KCDC2017 Demo Script

Make sure we have the latest versions of Rust. Run `rustup --show`.

Rustup is the version manager for Rust, which is officially supported by the Rust tools team. Rustup can install any Rust version, as well as all branches of it: stable, beta, and nightly.

Create the Rust package with Cargo, the build tool: `cargo new --bin cryptoserver`. This will create a new directory, a Git repo in it, and build a minimal package layout for an application binary in Rust.

With Cargo, you can make new packages, publish packages, install dependencies, format your code, and much more. Think of it as the equivalent of Ruby's Bundler for Rust.

Ensure we have the required packages. Add the following to Cargo.toml under dependencies:

```
rocket = "0.3.0"
rocket_codegen = "0.3.0"
rand = "0.3"
```

Install the packages with `cargo install`.

We'll open up the `src` directory and edit `main.rs`, which is where our web server will live. Let's use Atom with the supported Rust plugins.

Now, let's write some Rust code!

[See the code printouts for writing the Rust part]

main.rs

```rust
#![feature(plugin, custom_derive)]
#![plugin(rocket_codegen)]

extern crate rocket;
extern crate rand;

use rand::Rng;

const BASE62: &'static [u8] = b"0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";

#[derive(FromForm)]
struct IDSpec {
    size: usize
}

#[get("/")]
fn index() -> &'static str {
    "
        Welcome to CryptoServer, where you can generate random, secure, and unique IDs!

        USAGE:

            GET /generate/<size>

                returns a random, secure, and unique ID with the given size of characters generated on-the-fly
    "
}

fn get_secure_id(size: usize) -> String {
    let mut id = String::with_capacity(size);
    let mut rng = rand::thread_rng();
    for _ in 0..size {
        id.push(BASE62[rng.gen::<usize>() % 62] as char);
    }

    id
}

#[get("/generate?<idspec>")]
fn generate(idspec: IDSpec) -> String {
    get_secure_id(idspec.size)
}

fn main() {
    rocket::ignite().mount("/", routes![index, generate]).launch();
}
```

[Make sure to show the server generating the token with the browser.]

Let's load test this. Run the server with `cargo run --release`, which makes a release version with optimizations and smaller bundle size.

In our test.sh script, we're using the `wrk` tool to load test our app. It takes the amount of time to run with 400 HTTP connections and with 12 threads, as well as the size of token you want to generate.

If we run `test.sh 5s 128`, we'll see how many requests/s we get!

So, this is pretty fast for how easy it was to write this.

Let's compare this to a Python implementation of the same service, which would traditionally be much easier to write.

[Pull the Python source code up, go through it, explain it. Don't write it.]

Let's run the same load test!

If we run `test.sh 5s 128`, we'll see how many requests/s we get!

Nice! Rust is several orders of magnitude faster, for the same code, and it's totally memory safe. No more C/C++ bugs here.