PED Monitor technical documentation

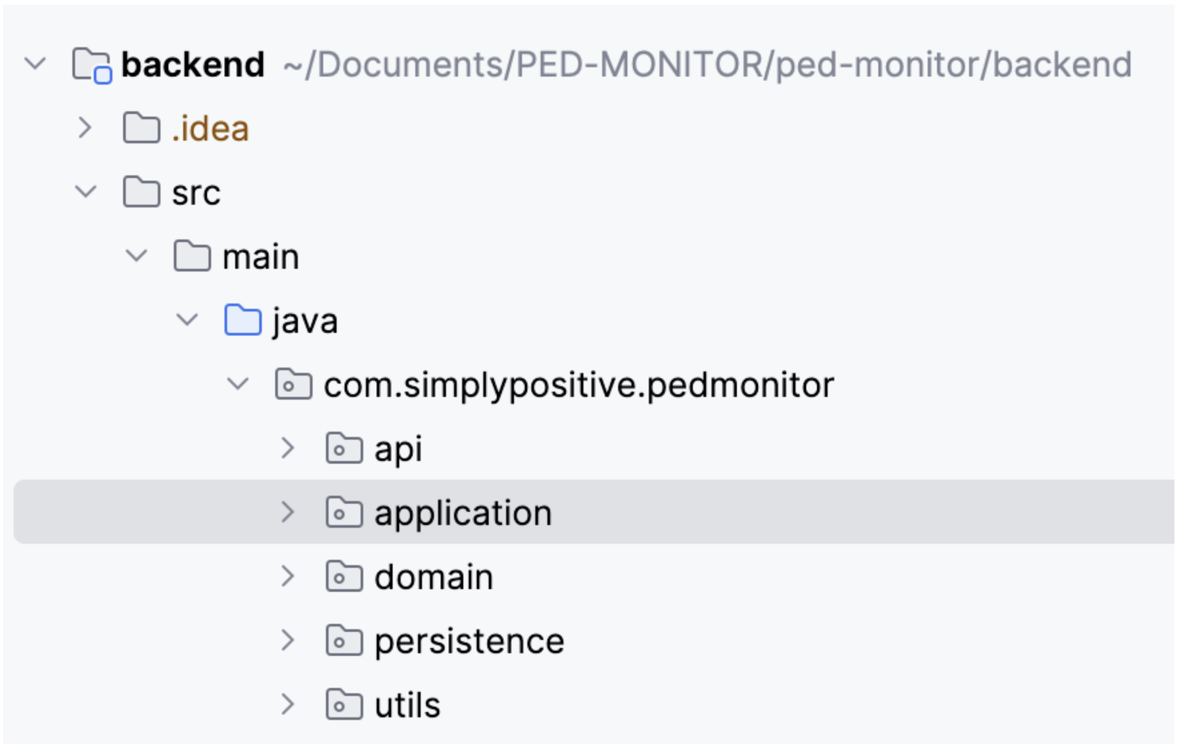
The application has two parts: backend application and frontend application (web user interface) packed together in an easily executable way.

1. Backend Application

Built in Java & Spring Boot framework exposes a REST API to manage data of Positive Energy Districts (PEDs) and Sustainability Indicators.

