step-1

What is Vert.x?

Eclipse Vert.x is a toolkit for building reactive applications on the JVM.

Core Vert.x concepts

There are 2 key concepts to learn in Vert.x:

1. what a *verticle* is, and
2. how the *event bus* allows verticles to communicate.

A minimally viable wiki written with Vert.x

Bootstrapping a Maven project

```
git clone https://github.com/vert-x3/vertx-maven-starter.git vertx-wiki
cd vertx-wiki
rm -rf .git
git init
```

```
mvn package exec:java
```

the Fabric8 project hosts a Vert.x Maven plugin. It has goals to initialize, build, package and run a Vert.x project.

```
mkdir vertx-wiki
cd vertx-wiki
mvn io.fabric8:vertx-maven-plugin:1.0.13:setup -DvertxVersion=3.8.0
git init
```

Adding the required dependencies

```
<dependency>
  <groupId>io.vertx</groupId>
  <artifactId>vertx-web</artifactId>
</dependency>
<dependency>
  <groupId>io.vertx</groupId>
  <artifactId>vertx-web-templ-freemarker</artifactId>
</dependency>
<dependency>
  <groupId>com.github.rjeschke</groupId>
  <artifactId>txtmark</artifactId>
  <version>0.13</version>
</dependency>
```

```
<dependency>
  <groupId>io.vertx</groupId>
  <artifactId>vertx-jdbc-client</artifactId>
</dependency>
<dependency>
  <groupId>org.hsqldb</groupId>
  <artifactId>hsqldb</artifactId>
  <version>2.3.4</version>
</dependency>
```

Anatomy of a verticle

```
public class MainVerticle extends AbstractVerticle {

  @Override
  public void start(Promise<Void> promise) {
    promise.complete();
  }
}
```

---

callback hell

```
foo.a(1, res1 -> {
  if (res1.succeeded()) {
    bar.b("abc", 1, res2 -> {
      if (res.succeeded()) {
         baz.c(res3 -> {
           dosomething(res1, res2, res3, res4 -> {
               // (...)
           });
         });
      }
    });
```

```
  }
});
```

Wiki verticle initialization phases

To get our wiki running, we need to perform a 2-phases initialization:

1. we need to establish a JDBC database connection, and also make sure that the database schema is in place, and
2. we need to start a HTTP server for the web application.

```
private Future<Void> prepareDatabase() {
  Promise<void> promise = Promise.promise();
  // (...)
  return promise.future();
}

private Future<Void> startHttpServer() {
  Promise<void> promise = Promise.promise();
  // (...)
  return promise.future();
}
```

```
@Override
public void start(Promise<Void> promise) throws Exception {
  Future<Void> steps = prepareDatabase().compose(v -> startHttpServer());
  steps.setHandler(ar -> {
    if (ar.succeeded()) {
      promise.complete();
    } else {
      promise.fail(ar.cause());
    }
  });
}
```

Database initialization

The wiki database schema consists of a single table `Pages` with the following columns:

Column	Type	Description
`Id`	Integer	Primary key
`Name`	Characters	Name of a wiki page, must be unique

Column	Type	Description
Content	Text	Markdown text of a wiki page

```
private static final String SQL_CREATE_PAGES_TABLE = "create table if not exists Pages (Id integer identity primary key, Name v
private static final String SQL_GET_PAGE = "select Id, Content from Pages where Name = ?"; (1)
private static final String SQL_CREATE_PAGE = "insert into Pages values (NULL, ?, ?)";
private static final String SQL_SAVE_PAGE = "update Pages set Content = ? where Id = ?";
private static final String SQL_ALL_PAGES = "select Name from Pages";
private static final String SQL_DELETE_PAGE = "delete from Pages where Id = ?";
```

```
<dependency>
    <groupId>com.guicedee.services</groupId>
    <artifactId>sl4j</artifactId>
    <version>1.0.13.5</version>
</dependency>
```

```
private JDBCClient dbClient;

private static final Logger LOGGER = LoggerFactory.getLogger(MainVerticle.class);
```

```
private Future<Void> prepareDatabase() {
  Promise<Void> promise = Promise.promise();

  dbClient = JDBCClient.createShared(vertx, new JsonObject()
    .put("url", "jdbc:hsqldb:file:db/wiki")
    .put("driver_class", "org.hsqldb.jdbcDriver")
    .put("max_pool_size", 30));

  dbClient.getConnection(ar -> {
    if (ar.failed()) {
      LOGGER.error("Could not open a database connection", ar.cause());
      promise.fail(ar.cause());
    } else {
      SQLConnection connection = ar.result();
      connection.execute(SQL_CREATE_PAGES_TABLE, create -> {
        connection.close();
        if (create.failed()) {
          LOGGER.error("Database preparation error", create.cause());
          promise.fail(create.cause());
        } else {
          promise.complete();
        }
      });
    }
  });

  return promise.future();
}
```

