Create project & scaffolding

1. New project wizard
    - Java 21
    - Maven
    - No sample code
    - Dependencies:
        - REST Jackson
        - Hibernate ORM Panache
        - PostgreSQL JDBC Driver
        - OpenAPI
        - Health
        - Jib
2. Edit `pom.xml`
    a. Add test dependencies (`tdd 2 - pomTestDeps` live template)
3. Go into `src/main/resources/application.properties` and add base config (`tdd 3 - baseConfig` live template)

Create Entity

1. Start dev mode
2. Discuss `Hibernate ORM is disabled because no JPA entities were found` & `Dev Services for default datasource (postgresql) started`
    - Show & explain Dev UI
3. Create the `com.example` package
4. Create `Pet` class (if using an AI assistant, use the `tdd 4 - aiPetEntity` live template), otherwise:
    - Add fields (`tdd 4 - petFields` live template)
    - Add getters/setters (IntelliJ assist)
    - Add `toString`/`hashCode`/`equals` (IntelliJ assist)
    - Add constructors (`tdd 5 - petConstructors` live template)
    - Turn into JPA Entity (`tdd 6 - petAnnotations` live template)
5. Add `src/main/resources/import.sql` file
    - Use `tdd 7 - importsql` live template

Create Repository layer tests

1. Back in dev mode console point out `No tests found` (duh!)
2. Create the `src/test/java` folder and the `com.example` package
3. Create `src/test/java/com/example/PetRepositoryTests.java`
    - Annotate with `@QuarkusTest`
4. Bring in test stubs with `tdd 8 - petRepositoryTests` live template
    - Notice in console that as soon as saved tests all run to success
5. Add `@TestTransaction` annotation & explain
    - Again all tests run when saving
6. Add `@Inject PetRepository`
    - Obviously this does not compile because `PetRepository` doesn't exist
    - Use IntelliJ assist to create class
7. Add `@ApplicationScoped` annotation so it can be found as a bean
8. Make `PetRepository` implement `PanacheRepository<Pet>`
9. Add method stubs with `tdd 9 - petRepository` live template
10. Return to `PetRepositoryTests` and implement `petsByKindFound` following steps:

```
setupTest();

// Persist a new pet to the repo
this.petRepository.persist(new Pet("fluffy", "cat"));

// Assert that finding the pets by kind returns the correct result
assertThat(this.petRepository.findPetsByKind("cat"))
        .isNotNull()
        .singleElement()
        .extracting(Pet::getName, Pet::getKind)
        .containsExactly("fluffy", "cat");
```

10. Implement `noPetsByKindFound` following steps:

```
setupTest();

// Assert that finding the pets by kind returns empty
assertThat(this.petRepository.findPetsByKind("cat"))
        .isNotNull()
        .isEmpty();
```

11. Implement `adoptFoundPet` following steps:

```
setupTest();

// Persist some pets
this.petRepository.persist(
        new Pet("fluffy", "cat"),
        new Pet("harry", "dog")
);

// Assert that adopting a found pet is correct
assertThat(this.petRepository.adoptPetIfFound("cat", "Eric"))
        .isNotNull()
        .get()
        .extracting(Pet::getKind, Pet::getName, Pet::getAdoptedBy)
        .containsExactly("cat", "fluffy", "Eric");
```

12. Implement `noAdoptablePetFound` following steps:

```
setupTest();

// Persist some pets
this.petRepository.persist(
        new Pet(null, "fluffy", "cat", "Eric"),
        new Pet("harry", "dog")
);

// Assert that no pet is found for adoption
assertThat(this.petRepository.adoptPetIfFound("cat", "Eric"))
        .isNotNull()
        .isEmpty();
```

