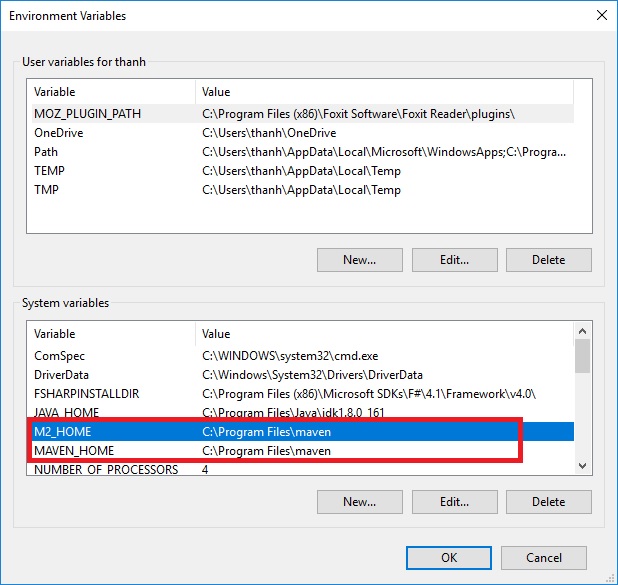
1. **Cài đặt Apache Maven trên Windows**

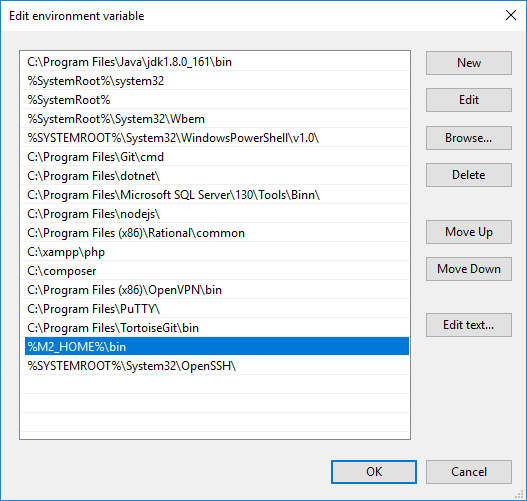
**Quy trình cài đặt Apache Maven:**

* Đã cài đặt JDK và thiết lập JAVA\_HOME
* Tải và giải nén Apache Maven tại link sau: <http://maven.apache.org/download.cgi>
* Lưu trữ thư mục giải nén.
* Thiết lập môi trường M2\_HOME và MAVEN\_HOME như sau:



***Lưu ý***: Đường dẫn trên dẫn đến nơi thư mục giải nén của Apache Maven.

* Thêm vào Path thư mục bin của Maven như sau:

