tools. Illustrations were done in Adobe Illustrator (adobe.com/products/illustrator), a vector graphics tool, and exported in an SVG format. Figma (figma.com) was used to design and prototype animations of the illustrations, while CSS was used for the implementation.

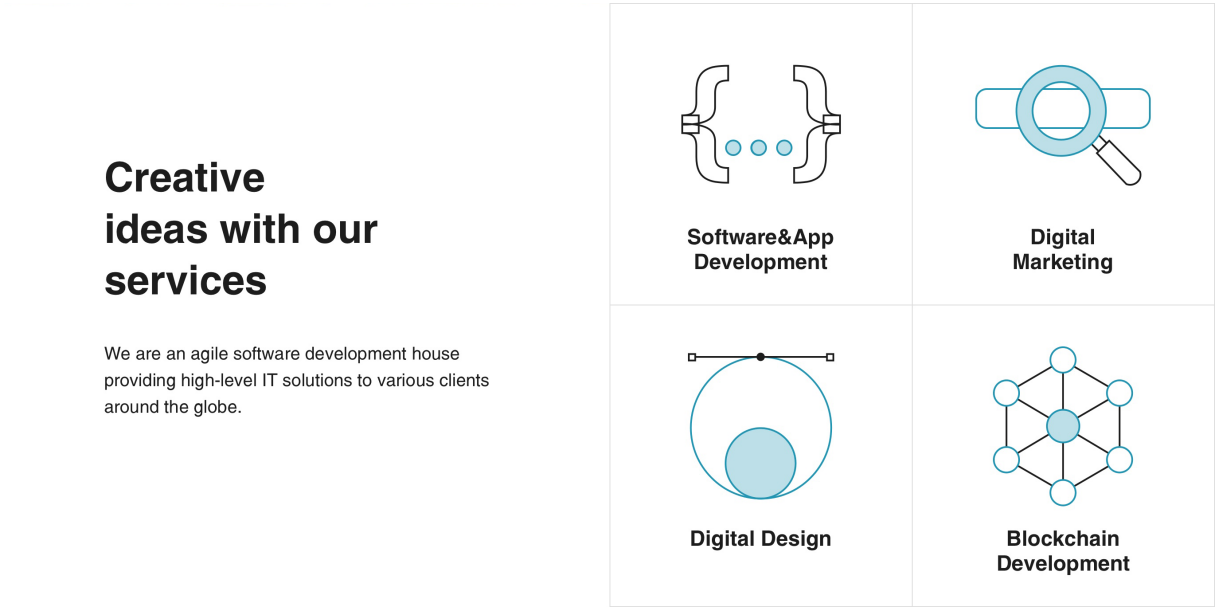

Figure 27: Async Labs services illustrations

The HTML structure for the entire section is as it follows.

```
<section class="services">
<div class="header">
<h2>Creative ideas with our services</h2>
<p>We are an agile software development house providing high-level IT
solutions to various clients around the globe.</p>
</div>
<div class="body">
            <div class="single-service development">
                  <a href="<?=
                  $softwareDevelopmentGroup['link'] ?>" class="name">
                        <div class="icon"> <?= getSvg('icon-development')
                        ?> </div>
                        <?= $softwareDevelopmentGroup['title'] ?>
                  </a>
            </div>
            <div class="single-service marketing">
                  <a href="<?=
                  $digitalMarketingGroup['link'] ?>" class="name">
                        <div class="icon"> <?= getSvg('icon-marketing') ?>
                        </div>
                        <?= $digitalMarketingGroup['title'] ?>
                  </a>
            </div>
            <div class="single-service design">
                  <a href="<?= $digitalDesignGroup['link'] ?>"
                  class="name">
                        <div class="icon"> <?= getSvg('icon-design')
                        ?> </div>
                        <?= $digitalDesignGroup['title'] ?>
                  </a>
            </div>
            <div class="single-service blockchain">
                  <a href="<?=$blockchainDevelopmentGroup['link'] ?>"
                  class="name">
                        <div class="icon"> <?= getSvg('icon-
                        blockchain') ?> </div>
                        <?=$blockchainDevelopmentGroup['title'] ?>
                  </a>
            </div>
      </div>
</section>
```

7.2.1. First (Software Development) Animation

The first animation was an illustration for "Software Development". It consists of curly brackets, commonly used to illustrate programming code, and three dots representing the content. We animated the three dots to represent typing of the code.

Directing focus principle was used as a purpose of animation, as it directs user's attention to the service service. ***Linear easing*** was used, despite the fact that linear is known to be unatural, and thus avoided in UI animation. However, the distance and duration are both small, so easing is not as noticable. ***Timing*** comes into play with creating an optical illusion of a grouped easing, as each dot is delayed after the previous one.

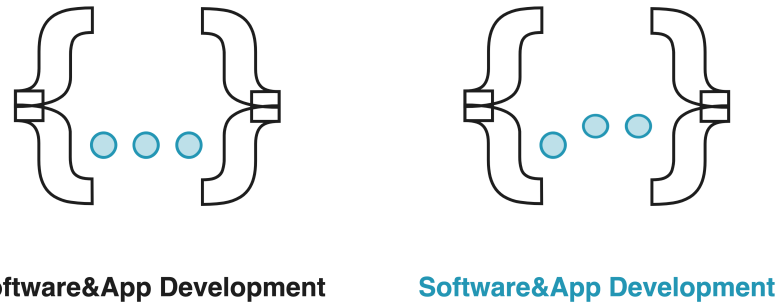

Figure 28: Software Development animation

After the initial phase of ideating and sketching out the illustration on paper, it was transferred to Adobe Illustrator. Since the idea was to animate different parts of the illustration separately, making a clean vector illustration was essential. With expanded strokes and neatly grouped elements, we still needed to clean the SVG code, which we would get to before implementing the animation.

