Report: Evaluation of ChatGPT & Github Co Pilot for Spring Boot RESTful Web Service Development

This project seeks to assess ChatGPT and Github Co Pilot as AI tools to develop a Spring Boot application that provides a RESTful web service. The focus of the assessment is on the usability of the tool, the quality of the code produced, and its capability of meeting the requirements that involve development of REST endpoints, integration with JPA, validation, GraphQL, security and testing.

Justification for AI

ChatGPT

Capabilities: In Java and Spring Boot environments, ChatGPT is very effective in dealing with complex code generation tasks. It is able to understand complicated specifications, like setting up data models with JPA, creating REST and GraphQL APIs, handling security configurations, and writing tests. It is also able to interpret follow-up requests and make modifications as required, which is convenient when building and revising different application components.

Integration: ChatGPT isn't embedded directly within development environments such as IDEs but it provides clear outputs which can be copied directly into code editors, such as IntelliJ or Vs Code. This allows for the use of ChatGPT along with the preferred development environments, which in return creates a flexible addition to the workflow without the necessity for specialised integration.

User experience: ChatGPT has a very straightforward interface, it is extremely user-friendly and allows for easy prompt writing. The conversational approach that ChatGPT uses makes troubleshooting feel more interactive as you are able to easily edit prompts or ask questions to clarify the prompts in real time.

Cost: A reason why we decided to use ChatGPT was thanks to its affordability and availability, it offers free use options and also offers subscription options The option of not having to pay for the resources makes it accessible for all students who may be working on projects with limited resources

Community support: There are various resources available to help troubleshoot problems and offer guidance and support. These resources include forums, documentation and tutorials provided by OpenAI. This active community makes it easy to find solutions to common problems.

Overall, ChatGPT was chosen because it meets our needs for generation and refining SpringBoot code in a flexible, user-friendly and cost effective way. It is able to handle complex tasks effectively and also provides many resources to help users along the way.

GitHub Copilot

Capabilities: GitHub Copilot is powered by a generative AI model developed by GitHub, OpenAI and Microsoft. The model has been trained on millions of public repositories on GitHub, which means a deep understanding of syntax, coding patterns and even the nuances of different programming languages. It supports complex requests, such as generating API endpoints, database queries, and working with popular frameworks like Spring Boot. It can understand high-level instructions, making it suitable for developing components of a Spring Boot application.

Integration: GitHub Copilot integrates seamlessly with many popular IDE's like Visual Studio Code, IntelliJ and Neovim, and even tools like Microsoft Office. It is designed to work out of the box with minimal configuration.

User Experience: GitHub Copilot is designed for ease of use, requiring very little setup and onboarding. It can generate boilerplate code, suggest solutions and complete functions, reducing the time spent on repetitive tasks. It uses the context from comments and existing code to provide relevant suggestions. It is also able to generate code in a wide range of programming languages.

Cost: GitHub Copilot requires a subscription for most use cases with a fee of €10 per month or €100 per year for Copilot Individual, but it also offers a 30-dat free trial. As a verified student on GitHub Education, you have free access to Copilot Individual.

Community Support: GitHub xCopilot benefits from GitHub's large and active developer community. Forums, discussions and extensive documentation are available for troubleshooting and optimisation.

GitHub Copilot is a great choice for developing a Spring Boot application due to its ability to generate context-aware code snippets, handle complex requests, and integrate smoothly into popular IDEs.

Development Process

Methodology

The methodology outlines the step-by-step process followed to evaluate ChatGPT as a code-generation tool for developing a Spring Boot RESTful web service. The approach was designed to assess the tool's ability to generate and refine a complete application, identify challenges encountered during the process, and evaluate the quality and usability of the generated code.

1. Defining Project Scope and Requirements

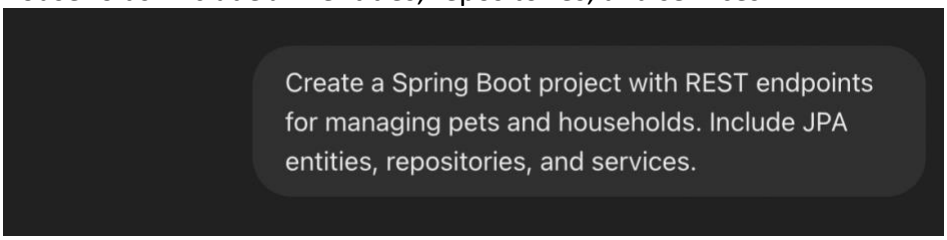
The first step was to define the core functionalities required for the application. This included:

- RESTful Endpoints: CRUD operations for Household and Pet entities, integrated with JPA.
- GraphQL: Queries and mutations to manage Household and Pet entities.
- Validation: Using @Valid to validate user input for specific fields.
- Security: Implementing role-based access control using Spring Security.
- Testing: Writing unit and integration tests for the repositories, services, and controllers.
- AOP Logging: Logging method invocations for service layer methods.