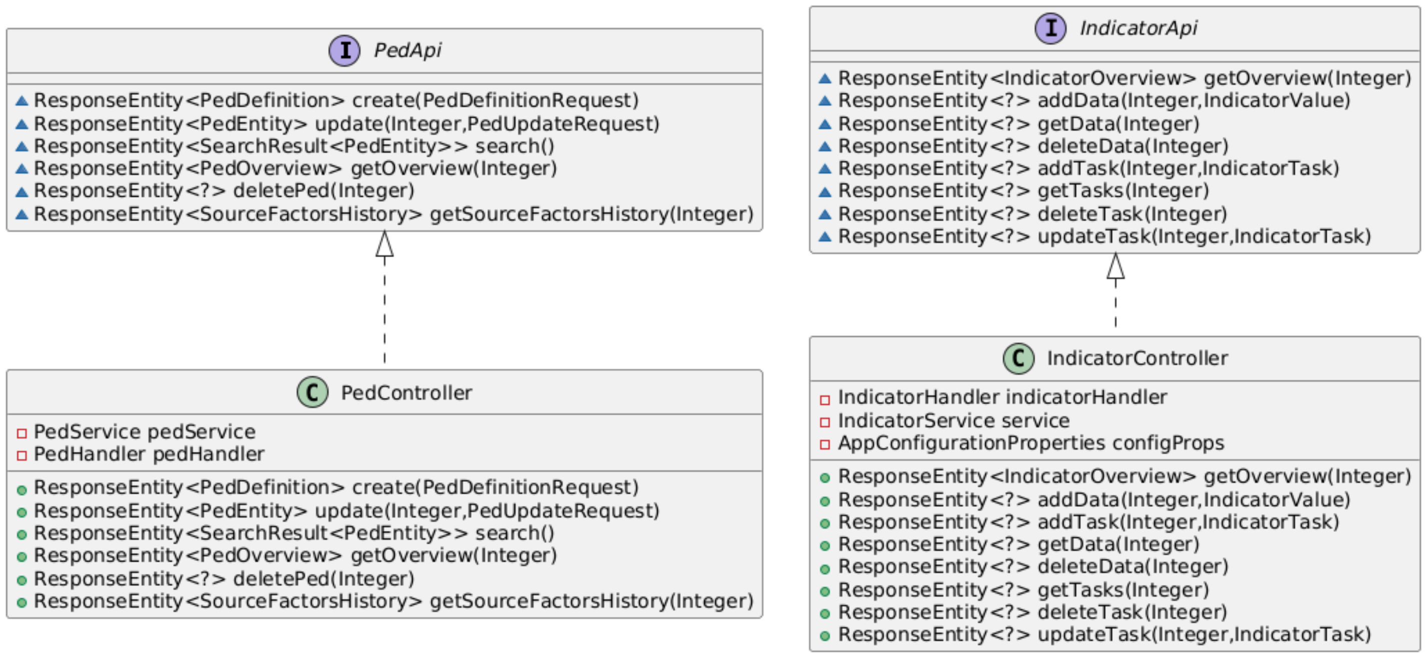
The application's structure follows the ONION architectural style:

- persistence package– contains entities or data model used for CRUD operations against the database
- domain package – contains all the business logic and associated models
- application package – coordinates the domain services
- api package – contains the REST API definition and its implementation



```
backend ~/Documents/PED-MONITOR/ped-monitor/backend
  .idea
  src
    main
      java
        com.simplypositive.pedmonitor
          api
          application
          domain
          persistence
          utils
```

API Package



PedApi
▲ ResponseEntity<PedDefinition> create(PedDefinitionRequest)
▲ ResponseEntity<PedEntity> update(Integer,PedUpdateRequest)
▲ ResponseEntity<SearchResult<PedEntity>> search()
▲ ResponseEntity<PedOverview> getOverview(Integer)
▲ ResponseEntity<?> deletePed(Integer)
▲ ResponseEntity<SourceFactorsHistory> getSourceFactorsHistory(Integer)
PedController
□ PedService pedService
□ PedHandler pedHandler
● ResponseEntity<PedDefinition> create(PedDefinitionRequest)
● ResponseEntity<PedEntity> update(Integer,PedUpdateRequest)
● ResponseEntity<SearchResult<PedEntity>> search()
● ResponseEntity<PedOverview> getOverview(Integer)
● ResponseEntity<?> deletePed(Integer)
● ResponseEntity<SourceFactorsHistory> getSourceFactorsHistory(Integer)
IndicatorApi
▲ ResponseEntity<IndicatorOverview> getOverview(Integer)
▲ ResponseEntity<?> addData(Integer,IndicatorValue)
▲ ResponseEntity<?> getData(Integer)
▲ ResponseEntity<?> deleteData(Integer)
▲ ResponseEntity<?> addTask(Integer,IndicatorTask)
▲ ResponseEntity<?> getTasks(Integer)
▲ ResponseEntity<?> deleteTask(Integer)
▲ ResponseEntity<?> updateTask(Integer,IndicatorTask)
IndicatorController
□ IndicatorHandler indicatorHandler
□ IndicatorService service
□ AppConfigurationProperties configProps
● ResponseEntity<IndicatorOverview> getOverview(Integer)
● ResponseEntity<?> addData(Integer,IndicatorValue)
● ResponseEntity<?> addTask(Integer,IndicatorTask)
● ResponseEntity<?> getData(Integer)
● ResponseEntity<?> deleteData(Integer)
● ResponseEntity<?> getTasks(Integer)
● ResponseEntity<?> deleteTask(Integer)
● ResponseEntity<?> updateTask(Integer,IndicatorTask)

