Web backend development in Rust

Atte Lautanala – Wolt

Atte Lautanala

- Software Engineer at Wolt
- Backend and some frontend web development
- Python, Rust, Javascript…
- CSE studies at Aalto University

```
commit 65a774786444065eaa59a727e9c8851ef2bb40d5
Author: Atte Lautanala <atte.lautanala@gmail.com>
Date:   Sat Oct 22 18:37:36 2016 +0300

    Implement response API Clone step 1

diff --git a/components/script/dom/response.rs b/components/script/dom/response.rs
index a122620441..7607cacc97 100644
--- a/components/script/dom/response.rs
+++ b/components/script/dom/response.rs
@@ -298,7 +298,9 @@ impl ResponseMethods for Response {
     // https://fetch.spec.whatwg.org/#dom-response-clone
     fn Clone(&self) -> Fallible<Root<Response>> {
         // Step 1
-        // TODO: This step relies on body and stream, which are still unimplemented.
+        if self.is_locked() || self.body_used.get() {
+            return Err(Error::Type("cannot clone a disturbed response".to_string()));
+        }

         // Step 2
         let new_response = Response::new(&self.global());
diff --git a/tests/wpt/metadata/fetch/api/response/response-clone.html.ini b/tests/wpt/metadata/f
etch/api/response/response-clone.html.ini
index ecd4656a2d..dc3c577e41 100644
--- a/tests/wpt/metadata/fetch/api/response/response-clone.html.ini
+++ b/tests/wpt/metadata/fetch/api/response/response-clone.html.ini
@@ -6,9 +6,6 @@
   [Check cloned response's body]
     expected: FAIL

-  [Cannot clone a disturbed response]
-    expected: FAIL
-
   [Cloned responses should provide the same data]
     expected: FAIL
```

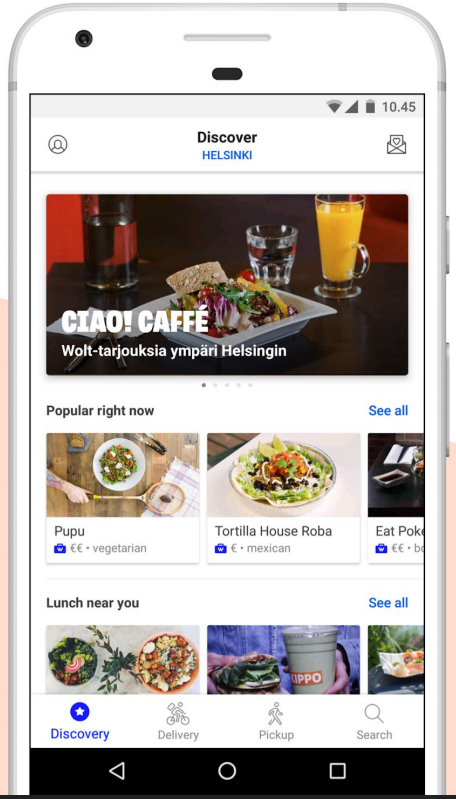
Rust at Wolt

The problem: images

- Discovery (left) uses slightly smaller and more compressed than the menu image (right)
- Two different versions of the images were stored to S3
- AWS Lambda, which was triggered by S3 file upload, resized and compressed the images with an external service
- The service had quotas and occasionally downtime which caused some images to be missing

