



---

# **Functional reactive programming in real-time web applications**

by Elias MEIRE

---

*Internship company:*  
Showpad

*Supervisor:* Rogier van der Linde  
*Mentor:* Laurens Dewaele

2016-2017

3<sup>rd</sup> year bachelor Electronics-IT

Odisee university college

## **Abstract**

To be written after the main content is completed.

**Keywords:** *Functional reactive programming, real-time, web development*

## **Acknowledgments**

To be written after the main content is completed.

# Contents

<b>Abstract</b>	<b>1</b>
<b>Acknowledgments</b>	<b>2</b>
<b>List of Abbreviations</b>	<b>5</b>
<b>1 Internship company</b>	<b>6</b>
1.1 Internal structure . . . . .	6
1.1.1 Engineering . . . . .	6
1.2 Culture . . . . .	8
1.2.1 Showings . . . . .	9
1.2.2 Events . . . . .	10
<b>2 Internship assignment</b>	<b>11</b>
2.1 Web app rewrite . . . . .	11
2.2 Web app speedmonitor . . . . .	11
2.3 Showkiosk . . . . .	11
2.4 Relation to bachelor thesis . . . . .	11
<b>3 Planning</b>	<b>12</b>
<b>4 Research question</b>	<b>14</b>
4.1 Metrics . . . . .	14
4.2 Approach . . . . .	14
<b>5 Real-time on the web</b>	<b>15</b>
5.1 HTTP Polling . . . . .	15
5.2 HTTP long polling . . . . .	15
5.3 WebSockets . . . . .	16
5.3.1 Libraries . . . . .	16
5.4 WebRTC . . . . .	16
<b>6 Functional reactive programming</b>	<b>18</b>
6.1 Functional programming . . . . .	18
6.1.1 Declarative programming . . . . .	18
6.1.2 First-class functions . . . . .	19
6.1.3 Immutability . . . . .	19
6.1.4 Other functional jargon . . . . .	20
6.2 Reactive programming . . . . .	20
6.3 FRP in JavaScript . . . . .	21

6.3.1 Asynchronous Javascript . . . . .	22
6.3.2 Current state . . . . .	24
<b>7 Case study: FRP in real-time data flows</b>	<b>25</b>
<b>Conclusion</b>	<b>26</b>
<b>References</b>	<b>27</b>
<b>Bibliography</b>	<b>28</b>

## List of Abbreviations

<b>SaaS</b>	Software as a Service
<b>B2B</b>	Business-to-business
<b>ARR</b>	Annual recurring revenue
<b>FRP</b>	Functional reactive programming
<b>QA</b>	Quality assurance
<b>UI</b>	User interface
<b>Ajax</b>	Asynchronous JavaScript and XML
<b>P2P</b>	Peer-to-peer
<b>I/O</b>	Input/output
<b>API</b>	Application programming interface

# 1 Internship company

I did my internship at Showpad in Ghent. Showpad is a software as a service start-up that was founded in 2011 by ex-employees of In The Pocket, a mobile agency. Since then the company has grown at a fast pace, today Showpad has over 1000 customers and over 200 employees in offices in Ghent, London, San Francisco and Portland [1].

Showpad provides a B2B service that aims to align marketing and sales departments by making marketing materials easily accessible and shareable. Showpad achieves this by hosting a platform and developing native applications for Android and iOS as well as a web application.

Showpad aims its service at enterprise customers with over 250 employees. Showpad's customers include Coca-cola, Audi, Johnson & Johnson and many other big names [2]. These customers pay Showpad a monthly subscription fee per user for the SaaS.

Showpad has been doubling its revenue every year for the past 4 years. At the end of 2015 Showpad's annual recurring revenue exceeded \$10 million [3]. In May 2016 Showpad closed a \$50 million funding round led by Insight Venture Partners [1].

## 1.1 Internal structure

Internally Showpad is divided into 5 departments, listed in order of descending size these are: customer success, engineering, sales, marketing and employee success. As can be deduced from this order, Showpad invests heavily in customer experience. Showpad prides itself on its very short customer issue cycle time and a strong synergy between customer success and engineering is essential to achieve this.

### 1.1.1 Engineering

#### Teams

The engineering department is divided into 6 smaller teams with around 10 members. These teams are not divided by product or discipline but rather by the functionality they implement and maintain. An example of this is the create & present team. That team maintains all user functionality related to creating and presenting content in Showpad.

The engineering teams in Showpad are discover & measure, distribute & collaborate, share & personalize and create & present. Every team consists of front-end and back-end developers as well as at least one quality assurance engineer and a team coach. There are also two teams that aren't linked to specific functionality. Those teams are the mobile & learn team and the architecture team because these are two

smaller teams that can't be linked to a specific user functionality. Even though these are separate teams there is still a lot of collaboration between different teams.

During my internship I was a part of the share & personalize team. This team maintains all features related to sharing and personalizing content in all Showpad products (except the mobile apps).