TIP

> The SQL database modules supported by the Vert.x project do not currently offer anything beyond passing SQL queries (e.g., an object-relational mapper) as they focus on providing asynchronous access to databases. However, nothing forbids using more advanced modules from the community, and we especially recommend checking out projects like this jOOq generator for Vert.x or this POJO mapper.

```
<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>1.2.3</version>
</dependency>
```

src/main/resources/logback.xml

```
<configuration>

  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <encoder>
      <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
    </encoder>
  </appender>

  <logger name="com.mchange.v2" level="warn"/>
  <logger name="io.netty" level="warn"/>
  <logger name="io.vertx" level="info"/>
  <logger name="io.vertx.guides.wiki" level="debug"/>

  <root level="debug">
    <appender-ref ref="STDOUT"/>
  </root>

</configuration>
```

HTTP server initialization

```
private FreeMarkerTemplateEngine templateEngine;

private Future<Void> startHttpServer() {
  Promise<Void> promise = Promise.promise();
  HttpServer server = vertx.createHttpServer();

  Router router = Router.router(vertx);
  router.get("/").handler(this::indexHandler);
  router.get("/wiki/:page").handler(this::pageRenderingHandler);
  router.post().handler(BodyHandler.create());
  router.post("/save").handler(this::pageUpdateHandler);
  router.post("/create").handler(this::pageCreateHandler);
  router.post("/delete").handler(this::pageDeletionHandler);

  templateEngine = FreeMarkerTemplateEngine.create(vertx);

  server
    .requestHandler(router)
    .listen(8080, ar -> {
      if (ar.succeeded()) {
        LOGGER.info("HTTP server running on port 8080");
```

```
        promise.complete();
      } else {
        LOGGER.error("Could not start a HTTP server", ar.cause());
        promise.fail(ar.cause());
      }
    });

  return promise.future();
}
```

HTTP router handlers

indexHandler

```
private void indexHandler(RoutingContext context) {
  dbClient.getConnection(car -> {
    if (car.succeeded()) {
      SQLConnection connection = car.result();
      connection.query(SQL_ALL_PAGES, res -> {
        connection.close();

        if (res.succeeded()) {
          List<String> pages = res.result()
            .getResults()
            .stream()
            .map(json -> json.getString(0))
            .sorted()
            .collect(Collectors.toList());

          context.put("title", "Wiki home");
          context.put("pages", pages);
          templateEngine.render(context.data(), "templates/index.ftl", ar -> {
            if (ar.succeeded()) {
              context.response().putHeader("Content-Type", "text/html");
              context.response().end(ar.result());
            } else {
              context.fail(ar.cause());
            }
          });

        } else {
          context.fail(res.cause());
        }
      });
    } else {
      context.fail(car.cause());
    }
  });
}
```

src/main/resources/templates/index.ftl

```
<#include "header.ftl">

<div class="row">

  <div class="col-md-12 mt-1">
    <div class="float-right">
      <form class="form-inline" action="/create" method="post">
        <div class="form-group">
          <input type="text" class="form-control" id="name" name="name" placeholder="New page name">
        </div>
        <button type="submit" class="btn btn-primary">Create</button>
      </form>
    </div>
    <h1 class="display-4">${title}</h1>
  </div>

  <div class="col-md-12 mt-1">
```

```
  <#list pages>
    <h2>Pages:</h2>
    <ul>
      <#items as page>
        <li><a href="/wiki/${page}">${page}</a></li>
      </#items>
    </ul>
  <#else>
    <p>The wiki is currently empty!</p>
  </#list>
  </div>

</div>

<#include "footer.ftl">
```

src/main/resources/templates/header.ftl

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <meta http-equiv="x-ua-compatible" content="ie=edge">
  <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css"
        rel="stylesheet"
        integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO"
        crossorigin="anonymous">
  <title>${title} | A Sample Vert.x-powered Wiki</title>
</head>
<body>

<div class="container">
```

src/main/resources/templates/footer.ftl

```
</div> <!-- .container -->

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"
        integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"
        crossorigin="anonymous"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js"
        integrity="sha384-ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAdn689aCwoqbBJiSnjAK/l8WvCWPIPm49"
        crossorigin="anonymous"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js"
        integrity="sha384-ChfqqxuZUCnJSK3+MXmPNIyE6ZbWh2IMqE241rYiqJxyMiZ6OW/JmZQ5stwEULTy"
        crossorigin="anonymous"></script>

</body>
</html>
```

---

pageRenderingHandler

```
private static final String EMPTY_PAGE_MARKDOWN =
  "# A new page\n" +
    "\n" +
    "Feel-free to write in Markdown!\n";

private void pageRenderingHandler(RoutingContext context) {
  String page = context.request().getParam("page");

  dbClient.getConnection(car -> {
    if (car.succeeded()) {

      SQLConnection connection = car.result();
      connection.queryWithParams(SQL_GET_PAGE, new JsonArray().add(page), fetch -> {
```