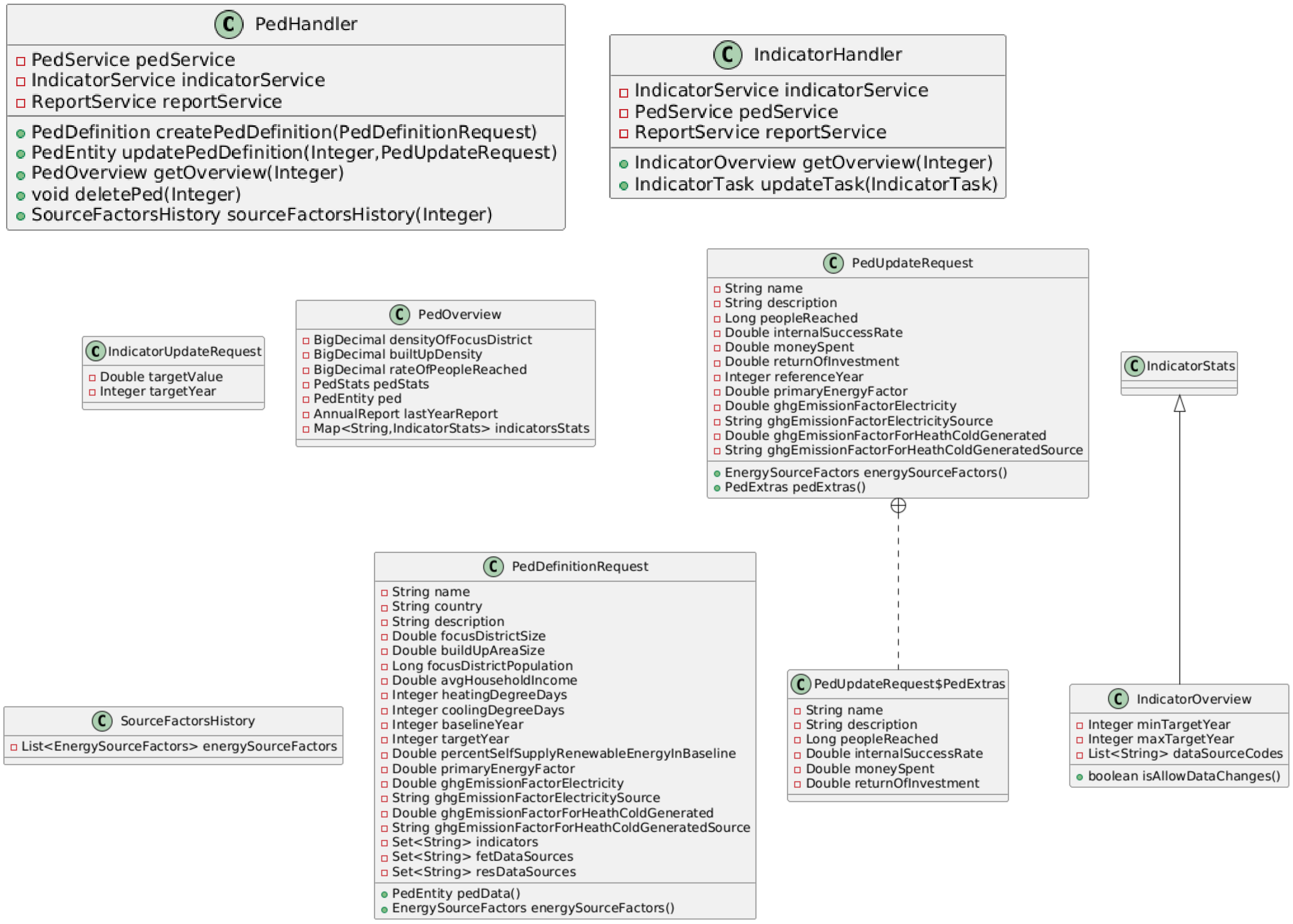
API interfaces & implementations

The APP exposes two API interfaces:

- PedApi – for actions on PED resource as: definition, update, deletion, search, overview and the history of the source factors
- IndicatorApi – for actions on Sustainability Indicator resource as:
    - retrieve/update/delete data or energy consumption for every year
    - retrieve/update/delete tasks defined in order to reach the goals and
    - obtain an overview of the indicator

Application Package

Coordinates the interaction between the API layer and domain services, where the business logic resides, providing in the same time the data in a suitable structure for the external clients/ frontend app.


PedHandler
PedService pedService
IndicatorService indicatorService
ReportService reportService
PedDefinition createPedDefinition(PedDefinitionRequest)
PedEntity updatePedDefinition(Integer,PedUpdateRequest)
PedOverview getOverview(Integer)
void deletePed(Integer)
SourceFactorsHistory sourceFactorsHistory(Integer)
IndicatorHandler
IndicatorService indicatorService
PedService pedService
ReportService reportService
IndicatorOverview getOverview(Integer)
IndicatorTask updateTask(IndicatorTask)
PedUpdateRequest
String name
String description
Long peopleReached
Double internalSuccessRate
Double moneySpent
Double returnOfInvestment
Integer referenceYear
Double primaryEnergyFactor
Double ghgEmissionFactorElectricity