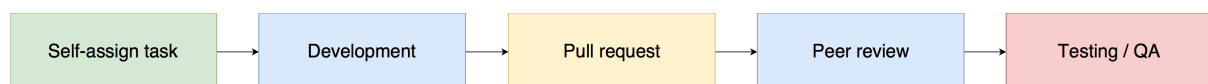
## Guilds

In Showpad engineering there are guilds for every development discipline. These 4 guilds are the front-end guild, the back-end guild, the architecture guild and the QA guild. Every guild has a separate group chat to share interesting articles or new developments in their areas of expertise. The guilds also have a weekly meeting to talk about any topic within their field. In these meetings every guild member has the opportunity to give a presentation or workshop on something they find interesting in their discipline. During my internship I was a part of the front-end guild.

## Kanban

Every team has a Kanban board [4] on which they receive and create new development tickets. These tickets are then taken through a development cycle which includes a peer code review and a testing session (see figure 1.1). Sometimes there are separate Kanban boards for epics. Epics are large features that typically require a long time in development and are not specific to one team. An example of an epic I took part in during my internship is the complete rewrite of the web application in Angular 2.

For distributing tickets between developers Showpad engineering uses a pull system instead of a push system. This means that developers self-assign tasks they feel comfortable working on instead of tasks being pushed to developers by a team coach.

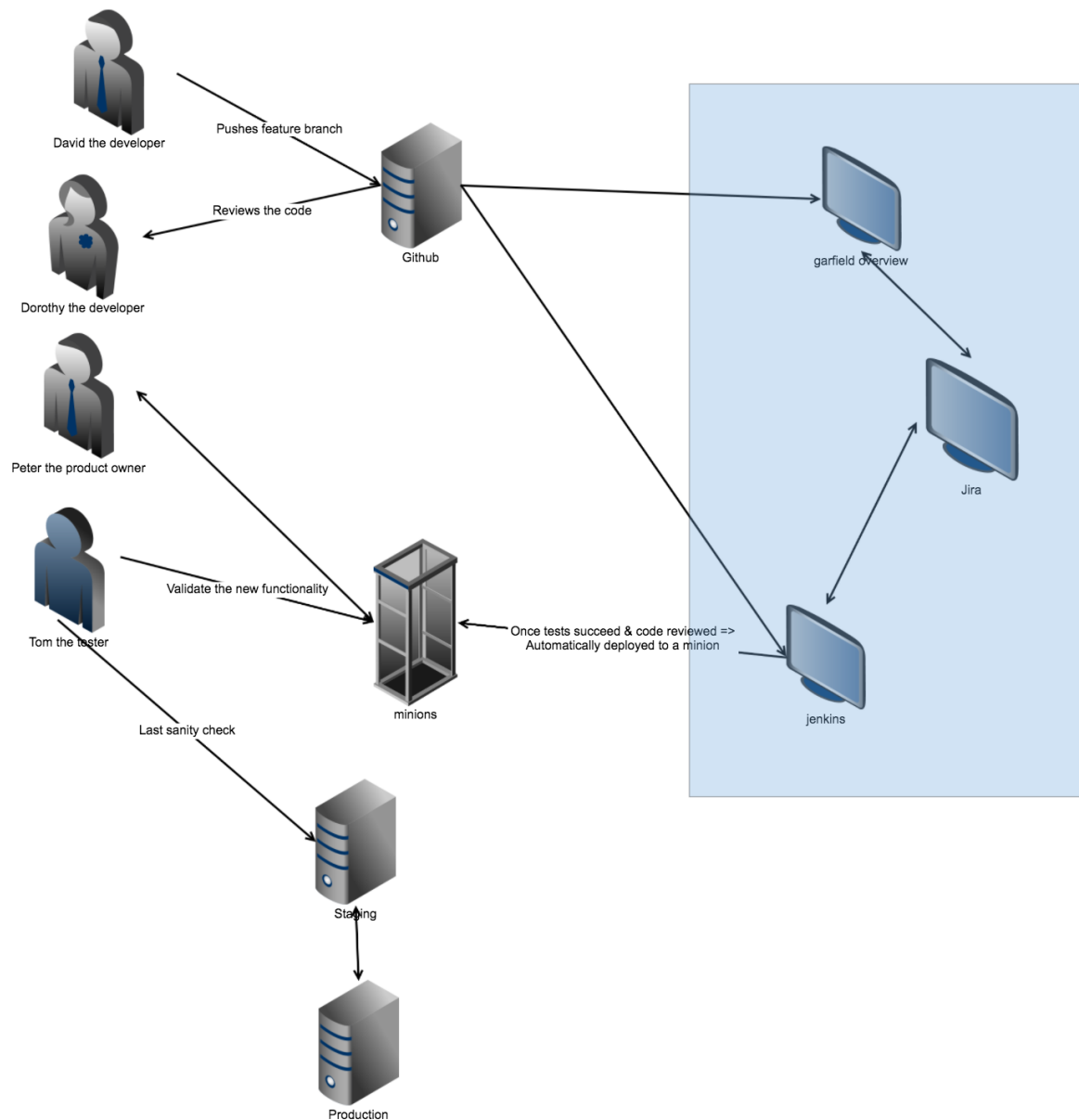


**Figure 1.1:** Ticket development cycle in Showpad

## Continuous integration

Showpad engineering puts a lot of effort into continuous integration and deployment. The development process is largely automated (see figure 1.2). Every commit that is pushed to a git repository gets built and tested automatically by a build server. Furthermore every pull request that is approved in a code review gets deployed to a minion where it gets tested by a QA engineer. If the QA engineer approves the changes and verifies the functionality the pull request will be deployed on the staging servers. On the staging servers a last sanity check is done to ensure everything is working. If there are no issues on staging the code will be added to a release and deployed to production in cycles of about a week on average. This very fast development cycle results in a rapidly improving product and a very quick response time to customer reported issues.





