Software Architecture

The web wrapper services were constructed with **annotation driven spring boot**

- Repository Location
- Core Stack
- Configuration
- Logging Configuration
- Authentication and Authorisation
- Error Handling
- Json Serialisation

Repository Location

https://git.virginaustralia.com/itdev/rest-services

Core Stack

- Spring boot
- Spring Security using Keycloak
- Jaxb
- Jackson

Configuration

configuration for the spring boot wrappers is in 3 places

- application.properties
- ws-config.properties
- logback xml files

Logging Configuration

Logging is controlled by Logback rather than log4j

logback info can be found here https://www.baeldung.com/logback

https://www.baeldung.com itself is also good for sprint related information, as is https://spring.io/guides

in this package logging is controlled by 3 files

- logback-sprint-local.xml
- logback-spring-uat.xml
- logback-sprint.xml

this logging file can **specified by** the embedded **application.properties**, or **ws-config.properties e.g.**

```
logging.config=classpath:logback-spring.xml
```

There are two log files, one for the events of the wrapper service

and one for events of the network communication between the wrapper app and the soa service

these are specified in the above 3 files

Authentication and Authorisation

- TODO

Error Handling

in the partner-points-conversion project

see com.virginaustralia.lp.exception.advices.GlobalExceptionHandler

GlobalExceptionHandler

```
@ControllerAdvice
public class GlobalExceptionHandler {

 private final Logger log = LoggerFactory.getLogger(this.getClass());

    @ResponseStatus(INTERNAL_SERVER_ERROR)
    @ExceptionHandler(Exception.class)
    @ResponseBody
    public ServiceExceptionDTO unknownError(Exception ex) {
     log.error(ex.getMessage());
     ServiceExceptionDTO serviceException =
ServiceExceptionHelper.createServiceException("An unknown error has
occurred in one of Virgin Australia's systems.", "Unknown Error");
  return serviceException;
    }

    @ExceptionHandler(SoaBusinessException.class)
    @ResponseBody
    public ResponseEntity<?> soaBusinessException(SoaBusinessException
ex) {

     log.error(ex.getMessage());
     ResponseEntity<?> responseEntity;

     ServiceExceptionDTO serviceException =
ServiceExceptionHelper.createServiceException(ex.getSoapFaultString(),
ex.getDescription());

  responseEntity = new
ResponseEntity<ServiceExceptionDTO>(serviceException,
HttpStatus.BAD_REQUEST);

  return responseEntity;
    }


....
}
```

The first Annotation to focus on is **@ControllerAdvice**, it allows exception handling across the whole application. it intercepts Exceptions.

It's utilisation reduces boiler plate code, and in general just code that would otherwise mess up the controller class.

the second important annotation is **@ExceptionHandler**. this annotation takes an Exception class as a parameter.

When this exception class is thrown, anywhere within the application, and is not implicitly caught in a method, then it will be processed in the method with this annotation

Json Serialisation

Business Object shouldn't be modified to aide in communication, that should be covered via a DTO

in Partner-points-conversion we had an instance where the business object was as expected, except a calendar attribute was being shown as an ISO string when the requirements needed a String of "yyyy-MM-dd".

to fulfil that requirement I utilised a json serialiser

The business object was PartnerPointsTransferConfigType, I extended it to a subclass called TransformedPartnerPointsTransferConfigType which overrides the getStartDate() method with the annotation of

```
@JsonSerialize(using = XmlGregorianCalendarToStringSerialiser.class)
```

on serialise it calls the serialiser and overrides the json