# “Ari on tour” concert ticketing website
# – Research document –

Irina-Maria David – 3371581

Student at Fontys University of Applied Sciences – S3CB04

Document version 1.1

## Questions/Sub questions

1. Understanding dependency inversion and dependency injection.

   **Main question**

   How will I make use of dependency inversion and injection in my project?

   **Sub question**

   How will I be applying the SOLID design principles in my application?

   **DOT Framework**

   Good and bad practices, Community research, Design pattern research

   I will be looking up to understanding how dependency injection, inversion, and the SOLID principles work, I will be researching on how examples of how the principles can be applied and map them to possible situations in my application. I will be analyzing good and bad practices through code examples of how the patterns should be applied and then immediately go back to my own application to be able to apply all the acquired knowledge.

   **Results and conclusion**

   After researching extensively on how each of them acts and what they do, what are their aims, I checked out the structuring of my project. I managed to understand the design patterns and how they are applied in my project better while writing on my Design document and reading from sources and reviewing examples.

   While reading from Vogel (n.d., pt.1) I understood the benefit of the dependency injection – decreased coupling, increased chances to reuse classes, the use of mocks and the fact that dependency injection helps me test classes in isolation. In my project for example it is used in the RepositoryTest classes when the customerRepository or the countryRepository are injected in the test classes.

   Spring Framework Guru (2019) depicts the Dependency inversion principle nicely in a schema, but also completes it with an explanation which lead me to understanding its applicability in my project. Dependency inversion principle helps in connecting classes together while keeping them independent at the same time. Dependency inversion says that high modules should not depend on low level modules, their communication is established through abstractions. In our case an example is the constructor of the CustomerController which communicates with the CreateUseCaseImpl through the interface CreateCustomerUseCase.

   It is a structured design approach which will be used in this project as well and it ensures that the software is modular and easy to maintain, understand, debug, and refactor – improving the design of the existing code (restructuring) without changing its original functionality (SOLID Principles Java - Javatpoint, n.d.). While reading about them and reviewing the code examples I took some time mapping the examples to situations in my own project and all of them were documented and explained in my Design document.

   **Recommendations**

I would recommend first of all reading about each of the design patterns and then easily try to recognize their behaviors into a small project. Having an example made from scratch or somewhere where the patterns apply makes it easier to visualize the way they act. It helped me a lot to check out other code examples or explanations that came along with schemas which gave me an overview of their way of working. Reading and trying to map the behaviors of the Dependency inversion as explained in Spring Framework Guru (2019) and the Dependency injection from Vogel (n.d.) with the case that you are trying to apply them on is a great chance of getting a clearer view on how they work within your own situation.

## References

Vogel, G. V. 2. 2. L.-. (n.d.). *Using dependency injection in Java - Introduction - Tutorial*. https://www.vogella.com/tutorials/DependencyInjection/article.html

Spring Framework Guru. (2019, June 5). *Dependency Inversion Principle*. https://springframework.guru/principles-of-object-oriented-design/dependency-inversion-principle/

*SOLID Principles Java - Javatpoint*. (n.d.). www.javatpoint.com. https://www.javatpoint.com/solid-principles-java