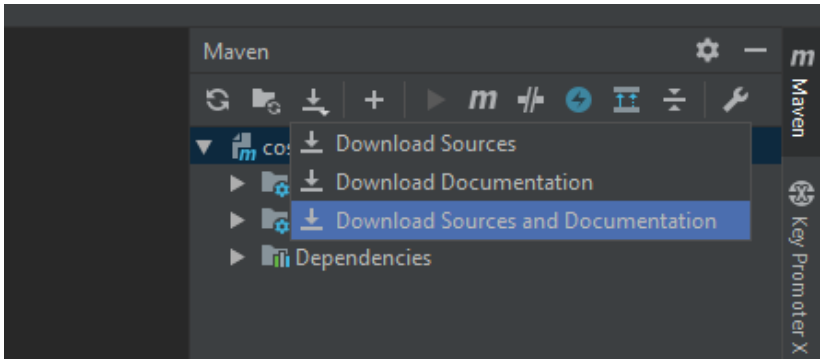
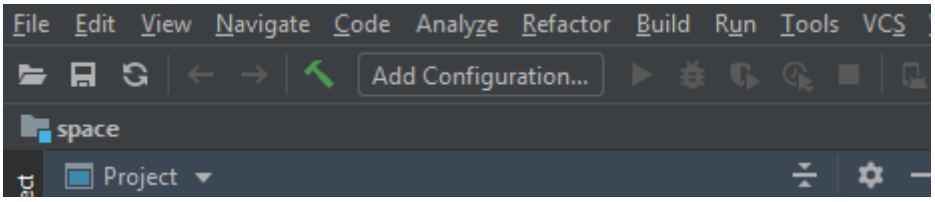
Как попасть на стажировку?

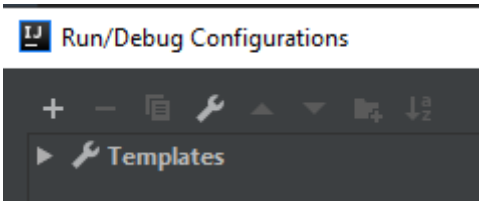
1. Скачать Intellij IDEA Ultimate Edition. Ее пробный период — 30 дней, которых хватит на решение тестового задания. А на стажировке ты получишь ключ для доступа на протяжении 6 месяцев.
2. Открыть проект, используя файл pom.xml. Так он сразу станет Maven-проектом. Maven встроен в Intellij IDEA Ultimate Edition, но также можно скачать и установить его отдельно.
3. Справа во вкладке Maven нажать на Download Sources and Documentation для загрузки исходного кода и документации используемых фреймворков.



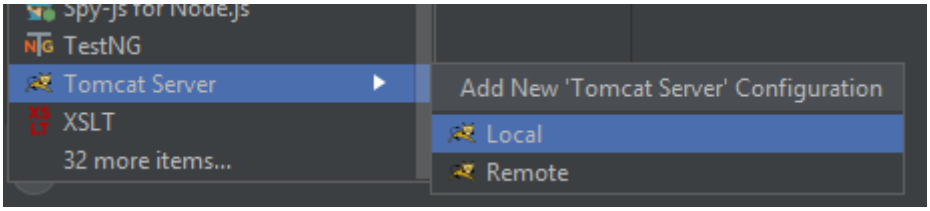
4. Установить MySQL сервер. Логин и пароль **root**. Порт **3306**. Залогиниться и выполнить скрипт init.sql, который ты найдешь в корне проекта.
5. Собрать проект (в терминале: mvn -DskipTests=true clean install)
   (чтобы Maven работал в терминале, он должен быть прописан в системной переменной PATH)
6. Настроить запуск приложения через Tomcat (если не установлен, то можно скачать отсюда):
   6.1 Выбрать Add Configuration... в параметрах запуска:



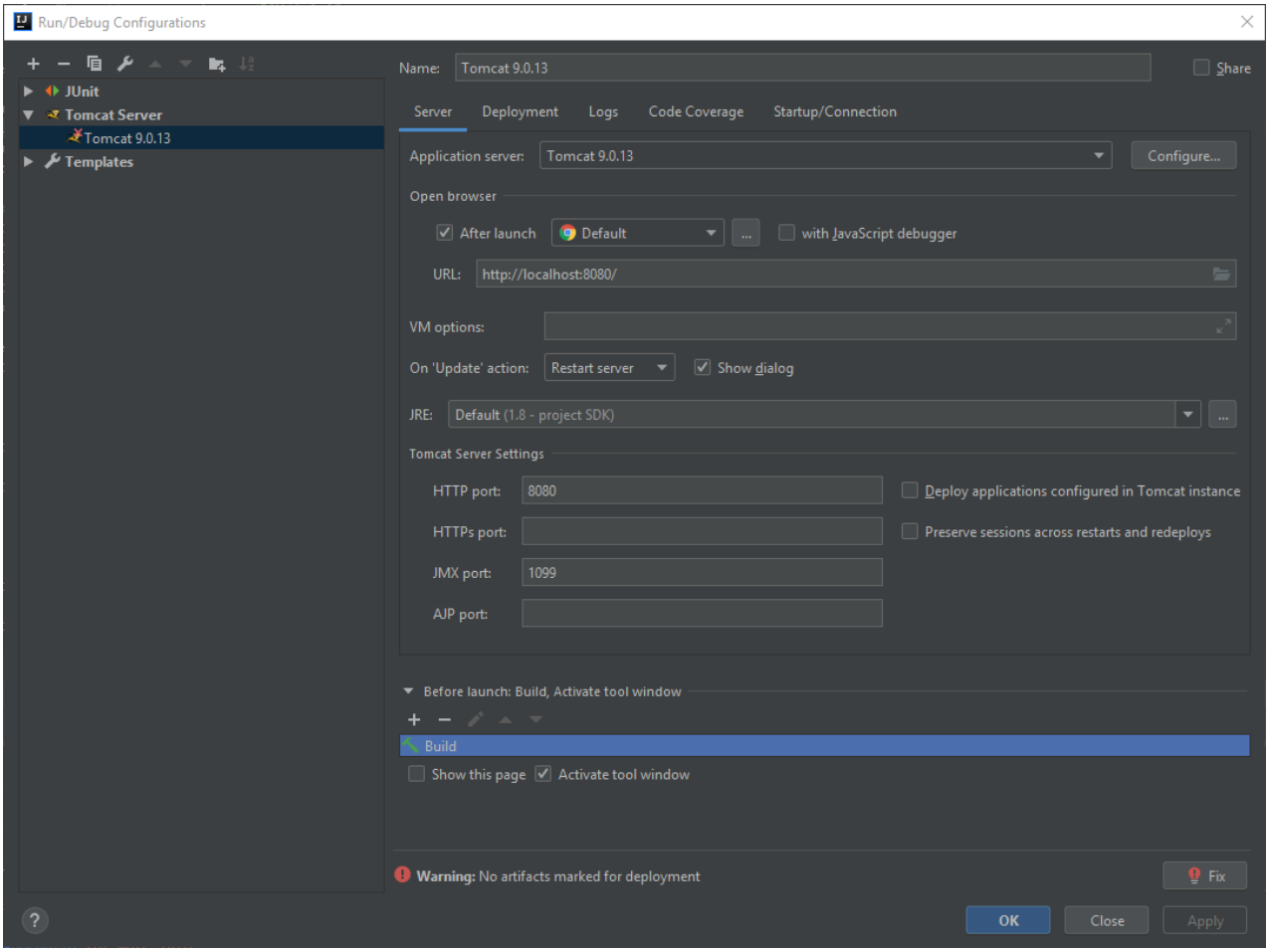
   6.2 Нажать на «плюсик» чтобы раскрыть список возможных конфигураций. Tomcat может находиться в самом низу. При необходимости нажми на «more items».