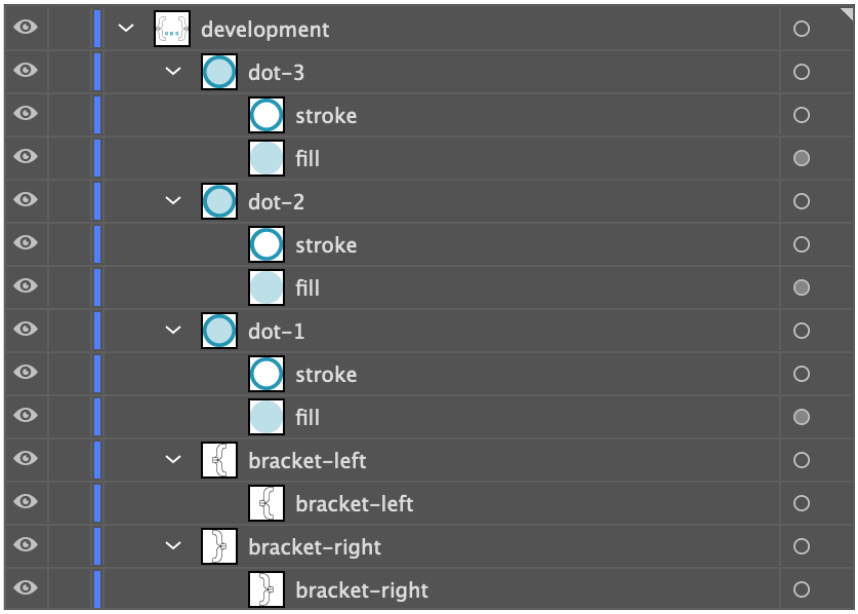

Figure 29: "Software Development" Illustrator layers

Before that, it might be a good idea to demonstrate how the animation should act, especially if handing off the implementation to the developers. Although, in this case, the animation was made without an example, we will demonstrate how it would have been made.

As shown before, Figma can be a great tool to prototype animations, while sometimes not as straightforward as the implemented version will be.

As said previously, the idea was to animate three of the dots inside the brackets, the first one with a 0-second delay, the second with a 0.4-second delay, and the third one with a 0.6-second delay. The duration of a single dot going from the bottom to the top and back is 1 second, with linear behavior, meaning it has the same speed from beginning to end.

Each of the dots was made into a component with multiple variants. The first component had three variants - a dot with a starting position, a dot with a middle position, and the last one with an ending position. The second and the third components had four variants - a dot with a starting position, a dot with a middle position, a dot with the ending position, and lastly, a dot that replaces the first one after the first animation loop to remove the 0.4 and 0.6 seconds delay at the start.

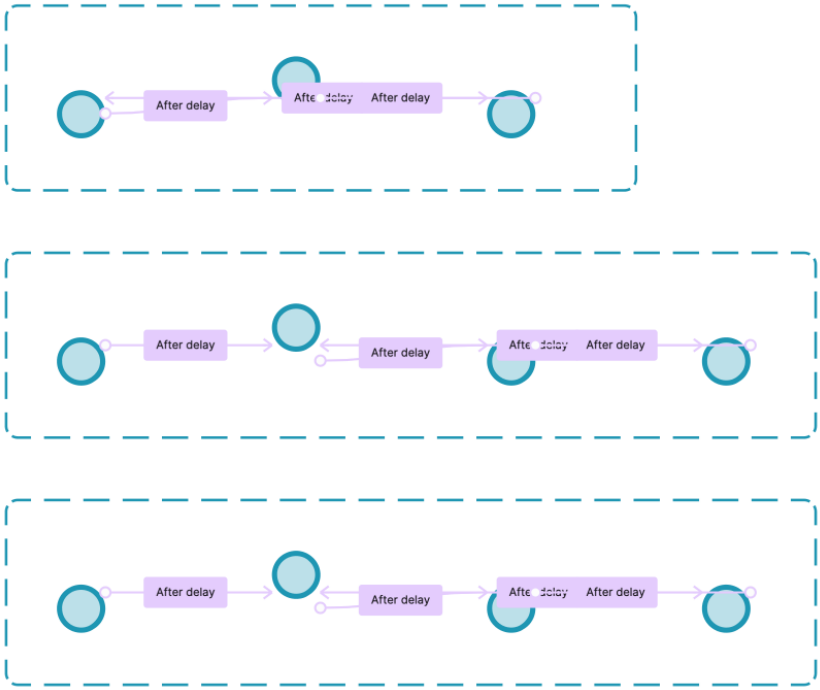

Figure 30: Software Development animation prototype

Below are interaction details for all four variants of the second component, or dot. The animation starts with 0.4 seconds or a 400 milliseconds delay. It lasts 500 milliseconds from the starting position, of the first variant, to the top position or the middle of the dots path and the second variant. After a one millisecond delay, it goes back to the bottom, the ending position, and the third variant, again in 500 milliseconds. This totals to 1000 milliseconds, or 1-second duration.

Then, to avoid looping back with a 400 milliseconds delay, we instantly switch to the last variant, with a one millisecond delay and duration. From there, we go to the second variant. This closes the loop to only a second, third and fourth variant after the first iteration.