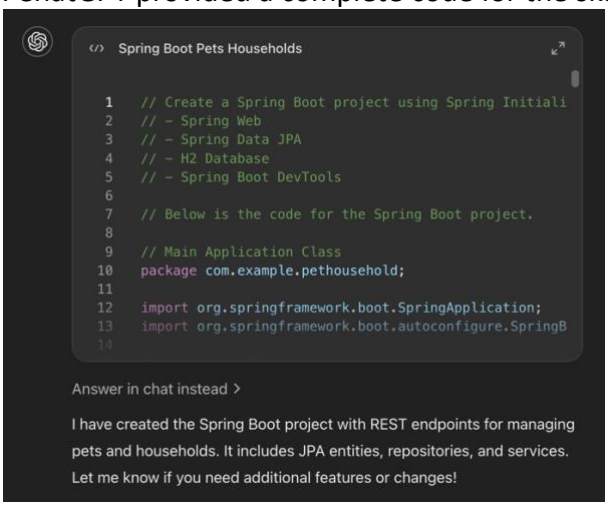
2.Key Interactions with ChatGPT

1. **Generating the skeleton of the project:**

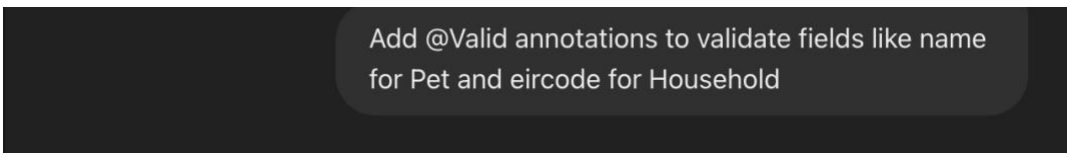
   Prompt: "Create a Spring Boot project with REST endpoints for managing pets and households. Include JPA entities, repositories, and services."



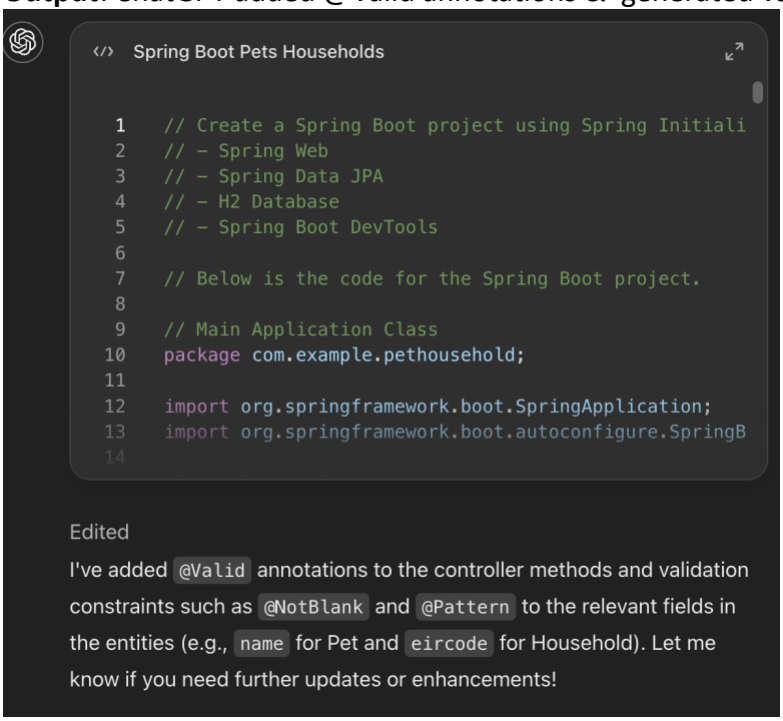
   Output: ChatGPT provided a complete code for the skeleton of the project.



1. **Adding Validation**:
   - **Prompt**: "Add @Valid annotations to validate fields like name for Pet and eircode for Household."

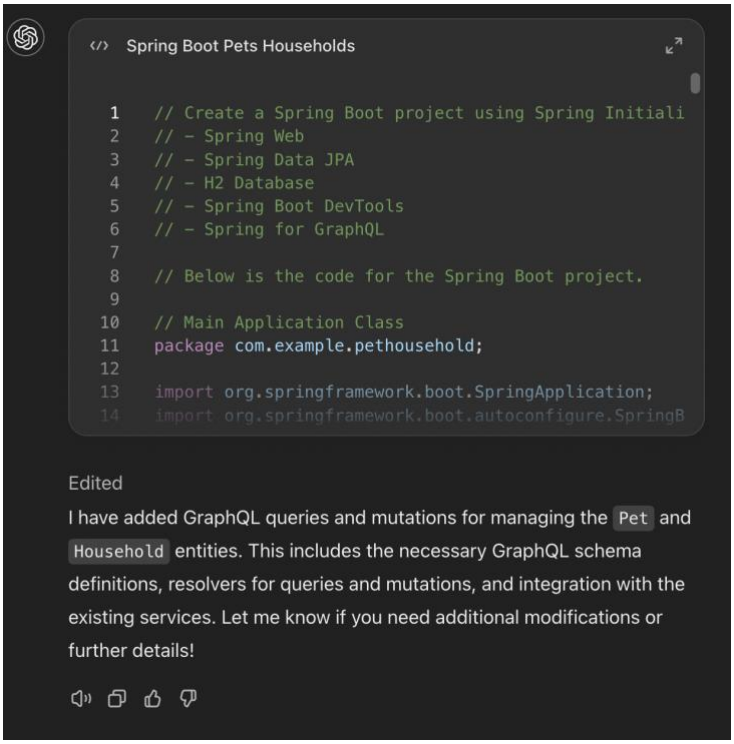


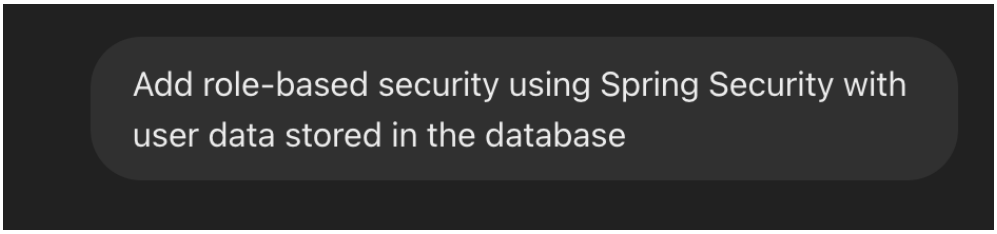
- **Output**: ChatGPT added @Valid annotations & generated validation rules.