13. At this point all 4 tests should be failing!

Implement Repository

```
public List<Pet> findPetsByKind(String kind) {
    Log.infof("Looking for all pets of kind '%s'", kind);
    return list("kind", kind);
}

@Transactional
public Optional<Pet> adoptPetIfFound(String kind, String owner) {
    Log.infof("Looking for an adoptable pet of kind '%s'", kind);
    var pet = find("kind = ?1 AND adoptedBy IS NULL ORDER BY RANDOM() LIMIT 1", kind)
       .withLock(LockModeType.PESSIMISTIC_WRITE)
       .firstResultOptional();

    pet.ifPresentOrElse(
        p -> {
            Log.infof("Found pet for adoption: %s", pet);
            p.setAdoptedBy(owner);
            persist(p);
        },
        () -> Log.infof("No pet of kind '%s' available for adoption", kind)
    );

    return pet;
}
```

Create REST layer tests

1. Create `src/test/java/com/example/PetResourceTests.java`
    - Annotate with `@QuarkusTest`
2. Add `@InjectMock PetRepository`
3. Bring in test stubs with `tdd 10 - petResourceTests` live template
    - Notice in console that as soon as saved tests all run to success
4. Implement `getAll` method

```
// Set up mock to return a pet when repo.listAll() is called
when(this.petRepository.listAll())
        .thenReturn(List.of(new Pet(1L, "fluffy", "cat")));

// Execute GET to /pets & assert
get("/pets").then()
        .statusCode(Status.OK.getStatusCode())
        .contentType(ContentType.JSON)
        .body("$.size()", is(1))
        .body("[0].name", is("fluffy"))
        .body("[0].kind", is("cat"))
        .body("[0].adoptedBy", blankOrNullString());

// Verify interactions
verify(this.petRepository).listAll();
verifyNoMoreInteractions(this.petRepository);
```

5. Highlight all remaining tests and implement in one shot with `tdd 11 - petResourceTestsOthers` live template

Implement REST layer

1. Create `src/main/java/com/example/PetResource.java`
2. *Optional*: Use AI assist to create the class (highlight `PetResourceTests` contents & `tdd 11a - aiCreatePetResource` live template)
3. *If not using AI to create*: Explain how we know what we need to create
   - `/pets` returns all `Pet` s
   - `/pets?kind={kind}` returns all `Pet` s of a certain kind
   - `/pets/{id}` returns a `Pet` given an id
     - **OR** returns a `404` if that `Pet` is not found
4. Implement methods:

```
private final PetRepository petRepository;

public PetResource(PetRepository petRepository) {
    this.petRepository = petRepository;
}

@GET
public List<Pet> getAll(@QueryParam("kind") Optional<String> kind) {
    return kind.map(this.petRepository::findPetsByKind)
        .orElseGet(this.petRepository::listAll);
}

@GET
@Path("/{id}")
public Response getPetById(@PathParam("id") Long id) {
    return this.petRepository.findByIdOptional(id)
        .map(Response::ok)
        .orElseGet(() -> Response.status(Status.NOT_FOUND))
        .build();
}
```

5. Add `@RunOnVirtualThread` to class and explain
6. *Optional*: Run `http :8080/pets` to see all pets
7. *Optional*: Run `http ":8080/pets?kind=cat"` to see all cats
8. *Optional*: Run `http ":8080/pets?kind=horse"` to see that there aren't any horses
9. *Optional*: Run `http :8080/pets/1` to see a certain pet
10. *Optional*: Run `http :8080/pets/5` to see a pet not found (`404` error)

Set up for Kafka

1. Explain that now we have to listen on Kafka for incoming adoption request messages
   - If we have an available pet, process the adoption and put the adoption message on another Kafka topic
   - If we don't have an available pet, do nothing
2. In new terminal add Kafka dependency with `quarkus ext add messaging-kafka`
3. Open `pom.xml` and add Kafka test dependencies (`tdd 12 - pomKafkaTestDeps` live template)
4. Go back to dev mode terminal to see reload
   - And also see that there is now a Kafka broker running (can verify in dev ui)
5. Go into `src/main/resources/application.properties` and add kafka config (`tdd 13 - petkafkaconfig` live template)