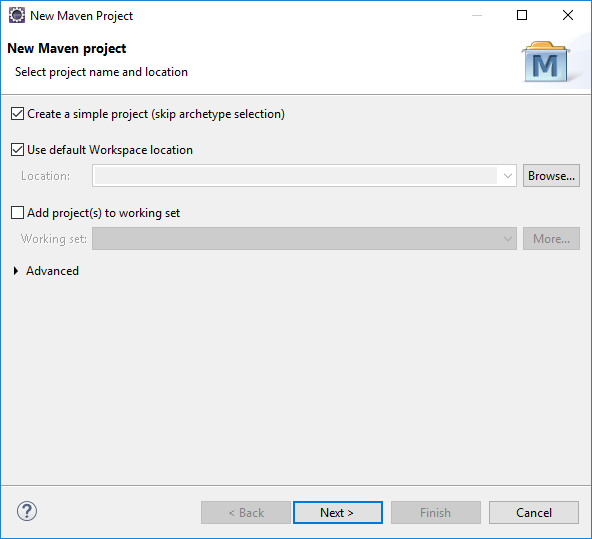


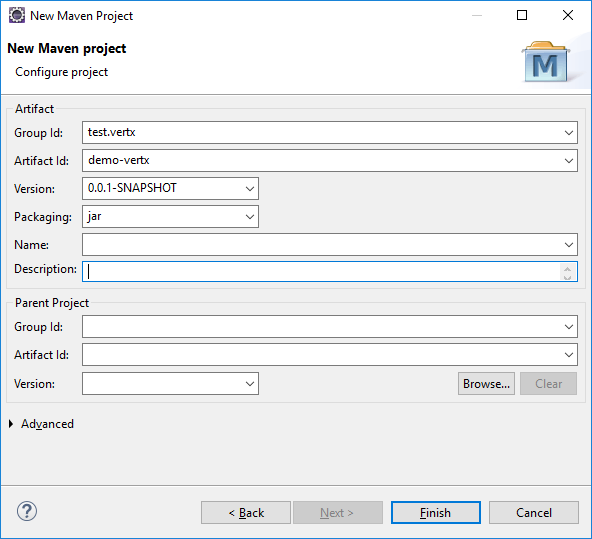
* Hoàn thành việc cài đặt Apache Maven trên máy, tiếp theo gõ lệnh **mvn –version** trên cmd để xem phiên bản của Apache Maven.

**Link tham khảo**: <https://www.mkyong.com/maven/how-to-install-maven-in-windows/>

1. **Khởi tạo project với Maven**

Đa số những phiên bản mới của Eclipse đều tích hợp sẵn việc khởi tạo project Maven.

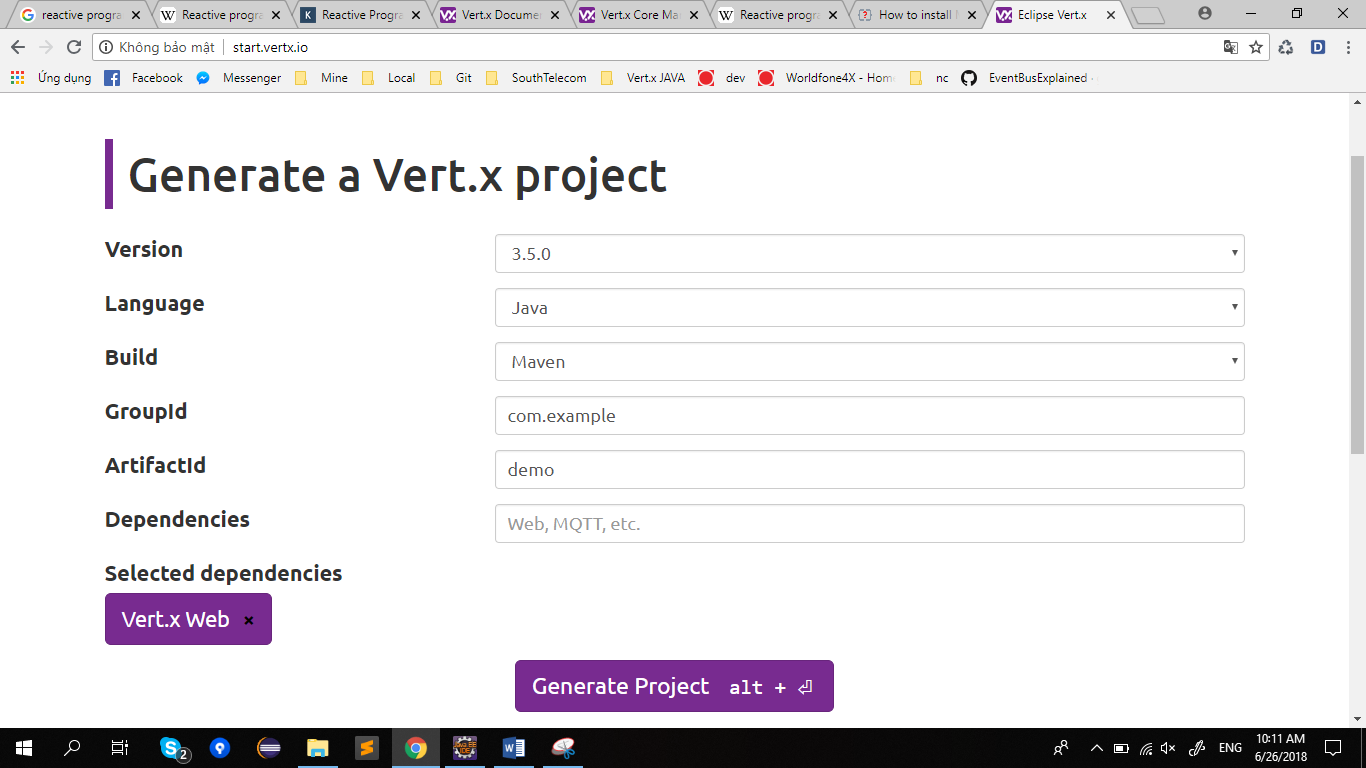




Khi click Finish thì ta đã khởi tạo thành công project Maven. Tiếp theo ta cần thiết lập project cũng như cài đặt những gói thư viện.