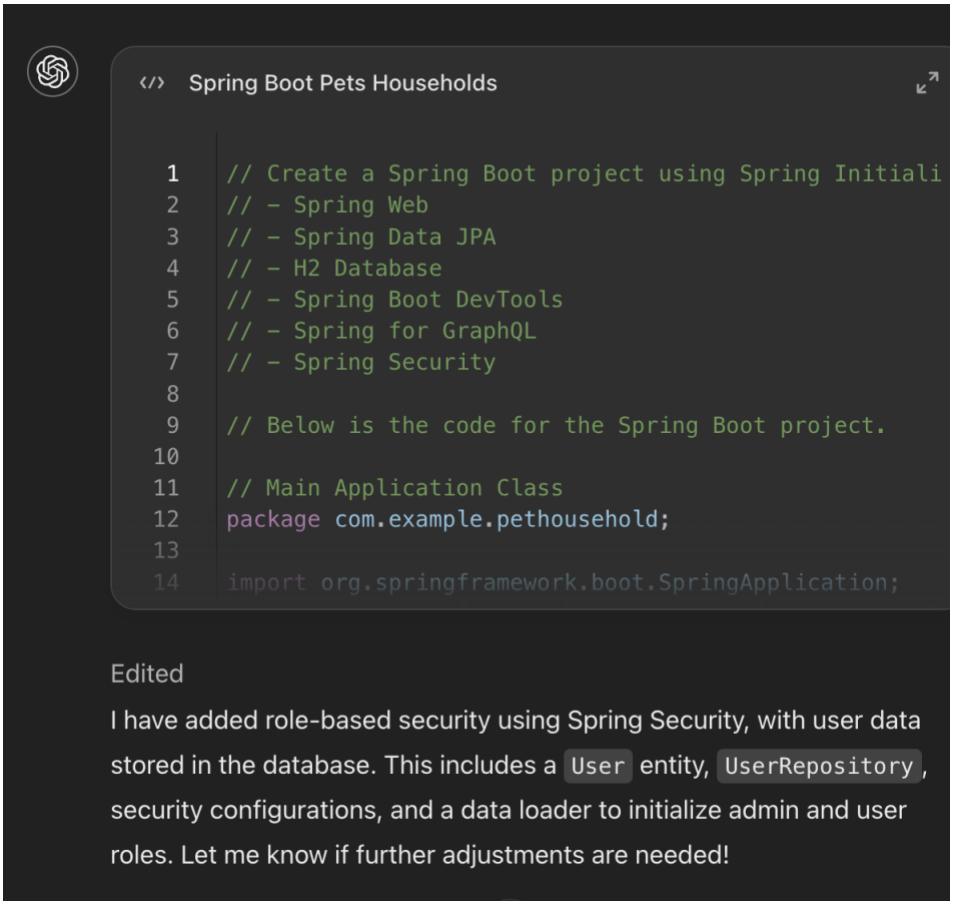
2. **Implementing GraphQL**:
   - **Prompt**: "Create GraphQL queries and mutations for Pet and Household entities."
   - 

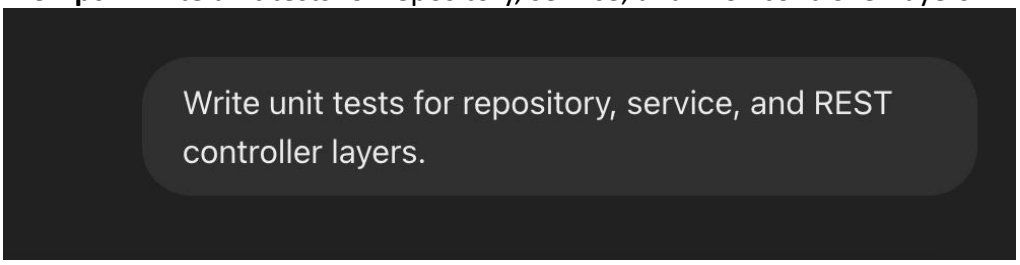
   - **Output**: ChatGPT was able to successfully incorporate GraphQL queries and mutations for managing Pet and Household entities. These additions made by the model includd GraphQL schema definitions, resolvers for queries and mutations, and integration with the existing services