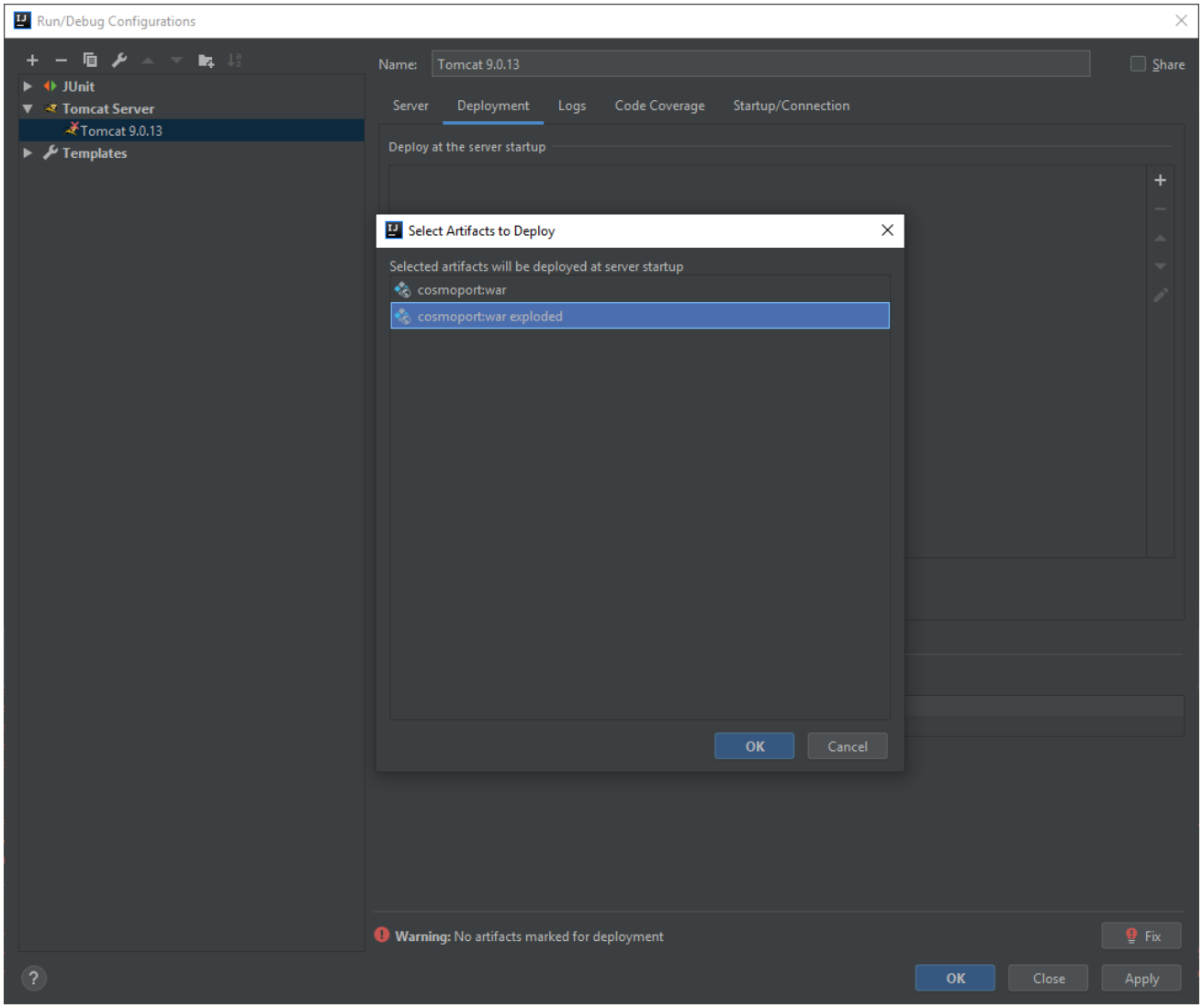
6.3 В выпадающем меню выбрать Local.