**Figure 1.2:** *Showpad developer workflow diagram*

## 1.2 Culture







Showpad has a very open culture that is typical for a start-up in the technology scene. The office is designed to be as open as possible to encourage interdepartmental collaboration and create a good atmosphere in general (see figure 1.3).



**Figure 1.3:** *Showpad office atmosphere*

## 1.2.1 Showings

Showpad outlines 6 characteristics called showings that a good Showpad employee should have. These showings are shown in figure 1.4.

- 
**KEEP IT SIMPLE**
  - We understand that less is more
  - We aim for **maximum impact**
- 
**TRANSPARENCY BY DEFAULT**
  - **We share** fast
  - We share the why
- 
**GIVE TO GROW**
  - We share knowledge for mutual growth
  - We believe **feedback** is a gift
- 
**BE HUMBLE**
  - We take our job seriously but ourselves not so much
  - We value **team success** over personal wins
- 
**TAKE OWNERSHIP**
  - We think and act like leaders
  - We are **pro-active**
- 
**BE PASSIONATE**
  - We love **what we do**
  - We do what we love

**Figure 1.4:** *The 6 showings*

### **1.2.2 Events**

Showpad organizes a lot of events for its employees and customers. Every Friday evening there is an event for all employees where they can have drinks together to set off the weekend. Every other week there is also a Showpie on Friday. This is an event where all employees in all offices get together in a video call and there is a moment to ask questions to the leadership. Sometimes Showpie also includes a speech or presentation by the CEO.

Showpad engineering organizes their own yearly developer conference in Ghent called GEARS [5]. This is a free event open to all web developers. The event hosts talks and presentations by Showpad engineers and external speakers from leading technology companies. There are open donations on the event that go to a charity of Showpad's choosing.

Lastly Showpad organizes a yearly event for its customers called Showtime. Showtime gives Showpad customers a chance to share their experience with the product [6].

## **2 Internship assignment**

My internship assignment at Showpad will consist of 3 main components: the web app rewrite, speed monitor and the Showkiosk.

### **2.1 Web app rewrite**

The largest part of my internship at Showpad will be spent doing work on the web app v2. Showpad web app v2 is a complete rewrite of the original web app in Angular 2+ and RxJS. The decision to rewrite the web app was made in July 2016 because of two major reasons. The first reason was that web app v1 was written as one big monolith. As a result development on the project was tedious, especially for new developers. The second reason is that the technologies that v1 was made with have become heavily outdated (ex. Angular 1).

During my internship I will work on the web app v2 until it gets released to customers. Afterwards I will still be part of development team on web app v2. This development will include fixing bugs and implementing new features.

### **2.2 Web app speedmonitor**

The web app speed monitor will be a data visualisation that will measure and display the performance of the web app in real-time. Some of the technologies used in this project will be React, RxJS, Redux and Websockets. This application will be developed from scratch during my internship.

### **2.3 Showkiosk**

The Showkiosk is an internal tool to display important message on screens throughout the company. Key technologies used in this project are Angular 2+ and Google Firebase.

### **2.4 Relation to bachelor thesis**

The assignments in my internship are all related to the topic of my bachelor thesis. These assignments are outside the scope of this thesis but lessons learned from using the technologies on these projects will be used as a basis to understand them and write this bachelor thesis.

### 3 Planning

Step	Content	Target date	Real date
<b>Deadlines</b>			
1.	Create planning		
2.	Decide title for bachelor thesis	20/03	16/03
3.	Submit first part of bachelor thesis	27/03	27/03
4.	Intermediate evaluation by internship mentor	31/03	
5.	Submit second part of bachelor thesis	24/04	
6.	Submit bachelor thesis	26/05	
<b>Research</b>			
1.	Functional Reactive programming	20/02	19/02
1.	Declarative vs. imperative programming	20/02	19/02
2.	Real time on the web (Websockets, long polling...)	27/02	28/02
2.	Reactive implementations in Javascript	27/02	26/02
3.	Elm: A purely functional language for the web	06/03	10/03
4.	FRP in user interfaces	13/03	13/03
4.	Further research on FRP	13/03	13/03
5.	Matching FRP with Real-time applications	20/03	23/03
<b>Internship</b>			
1.	<u>Web app v2 (Angular 2+, RxJS rewrite)</u>		
	Verify existence of issues reported in v1 that are still in v2 and help solving them	20/03	17/03

	Maintenance, bug fixes and improvements	12/05	
	Help develop new feature (Locked pages project)	12/05	
2.	<u>Speed monitor (Websockets, React)</u>		
	Help develop team tool to monitor speed of web app	12/05	
3.	<u>Showkiosk (Angular 2+, Google Firebase)</u>		
	Help develop Internal communication tool to display messages throughout the entire company (1day/week)	12/05	

**Table 3.1:** *Planning for internship and bachelor thesis*

## **4 Research question**

In this bachelor thesis research will be conducted on how functional reactive programming can be beneficial to development of real-time web applications. The research question that will be answered is the following: "Can functional reactive programming be used to make developing real-time web applications more efficient?".