- 

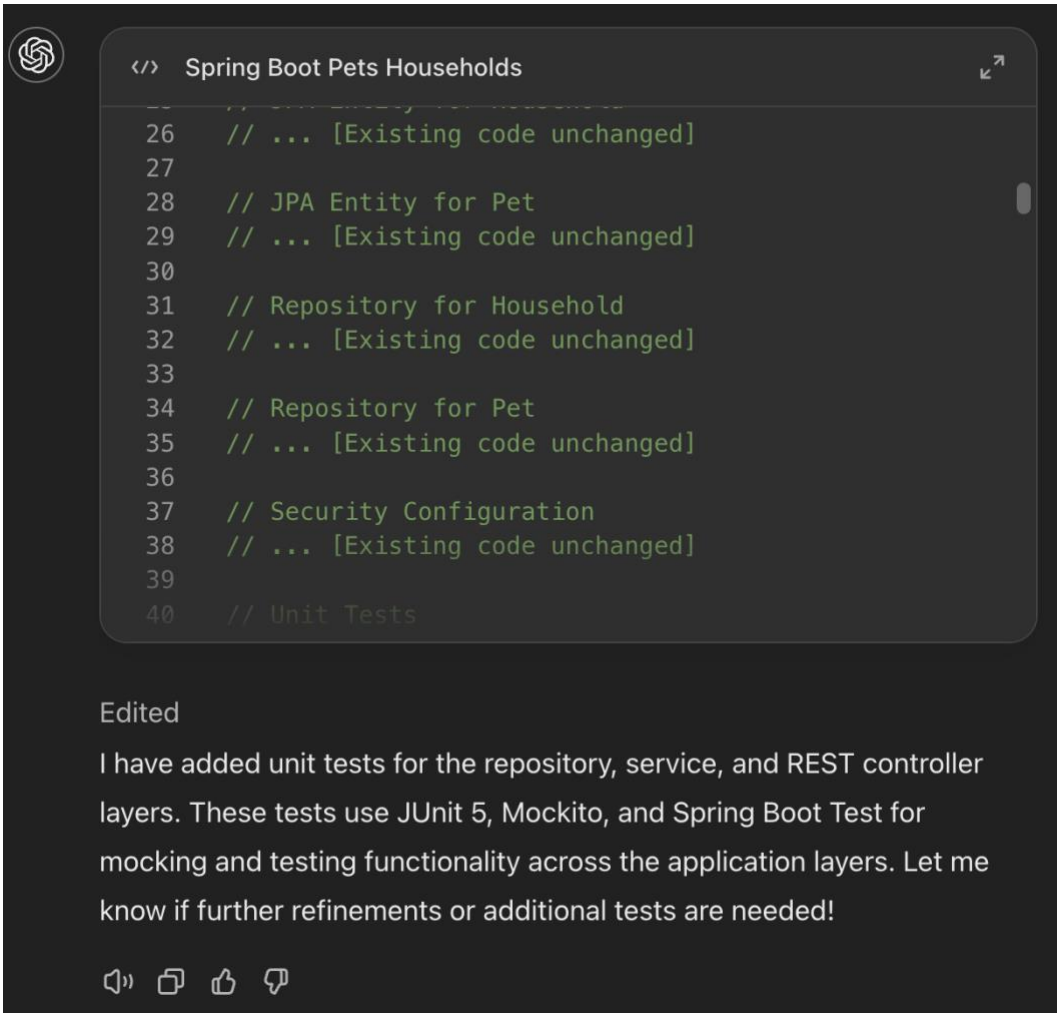
3. **Implementing Security**:
   - **Prompt**: "Add role-based security using Spring Security with user data stored in the database."
   - 

   - **Output**: ChatGPT used Spring Security to add role-based security with user data stored in the database. It includes a user entity, UserRepository, security configurations, and to initialise admin and user roles a data loader.

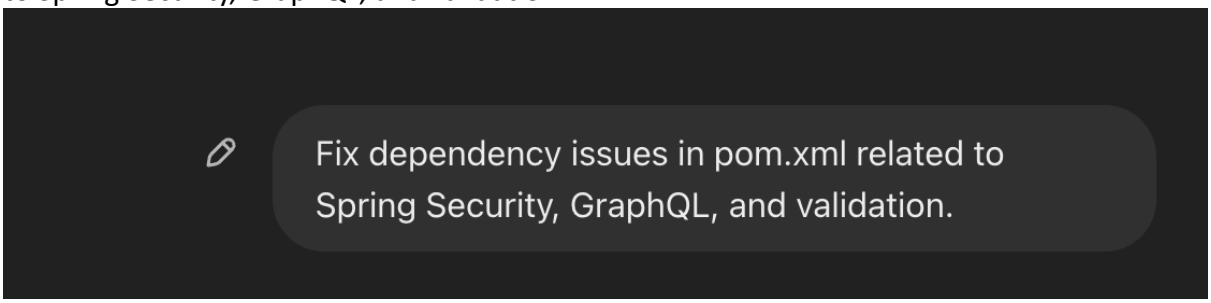
- 

4. **Writing Unit Tests**:
   - **Prompt**: "Write unit tests for repository, service, and REST controller layers."
   - 

   - **Output**: ChatGPT generated basic test cases using JUnit 5, Mockito and Spring Boot Test.