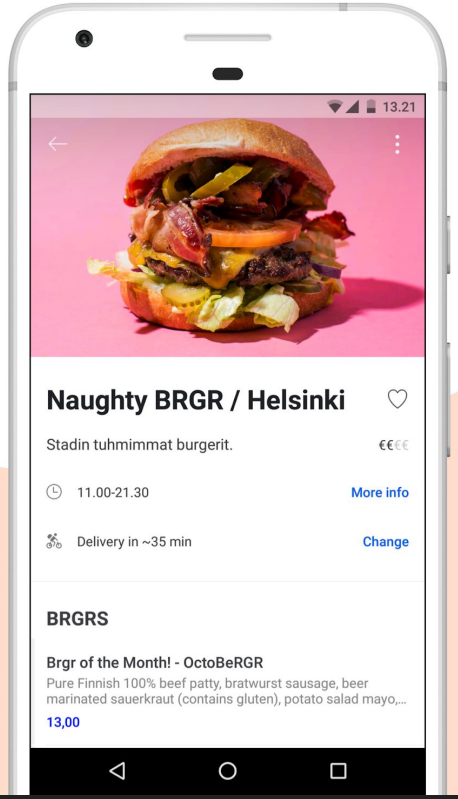
DISCOVER

First, choose your favourite restaurant from the list



CHOOSE

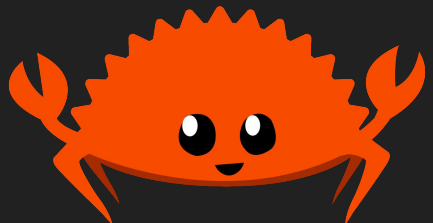
Build your order with a couple taps only



Why Rust?

Rust

- Safe system programming language with high-level ergonomics and low-level control
- Multiple web frameworks exist
- Foreign Function Interface
  - Ability to use C libraries
- Strong static typing enables *fearless* refactoring
- Fast release cycle
- Trying out new technology – Rust has not been previously used at Wolt
- Syntax feels familiar



```rust
fn foo(name: &str) -> u32 {
    let string = "Hello";

    if name.len() > 0 {
        println!("{}, {}", string, name);
    }

    42
}
```

```python
def foo(name):
    string = 'Hello'

    if len(name) > 0:
        print(f'{string}, {name}')

    return 42

```

```rust
fn foo(name: &str) -> u32 {
    let string = "Hello";

    if name.len() > 0 {
        println!("{}, {}", string, name);
    }

    42
}
```

```python
def foo(name: str) -> int:
    string = 'Hello'

    if len(name) > 0:
        print(f'{string}, {name}')

    return 42
```

```rust
fn foo(name: &str) -> u32 {
    let string = "Hello";

    if name.len() > 0 {
        println!("{}, {}", string, name);
    }

    42
}
```

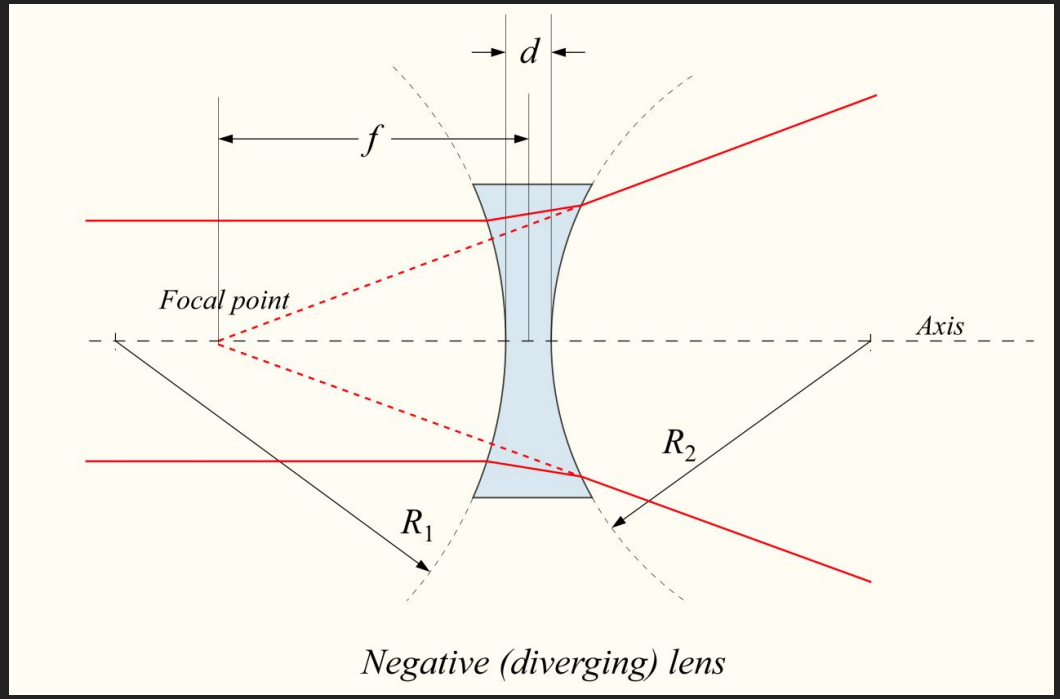
```c
uint32_t foo(const char* name) {
    const char* string = "Hello";

    if (strlen(name) > 0) {
        printf("%s, %s\n", string, name);
    }

    return 42;
}
```

