Food Orders Project

Technology Learned and Used

Framework

Symfony 3.2 (The latest stable release)

External Symfony Bundles

With the assistance of Symfony community, development is easier ..

	Version Status	Why?
FOSUserBundle 2.0	Under Development	Users Management
FOSRestBundle 2.0	Under Development	Restful API
KnpPaginatorBundle	Under Development	Paginator
LexikJWTAuthenticationBundle	Under Development	JWT Layer

Other Open Source Framework/Libraries

With the assistance of open sourc community, life is more easier ..

	Version Status	Why?
Bootstrap 3.3.7	Latest Stable Version	UI Layout
PHPUnit 6.0	Latest Stable Version	Unit Testing

Version Control System

Github Repository

Security

- The application is secured from most popular security vulnerabilities (e.g CSRF attacks and SQL injections)
- Authorization settings are handled smartly, for instance: admins can't access admin area based on a remember me session.
- JWT layer secures all API communications.

Performance

- The application responds to requests in a perfect timing.
- The system is not big enough to brutally test its performance, but following the framework best practices and recommendations helps archiving high performance.

Maintenance

- I wrote Symfony Cron Jobs but it was not apply to the application yet
- One cron job could clear unused orders after a certain period of inactivity.
- The readability of the code makes it easy for any developer to fix a bug or develop a new feature.

Logic Protection and Data Validation

- Business login is protected via input validations(eg. Adding items to inactive order, or proceeding with order of zero items are not possible)

Integration With Smart Phones and Third Parties

- API layer works correctly, and deals with known formats like JSON and XML
- It is enhanced by the support of FOSRestBundle and secured with JWT

Testing

Unit Tests

Currently there is one unit test with two assertions:

- one makes sure that “current orders” displays orders of states (active, ready and waiting).
- the latter makes sure that “history” displays orders of states (delivered and complete).

Functional Tests

Currently there is one functional test with two assertions:

- one makes sure that visitors can’t access admin area.
- the latter makes sure that visitors can’t access users area.

How did the project help me develop my ability to make rational technical decisions?

- Presence comparisons with Laravel framework helped me know more in-depth details about both frameworks and know what they are good for.
- The day before the second demo, I took a decision to deliver more business value on the demo instead of perfectly completing one technical feature.
- Choosing which library/third party to rely on and integrate with is a decision.
- I made a lot of decisions from planning, designing, and developing stages.

Main Challenges Encountered

- Learning a new framework and its components in a short time
- Circular reference technical issue

How it was planned (Not 100% Accurate)

- Determining main entities and players on the system and their roles, characteristics and behaviors.
- designing the ERD
- Acting on paper, and updating the ERD if necessary.
- building the correspondent data layer (I.e. Entities, relations and repositories)
- Building the Database
- Testing the empty data layer
- setup configurations and general routes
- Test every single bundle before adding it to the system to see how compatible is it.
- Every development cycle ends with testing
- General modules and UI first.
- Building admin area

Good Things Done (Best Practices)

- High code base readability.
- High modules re-usability.
- Well organized code (e.g. Business logic entirely lives in the service layer, which leads to thin controller pattern.)

Bad Things Done (Mistakes Happened)

- Phone numbers and restaurant URL were not used at all.
- Forder was not a good naming for this class (Descriptive naming should be used instead of short naming)
- AdminService has to be StatisticsServices (Convention Broken)

Scalability and Recommended Migration Plans

Migration to NoSQL Database

- The possibility to migrate to MongoDB is fine, as DoctrineMongoDBBundle 3 is compatible with Symfony 3.*
- Why should we migrate to MongoDB is a good topic to discuss.

Turning The Product Into a Service

The application was written as a product and it was not intended to be a service, but it could be upgraded to do it.

The following are some hypothesis thoughts about the steps needed to do it.

- Backing up the data and the code base
- Redesigning the schema to do the following.
- Adding an entity (Table) for companies
- Adding the current operator(Softxpert) as the company number 1
- Adding an attribute(Column) for all entities to refer to the company, as there will not be a company-less data any more.
- Setting it to 1, as all current data belongs to it.
- Updating the code base …

This part needs to be updated and completed