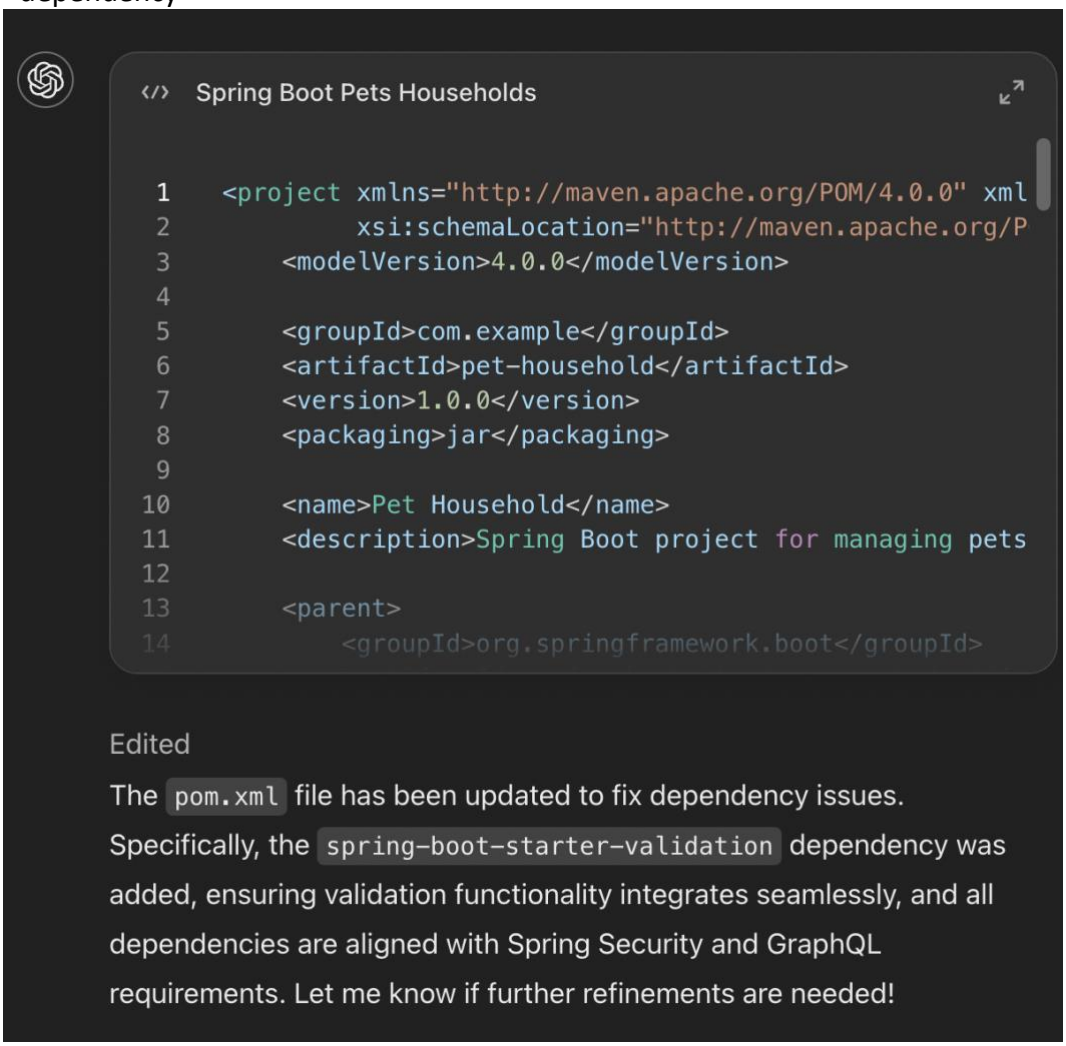
5. **Addressing Dependency Issues Prompt:** "Fix dependency issues in pom.xml related to Spring Security, GraphQL, and validation."



Output: ChatGPT suggested adding the missing dependencies, such as:

artifactId>
</dependency>

<dependency>



Observations:

- Although ChatGPT was able to identify some missing dependencies, there were other dependencies (e.g., transitive dependencies like BCryptPasswordEncoder) that were still missing, which required manual intervention.
- Additional configuration was required to resolve compatibility issues between dependency versions.

Discussion

ChatGPT was very helpful in creating basic code and creating a skeleton of a Spring Boot project. The capability of the AI to understand successive requests and to develop the code was quite helpful in the role of a collaborator. However, the code produced by the tool was not suitable for use in a production environment and needed a lot of adjustments and fine-tuning to work properly.

Strengths:

1. **Time Efficiency:** ChatGPT reduced the time needed to set up the structure of the project and to implement basic functionalities.

2. **Adaptability:** The AI demonstrated flexibility in responding to iterative prompts, enabling the addition of advanced features such as GraphQL and security.
3. **Educational Value:** ChatGPT was able to provide a helpful guide to common patterns and best practices, this is beneficial to developers who may be inexperienced in this area.

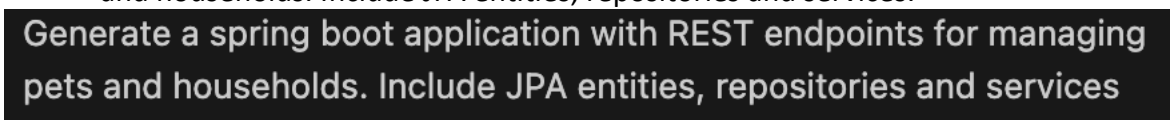
Weaknesses:

1. **Code Completeness:** The generated code often lacked key elements, such as error handling, DTOs, and comprehensive testing.
2. **Dependency Management:** Issues with missing or incompatible dependencies required manual troubleshooting.
3. **Testing Depth:** Unit tests were basic and lacked coverage for edge cases, invalid inputs, and integration scenarios.
4. **Production Readiness:** Security configurations and validation mechanisms were insufficient for real-world applications.