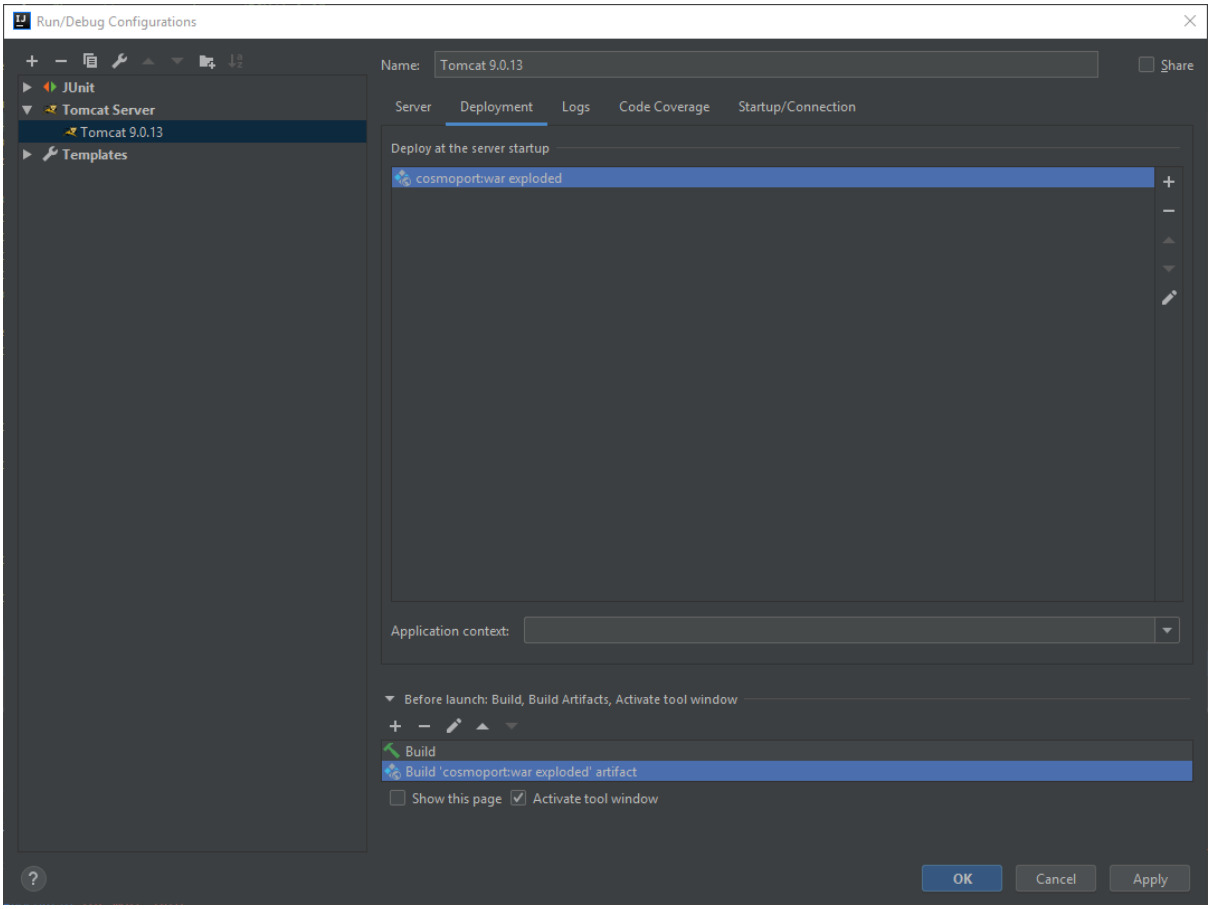
6.4 Во вкладке Server настроить все так, как показано на скриншоте (кроме версии Tomcat – используй последнюю):



6.5 Во вкладке Deployment нажать на «плюсик» и добавить артефакт «cosmoport:war exploded», как указано на скриншоте.



