Test Layout

This document serves to give some substantiation for the approach of test layout within the ICAP project.

Single Test Project for All Microservices

1. **Centralized Management**: Easier to manage dependencies and configurations from a single location.
2. **Consistency**: Ensures uniformity in testing standards and practices across all microservices.
3. **Integrated Testing**: Facilitates testing scenarios that involve multiple services, especially for end-to-end testing.
4. **Resource Optimization**: Less overhead in terms of project maintenance, build, and deployment processes.

Separate Test Project for Each Microservice

1. **Modularity**: Aligns with the microservices philosophy of independent, self-contained units.
2. **Scalability**: As the number of microservices grows, having separate projects can make scaling and managing each service's tests more manageable.
3. **Isolation**: Reduces the risk of changes in one service's tests affecting others.
4. **Focused Testing**: Easier to develop, maintain, and execute tests that are specific to the business logic of each service.
5. **Parallel Development**: Supports concurrent development and testing by different teams without causing dependencies or conflicts.

Best Practices and Considerations

- **Project Size and Complexity**: For a small number of microservices with high inter-dependency, a single project might be more efficient. For larger, more complex landscapes with clear domain boundaries, separate projects can be beneficial.
- **Team Structure**: If different teams are responsible for different microservices, separate projects can provide autonomy and reduce coordination overhead.
- **Continuous Integration/Continuous Deployment (CI/CD)**: Consider how your CI/CD pipelines are set up. Separate projects can align better with a CI/CD approach where each microservice is built, tested, and deployed independently.
- **Tooling and Infrastructure**: Ensure that your testing infrastructure (like build servers and test environments) can support your chosen approach effectively.
- **Test Types**: Differentiate between unit/integration tests, which are typically service-specific, and end-to-end tests, which might require a more integrated approach.

Conclusion

There is no one-size-fits-all answer, and the best approach depends on your specific context, including the size and complexity of your microservice landscape, team structure, and CI/CD practices. In many cases, a hybrid approach is also viable, where you maintain separate test projects for each microservice for unit and integration tests, but have a common project for broader scope tests like end-to-end tests. This provides a balance between modularity and integrated testing capabilities.

In the case of ICAP a hybrid approach will be used. Each microservice will have its own SpecFlow test project for unit and integration tests, but there will eventually also be a common test project or end-to-end test integration within the ICAP_Client.Specs project.