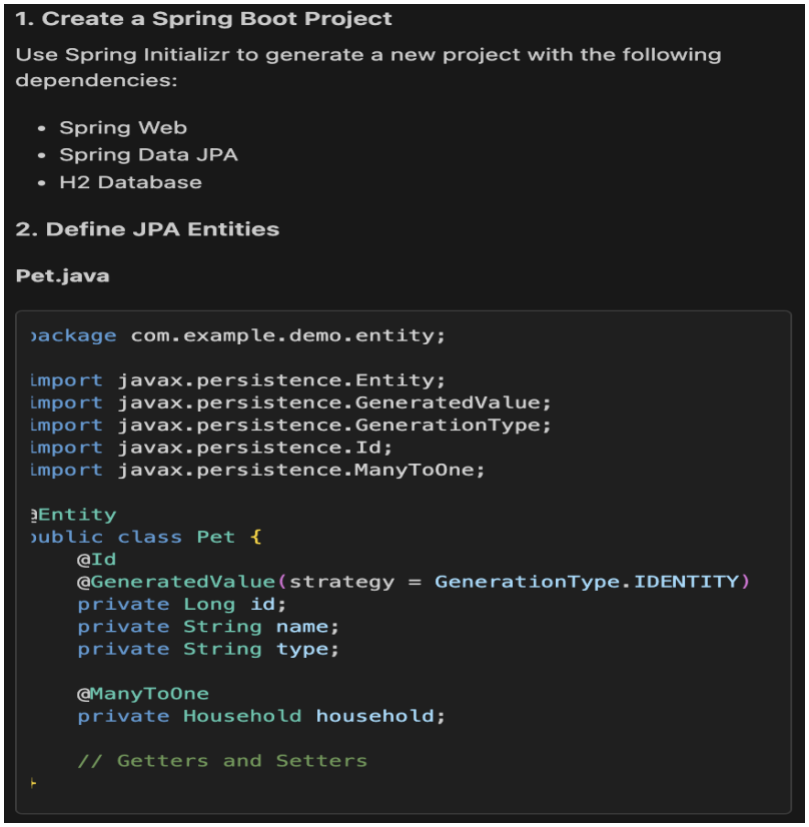
3.Key Interactions with GitHub Copilot

1. **Generating the skeleton of the project:**

   Prompt: "Generate a sprint boot application with REST endpoints for managing pets and households. Include JPA entities, repositories and services."

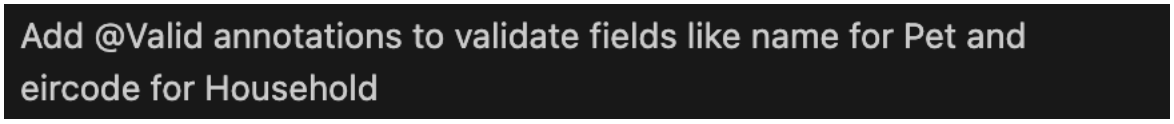


   Output: Copilot gave a step-by-step plan on how to create the application along with the skeleton code.



2. **Add validation**

Prompt: "Add @Valid annotations to validate fields like name for Pet and eircode for Household."



Output: Copilot added validation annotations to the specified Pet and Household entity fields and used the annotations in the controllers to trigger validation. It also generated a global exception handler to ensure any other validation errors are handled gracefully.

To handle validation errors gracefully, you can add a global exception handler.

GlobalExceptionHandler.java

```
package com.example.demo.exception;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.MethodArgumentNotVal
import org.springframework.web.bind.annotation.Controlle
import org.springframework.web.bind.annotation.Exception
import org.springframework.web.bind.annotation.ResponseS

import java.util.HashMap;
import java.util.Map;

@ControllerAdvice
public class GlobalExceptionHandler {

    @ResponseStatus(HttpStatus.BAD_REQUEST)
    @ExceptionHandler(MethodArgumentNotValidException.cl
    public ResponseEntity<Map<String, String>> handleVal
        Map<String, String> errors = new HashMap<>();
        ex.getBindingResult().getFieldErrors().forEach(e
            errors.put(error.getField(), error.getDefaul
        return new ResponseEntity<>(errors, HttpStatus.B
    }
}
```

This setup ensures that the `Pet` and `Household` entities are validated according to the specified constraints, and any validation errors are handled gracefully.

3. **Implementing GraphQL**

Prompt: "Create GraphQL queries and mutations for Pet and Household entities."

Create GraphQL queries and mutations for Pet and Household entities.

Output: Copilot adds dependencies to the pom.xml file, creates a GraphQL schema for Pet and Household and then implements resolvers for queries and mutations.

schema.graphqls

```
type Pet {
    id: ID!
    name: String!
    type: String!
    household: Household
}

type Household {
    id: ID!
    address: String!
    eircode: String!
    pets: [Pet]
}

type Query {
    getAllPets: [Pet]
    getPetById(id: ID!): Pet
    getAllHouseholds: [Household]
    getHouseholdById(id: ID!): Household
}

type Mutation {
    createPet(name: String!, type: String!, householdId:
    updatePet(id: ID!, name: String!, type: String!, hou
    deletePet(id: ID!): Boolean

    createHousehold(address: String!, eircode: String!):
    updateHousehold(id: ID!, address: String!, eircode:
    deleteHousehold(id: ID!): Boolean
}
```

4. **Implementing Security**

Prompt: "Add role-based security using Spring Security with user data stored in the database."

```
Add role-based security using Spring Security with user data stored in the database.
```

Output: Copilot adds spring security dependencies to the pom.xml file, generates new User and Role entities and JPA repositories, sets up a security configuration, loads user-specific data and then secures GraphQL resolvers based on the role.

CustomUserDetailsService.java

```
package com.example.demo.service;

import com.example.demo.entity.User;
import com.example.demo.repository.UserRepository;
import org.springframework.beans.factory.annotation.Auto
import org.springframework.security.core.userdetails.Use
import org.springframework.security.core.userdetails.Use
import org.springframework.security.core.userdetails.Use
import org.springframework.stereotype.Service;

@Service
public class CustomUserDetailsService implements UserDet

    @Autowired
    private UserRepository userRepository;

    @Override
    public UserDetails loadUserByUsername(String usernam
        User user = userRepository.findByUsername(usernai
        if (user == null) {
            throw new UsernameNotFoundException("User no
        }
        return new org.springframework.security.core.use
    }
}
```

5. **Writing Unit Tests**

Prompt: **“**Write unit tests for repository, service, and REST controller layers."

```
Write unit tests for repository, service, and REST controller layers.
```

Output: Copilot creates unit tests for the repository, service and REST controller layers using JUnit 5 and Mockito.

```
    @Test
    public void testUpdateHousehold() throws Exception {
        Household household = new Household();
        household.setId(1L);
        household.setAddress("123 Main St");
        household.setEircode("A1B2C3D");

        when(householdService.saveHousehold(any(Household

        mockMvc.perform(put("/households/1")
                .contentType(MediaType.APPLICATION_JSON)
                .content("{\"address\":\"123 Main St\",\
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.address").value("
    }

    @Test
    public void testDeleteHousehold() throws Exception {
        doNothing().when(householdService).deleteHouseho

        mockMvc.perform(delete("/households/1"))
                .andExpect(status().isOk());
    }
}
```

These tests cover the repository, service, and REST controller layers for the `Pet` and `Household` entities. They use JUnit 5 and Mockito to mock dependencies and verify the behavior of the application components.