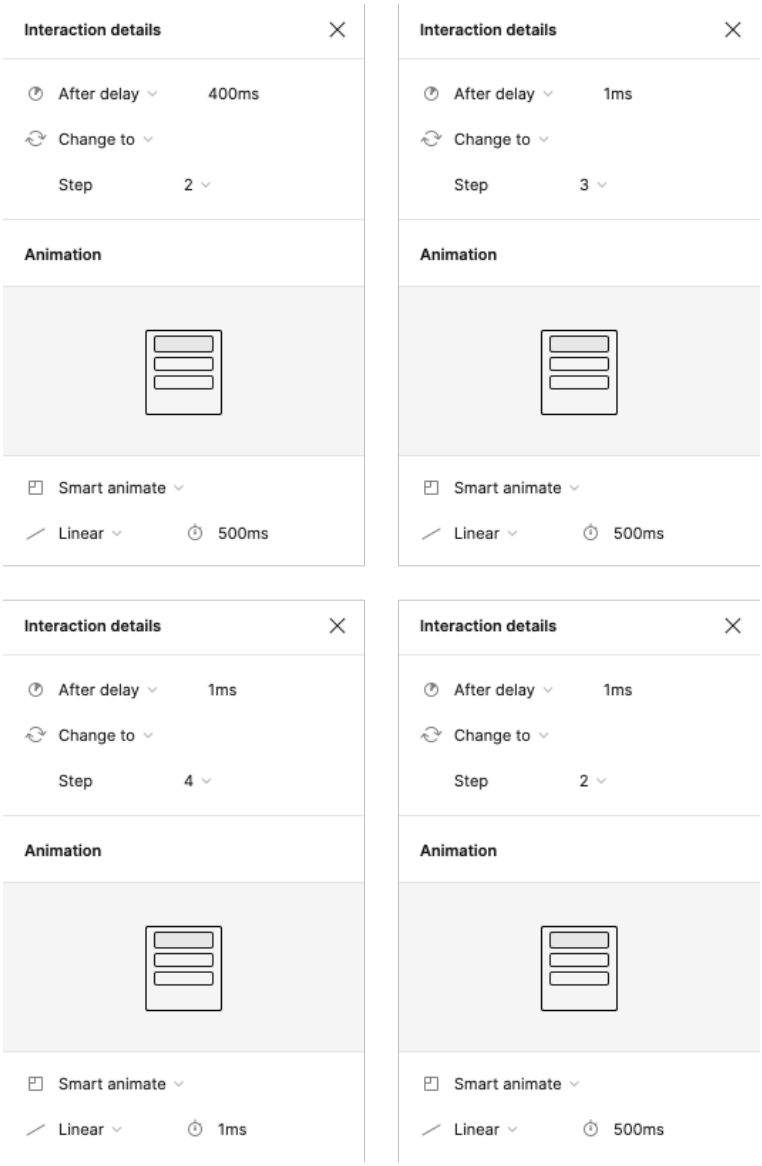

Figure 31: Software Development animation properties

Prototype for this animation, as well as all the prototype elements, can be seen on Figma [Appendix 3].

When the animation is confirmed, it is time to implement it. To keep things as neat as possible, XML, comments, metadata and other unnecessary parts were removed from the SVG code. What was left was a clean code which represented the vector illustration, clearly defining the positions and style of all the elements that make it up.

```
<div>
      <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 200 200">
            <g fill="none" stroke="#1d1d1d" stroke-width="2">
<path d="M11 100.5V90c6 0 10.1-1.2 12.4-3.6 2.3-2.4 3.5-6.3 3.5-
11.6V63.3c0-6.4.9-11.6 2.7-15.5 1.6-3.6 4.2-6.7 7.5-8.8 3.5-2.1 7.3-3.4
```

```
11.4-4 4.8-.7 9.6-1 14.4-1v16.6c-4.1 0-7.2.5-9.3 1.6-2.1 1-3.6 2.8-4.3 4.9-
.8 2.7-1.2 5.5-1.1 8.3v15.2c0 2.6-.5 5.1-1.5 7.5-1.1 2.6-3 4.8-5.4 6.3-2.6
1.8-6.4 3.3-11.2 4.3-6.4 1.4-12.7 1.9-19.1 1.8h0z" />

<path d="M11 90h18.6v20H11z" />

<path d="M62.8 166c-4.8 0-9.6-.3-14.4-1-4-.5-7.9-1.9-11.4-4-3.3-2.1-5.9-
5.2-7.5-8.8-1.8-3.9-2.7-9-2.7-15.5v-11.5c0-5.3-1.2-9.2-3.5-11.6-2.2-2.4-
6.3-3.6-12.3-3.6V99.5c6.4-.1 12.7.4 19 1.6 4.9 1.1 8.6 2.5 11.2 4.3 2.4 1.5
4.2 3.7 5.4 6.3 1 2.4 1.5 5 1.4 7.6v15.2c-.1 2.8.3 5.6 1.1 8.3.7 2.2 2.3
3.9 4.3 4.9 2.1 1.1 5.2 1.6 9.3 1.6l.1 16.7h0z" />

</g>

<g fill="none" stroke="#1d1d1d" stroke-width="2">

<path d="M137.2 34.5c4.8 0 9.6.3 14.4 1 4 .5 7.9 1.9 11.3 4 3.3 2.1 5.9 5.2
7.5 8.8 1.8 3.9 2.7 9 2.7 15.5v11.5c0 5.3 1.2 9.2 3.5 11.6 2.3 2.4 6.5 3.6
12.4 3.6V101c-6.4.1-12.7-.4-19-1.6-4.9-1.1-8.6-2.5-11.3-4.3-2.4-1.5-4.2-
3.7-5.4-6.3-1-2.4-1.5-4.9-1.4-7.5V66c.1-2.8-.3-5.6-1.1-8.3-.7-2.2-2.3-4-
4.3-5-2.1-1.1-5.2-1.6-9.3-1.6V34.5z" />

<path d="M189 100v10.5c-5.9 0-10.1 1.2-12.4 3.6-2.3 2.4-3.5 6.2-3.5
11.6v11.5c0 6.4-.9 11.6-2.7 15.5-1.6 3.6-4.2 6.7-7.5 8.8-3.5 2.1-7.3 3.4-
11.3 4-4.8.7-9.6 1-14.4 1v-16.6c4.1 0 7.2-.5 9.3-1.6 2-1 3.6-2.7 4.3-4.9.8-
2.7 1.2-5.5 1.1-8.3v-15.2c0-2.6.5-5.2 1.4-7.6 1.1-2.6 3-4.8 5.4-6.3 2.6-1.8
6.4-3.3 11.2-4.3 6.4-1.3 12.7-1.8 19.1-1.7h0z" />

<path d="M189 90.4v20.1h-18.6v-20z" />

</g>

            <g fill="#2396b4" class="dot-3">

<path d="M133.2 121.4c-1.3-1.4-3.2-2.1-5.1-2.1-1.9 0-3.8.7-5.1 2.1-1.4 1.3-
2.2 3.2-2.1 5.1 0 4 3.3 7.3 7.3 7.3 1.3 0 2.5-.3 3.6-1s2-1.6 2.7-2.7c.7-1.1
1-2.4 1-3.6-.1-1.9-.9-3.8-2.3-5.1z" opacity=".3" />

<path d="M134.6 120c-1.7-1.7-4-2.7-6.5-2.7-5.1 0-9.2 4.1-9.2 9.2s4.1 9.3
9.2 9.3c1.6 0 3.2-.4 4.6-1.3 1.4-.8 2.5-2 3.3-3.3 2.2-3.6 1.6-8.3-1.4-
11.2zm-6.4 13.8c-4 0-7.2-3.3-7.3-7.3 0-1.9.7-3.8 2.1-5.1 1.3-1.4 3.2-2.2
5.1-2.1 1.9 0 3.7.7 5.1 2.1 1.4 1.3 2.2 3.2 2.2 5.1 0 1.3-.3 2.5-1 3.6s-1.6
2-2.7 2.7c-1 .7-2.3 1-3.5 1z" />

            </g>

            <g fill="#2396b4" class="dot-2">

<path d="M104.3 121.4c-1.3-1.4-3.2-2.1-5.1-2.1-1.9 0-3.8.7-5.1 2.1-1.4 1.3-
2.2 3.2-2.1 5.1 0 4 3.3 7.3 7.3 7.3 1.3 0 2.5-.3 3.6-1s2-1.6 2.7-2.7c.7-1.1
1-2.4 1-3.6-.1-1.9-.9-3.8-2.3-5.1z" opacity=".3" />

<path d="M105.7 120c-1.7-1.7-4-2.7-6.5-2.7-5.1 0-9.2 4.1-9.2 9.2s4.1 9.3
9.2 9.3c1.6 0 3.2-.4 4.6-1.3 1.4-.8 2.5-2 3.3-3.3 2.2-3.6 1.6-8.3-1.4-
11.2zm-6.5 13.8c-4 0-7.2-3.3-7.3-7.3 0-1.9.7-3.8 2.1-5.1 1.3-1.4 3.2-2.2
5.1-2.1 1.9 0 3.7.7 5.1 2.1 1.4 1.3 2.2 3.2 2.2 5.1 0 1.3-.3 2.5-1 3.6s-1.6
2-2.7 2.7c-1 .7-2.2 1-3.5 1z" />

            </g>

            <g fill="#2396b4" class="dot-1">

<path d="M75.6 121.4c-1.3-1.4-3.2-2.1-5.1-2.1-1.9 0-3.8.7-5.1 2.1-1.4 1.3-
2.2 3.2-2.1 5.1 0 4 3.3 7.3 7.3 7.3 1.3 0 2.5-.3 3.6-1s2-1.6 2.7-2.7c.7-1.1
1-2.4 1-3.6-.1-1.9-.9-3.8-2.3-5.1z" opacity=".3" />

<path d="M77 120c-1.7-1.7-4-2.7-6.5-2.7-5.1 0-9.2 4.1-9.2 9.2s4.1 9.3 9.2
9.3c1.6 0 3.2-.4 4.6-1.3 1.4-.8 2.5-2 3.3-3.3 2.3-3.6 1.7-8.3-1.4-11.2zm-
6.4 13.8c-4 0-7.2-3.3-7.3-7.3 0-1.9.7-3.8 2.1-5.1 1.3-1.4 3.2-2.2 5.1-2.1
```

```
1.9 0 3.7.7 5.1 2.1 1.4 1.3 2.2 3.2 2.2 5.1 0 1.3-.3 2.5-1 3.6s-1.6 2-2.7
2.7c-1 .7-2.3 1-3.5 1z" />
</g>
      </svg>
</div>
```