### **4.1 Metrics**

The metrics that will be used to assess whether the use of FRP is beneficial to real-time web application development are the following: Amount of code, readability, learning curve, performance and extensibility.

### **4.2 Approach**

The first part of this bachelor thesis will research FRP and real-time dataflow on the web today. Because this bachelor thesis is applied to the web all code examples will be in JavaScript.

To come to a conclusion a practical case study will be conducted. The research question will be evaluated by implementing a simple case in a traditional imperative style and in a declarative FRP style. The metrics defined in section 4.1 will then be used to compare the two implementations and formulate a conclusion.

The case that will be implemented in this bachelor thesis will be a simple text-based chat application. This is the simplest example of a real-time application with full-duplex communication. To measure extensibility a feature will be added to the finished application and the degree of difficulty will be assessed.

## 5 Real-time on the web

### 5.1 HTTP Polling

The simplest way to do real-time on the web is to poll a server via Ajax at a certain interval. This is a simple but very primitive way to update data in real-time. This technique has multiple downsides. Firstly the communication is not really real-time and your data will always be out of date because of the usage of an interval. Secondly this technique also increases load on your server because it has to handle a lot of requests from the clients. This makes scaling with a HTTP polling architecture very cumbersome [7]. Lastly this technique cannot do two-way communication between server and client so this it is not suitable for real-time communication [7].

### 5.2 HTTP long polling

HTTP long polling is a variation on HTTP polling. The client will still poll the server for new data but the server will keep the request open until it has new data. After the client gets a response it will immediately send a new request, repeating the process [8] (see figure 5.1). This technique resolves most downsides of classic HTTP polling but it still cannot facilitate two-way communication between server and client [7].

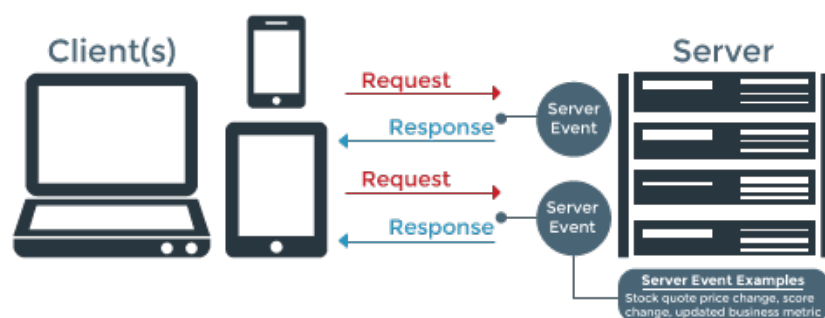


Figure 5.1: Long polling [8]



## 5.3 WebSockets

HTML5 WebSockets are a full-duplex communication channel (see figure 5.2) that operates through a single socket over the web. HTML5 WebSockets is not just another small improvement over conventional HTTP communications [7], it is a completely new communication protocol. Under the hood WebSockets are built on TCP; they do not use HTTP except for establishing the connection [9]. Browser support for WebSockets is excellent, all major browsers support them [10].