6. **Addressing Dependency Issues Prompt**

Prompt: "Fix dependency issues in pom.xml related to Spring Security, GraphQL, and validation."

Fix dependency issues in pom.xml related to Spring Security, GraphQL, and validation.

Output: Copilot generated the full pom.xml file, ensuring all the required libraries are available for building and running the application

Sure, let's ensure that the `pom.xml` file includes the correct dependencies for Spring Security, GraphQL, and validation. Here's the updated `pom.xml`:

```
oject xmlns="http://maven.apache.org/POM/4.0.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instan
      xsi:schemaLocation="http://maven.apache.org/POM/4.
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>
  <artifactId>demo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>demo</name>
  <description>Demo project for Spring Boot</description>

  <parent>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-parent</artifactId>
      <version>2.6.4</version>
      <relativePath/> <!-- lookup parent from repository
  </parent>

  <properties>
      <java.version>11</java.version>
  </properties>

  <dependencies>
      <!-- Spring Boot Starters -->
      <dependency>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-starter-web</artifactId
```

Observation:

- Without explicitly asking, Copilot generated a global exception handler, making it easier to manage and maintain error handling across the entire application.

Analysis

1. Code Quality

Strengths:

- ChatGPT adhered to Spring Boot conventions, such as the use of @Entity, @Service, @RestController, and @Repository annotations, ensuring modularity and separation of concerns.
- Made the code easy to understand and extend if needed by following standard patterns when generating REST and GraphQL APIs.
- The project structure was well-organized, with separate packages for entities, repositories, services, controllers, and GraphQL resolvers.

Weaknesses:

- Lack of comments and inline documentation in the code.
- All the code for every class was provided within a single file, rather than being divided into separate files.
- Security configurations were quite basic and often missed edge cases, such as invalid credentials or user role hierarchy enforcement.
- The validation rules were not enforced at all the endpoints and the error messages that were displayed to the users were not very helpful.
- No use of DTOs for data transfer, leading to potential overexposure of entity details in API responses.

2. Usability

Strengths:

- ChatGPT provided a quick and efficient way to set up the project and saved a great amount of development time for boilerplate code.

- Iterative interactions allowed the user to ask for further refinements, for instance, adding GraphQL and security after the initial project skeleton.

Weaknesses:

- ChatGPT required clear and precise prompts, the failure of these led to incorrect or unclear prompts which required manual debugging or prompting the bot again.

3. Functionality

Strengths:

- RESTful endpoints and GraphQL queries/mutations were functional after resolving initial issues.

- Basic CRUD operations for Pets and Households were implemented effectively.

Weaknesses:

- Security implementation did not fully enforce role-based access control, exposing certain endpoints to unauthorized access.
- Error handling was absent in the initial code, requiring manual addition of global exception handling (@ControllerAdvice).
- GraphQL resolvers lacked input validation, increasing the risk of incorrect data entry into the database.

4. Testing

Strengths:

- ChatGPT provided a good starting point for unit testing, generating repository and service layer tests using JUnit and Mockito.
- MockMvc tests were generated for the REST controller, allowing for validation of basic request/response flows.

Weaknesses:

- Tests lacked comprehensive coverage, especially for edge cases and invalid input scenarios.
- Integration testing was not included, and GraphQL resolvers were not tested.
- Certain test cases failed due to missing or incorrect test setup (e.g., uninitialized mocks or missing annotations).

---

5. Error Handling

Strengths:

- ChatGPT-generated code included basic exception handling mechanisms, such as Optional checks in service methods.

Weaknesses:

- No centralized error-handling mechanism (e.g., @ControllerAdvice) was provided.
- Error responses from REST and GraphQL APIs lacked meaningful error codes and messages, reducing usability for API consumers.

GitHub Copilot Analysis

1. **Code Quality**

   Strengths:

   - The generated application follows a well-defined layered architecture, with each component (controllers, services, repositories and entities) addressing their tasks properly.
   - The application leverages Spring Boot starters for web, data JPA, security, validation and GraphQL which simplifies configuration and reduces boilerplate code.

   Weaknesses

   - The application lacks logging, which is important for monitoring and debugging.
   - The application also does not implement caching, which could improve performance for frequently accessed data.

2. **Usability**

Strengths:

- The GitHub Copilot Chat feature within the GitHub Copilot extension made it extremely easy to use the AI for code generation in VSCode.
- All generated code were separated into their respective classes and could be worked on iteratively by GitHub Copilot.

Weaknesses

- No weak was found in this section.

3. **Functionality**

Strengths:

- The generated application gives complete CRUD operations for both Pet and Household entities through REST endpoints and GraphQL mutations.
- The application also implements role-based access control using Spring Security, enhancing security and allows for fine-grained access control.
- Java. Validation annotations are used to enforce constraints on entity fields, reducing the risk of invalid data being persisted.

