}

Concurrent File-Based Key-Value Store in BUST

This project showcases a concurrent, thread-safe file-based key-value store written in the BUST programming language. It demonstrates modular architecture, mutex locking, file I/O operations, and performance benchmarking—crafted to impress recruiters at top tech companies.

Directory Structure

```
/bust-kvstore
  ├── main.bust
  ├── store.bust
  ├── utils.bust
  ├── bench.bust
  └── README.md
```

main.bust

```
import store from "./store.bust"
import utils from "./utils.bust"

fn main() -> void {
    print("□ Initializing BUST Key-Value Store...\n")
    let kv = store::new_store("./kv-data")
    kv.set("username", "Maria@TechGiant")
    kv.set("email", "maria@fakemail.com")
    kv.set("language", "BUST")

    let name = kv.get("username")
    print("□ Retrieved username: ", name)

    kv.delete("email")
    match kv.get("email") {
        Some(val) => print("□ Found: ", val),
        None => print("□  Email successfully deleted.")
    }

    bench::run(kv)
}
```

store.bust

```
module store

type Store = struct {
   path: string
   lock: Mutex
}

fn new_store(path: string) -> Store {
   utils::create_dir_if_missing(path)
   return Store { path: path, lock: Mutex::new() }
}

fn (s: Store) set(key: string, value: string) -> void {
   let filepath = s.path + "/" + key + ".kv"
   s.lock.lock()
   defer s.lock.unlock()

   let file = open(filepath, "w")
   file.write(value)
   file.close()
}

fn (s: Store) get(key: string) -> Option<string> {
   let filepath = s.path + "/" + key + ".kv"
   if !utils::file_exists(filepath) {
      return None
   }
   s.lock.lock()
   defer s.lock.unlock()

   let file = open(filepath, "r")
   let content = file.read_all()
   file.close()

   return Some(content)
}

fn (s: Store) delete(key: string) -> void {
   let filepath = s.path + "/" + key + ".kv"
```

```
    s.lock.lock()
    defer s.lock.unlock()

    if utils::file_exists(filepath) {
        utils::remove_file(filepath)
    }
}
```

utils.bust

```
module utils

fn create_dir_if_missing(path: string) -> void {
    if !dir_exists(path) {
        mkdir(path)
    }
}

fn file_exists(path: string) -> bool {
    try {
        let file = open(path, "r")
        file.close()
        return true
    } catch (FileNotFound) {
        return false
    }
}

fn remove_file(path: string) -> void {
    rm(path)
}
```

bench.bust

```
module bench
import time

fn run(store: Store) -> void {
    print("\n Starting Performance Benchmark...\n")

    let start = time::now()
    let iterations = 10000
    for i in 0..iterations {
        let key = "k" + str(i)
        let val = "v" + str(i)
        store.set(key, val)
    }

    for i in 0..iterations {
        let key = "k" + str(i)
        let val = store.get(key)
        assert(val.is_some())
    }

    let end = time::now()
    let duration = end - start
    print("□ Benchmark Completed: ", iterations, " ops in ", duration, " ms")
}
```