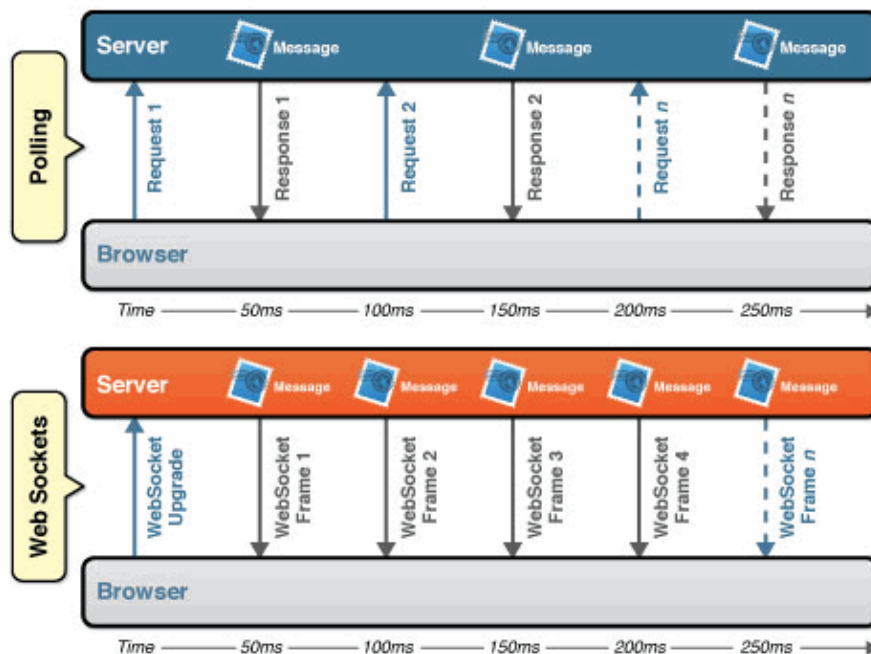


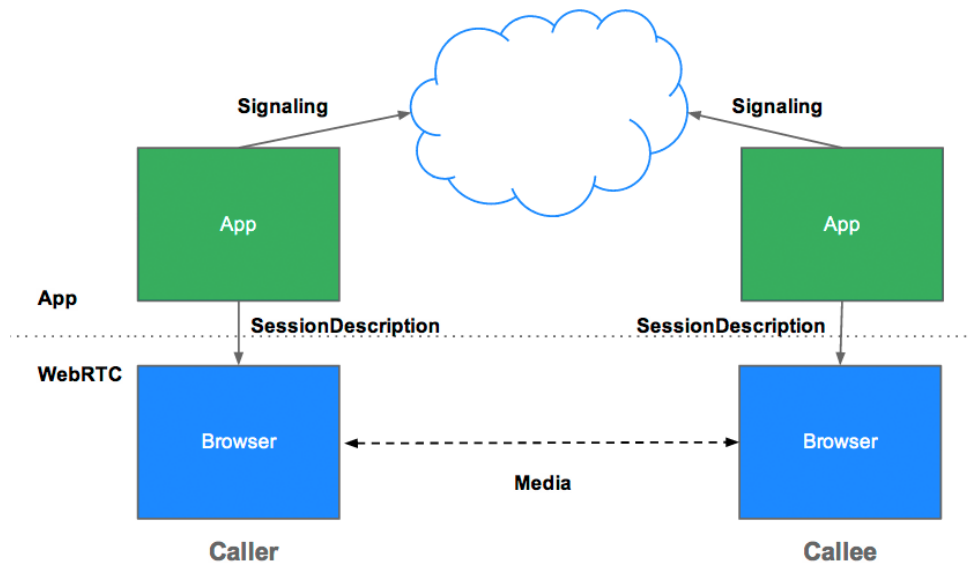
Figure 5.2: Comparison of HTTP polling and WebSockets [7]

### 5.3.1 Libraries

There are many libraries that abstract away some of the complexity from the WebSocket protocol. The most popular WebSocket libraries at the time of writing are Socket.IO, Primus, ws and Faye. The most popular reasons to choose a library over using the vanilla version are elegant fallbacks for browsers that do not support WebSockets and automatic reconnection.

## 5.4 WebRTC

WebRTC is full-duplex communication protocol developed and open sourced by Google in 2010 [11]. After that WebRTC was standardised by the IETF and the W3C [11]. Now there are implemented open standards for real-time, plugin-free video, audio and data communication. WebRTC allows real-time peer-to-peer communication of audio, video and arbitrary data between browsers. A server is only needed to set up the initial connection between peers, this process is called signaling [12] (see figure 5.3).



**Figure 5.3:** *Signaling process for WebRTC [11]*

WebRTC's P2P connection model further increases scalability and simplicity of real-time applications [12]. However WebRTC also has some caveats. Even though it has been around for 4 years now browser support is still lacking at the time of writing [13]. It is also hard to implement WebRTC in a client-server architecture [14] as it was primarily made for P2P communication.

## 6 Functional reactive programming

FRP is a programming paradigm that integrates the idea of asynchronous dataflows into functional programming. This provides an elegant way to express computation for areas that involve a lot of time related variables such as animation and user interface rendering [15][16].

### 6.1 Functional programming

Functional programming is a programming paradigm that models computation as a composition of pure functions without mutating any state [17]. It is a declarative programming paradigm. One of the biggest advantages of functional programming is that by using pure functions it becomes much easier to reason about the behavior of a program [17].

Functional programming has its origins in lambda calculus, a system in mathematical logic for expressing computation based on operations and variables [18].

#### 6.1.1 Declarative programming

In declarative programming a program describes what to do, rather than how to do it. This is the opposite of classic imperative programming [19]. The difference between the two is best explained by example. Listing 6.1 and listing 6.2 both show an implementation of a function called double. The function takes an array and returns another array with all elements doubled. Listing 6.1 shows an imperative implementation of this function, the function explicitly says to loop over the elements, double them and add them to an empty array. Listing 6.2 shows a declarative implementation of this function, it only says what we want to happen to the elements in the array.

---

**Listing 6.1:** *Imperative implementation of the double function*

---

```
function double (arr) {  
  let results = [];  
  for (let i = 0; i < arr.length; i++) {  
    results.push(arr[i] * 2);  
  }  
  return results;  
}
```

---

---

**Listing 6.2:** *Declarative implementation of the double function*

---

```
function double (arr) {  
  return arr.map(item => item * 2);  
}
```

---

### 6.1.2 First-class functions

In functional programming functions are first-class citizens. This means that functions can be treated like any other value. They can be created, stored in data structures and passed into or returned from other functions.

#### Pure functions

Pure functions have two defining characteristics. The first is that the function's result only depends on its arguments, given the same arguments the function will always return the same value [19]. The second characteristic is that pure functions do not cause any side effects such as mutations to data structures or output to I/O devices [19].

#### Higher order functions

Higher order functions are functions that either take a function as an argument, return a function or both. This is a simple concept that makes function composition possible. Function composition helps keeping functional programs readable and easy to reason about [19].

An example of a higher order function in JavaScript is the map function of the Array prototype (see listing 6.2). This function takes a function as an argument. The invokeTwice function in listing 6.3 is an example of a higher-order function that returns a function.

