1. mvn archetype:generate -DarchetypeArtifactId=maven-archetype-webapp -DgroupId=com.howtographql.sample -DartifactId=hackernews-graphql-java -Dversion=1.0-SNAPSHOT Command usage
2. resolver functions
3. [WARNING] \* org.apache.maven.plugins:maven-site-plugin:3.12.1
4. [WARNING] \* org.apache.maven.plugins:maven-archetype-plugin:3.2.1
5. Still, to make dynamic resolver wiring easy, you’ll also want to use graphql-java-tools, the library inspired by Apollo’s graphql-tools. Additionally, because the goal is to expose the API over the web, you’ll also make use of graphql-java-servlet (a simple helper library containing a ready-made servlet for accepting GraphQL queries) and javax.servlet-api (the servlet specification implementation).
6. what is jetty server
7. uses of the mentioned dependencies and plugins
8. strong typing and intuitive deisign
9. Serializable interface – this class can be written into things like database and files

import com.coxautodev.graphql.tools.SchemaParser;

import javax.servlet.annotation.WebServlet;

import graphql.servlet.SimpleGraphQLServlet;

@WebServlet(urlPatterns = "/graphql")

public class GraphQLEndpoint extends SimpleGraphQLServlet {

public GraphQLEndpoint() {

super(SchemaParser.newParser()

.file("schema.graphqls") //parse the schema file created earlier

.build()

.makeExecutableSchema());

}