To this, animation is added through CSS code and activated on hover. For this particular one, each dot has its own class, where we called upon *jump-animation*, whose construct I'll explain later on. We call upon the animation, set the duration of 1 second, the delay of 0, 0.4 and 0.6 seconds, and define the animation as linear.

```
.development {
      &:hover {
            .dot-1 {
                  animation: jump-animation 1s linear;
            }
            .dot-2 {
                  animation: jump-animation 1s .4s linear;
            }
            .dot-3 {
                  animation: jump-animation 1s .6s linear;
            }
      }
}
```

In this section, we define the behavior of the custom *jump-animation* function with the *@keyframes* at-rule. CSS function *rotateX()*, which defines a transformation that rotates an element, in this case the dot, around the horizontal X axis, was used. Function *rotateX()* is just one of many transform functions, which are CSS data types that represent a transformation of an element. These transformations can rotate, resize, distort or move an object in 2D or 3D space. [34]

Here we defined 5 stages of the dot animation, with the first and the last one set to 0 degree rotation, second and the second to last one set to 20 degree rotation, and the middle stage set to the highest, 30 degree rotation. This rotates, or transforms, the dot across the x axis until the middle stage, and then brings it back to the initial position. With that, we've defined the *jump-animation* and animated the illustration.

```
@keyframes jump-animation {
```

```
        0% {
              transform: rotateX(0deg); }
        25% {
              transform: rotateX(20deg); }
        50% {
              transform: rotateX(30deg); }
        75% {
              transform: rotateX(20deg); }
        100% {
              transform: rotateX(0deg); }
}
```

Animation demonstration, as well as the entire code, can be viewed on CodePen [Appendix 4].

7.2.2. Second (Marketing) Animation

The second, "Digital Marketing" animation, consisted of a search bar with the magnifying glass that moves across the bar. The illustration represents SEO analysis in digital marketing.

Once again, ***Directing focus*** principle was used as a purpose of animation to direct the user's attention. ***Ease in, ease out***, easing specified with a slow start and an end was used to create a gliding, scanning effect of the magnifying glass movement. To avoid dragging, animation ***timing*** was set only to 2 seconds.

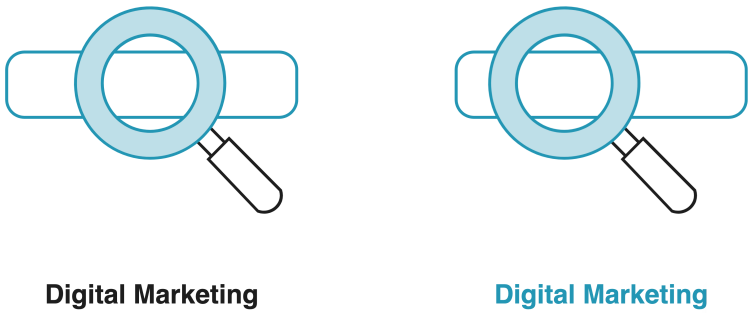

Figure 32: Digital Marketing Animation

This animation was defined through a custom *search-animation* function, which has been set to a duration of 2 seconds and a ease speed curve of the animation

```
.marketing {
	&:hover {
		.magnifier {
			animation: search-animation 2s ease;
		}
	}
}
```

Animation function *search-animation*, that we called upon for the *.magnifier* class, is defined similar to the animation for the previous illustration. CSS function *translateX()*, which is a part of transform function and uses *transform* property, repositions an element, in this case the *.magnifier* class, horizontally. *@keyframes* are used to set the stages of the animation. [34]

```
@keyframes search-animation {
	0% {
		transform: translateX(0); }
	33% {
		transform: translateX(-24px); }
	66% {
		transform: translateX(10px); }
	100% {
		transform: translateX(0); }
}
```

7.2.3. Third (Digital Design) Animation

"Digital Design" illustration represents an anchor point with two round paths as a simplified representation of creating vector illustrations. Its animation moves the smaller circle circularly along the outlines of the bigger circle.