---

**Listing 6.3:** *Higher order function composition*

---

```
const invokeTwice = f => subject => f(f(subject));
const double = x => x * 2;
const quad = invokeTwice(double);

quad(10); // 40
invokeTwice(double)(10); // 40
```

---

The invokeTwice function will take a function as an argument and return a function that applies the first function twice on its argument. The example also shows how invokeTwice can be composed together with other functions such as double. A function like invokeTwice that takes one argument at a time and returns a function taking the next argument is called a curried function. [17].

### 6.1.3 Immutability

Functional programming promotes the use of immutable data structures, most functional languages even enforce it [17]. Immutability means that a data structure cannot be changed. If an operation is done on an immutable object it copies the value, mutates it and passes back the result. This helps eliminate bugs where an object enters a state that was not predicted by the programmer [19].

### 6.1.4 Other functional jargon

There are many other terms associated with functional programming but these are out of scope for this thesis. This chapter only serves to give a basic understanding of the functional programming paradigm.

## 6.2 Reactive programming

Reactive programming is programming paradigm that deals with asynchronous data streams [20]. These data streams will then propagate their changes to other parts of the application; this is the observer pattern [21]. Reactive programming has three main objects: an observable or a stream, an observer or subscriber and a scheduler [20].

A stream will emit values over time. A stream can be infinite or it can complete after a certain number of emitted values. Observers can subscribe to these streams and get notified whenever the stream emits a value, throws an error or completes (see listing 6.4 for an example). Schedulers decide on which thread an observer's code should run and when.

---

**Listing 6.4:** *An observable and observer in RxJS*

---

```
const someObservable$ = Observable.of([1,2,3]);
```

```
someObservable$  
  .subscribe(  
    value => console.log(value),  
    error => console.err(error),  
    completed => console.log('completed')  
  );
```

```
// output: 1, 2, 3, completed
```

---

Typically a reactive framework or language includes a lot of operators [20] so that these three objects can be easily worked with. An example of operators on an data stream can be seen in figure 6.1, this example illustrates a stream of clicks by a user. The events in the stream are first grouped by throttling them for 250ms. After that they are mapped to the length of the groups and finally they are filtered so that only groups of clicks larger than 3 remain. This example shows really well how reactive programming helps with handling asynchronism and concurrency in modern applications.

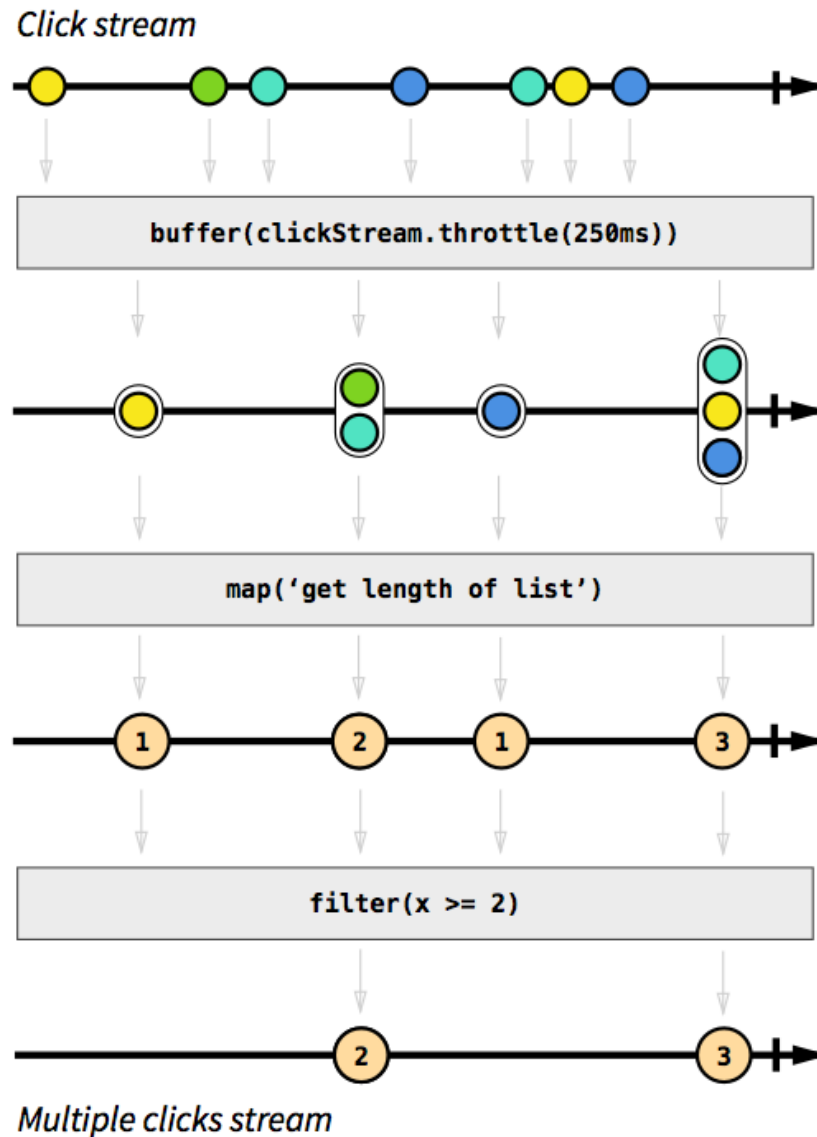


Figure 6.1: Stream of clicks [20]

## 6.3 FRP in JavaScript

Because of JavaScript's unopinionated nature, choice of programming paradigm is left to the developer. JavaScript also has a lot of the ingredients needed for FRP built into the language. Functions in JavaScript are already first-class citizens of the language. Arrow function allow JavaScript functions to be reasoned about as lambda calculus and finally ES2015 introduced a few functional array functions such as map, reduce and filter [22].