File **pom.xml** sẽ giúp chúng ta việc này. Chúng ta có thể cấu hình file pom.xml để tải các gói thư viện cho vert.x.

Tuy nhiên trên trang chủ của Vert.x có tích hợp sẵn việc download project Vert.x, cũng như tích hợp sẵn những gói cho Vert.x. Truy cập đường dẫn <http://start.vertx.io/> để tải project Vert.x

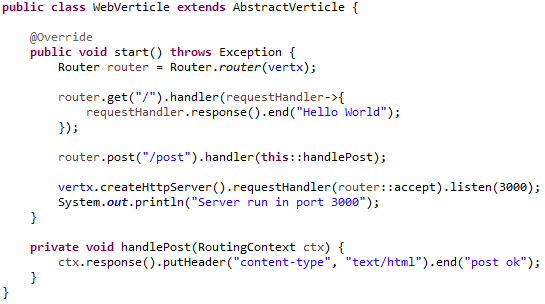


Sau khi tải thành công ta mở bằng IDE có hỗ trợ Maven ta sẽ được một project Vert.x.

1. **Các tính năng của Vert.x**

* **WEB**

1. **Web**



Cần cấu hình file **pom.xml** thêm một dependency **vertx-web.**

1. **Web Client**



Cần cấu hình file **pom.xml** thêm một dependency **vertx-web-client.**

Giúp dễ dàng sử dụng Http request response.

* **DATA ACCESS**

1. **MongoDB**



Cần cấu hình file **pom.xml** thêm một dependency **vertx-mongo-client.**

Kết nối đến cơ sở dữ liệu MongoDB

* **MQTT**

**MQTT = Message Queue Telemetry Transport**

Đây là một giao thức truyền thông điệp (message) theo mô hình publish/subscribe (xuất bản – theo dõi), sử dụng băng thông thấp, độ tin cậy cao và có khả năng hoạt động trong điều kiện đường truyền không ổn định.

Kiến trúc mức cao (high-level) của MQTT gồm 2 phần chính là Broker và Clients.

**Tài liệu tham khảo**: <https://techmaster.vn/posts/34394/iot-giao-thuc-mqtt-va-ung-dung-trong-iot> ; <https://devlog.vn/5-dieu-can-biet-ve-mqtt-de-xay-dung-ung-dung-iot/>

Code mẫu giao thức MQTT trên Vert.x

Cần cấu hình file **pom.xml** thêm một dependency **vertx-mqtt.**

* **Server**

**public** **class** ServerMqtt **extends** AbstractVerticle {

@Override

**public** **void** start() **throws** Exception {

MqttServer mqttServer = MqttServer.*create*(vertx);

mqttServer.endpointHandler(endpoint -> {

// shows main connect info

System.***out***.println("MQTT client [" + endpoint.clientIdentifier() + "] request to connect, clean session = "

+ endpoint.isCleanSession());

System.***out***.println("[keep alive timeout = " + endpoint.keepAliveTimeSeconds() + "]");

// accept connection from the remote client

endpoint.accept(**false**);

// handling requests for subscriptions

endpoint.subscribeHandler(subscribe -> {

List<MqttQoS> grantedQosLevels = **new** ArrayList<>();

**for** (MqttTopicSubscription s : subscribe.topicSubscriptions()) {

System.***out***.println("Subscription for " + s.topicName() + " with QoS " + s.qualityOfService());

grantedQosLevels.add(s.qualityOfService());

}

// ack the subscriptions request

endpoint.subscribeAcknowledge(subscribe.messageId(), grantedQosLevels);