As with the first animation, ***directing focus*** was used as a purpose of the animation, and ***easing*** was done linear. This was done to simplify the gravity's effect on a swinging animation, as using ease in and/or out property would go again against the natural movement. ***Timing*** was used to help with the natural movement of a swinging animation. The first, left side part of the animation, lasts longer than the second right one, as the ball loses momentum.

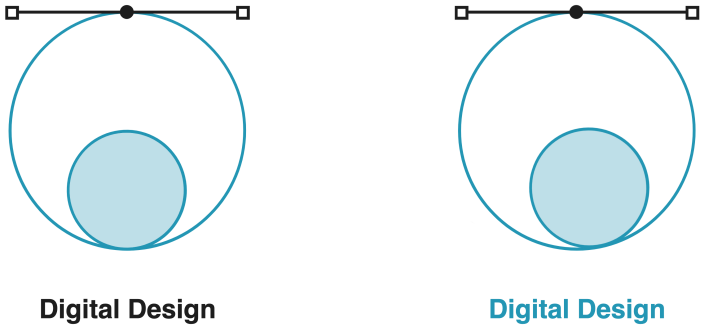

Figure 33: Digital Design animation

As with previous animations, custom function *curve-animation* is set to last 2 seconds and has a linear movement. CSS property *transform-origin* is used to set the origin of an element's transformation. [34] Default value of this property is *transform-origin 50% 50%,* which sets the origin to exactly the center of an element. On this animation, *transform-origin* is set on 50% for the x-axis and 60% for the y-axis. Horizontally, this leaves the origin in the center, but vertically places it just below the middle the illustration.

```
.design {
	&:hover {
		.circle_filled {
			animation: curve-animation 2s linear;
			transform-origin: 50% 60%;
		}
	}
}
```

Custom function *curve-animation* is defined with a CSS function *rotateZ()*, which rotates the element around the z-axis. [34] Same as with the animation of the first illustration, we defined 5 stages of the animation and set the rotation degree for each of them. Setting the rotation to be around the z-axis is what gives the circle a swinging effect.

```
@keyframes curve-animation {
	0% {
		transform: rotateZ(0deg); }
	25% {
		transform: rotateZ(35deg); }
	50% {
```

```
            transform: rotateZ(0deg); }
      75% {
            transform: rotateZ(-35deg); }
      100% {
            transform: rotateZ(0deg); }
}
```

7.2.4. Fourth (Blockchain Development) Animation

An illustration representing connections within the blockchain, the one for "Blockchain Development", is the only one whose animation rotates the entire illustration.

This animation has, perhaps, the most notable easing property. ***Ease out*** was used to simulate the rotation of a spinning wheel, which the illustration resembles. Again, ***directing focus*** was used for attentio grabbing.

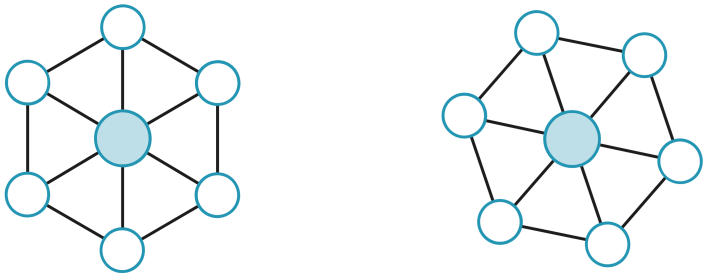

Figure 34: Blockchain Development animation

This custom animation, *rotate-animation*, is set to 2 seconds duration and has an ease-out movement which specifies an animation with a slow end. We also have a *transform-origin* property set to center, and *transform-box* property to fill-box. *Transform-box* is a CSS property defined the layout box to which the transform relates to, and *transform-box: fill-box* sets the object bounding box as the reference box and restricts our animation inside of the object. [34]

```
.blockchain {
	overflow: hidden;
	&:hover {
		.blockchain-group {
			animation: rotate-animation 2s ease-out;
			transform-origin: center;
			transform-box: fill-box;
		}
	}
}

@keyframes rotate-animation {
	0% { transform: rotateZ(0deg) }
	50% { transform: rotateZ(35deg) }
	100% { transform: rotateZ(0deg) }
}
```

7.3. Microinteractions

Microinteractions on input forms make them more intuitive by clearly indicating actions through transitions. The process of their design and implementation will be demonstrated on an example of changing input field states.

This animation process did not require designing illustrations, but rather specifying different states for the input field. This was done as a part of our design system in Figma (figma.com). Prototyping was also done in Figma, while the implementation was done in CSS, with JavaScript used for trigger validations.

Ease in, ease out was used in all of the transitions, which also played into ***visual continuity***. Transitions were smooth and connected, without any hard cuts. Transitions followed logical direction, which created ***clarity***, and ***directed focus*** of an user to either a need to input text or take note of an error.

Here are the *default*, *typing* and an *error* state, on which microinteractions will be demonstrated.

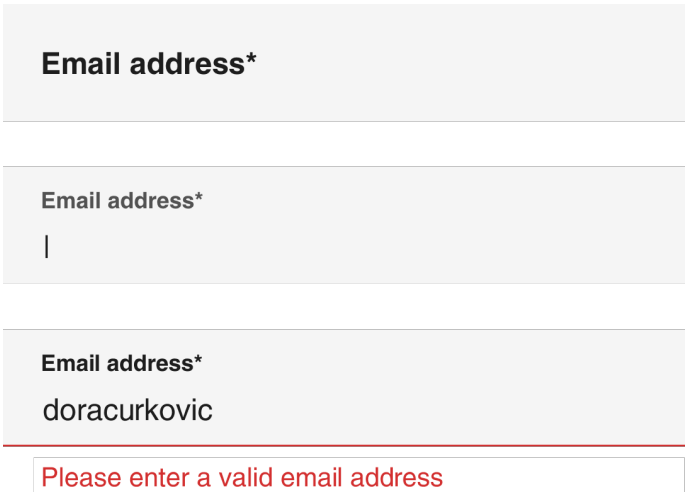

Figure 35: Input field states (default, typing, error)

As said, Figma was used tweak how the transitions will look like. In this case, we demonstrated simple interactions users will have with our input field. We first defined input field states as variants, then connected their interactions in a prototype.