Because of these characteristics JavaScript is an excellent language for functional reactive programming.

### 6.3.1 Asynchronous Javascript

Dealing with asynchronism has always been one of the biggest challenges of programming for the web. Over the years developers have employed different techniques to handle asynchronous code in JavaScript.

#### Callbacks

Callbacks are the simplest way to handle asynchronism in JavaScript. The principle is simple; a callback function is passed as an argument with another function. That function will then call the callback function when it finishes its work.

The problem with callbacks is that they quickly become unreadable when the programmer needs to compose different asynchronous actions. Listing 6.5 is an example of 3 asynchronous functions running one after the other. This example code is already pretty complex even though it accomplishes a very simple task.

---

**Listing 6.5:** *Fetching data with nested callbacks*

---

```
getData(function (err, x) {  
  if(err) {  
    console.error('Oh no! an error: ${err}');  
    return;  
  }  
  getMoreData(x, function (err, y) {  
    if(err) {  
      console.error('Oh no! an error: ${err}');  
      return;  
    }  
    other(x, y, function (err, z) {  
      if(err) {  
        console.error('Oh no! an error: ${err}');  
        return;  
      }  
      ...  
    });  
  });  
});
```

---

#### Promises

Promises are an alternative way to handle asynchronous code. Promises were standardized in ES2015 but have been around for years in various libraries such as Bluebird [23]. The Promise API allows the developer to compose asynchronous code in a more declarative manner. A promise can either resolve and deliver a value or reject and throw an error.

Listing 6.6 shows the example of listing 6.5 rewritten with the JavaScript Promise API. The catch function will handle all errors on a chain of promises.

---

**Listing 6.6:** *Fetching data with the Promise API*

---

```
getDataPromise
  .then(x => getMoreDataPromise(x))
  .then(y => otherPromise(y))
  .then(z => ...)
  .catch(err => console.error('Oh no! an error: ${err}'));
```

---

If the promises in the example would not depend on the result of the previous one the Promise.all function could be used. This function will complete when all promises have resolved. The Promise.all method will throw an error if one of the promises rejects.

## Async/await

Async functions and the async/await syntax was recently introduced to JavaScript in ES2017 [24]. It is a new language construct to handle asynchronous code. The await operator will block code execution until an async function has completed. The example used in listing 6.5 and listing 6.6 is rewritten below in listing 6.7 using the async/await syntax.

---

**Listing 6.7:** *Fetching data with the async/await keywords*

---

```
async function getAllData() {
  try {
    const x = await getData();
    const y = await getMoreData(x);
    const z = await other(y);
    ...
  } catch(err) {
    console.error('Oh no! an error: ${err}');
  }
}
```

---

## Functional Reactive Programming

FRP provides a more versatile API to handle asynchronous code than Promises and async/await. As Akash Agrawal correctly notes in his article "What Promises Do That Observables Can't" observables completely overshadow promises in terms of functionality [25].

Listing 6.8 shows the example from the listings in the previous paragraphs rewritten using RxJS observables. The getMoreDataObservable and getOtherObservable function in this example return an observable based on their parameter. While the implementations with promises and async/await are slightly more concise FRP offers a lot more functionality than both. RxJS observables for example



provide operators to cancel, buffer and debounce asynchronous code [26]. Promises or `async/await` provide no API for those functionalities.

---

**Listing 6.8:** *Fetching data with FRP (RxJS)*

---

```
dataObservable$
  .flatMap(x => getMoreDataObservable(x))
  .flatMap(y => getOtherObservable(x))
  .catch(err => {
    console.error('Oh no! an error: ${err}');
  })
  .subscribe(z => {
    ...
  });
```

---

### 6.3.2 Current state

Many frameworks already bring FRP to the language. The most popular ones at the time of writing are RxJS, xstream and Bacon.js [20]. There is also a proposal out there to bake an observable data structure into a future version of ECMAScript [27].

In 2013 Evan Czaplicki and Stephen Chong published a thesis at the Harvard University about the creation of Elm [28]. Elm is a practical FRP language with two important features: high-level abstractions to support simple, declarative FRP and purely functional graphical layout calculation [28]. Elm is also known for its excellent tooling and the absence of runtime errors in the language. Most importantly the Elm language is built on top of JavaScript and is made with the web in mind.

The FRP paradigm has been around for a very long time but it is still very new to the web platform. The need for a good abstraction to handle asynchronous code can be a driving factor to increase adoption of FRP in JavaScript. Possible support for observables in the JavaScript language can also further increase developer adoption in the future.