// just as example, publish a message on the first topic with requested QoS

endpoint.publish(subscribe.topicSubscriptions().get(0).topicName(),

Buffer.*buffer*("Hello from the Vert.x MQTT server"),

subscribe.topicSubscriptions().get(0).qualityOfService(), **false**, **false**);

});

// handling requests for unsubscriptions

endpoint.unsubscribeHandler(unsubscribe -> {

**for** (String t : unsubscribe.topics()) {

System.***out***.println("Unsubscription for " + t);

}

endpoint.unsubscribeAcknowledge(unsubscribe.messageId());

});

// handling ping from client

endpoint.pingHandler(v -> {

System.***out***.println("Ping received from client");

});

// handling disconnect message

endpoint.disconnectHandler(v -> {

System.***out***.println("Received disconnect from client");

});

// handling closing connection

endpoint.closeHandler(v -> {

System.***out***.println("Connection closed");

});

}).listen(1883, "0.0.0.0", ar -> {

**if** (ar.succeeded()) {

System.***out***.println("MQTT server is listening on port " + mqttServer.actualPort());

} **else** {

System.***err***.println("Error on starting the server" + ar.cause().getMessage());

}

});

}

}

* **Client**

**public** **class** ClientMqtt **extends** AbstractVerticle {

**private** **static** **final** String ***MQTT\_TOPIC*** = "/my\_topic";

**private** **static** **final** String ***MQTT\_MESSAGE*** = "Hello Vert.x MQTT Client";

**private** **static** **final** String ***BROKER\_HOST*** = "localhost";

**private** **static** **final** **int** ***BROKER\_PORT*** = 1883;

@Override

**public** **void** start() **throws** Exception {

MqttClientOptions options = **new** MqttClientOptions().setKeepAliveTimeSeconds(2);

MqttClient client = MqttClient.*create*(Vertx.*vertx*(), options);

// handler will be called when we have a message in topic we subscribing for

client.publishHandler(publish -> {

System.***out***.println("Just received message on [" + publish.topicName() + "] payload ["

+ publish.payload().toString(Charset.*defaultCharset*()) + "] with QoS [" + publish.qosLevel() + "]");

});

// handle response on subscribe request

client.subscribeCompletionHandler(h -> {

System.***out***.println("Receive SUBACK from server with granted QoS : " + h.grantedQoSLevels());

// let's publish a message to the subscribed topic

client.publish(***MQTT\_TOPIC***, Buffer.*buffer*(***MQTT\_MESSAGE***), MqttQoS.***AT\_MOST\_ONCE***, **false**, **false**,

s -> System.***out***.println("Publish sent to a server"));

// unsubscribe from receiving messages for earlier subscribed topic

vertx.setTimer(6000, l -> client.unsubscribe(***MQTT\_TOPIC***));

});

// handle response on unsubscribe request

client.unsubscribeCompletionHandler(h -> {

System.***out***.println("Receive UNSUBACK from server");

vertx.setTimer(6000, l ->

// disconnect for server

client.disconnect(d -> System.***out***.println("Disconnected form server")));

});

// connect to a server

client.connect(***BROKER\_PORT***, ***BROKER\_HOST***, ch -> {

**if** (ch.succeeded()) {

System.***out***.println("Connected to a server");

client.subscribe(***MQTT\_TOPIC***, 0);

} **else** {

System.***out***.println("Failed to connect to a server");

System.***out***.println(ch.cause());

}

});

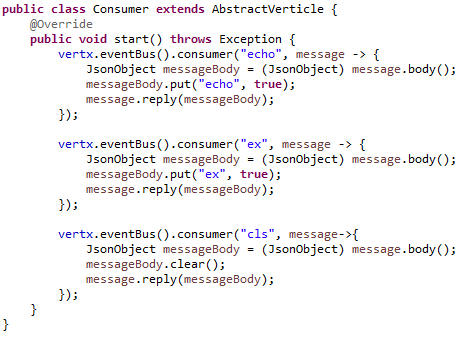
}

}

* **EVENTBUS**

Eventbus là cơ chế giúp các components khác nhau giao tiếp với nhau. Một component có thể publish một event mà không cần biết ai hay bao nhiêu component khác sử dụng nó.

* Class Consumer sẽ publish các event đến Eventbus cho các class khác gọi sử dụng



* Class Server gọi đến Eventbus sử dụng event

