PROYECTO INTEGRADOR REDES-S.OPERATIVOS

AVANCE I

Ariel Arévalo Alvarado B50562
Antonio Badilla Olivas B80874
Geancarlo Rivera Hernández C06516
Jean Paul Chacón González C11993

Logger

```
high_resolution_cloc ::time_point Logger::start{
k high_resolution_cloc ::now()
};k
void Logger::print(const string &message) {
  cout << "[" << duration() << " ms]" << "[INFO]: "
<< message << endl;
}
void Logger::info(const string &message) {
  cout << "[" << duration() << " ms]" << "[INFO]: "
<< message << endl;
}
void Logger::error(const string &message) {
  cout << "[" << duration() << " ms]" << "[ERROR]: "
<< message << endl;
}
void Logger::error(const string &message, const
exception &e) {
  error(message);
  print_exception(e);
}
u_int64_t Logger::duration() {
  return duration_cast<milliseconds>(
high_resolution_cloc ::now() - start).count();
}
```

```
void Logger::print_exception(const exception &e, int
level) {
  cout << "Caused by:   << e.what() << endl;
  try {   "
    rethrow_if_nested(e);
  }
  catch (const exception &ne) {
    print_exception(ne, level + 1);
  }
}
```

Socket

```
string Ipv4SslSocket::sslRead() {
  string output{};
  if (isReadReady()) {
    char buf[CHUNK_SIZE]{};
    int bytesRead;
    do {
      bytesRead = SSL_read(static_cast<SSL *>(this->ssl),
        static_cast<void *>(buf), CHUNK_SIZE);
      if (0 > bytesRead) {
        int sslError{SSL_get_error(static_cast<SSL *>(this->ssl), bytesRead)};
        if (sslError == SSL_ERROR_WANT_READ || sslError == SSL_ERROR_WANT_WRITE) {
          continue;
        } else {
          throw runtime_error(
          appendSslErr("Ipv4SslSocket::sslRead: Failed to read from socket: "));
        }
      }
      output.append(buf, bytesRead);
    } while (bytesRead > 0);
  }
  return output;
}
```

Socket

```
void Ipv4SslSocket::sslWrite(const string &text) const {
  int st{SSL_write(static_cast<SSL *>(this->ssl),
    static_cast<const void *>(text.c_str()),
    static_cast<int>(text.size()))};
  if (0 >= st) {
    int sslError{SSL_get_error(static_cast<SSL *>(this->ssl), st)};
    if (sslError == SSL_ERROR_WANT_REA  || sslError == SSL_ERROR_WANT_WRITE) {
      sslWrite(text);
    } else {
      throw runtime_error(appendSslErr("Ipv4SslSocket::sslWrite: Failed to write to socket:  ));
    }                                    "
  }
}
```

HttpClient

```
static constexpr char GET[]{"GET "};
static constexpr char CRLF[]{"\r\n"};
static constexpr char HOST[]{"Host: "};
static constexpr char HTTPS[]{"https"};
```

```
string HttpsClient::get(const std::string &host, const std::string &resource) const {
  try {
    Ipv4SslSocket socket{};
    socket.sslConnect(host, HTTPS);
    const string request{string{GET} + resource + CRLF + HOST + host + CRLF + CRLF};
    socket.sslWrite(request);
    string response{socket.sslRead()};
    return response;
  } catch (exception const &e) {
    throw_with_nested(runtime_error("Failed to GET from:   + resource));
  }                                  "
}
```

FigureRepository

```
//hpp
 public:
  [[nodiscard]] Figure findByName(const std::string&
name) const;
 private:
  static constexpr char HOST[]{"os.ecci.ucr.ac.c  };
  static constexpr char URL_TEMPLATE[]{
"/lego/list.php?figure  };
="
  HttpsClient httpsClient{};
```

```
// cpp
Figure FigureRepository::findByName(const string& name
) const {
    const string url{URL_TEMPLATE + name};
    const string html{httpsClient.get(HOST, url)};

    return Figure::fromHtml(html);
}
```

Figure

```
public:
  static Figure fromHtml(const std::string &html);
  friend std::ostream &operator<<(std::ostream &os, const Figure
 &figure);
  explicit operator std::string() const;
  const std::string name;
  const std::vector<Row> parts;
private:
  Figure(std::string name, const std::vector<Row> &parts)
    : name(std::move(name)), parts(parts) {}
};
```

Figure

```
Figure Figure::fromHtml(const string &html) {
  string::const_iterator searchStart(html.cbegin());
  string::const_iterator searchEnd(html.cend());
  smatch matchName;
  regex regName{"/lego/(?:[a-zA-Z]+)/([a-zA-Z]+)(?=.jpg\" width=500 height=500)"};
  string name;
  if (regex_search(searchStart, searchEnd, matchName, regName)) {
    name = matchName[1];
  }
  smatch matchParts;
  regex regParts{"(?:brick|plate|flag) (?:[0-9]x[0-9] )?(?:[a-z ]+)"};
  smatch matchAmount;
  regex regAmount{"[0-9]+(?=</TD>)"};
  vector<Row> inputParts;
  while (regex_search(searchStart, searchEnd, matchParts, regParts) &&
         regex_search(searchStart, searchEnd, matchAmount, regAmount)) {
    inputParts.emplace_back(matchParts[0], stoi(matchAmount[0]));
    searchStart = matchParts.suffix().first;
  }
  return {name, inputParts};
}
```

FigureController

```
void FigureController::printFigureByName(const string
&name) const {
  Figure figure{figureRepository.findByName(name)};

  if (figure.name.empty() || figure.parts.empty()) {
    Logger::error("Figure is empty, cannot prin   );
  } else {        t."
    Logger::info(string(figure));
  }
}
```

LegoClient

```
int main(int argc, char *argv[]) {
  Logger::initialize();
  try {
    if (argc < 2) {
      Logger::error("Error: No figure name provide   );
      exit(1);      d."
    }
    string figureName{argv[1]};
    FigureController().printFigureByName(figureName);
  }
  catch (exception const &e) {
    Logger::error("Client has crashe   , e);
    exit(1);      d."
  }
  Logger::info("Finished.");
  exit(0);
}
```