## 7 Case study: FRP in real-time data flows

TODO

**Listing 7.1:** *Some Javascript code*

---

```
// Hello.js

const message = 'Hello';
const numbers = [1, 2, 3];

numbers
  .map(n => `${message} ${n}`)
  .forEach(n => console.log(n));

// Hello 1
// Hello 2
// Hello 3
```

---

## Conclusion

## References

- [1] N. Lomas. (2016). Showpad closes \$50m series c to keep growing its saas sales productivity platform, [Online]. Available: <https://techcrunch.com/2016/05/19/showpad-closes-50m-series-c-to-keep-growing-its-saas-sales-productivity-platform/> (visited on 2017-03-02).
- [2] Showpad. (2017). Smarter sales enablement, [Online]. Available: <https://www.showpad.com/who-we-are/> (visited on 2017-03-02).
- [3] J. M. Lemkin. (2015). Coming to the u.s.: Learnings at ~\$10m arr from showpad and conversocial, [Online]. Available: <https://saastr.quora.com/Coming-to-the-U-S-Learnings-at-10m-ARR-From-Showpad-and-Conversocial> (visited on 2017-03-06).
- [4] D. Radigan. (2016). How the kanban methodology applies to software development, [Online]. Available: <https://www.atlassian.com/agile/kanban>.
- [5] N.N. (2017). Gears17, [Online]. Available: <http://gears.gent> (visited on 2017-03-15).
- [6] N.N. (2016). Showtime16, [Online]. Available: <http://showtime.showpad.com> (visited on 2017-03-13).
- [9] N.N. (2017). Websockets, [Online]. Available: <https://en.wikipedia.org/wiki/WebSocket> (visited on 2017-03-18).
- [10] N.N. (2017). Websockets, [Online]. Available: <http://caniuse.com/#feat=websockets> (visited on 2017-03-18).
- [13] N.N. (2017). Webrtc peer-to-peer connections, [Online]. Available: <http://caniuse.com/#feat=rtcpeerconnection> (visited on 2017-03-20).
- [14] N.N. (2017). Webrtc: The future of web games, [Online]. Available: <https://news.ycombinator.com/item?id=13266692> (visited on 2017-03-18).
- [21] N.N. (2017). Observer, [Online]. Available: [https://sourcemaking.com/design\\_patterns/observer](https://sourcemaking.com/design_patterns/observer) (visited on 2017-03-19).
- [22] A. Susiripala. (2015). Top es2015 features in 15 minutes, [Online]. Available: <https://kadirai.io/blog/other/top-es2015-features-in-15-minutes> (visited on 2017-03-10).
- [23] B. McCormick. (2015). Es6 patterns: Converting callbacks to promises, [Online]. Available: <https://benmccormick.org/2015/12/30/es6-patterns-converting-callbacks-to-promises/> (visited on 2017-04-02).
- [24] N.N. (2017). Async functions, [Online]. Available: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async\\_function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function) (visited on 2017-04-13).
- [25] A. Agrawal. (2017). What promises do that observables can't, [Online]. Available: <http://moduscreate.com/observables-and-promises> (visited on 2017-04-15).
- [27] N.N. (2017). Observable proposal, [Online]. Available: <https://github.com/tc39/proposal-observable> (visited on 2017-03-19).

## Bibliography

- [7] P. Lubbers. (2014). Html5 websocket: A quantum leap in scalability for the web, [Online]. Available: <https://websocket.org/quantum.html> (visited on 2017-03-13).
- [8] J. Hanson. (2014). Http long polling, [Online]. Available: <https://www.pubnub.com/blog/2014-12-01-http-long-polling/> (visited on 2017-03-06).
- [11] S. Dutton. (2014). Getting started with web rtc, [Online]. Available: <https://www.html5rocks.com/en/tutorials/webrtc/basics> (visited on 2017-03-18).
- [12] D. Ristic. (2014). Web rtc data channels, [Online]. Available: <https://www.html5rocks.com/en/tutorials/webrtc/datachannels/> (visited on 2017-03-22).
- [15] N.N. (2017). Functional reactive programming, [Online]. Available: [https://en.wikipedia.org/wiki/Functional\\_reactive\\_programming](https://en.wikipedia.org/wiki/Functional_reactive_programming) (visited on 2017-03-03).
- [16] N.N. (2017). Functional reactive programming, [Online]. Available: [https://wiki.haskell.org/Functional\\_Reactive\\_Programming](https://wiki.haskell.org/Functional_Reactive_Programming) (visited on 2017-03-02).
- [17] M. Fogus, *Functional javascript*. O'Reilly Media, 2013, ISBN: 9781449360726.
- [18] N.N. (2016). Lambda calculus, [Online]. Available: [https://en.wikipedia.org/wiki/Lambda\\_calculus](https://en.wikipedia.org/wiki/Lambda_calculus) (visited on 2017-02-24).
- [19] M. R. Cook. (2016). A practical introduction to functional programming, [Online]. Available: <https://maryrosecook.com/blog/post/a-practical-introduction-to-functional-programming> (visited on 2017-02-24).
- [20] A. Staltz. (2016). The introduction to reactive programming you've been missing, [Online]. Available: <https://gist.github.com/staltz/868e7e9bc2a7b8c1f754> (visited on 2017-02-22).
- [26] B. Lesh. (2016). Netflix javascript talks - rxjs version 5, [Online]. Available: <https://www.youtube.com/watch?v=C0viCoUtwx4> (visited on 2017-04-03).
- [28] E. Czaplicki and S. Chong. (2013). Asynchronous functional reactive programming for guis, [Online]. Available: <http://people.seas.harvard.edu/~chong/pubs/pldi13-elm.pdf> (visited on 2017-04-03).