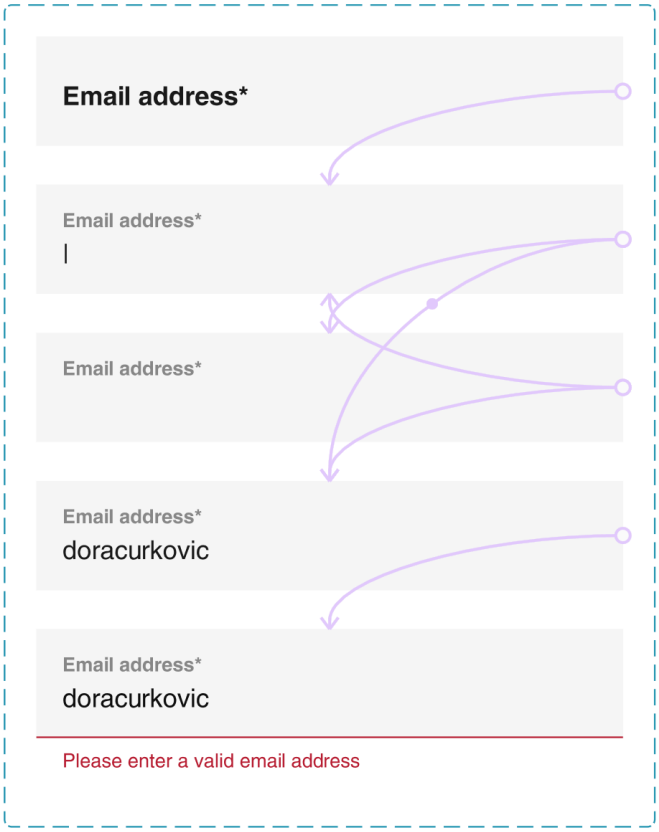

Figure 36: Input field microinteraction prototyping

We defined two transitions carefully - from the *default* to *typing* state, where the label shrinks, changes color and moves up, and from *typing* to *enter* state, where we have the appearance of the red bottom border and an error label.

Among many options, Figma uses the same animation properties we can later define with CSS - trigger, transition type and duration.

For the transition from the *default* to *typing* state, we defined a smart animate, easy in and out, duration of 300 milliseconds, that happens on the click of a mouse. In this case, Smart animate recognizes if the element's location has changed, and animates it to move from current to destination position, as well as its change of color and font size. [20]

For the transition from *typing* to *enter* state, we defined a dissolve, easy in and out, duration of 300 milliseconds, that happens with the confirmation of an input which does not pass the validation for that input field. We use dissolve because we don't need the element to change positions, just to appear on the screen.

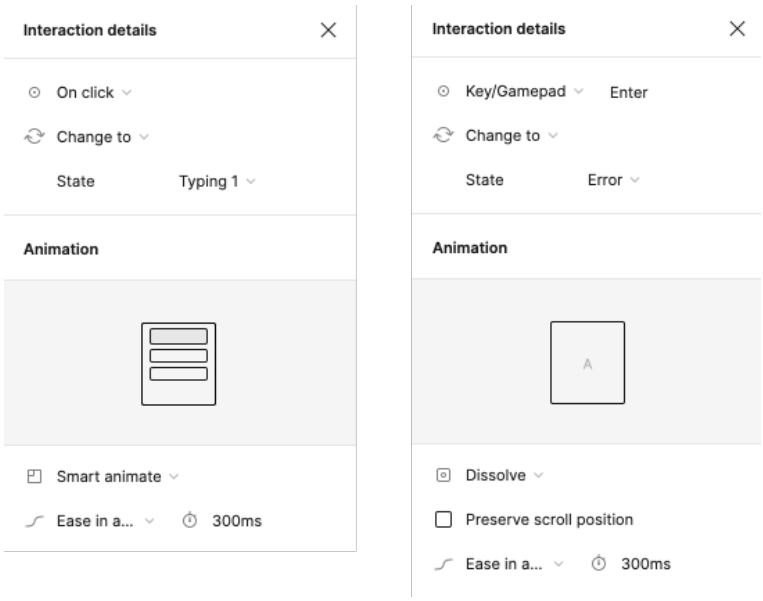

Figure 37: Input field microinteraction properties

Demonstration of this microinteraction, as well as all the prototype elements, can be seen on Figma [Appendix 3].

After clearly defining how our transitions will look, it is time for the implementation. Input field as such is defined with an HTML, with an input, label of the entire input field and an error message.

```
<div class="fieldset">
	<div class="input">
		<input id="email" type="text" data-targetable="true"/>
	<label for="email">Email address*</label>
	<span class="error">Please enter a valid email address</span>
	</div>
</div>
```

Defining the style, dimensions of an element and the animation was done in SCSS. CSS property *transition* was used to the animation, which enables us to define the transition between states of an element. [34] Parts of the code have been commented to better understand what different sections are for.

```
div.fieldset {
	display: grid;
	grid-template-columns: repeat(10, calc((100% - 9 * 30px) / 10));
	grid-column-gap: 30px;
	padding: 0;
	margin: 8px 0;

	.input {
	position: relative;

// Layout placement on the website grid
	&:nth-child(1) {
		grid-column: 2/6;
		&:last-child {
			grid-column: 2/10;
		}
	}

	&:nth-child(2) {
		grid-column: 6/10;
	}

	input {
		width: 100%;
		border: none;
		padding: 28px 16px 24px;
```

```
            cursor: inherit;
      }

// Defining style for label elements and span elements that have a class of
"file-name". Here we also define the transition, or the animation, of label
element with the transition property.
      label, span.file-name {
            position: absolute;
            font-weight: bold;
            color: $primary-dark;
            font-size: 16px;
            user-select: none;
            pointer-events: none;
            transition: all 0.3s ease-in-out;
            top: 32px;
            left: 16px;
      }

// We shift the position of label if the input is currently in focus or if
it contains "is-filled" class
      input:focus + label, &.is-filled label {
            top: 0;
            left: 8px;
            font-size: 12px;
            color: $header-grey;
      }

// Error message style
      .error {
            display: block;
            color: $secondary-light;
            font-size: 12px;
            text-align: left;
            padding: 8px 16px;
            opacity: 0;
            transition: opacity 0.3s ease-in-out;
}
      // Input filed style for error state
      &.is-error {
            input {
            border-bottom: 1px solid $secondary-light;
```

```
}
      .error {
            opacity: 1; }
            }
      }
}

input:focus,
input:valid {
      font-weight: bold;
}
```