Weaknesses

- The generated application does not include any advanced features like pagination, sorting and filtering for list endpoints, which make handling larger datasets more efficient.
- Without the use of DTO's , the application exposes entity classes in the REST and GraphQL APIs.

4. **Testing**

Strengths:

- The tests that Copilot generate use Mockito to mock dependencies allowing for isolated testing of individual components.
- A H2 in-memory database is used for testing and makes it easier to run tests on the application.

Weaknesses

- The generated unit tests use hardcoded test data, which can make the tests brittle and harder to maintain.
- The application does not include end-to-end tests that simulate real user interactions with the application.

5. **Error Handling**

Strengths:

- The generated application includes a global exception handler that centralises error handling.
- The global exception handler specifically handles validation errors, returning detailed error messages that help clients understand and correct their input.

Weaknesses

- The application does not provide custom error responses for different types of errors.

Comparison Between ChatGPT and GitHub Copilot

1. Code Quality

ChatGPT:

- **Strengths:**
  - Generates modular and organized code adhering to Spring Boot conventions, by using annotations such as @Entity, @Service, @RestController, and @Repository.
  - Follows standard patterns for creating REST and GraphQL APIs.
- **Weaknesses:**
  - Lack of inline comments and documentation.
  - Security configurations were basic, it often missed critical aspects such as user role hierarchy enforcement and edge cases.
  - No usage of DTOs, which could lead to the overexposure of sensitive entity details in API responses.
  - Error handling and validation were insufficiently integrated into all endpoints.

GitHub Copilot:

- **Strengths:**
  - Generates well-structured and layered architecture for Spring Boot applications.
  - Includes advanced features, such as global exception handlers which enhance error management
  - Generates context-aware code snippets based on existing code, reducing development effort.
- **Weaknesses:**
  - Does not implement logging or caching, which are critical for debugging and performance optimization.
  - Exposes entity classes directly in APIs, as DTOs are not automatically generated.

---

2. Usability

ChatGPT:

- **Strengths:**
  - Allows for iterative development and refinement of code
  - Offers clear explanations of code logic, this makes it highly educational for developers who may inexperienced in this area.
  - Saves time by generating boilerplate code and handling complex queries on request.
- **Weaknesses:**
  - Requires precise and unambiguous prompts; ambiguous or incomplete inputs may lead to incorrect or unwanted responses by the bot.
  - Code snippets are sometimes not directly usable without adjustments in the project.

GitHub Copilot:

- **Strengths:**
  - Seamlessly integrates into IDEs like IntelliJ and VSCode, enhancing the coding experience.

    - It reduces the manual need for repetitive tasks by automatically suggesting code based on context.
    - Allows iterative improvements on existing code, making it highly intuitive for developers.
- **Weaknesses:**
    - Fairly dependent on the given code and background; Sudden or isolated prompts may not produce the most effective outcomes.

---

3. Functionality
ChatGPT:

- **Strengths:**
    - Successfully implements basic CRUD operations for RESTful endpoints and GraphQL queries/mutations.
    - Handles validation with @Valid annotations and basic input constraints.
- **Weaknesses:**
    - Security configurations lacked depth, leaving certain endpoints exposed.
    - GraphQL resolvers were functional but missed input validation, increasing the risk of data inconsistency.
    - Lacked advanced features like pagination, sorting, and filtering for larger datasets.

GitHub Copilot:

- **Strengths:**
    - Generates complete CRUD functionality for RESTful and GraphQL endpoints.
    - Role-based access control was well-implemented using Spring Security.
    - Input validation using Java Validation annotations reduces risks of invalid data.
- **Weaknesses:**
    - Advanced features such as pagination or data filtering is not automatically generated.
    - Entity exposure through APIs remains a security and design issue due to the absence of DTOs.

---

4. Testing
ChatGPT:

- **Strengths:**
    - Generates basic unit tests for repository and service layers using JUnit and Mockito.
    - Provides MockMvc tests for REST controllers, ensuring basic request/response validation.
- **Weaknesses:**
    - Lacked integration testing and comprehensive test coverage for edge cases and invalid inputs.
    - Test cases sometimes failed due to incomplete setup or missing annotations.

GitHub Copilot:

- **Strengths:**
    - Automatically generates unit tests with Mockito, mocking dependencies for isolated testing.

    - Uses H2 in-memory database for efficient test execution.
- **Weaknesses:**
    - Relies on hardcoded test data, which can lead to brittle and hard-to-maintain tests.
    - Does not include end-to-end testing, leaving gaps in real-world use case validation.

---

5. Error Handling

ChatGPT:

- **Strengths:**
    - Includes basic error handling with Optional checks in service methods.
- **Weaknesses:**
    - No centralized error-handling mechanism (e.g., @ControllerAdvice).
    - Error responses lacked detailed messages and codes, reducing API usability.

GitHub Copilot:

- **Strengths:**
    - Provides a global exception handler for consistent error handling.
    - Returns detailed validation error messages, aiding debugging.
- **Weaknesses:**
    - Does not differentiate error responses by type, leading to generic error messages.

Conclusion

Both ChatGPT and GitHub Copilot excel as AI-assisted development tools for Spring Boot projects, but their strengths lie in different areas:

- **ChatGPT** is ideal for learning and understanding, since it is able to explain the code it provided. However, the generated code often requires manual intervention to improve the quality of the code.
- **GitHub Copilot** integrates seamlessly into development environments, offering context-aware code generation that significantly reduces boilerplate coding efforts. However, it does not have the ability to explain the code it provides.

For complex production-ready applications, a combination of both tools could be the be the best. ChatGPT can be used for learning and planning while GitHub Copilot for efficiency and integration within the IDE.