6.6 Очистить поле Application context и нажать Ок:



7. Запустить приложение. В браузере откроется стартовая страница, на которой ты увидишь готовый интерфейс приложения, но он не работает пока на сервере нет соответствующего функционала. По мере реализации проекта, интерфейс будет правильно отображать данные и отправлять запросы на редактирование, удаление и создание кораблей.
8. Дописать нужный функционал (см. Задание на стажировку). **ВАЖНО: Файл pom.xml изменять нельзя!**
9. Протестировать свое приложение с помощью тестов, которые уже есть в проекте.
10. Если **ВСЕ ТЕСТЫ ПРОШЛИ** успешно, то отправить задание на проверку в JavaRush.

Задание на стажировку.

Нужно дописать приложение, которое ведет учет космических кораблей в далеком будущем (в 3019 году). Должны быть реализованы следующие возможности:

1. получать список всех существующих кораблей;
2. создавать новый корабль;
3. редактировать характеристики существующего корабля;
4. удалять корабль;
5. получать корабль по id;
6. получать отфильтрованный список кораблей в соответствии с переданными фильтрами;
7. получать количество кораблей, которые соответствуют фильтрам.

Для этого необходимо реализовать REST API в соответствии с документацией.

В проекте должна использоваться сущность Ship, которая имеет поля:

Long id	ID корабля
String name	Название корабля (до 50 знаков включительно)
String planet	Планета пребывания (до 50 знаков включительно)
ShipType shipType	Тип корабля
Date prodDate	Дата выпуска. Диапазон значений года 2800..3019 включительно
Boolean isUsed	Использованный / новый
Double speed	Максимальная скорость корабля. Диапазон значений 0,01..0,99 включительно. Используй математическое округление до сотых.
Integer crewSize	Количество членов экипажа. Диапазон значений 1..9999 включительно.
Double rating	Рейтинг корабля. Используй математическое округление до сотых.

Также должна присутствовать бизнес-логика:
Перед сохранением корабля в базу данных (при добавлении нового или при апдейте характеристик существующего), должен высчитываться рейтинг корабля и сохраняться в БД. Рейтинг корабля рассчитывается по формуле:

$$R = \frac{80 \cdot v \cdot k}{y_0 - y_1 + 1},$$

где:
v — скорость корабля;
k — коэффициент, который равен 1 для нового корабля и 0,5 для использованного;
y_0 — текущий год (не забудь, что «сейчас» 3019 год);
y_1 — год выпуска корабля.

В приложении используй технологии:

1. Maven (для сборки проекта);
2. Tomcat 9 (для запуска своего приложения);
3. Spring;
4. Spring Data JPA;
5. MySQL (база данных (БД)).

Вот ссылка на архив с полезными книгами, которые помогут тебе в решении тестового задания. Не нужно их все перечитать, используй как справочники.

Отправка готового задания на проверку.

Результат нужно выложить на GitHub в публичный репозиторий.
Перед этим убедись, что все тесты проходят. **ВАЖНО: Файл pom.xml изменять нельзя!**
На странице стажировки нажми кнопку «Отправить заявку», заполни все поля и нажми «Отправить заявку». Дождись результатов автоматической проверки задания (этот процесс занимает до 1 мин). В отдельных случаях может понадобиться ручная проверка задания, которая может длиться несколько дней.

Обрати внимание.

1. Если в запросе на создание корабля нет параметра “isUsed”, то считаем, что пришло значение “false”.
2. Параметры даты между фронтом и сервером передаются в миллисекундах (тип Long) начиная с 01.01.1970.
3. При обновлении или создании корабля игнорируем параметры “id” и “rating” из тела запроса.
4. Если параметр pageNumber не указан – нужно использовать значение 0.
5. Если параметр pageSize не указан – нужно использовать значение 3.
6. Не валидным считается id, если он:
    - не числовой
    - не целое число
    - не положительный
7. При передаче границ диапазонов (параметры с именами, которые начинаются на «min» или «max») границы нужно использовать включительно.

REST API.