```
        connection.close();
        if (fetch.succeeded()) {

          JsonArray row = fetch.result().getResults()
            .stream()
            .findFirst()
            .orElseGet(() -> new JsonArray().add(-1).add(EMPTY_PAGE_MARKDOWN));
          Integer id = row.getInteger(0);
          String rawContent = row.getString(1);

          context.put("title", page);
          context.put("id", id);
          context.put("newPage", fetch.result().getResults().size() == 0 ? "yes" : "no");
          context.put("rawContent", rawContent);
          context.put("content", Processor.process(rawContent));
          context.put("timestamp", new Date().toString());

          templateEngine.render(context.data(), "templates/page.ftl", ar -> {
            if (ar.succeeded()) {
              context.response().putHeader("Content-Type", "text/html");
              context.response().end(ar.result());
            } else {
              context.fail(ar.cause());
            }
          });
        } else {
          context.fail(fetch.cause());
        }
      });

    } else {
      context.fail(car.cause());
    }
  });
}
```

page.ftl

```
<#include "header.ftl">

<div class="row">

  <div class="col-md-12 mt-1">
      <span class="float-right">
        <a class="btn btn-outline-primary" href="/" role="button" aria-pressed="true">Home</a>
        <button class="btn btn-outline-warning" type="button" data-toggle="collapse"
                data-target="#editor" aria-expanded="false" aria-controls="editor">Edit</button>
      </span>
    <h1 class="display-4">
      <span class="text-muted">{</span>
    ${title}
      <span class="text-muted">}</span>
    </h1>
  </div>

  <div class="col-md-12 mt-1 clearfix">
  ${content}
  </div>

  <div class="col-md-12 collapsable collapse clearfix" id="editor">
    <form action="/save" method="post">
      <div class="form-group">
        <input type="hidden" name="id" value="${id}">
        <input type="hidden" name="title" value="${title}">
        <input type="hidden" name="newPage" value="${newPage}">
        <textarea class="form-control" id="markdown" name="markdown" rows="15">${rawContent}</textarea>
      </div>
      <button type="submit" class="btn btn-primary">Save</button>
    <#if id != -1>
      <button type="submit" formaction="/delete" class="btn btn-danger float-right">Delete</button>
    </#if>
    </form>
  </div>

  <div class="col-md-12 mt-1">
    <hr class="mt-1">
    <p class="small">Rendered: ${timestamp}</p>
  </div>

</div>

<#include "footer.ftl">
```

pageCreateHandler

```
private void pageCreateHandler(RoutingContext context) {
  String pageName = context.request().getParam("name");
  String location = "/wiki/" + pageName;
  if (pageName == null || pageName.isEmpty()) {
    location = "/";
  }
  context.response().setStatusCode(303);
  context.response().putHeader("Location", location);
  context.response().end();
}
```

pageUpdateHandler

```
private void pageUpdateHandler(RoutingContext context) {
  String id = context.request().getParam("id");   (1)
  String title = context.request().getParam("title");
  String markdown = context.request().getParam("markdown");
  boolean newPage = "yes".equals(context.request().getParam("newPage"));  (2)

  dbClient.getConnection(car -> {
    if (car.succeeded()) {
      SQLConnection connection = car.result();
      String sql = newPage ? SQL_CREATE_PAGE : SQL_SAVE_PAGE;
      JsonArray params = new JsonArray();   (3)
      if (newPage) {
        params.add(title).add(markdown);
      } else {
        params.add(markdown).add(id);
      }
      connection.updateWithParams(sql, params, res -> {   (4)
        connection.close();
        if (res.succeeded()) {
          context.response().setStatusCode(303);    (5)
          context.response().putHeader("Location", "/wiki/" + title);
          context.response().end();
        } else {
          context.fail(res.cause());
        }
      });
    } else {
      context.fail(car.cause());
    }
  });
}
```

pageDeletionHandler

```
private void pageDeletionHandler(RoutingContext context) {
  String id = context.request().getParam("id");
  dbClient.getConnection(car -> {
    if (car.succeeded()) {
      SQLConnection connection = car.result();
      connection.updateWithParams(SQL_DELETE_PAGE, new JsonArray().add(id), res -> {
        connection.close();
        if (res.succeeded()) {
          context.response().setStatusCode(303);
          context.response().putHeader("Location", "/");
          context.response().end();
        } else {
          context.fail(res.cause());
        }
      });
    } else {
      context.fail(car.cause());
    }
  });
}
```

Running the application

```
mvn clean package
```

```
java -jar target/wiki-step-1-1.3.0-SNAPSHOT-fat.jar
```