To demonstrate microinteraction without the JS validations and embedded design system, here is a mock version with the interaction between default and typing states. Again, CSS property *transition* was used to define the microinteraction.

```
<form action="">
      <div class="input-field">
            <input type="text" id="name" required />
            <label for="name">Email address*</label>
      </div>
</form>

.input-field {
      position: relative;
      width: 250px;
      height: 44px;
      line-height: 44px;
      background-color: #f5f5f5;
}

label {
      position: absolute;
      top: 16px;
      left: 16px;
      width: 100%;
      color: #1d1d1d;
      transition: all 0.3s ease-in-out;
      cursor: text;
```

```
	font-weight: bold;
}

input {
	width: 100%;
	border: 0;
	outline: 0;
	padding: 28px 16px 24px;
	box-shadow: none;
	color: #252525;
	background-color: #f5f5f5;
}

input:invalid {
	outline: 0;
}

input:focus ~ label,
input:valid ~ label {
	top: -6px;
	font-size: 12px;
	color: #8a8a8a;
}
```

Both the animation and the code are available on CodePen for demonstration. [Appendix 5]

8. Conclusion

Animation in visual interfaces has come a long way. Long gone are the days of flashy, colorful GIFs. As design became more and more phycology related and designers started learning and thinking about how users interact with interfaces, it became clear that this was not what users wanted and added value to the product.

Interfaces today strive to be responsive, functional, and logical, along with visually pleasing. Animation can be both functional and decorative. It can reduce cognitive load by making interfaces more intuitive and improving navigation and reduce stress by providing feedback on actions and status updates. It can also be fun and engaging, making an interface less boring.

The animation process can, depending on the navigation, be extremely delicate. It is good to rely on the traditional principles mentioned in this thesis. Considering traditional principles will give a natural feel to the animation. We have analyzed how those principles translate into modern animation principles for visual interfaces. Interface animation has the aspect of user interaction, so we must consider the expectations that come with it. Speed and duration, secondary actions, easing, and overlapping are all properties that should be adjusted and affect how the animation feels and looks. Animation principles are a valuable asset that should help guide and create an animation that would enrich the user experience of the interface. Spotify app was used as an example of incorporating these principles and immersing brand's personality into animation.

Further discussed were microinteractions, which are what a lot of people think about when talking animation in interfaces. Has the action been validated? Was the button pressed? Is the input correct? All of this is said through microinteractions. They provide feedback, and information about the result of user's action with an interface and the inclusion of well-designed microinteractions can have a big impact on an interface.

An increase in the need for animations and awareness about its benefits has led to disparity with the implementation techniques. There was a need for animation, but no efficient way to execute it. Designers and developers have relied on image formats, like PNG and GIF, as well as massive code for implementation. This has proved to be inefficient, as it led to slow interfaces and robust upkeeping. This drawback has resulted in designer and developer teams trying to find better solutions for the problems they were facing themselves, as it is the case with Lottie, which came from the AirBnB team. Designers today have a lot more options to work with. Adobe After Effects is versatile and allows for the creation of complex animation, which can later be parsed with Lottie and implemented as a lightweight asset to native mobile

and web platforms. Another option is implementation through code, which can be done through CSS or JavaScript, both by hand and by using libraries. This requires a lot of attention, as it is difficult to translate animation properties if designers are not the ones implementing animation. To make this process easier, designers use tools like Figma, Principle, and Framer, which are excellent prototyping tools and allow for code export and animation demonstration.

However, they should be used with caution, bearing in mind that too many can have an opposite effect on usability and distract from what is important. The same effect is the result of random animations that have not been thought through and do not serve the interface hierarchy. Another important point to consider is implementation. Consulting with developers and having in mind that badly implemented animations can increase loading time.

All that was discussed has been demonstrated in a practical part of the thesis, where animation design was shown on a real-life web interface. Emphasis was put on the preparation and the implementation of SVG illustration animations and microinteractions, with the process of SVG optimization, Figma prototyping and implementation through CSS.

To conclude this thesis, animations have become essential, as users are now used to the benefits they offer. It has to be thought through, have a meaning and a purpose, and be brief and subtle. Used for providing feedback, grabbing attention, navigating users, or enriching branding. Finding a balance between usability and using animation as a design tool can be challenging, but with a successful execution that comes from planning and attention to detail, animation can be a powerful communication tool.

9. Bibliography

[1] V. Head, Designing Interface Animation: Meaningful Motion for User Experience, Rosenfeld, 2016.

[2] N. Baskanderi, "Medium - UX Collective "UI Animation: Please Use Responsibly"," 2017. [Online]. Available: https://uxdesign.cc/ui-animation-please-use-responsibly-e707dbdb12d5.

[3] J. Pratt, . P. V. Radulescu and A. R. Abrams, "It's Alive!: Animate Motion Captures Visual Attention," *Psychological Science,* vol. 21, no. 11, 2010.

[4] Nielsen Norman Group, "Animation for Attention and Comprehension," 2014. [Online]. Available: https://www.nngroup.com/articles/animation-usability/.

[5] F. Thomas and O. Johnston, The Illusion of Life: Disney Animation, Hyperion, 1995.

[6] R. A. D. Stefano, "The Principles of Animation," Electronic Visualization Laboratory, University of Illinois, Chicago, 1996.

[7] J. Ritchie, "IdeaRocket," 2017. [Online]. Available: https://idearocketanimation.com/13721-12-principles-of-animation-gifs/.

[8] I. Willenskomer, "Medium - Creating Usability with Motion: The UX in Motion Manifesto The following manifesto represents my answer to the question — "As a UX or UI, designer, how do I know when and where to implement motion to support usability?" Over the last 5 years, it," 2017. [Online]. Available: https://medium.com/ux-in-motion/creating-usability-with-motion-the-ux-in-motion-manifesto-a87a4584ddc.

