**Table of Contents**

[1 Project Configuration 2](#__RefHeading___Toc232_2191818692)

[2 Test Cases 6](#__RefHeading___Toc234_2191818692)

# Project Configuration

Versioning  
  
PHP:

Symfony:

Apache:

Database:

**Recommendations from ChatGPT**

Thank you for providing the entity classes. It seems like you have a well-structured Symfony application with entities representing questions, answers, users, tags, and their relationships. The entities include various fields, relationships, and methods.

Here are a few observations and recommendations:

1. **Consistency in Naming:**
   * Ensure consistency in naming conventions across your entities. For example, some entities use camelCase (QuestionTag), while others use snake\_case (QuestionTagRepository).
2. **Validation and Constraints:**
   * Add validation constraints to your entities, especially for fields that should not be null or have specific length constraints.
   * For example, you can use the @Assert annotations to enforce constraints on fields like email or name.
3. **Entity Relationships:**
   * Consider using the cascade option in your entity relationships. For instance, in the Question entity, if deleting a question should also delete its associated answers, you might want to use cascade={"remove"} in the @OneToMany relationship.
4. **Use of Traits:**
   * The use of traits, like TimestampableEntity, is great for adding common fields like createdAt and updatedAt. Ensure that you've added these fields in the database schema.
5. **Default Values:**
   * Set default values for fields where it makes sense. For example, in the votes field of the Question entity, you've set a default value of 0, which is a good practice.
6. **Custom Methods:**
   * Review custom methods in your entities, such as getVotesString() in the Question entity. Ensure they return expected values and handle edge cases.
7. **TotpSecret Handling:**
   * Be cautious with handling totpSecret in the User entity. It's essential to keep such sensitive information secure. Make sure it's properly encrypted and handled.
8. **ToString Method:**
   * In the Question entity, the \_\_toString method returns $this->getId(), but it might be more useful to return the question's title or another descriptive field.
9. **Logging and Debugging:**
   * Consider adding logging statements or using Symfony's built-in logger for important actions or events in your entities, especially when handling votes or sensitive operations.
10. **DocBlocks:**
    * Add detailed DocBlocks to your entity methods, especially for methods like getTotpAuthenticationConfiguration() or any custom methods that might be reused.

Remember to adapt these recommendations based on your specific project requirements and coding standards.

**Recommendations for Forms**

Thank you for providing the form classes. Overall, the form classes appear well-structured and follow Symfony best practices. However, there are a few observations and recommendations:

1. **Duplicate Class Names:**
   * The class names for the AnswerFormType and QuestionFormType are the same. Make sure to use distinct names for each form type to avoid conflicts.
2. **Unused Fields:**
   * In both the AnswerFormType and QuestionFormType, you've commented out all the fields except for 'content'. If these fields are intended to be part of the form, uncomment them and ensure they are properly configured.
3. **Field Configuration:**
   * Uncomment the necessary fields in the AnswerFormType and QuestionFormType according to your application's requirements. Adjust the form fields based on what information you want to collect from users.
4. **Field Mapping:**
   * In the RegistrationFormType, you've used 'mapped' => false for 'agreeTerms' and 'plainPassword'. This is correct, as these fields are not directly mapped to the corresponding entity properties. Ensure that these fields are properly handled in your controller.
5. **Password Field:**
   * The 'autocomplete' => 'new-password' attribute in the plainPassword field is a good practice for password fields to prevent browsers from suggesting old passwords.
6. **Validation Constraints:**
   * The validation constraints in the RegistrationFormType are appropriate. Ensure that they align with your application's security and user registration requirements.
7. **Consistency:**
   * Make sure that the form fields align with your entity fields. For example, if there are additional fields in the Answer or Question entities, they should be included in the form types.
8. **Default Values:**
   * If applicable, consider setting default values for form fields that have default values in the corresponding entities.
9. **Documentation:**
   * Add comments or PHPDoc annotations to explain the purpose of each form field, especially for custom fields or fields with constraints.
10. **Testing:**
    * Ensure that you have functional tests covering the form submission and validation logic.

Make these adjustments based on your specific requirements and entity definitions.

# Test Cases

**Authentication**

1. Registration only accepts valid inputs

2. Registration will prevent registration with existing username/email

3. Log in works correctly

**Questions/Answers**

1. Creating a question works (all necessary fields are actually required, only valid input is accepted, entity is created.)

2. Loading list of questions returns the previously created result (and it matches the expected Entity class PHP's instanceof operator is a great way to check: [https://www.php.net/manual/en/language.operators.type.php](https://www.php.net/manual/en/language.operators.type.php?fbclid=IwAR1JOh7bpKamNYlvaPJRGdvi-1YTNF76Bhzpvj_cz63kPhOTfDH1gTua1sQ)) - An answer can be submitted to that question - The question's page now shows the new answer - Voting up/down works as expected