Tooling

- Helpful compiler
- Great build and dependency management tool: cargo
- Editor support: Rust language server for editors with Language Server Protocol
  - This was stabilized last year
- Documentation and docs.rs

Lens

- A web server implemented in Rust
- Created in August 2017
- Resizes and compresses images
- Iron framework and other Rust crates
- Ability to render text and complete images was added later



AL

Web Frameworks

Iron Framework

- The first version of Lens was built with Iron Web Framework
- Built on hyper, a HTTP library for Rust
- API similar to Express.js

Iron Framework: a simple HTTP server

```rust
use iron::prelude::*;
use iron::status;

fn main() {
    let _server = Iron::new(hello).http("localhost:3000").unwrap();
}

fn hello(_request: &mut Request) -> IronResult<Response> {
    Ok(Response::with((status::Ok, "Hello")))
}
```

Iron Framework: a s[illegible]erver

```
1   use iron::prelude::*;
2   use iron::status;
3
4   fn main() {
5       let _server = Ir[illegible]llu[illegible]"host:3000").un
6   }
7
8   fn hello(_request: [illegible]st) -> Iro[illegible]nse> {
9       Ok(Response::wi[illegible]::Ok, "Hello [illegible]
10  }
11
```

Issues with Iron framework

- Major one: new connections can not be accepted until a thread is free to handle it
- Vulnerable to simple DOS attack: open maximum amount of connections to server and keep them alive
- Routing
- The project has been barely maintained since it was created in 2015

NOTE: Iron is not actively maintained at the moment, please consider using a different framework

Iron

build passing | crates.io v0.6.0 | license MIT

- This note was added in Feb. 2018 and removed two months later
- No releases since Nov. 2017 (0.6.0)
- Maintainers complain about source code being fragmented into multiple repositories

New web framework: Actix web

- First release in October 2017
- Actively maintained: so far on average multiple releases a month
- API similar to Iron with some improvements
- Version 1.0 in beta at the moment
  - Some parts of the framework have been rewritten completely
  - Another major refactoring ahead for the web service
- Includes an asynchronous HTTP client

Actix web: a simple HTTP server

```rust
use actix_web::{server, App, HttpRequest, Responder};

fn main() {
    server::new(|| App::new().resource("/", |r| r.with(hello)))
        .bind("localhost:3000")
        .unwrap()
        .run();
}

fn hello(_request: HttpRequest) -> impl Responder {
    "Hello"
}
```

Actix web: extractors

```rust
use actix_web::{Path, Responder};

fn index(info: Path<(String, u32)>) -> impl Responder {
    format!("Hello {}! id:{}", info.0, info.1)
}

```

Other interesting Rust web frameworks: Rocket

- Earlier versions required a compiler plugin, which would never work on stable
- Still requires a nightly build of Rust due to some experimental Rust features
- Provides a simple interface for application development

Are we *web* yet?

You can build stuff!

www.arewewebyet.org