Get ships list

URL	/ships
Method	GET
URL Params	**Optional:** name=String planet=String shipType=ShipType after=Long before=Long isUsed=Boolean minSpeed=Double maxSpeed=Double minCrewSize=Integer maxCrewSize=Integer minRating=Double maxRating=Double order=ShipOrder pageNumber=Integer pageSize=Integer
Data Params	None
Success Response	**Code:** 200 OK **Content:** [{ “id”:[Long], “name”:[String], “planet”:[String], “shipType”:[ShipType], “prodDate”:[Long], “isUsed”:[Boolean], “speed”:[Double], “crewSize”:[Integer], “rating”:[Double] }, …]
Notes	Поиск по полям name и planet происходить по частичному соответствию. Например, если в БД есть корабль с именем «Левиафан», а параметр name задан как «иа» - такой корабль должен отображаться в результатах (Лев**иа**фан). pageNumber – параметр, который отвечает за номер отображаемой страницы при использовании пейджинга. Нумерация начинается с нуля pageSize – параметр, который отвечает за количество результатов на одной странице при пейджинге

Get ships count

URL	/ships/count
Method	GET
URL Params	**Optional:** name=String planet=String shipType=ShipType after=Long before=Long isUsed=Boolean minSpeed=Double maxSpeed=Double minCrewSize=Integer maxCrewSize=Integer minRating=Double maxRating=Double
Data Params	None
Success Response	**Code:** 200 OK **Content:** Integer
Notes	

Create ship

URL	/ships
Method	POST
URL Params	None

Data Params	{
	“name”:[String],
	“planet”:[String],
	“shipType”:[ShipType],
	“prodDate”:[Long],
	“isUsed”:[Boolean], --optional, default=false
	“speed”:[Double],
	“crewSize”:[Integer]
	}

Success Response	**Code:** 200 OK
	Content: {
	“id”:[Long],
	“name”:[String],
	“planet”:[String],
	“shipType”:[ShipType],
	“prodDate”:[Long],
	“isUsed”:[Boolean],
	“speed”:[Double],
	“crewSize”:[Integer],
	“rating”:[Double]
	}

Notes	Мы не можем создать корабль, если: - указаны не все параметры из **Data Params (**кроме isUsed**)**; - длина значения параметра “name” или “planet” превышает размер соответствующего поля в БД (50 символов); - значение параметра “name” или “planet” пустая строка; - скорость или размер команды находятся вне заданных пределов; - “prodDate”:[Long] < 0; - год производства находятся вне заданных пределов. В случае всего вышеперечисленного необходимо ответить ошибкой с кодом 400.

Get ship

URL	/ships/{id}
Method	GET
URL Params	id
Data Params	None

Success Response	**Code:** 200 OK
	Content: {
	“id”:[Long],
	“name”:[String],
	“planet”:[String],
	“shipType”:[ShipType],
	“prodDate”:[Long],
	“isUsed”:[Boolean],
	“speed”:[Double],
	“crewSize”:[Integer],
	“rating”:[Double]
	}

Notes	Если корабль не найден в БД, необходимо ответить ошибкой с кодом 404. Если значение id не валидное, необходимо ответить ошибкой с кодом 400.

Update ship

URL	/ships/{id}
Method	POST
URL Params	id

Data Params	{
	“name”:[String], --optional
	“planet”:[String], --optional
	“shipType”:[ShipType], --optional
	“prodDate”:[Long], --optional
	“isUsed”:[Boolean], --optional
	“speed”:[Double], --optional
	“crewSize”:[Integer] --optional
	}

Success Response	**Code:** 200 OK
	Content: {
	“id”:[Long],
	“name”:[String],
	“planet”:[String],
	“shipType”:[ShipType],
	“prodDate”:[Long],
	“isUsed”:[Boolean],
	“speed”:[Double],
	“crewSize”:[Integer],
	“rating”:[Double]
	}

Notes	Обновлять нужно только те поля, которые не null. Если корабль не найден в БД, необходимо ответить ошибкой с кодом 404. Если значение id не валидное, необходимо ответить ошибкой с кодом 400.

Delete ship

URL	/ships/{id}
Method	DELETE
URL Params	id
Data Params	
Success Response	**Code:** 200 OK
Notes	Если корабль не найден в БД, необходимо ответить ошибкой с кодом 404. Если значение id не валидное, необходимо ответить ошибкой с кодом 400.