[9] Material Design, "Material Design - Motion / Speed," [Online]. Available: https://material.io/design/motion/speed.html.

[10] P. Lewis, "Web.Dev - The basics of easing," 2019. [Online]. Available: https://web.dev/the-basics-of-easing/.

[11] M. Seelie, "Adobe Blog - Six Principles of Using Animation in UX Design," 2019. [Online]. Available: https://blog.adobe.com/en/2019/06/19/designing-animation-six-principles-using-animation-ux.

[12] G. Kaiser, M. Posniak and S. Bent, "Spotify - Reimagining Design Systems at Spotify," 2020. [Online]. Available: https://spotify.design/article/reimagining-design-systems-at-spotify.

[13] H. Winter, R. Marmelstein and F. Souza, "Spotify - Bringing the Spotify Heart to Life," 2020. [Online]. Available: https://spotify.design/article/bringing-the-spotify-heart-to-life.

[14] S. Gladkiy, "Medium - How You Can Improve UX with Microinteractions. Part I," [Online]. Available: https://medium.muz.li/how-you-can-improve-ux-with-microinteractions-part-i-73c8dfc01ca3.

[15] A. Joyce, "Microinteractions in User Experience," Nielsen Norman Group, 2018. [Online]. Available: https://www.nngroup.com/articles/microinteractions/.

[16] S. Suzuki, "Medium - Microinteractions in Twitter," 2018. [Online]. Available: https://medium.com/@shingo2000/microinteractions-in-twitter-e8affd76d0f1.

[17] R. Raghavan, "Acodez - All about microinteractions in UX," 2022. [Online]. Available: https://acodez.in/microinteractions-user-experience/#_3_Standby.

[18] Google, "Material Design - Text Fields," [Online]. Available: https://material.io/components/text-fields.

[19] A. Sl, "Figma: 5 ways to add animation to your designs," [Online]. Available: https://uxdesign.cc/figma-5-ways-to-add-animation-to-your-designs-e3c521aa8902.

[20] Figma, "Figma - Create advanced animations with smart animate," [Online]. Available: https://help.figma.com/hc/en-us/articles/360039818874-Create-advanced-animations-with-smart-animate.

[21] P. Chen, "Prototyping design animations with Framer X," 2018. [Online]. Available: https://uxdesign.cc/prototyping-design-animation-with-framer-x-240ae6c4caa8.

[22] Framer, "Framer Motion," [Online]. Available: https://www.framer.com/motion/.

[23] Jenn, "Creating a Principle prototype animation in 30 minutes," 2020. [Online]. Available: https://uxdesign.cc/creating-a-principle-prototype-animation-in-30-minutes-for-the-complete-beginner-93dd7b41a47e.

[24] E. Chau, "Crash Course in Prototyping with Principle for Web and UX/UI Designers," 2018. [Online]. Available: https://medium.com/sketch-app-sources/crash-course-in-prototyping-with-principle-for-web-and-ux-ui-designers-intro-1b5b32935d1c.

[25] B. Berger, "Principle : The Prototyping Tool you've got to try," 2015. [Online]. Available: https://medium.com/swlh/principle-the-prototyping-tool-you-got-to-try-93ab743fe4ae.

[26] Principle, "Principle Documentation," [Online]. Available: https://principleformac.com/docs.html#tap.

[27] C. Charniga, "Spotify - How we Brought 2020 Wrapped to Life in the Mobile App," 2020. [Online]. Available: https://spotify.design/article/how-we-brought-2020-wrapped-to-life-in-the-mobile-app.

[28] G. Pino, "Create fast & customizable animations with Smart Animate," 2020. [Online]. Available: https://bootcamp.uxdesign.cc/create-fast-customizable-animations-with-smart-animate-figma-aec3bf11208b.

[29] W3Schools, "W3Schools - SVG," [Online]. Available: https://www.w3schools.com/graphics/svg_intro.asp.

[30] Airbnb, "Airbnb - About Us," [Online]. Available: https://news.airbnb.com/about-us/.

[31] AirBnB, "AirBnB - Lottie," [Online]. Available: http://airbnb.io/lottie/#/.

[32] LottieFiles, "Why Use Lottie," 2020. [Online]. Available: https://lottiefiles.com/blog/working-with-lottie/why-use-lottie.

[33] Lottie, "Lottie - Community showcase," [Online]. Available: https://airbnb.io/lottie/#/community-showcase.

[34] MDN, "MDN Web Docs - CSS," 2022. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/CSS.

[35] I. Willenskomer, "Medium - "UI Animation Principles: Disney is Dead"," 2016. [Online]. Available: https://medium.com/ux-in-motion/ui-animation-principles-disney-is-dead-8bf6c66207f9.

[36] AirBnB, "GitHub AirBnB - Lottie Supported Features," 2021. [Online]. Available: https://github.com/airbnb/lottie/blob/master/supported-features.md.

[37] D. Eden, "Animate.css," [Online]. Available: https://animate.style/.

10. Figures

11. Appendix

Appendix 1: D. Ćurković and A. Jedvaj, Async Labs Behance casestudy. This is a casestudy of the design project discussed in the practical assigment. (behance.net/gallery/114104749/Async-Labs-Website).

Appendix 2: Async Labs, Async Labs website. This is the current website of Async Labs digital agency. (asynclabs.co).

Appendix 3: D. Ćurković, Async Labs animations prototype. This is a Figma prototype file with prototypes for software development animation and inputfield microinteractions. (figma.com/proto/sTQaWWltHbgLfoYaF8BBzC/Dora-%C4%86urkovi%C4%87---Master's-Thesis?node-id=31%3A5&scaling=min-zoom&page-id=0%3A1&starting-point-node-id=31%3A5&show-proto-sidebar=1)

Appendix 4: D. Ćurković, Async Labs software development animation on CodePen. This is a sandbox version of the animation, written in HTML and CSS. (codepen.io/DoraCurkovic/pen/QWgOrze)

Appendix 5: D. Ćurković, Async labs input microinteraction on CodePen. This is a sanbox version of the animation, written in HTML and CSS. (codepen.io/DoraCurkovic/pen/dyebNrm)