Create AdoptionListenerTests

1. Create `src/test/java/com/example/AdoptionListenerTests.java`
2. Add `@QuarkusTest` annotation to class
3. Use `tdd 14 - adoptionListenerTests` live template to insert class outline

4. Explain that we will need an `AdoptionRequest` class - create it using IntelliJ assist

```
@RegisterForReflection
public record AdoptionRequest(String owner, String kind) { }
```

5. Explain that we will need an `AdoptionListener` class
    - Use IntelliJ assist to create class from field
    - Add `@ApplicationScoped` to the class to add it as a bean
6. Add fields to `AdoptionListener` with `tdd 15 - adoptionListenerFields` live template
7. Implement `adoptablePetFound` following steps:

```
// Set up mock
when(this.petRepository.adoptPetIfFound(adoptionRequest.kind(), adoptionRequest.owner()))
    .thenReturn(Optional.of(pet));

// Send request to channel
this.inMemoryConnector.source(AdoptionListener.ADOPTION_REQUESTS_CHANNEL_NAME)
    .send(adoptionRequest);

// Create sink
var sink = this.inMemoryConnector.sink(AdoptionListener.ADOPTIONS_CHANNEL_NAME);

// Wait for messages to arrive in sink
await()
    .atMost(Duration.ofSeconds(10))
    .until(() -> sink.received().size() == 1);

// Perform assertions on received message(s)
assertThat(sink.received())
    .isNotNull()
    .singleElement()
    .extracting(Message::getPayload)
    .usingRecursiveComparison()
    .isEqualTo(new Pet(pet.getId(), pet.getName(), pet.getKind(), adoptionRequest.owner()));

// Verify interactions
verify(this.petRepository).adoptPetIfFound(adoptionRequest.kind(), adoptionRequest.owner());
verify(this.adoptionListener).handleAdoption(any(AdoptionRequest.class));
verifyNoMoreInteractions(this.petRepository);
```

8. Notice `handleAdoption` method doesn't exist
    - Use IntelliJ assist to create it
9. Add annotations to the `handleAdoption` method using the `tdd 16 - adoptionListenerHandleAdoptionAnnotations` live template
10. Implement `adoptablePetNotFound` following steps:

```
// Set up mock
when(this.petRepository.adoptPetIfFound(adoptionRequest.kind()  , adoptionRequest.owner()))
    .thenReturn(Optional.empty());

// Send request to channel
this.inMemoryConnector.source(AdoptionListener.ADOPTION_REQUESTS_CHANNEL_NAME)
    .send(adoptionRequest);

// Verify interactions (with timeout)
verify(this.petRepository, timeout(10_000)).adoptPetIfFound(adoptionRequest.kind(), adoptionRequest.owner());
verify(this.adoptionListener, timeout(10_000)).handleAdoption(any(AdoptionRequest.class));
verifyNoMoreInteractions(this.petRepository);
```

11. Tests should still be failing.
    - Now we need to implement `AdoptionListener`

Implement AdoptionListener

1. Add attributes:

```
private final PetRepository petRepository;
private final Emitter<Pet> petEmitter;

public AdoptionListener(PetRepository petRepository, @Channel(ADOPTIONS_CHANNEL_NAME) Emitter<Pet> petEmitter) {
    this.petRepository = petRepository;
    this.petEmitter = petEmitter;
}
```

2. Implement `handleAdoption`:

```
Log.infof("Handling adoption for request: %s", adoptionRequest);
this.petRepository.adoptPetIfFound(adoptionRequest.kind(), adoptionRequest.owner())
    .ifPresent(this.petEmitter::send);
```

Integration tests (if time permits)

1. Use `tdd - petResourceIT` live template for `PetResourceIT`
2. Use `tdd - adoptionListenerIT` live template for `AdoptionListenerIT`