String ghgEmissionFactorElectricitySource
Double ghgEmissionFactorForHeathColdGenerated
String ghgEmissionFactorForHeathColdGeneratedSource
EnergySourceFactors energySourceFactors()
PedExtras pedExtras()
IndicatorUpdateRequest
Double targetValue
Integer targetYear
PedOverview
BigDecimal densityOfFocusDistrict
BigDecimal builtUpDensity
BigDecimal rateOfPeopleReached
PedStats pedStats
PedEntity ped
AnnualReport lastYearReport
Map<String,IndicatorStats> indicatorsStats
IndicatorStats
PedDefinitionRequest
String name
String country
String description
Double focusDistrictSize
Double buildUpAreaSize
Long focusDistrictPopulation
Double avgHouseholdIncome
Integer heatingDegreeDays
Integer coolingDegreeDays
Integer baselineYear
Integer targetYear
Double percentSelfSupplyRenewableEnergyInBaseline
Double primaryEnergyFactor
Double ghgEmissionFactorElectricity
String ghgEmissionFactorElectricitySource
Double ghgEmissionFactorForHeathColdGenerated
String ghgEmissionFactorForHeathColdGeneratedSource
Set<String> indicators
Set<String> fetDataSources
Set<String> resDataSources
PedEntity pedData()
EnergySourceFactors energySourceFactors()
SourceFactorsHistory
List<EnergySourceFactors> energySourceFactors
PedUpdateRequest$PedExtras
String name
String description
Long peopleReached
Double internalSuccessRate
Double moneySpent
Double returnOfInvestment
IndicatorOverview
Integer minTargetYear
Integer maxTargetYear
List<String> dataSourceCodes
boolean isAllowDataChanges()

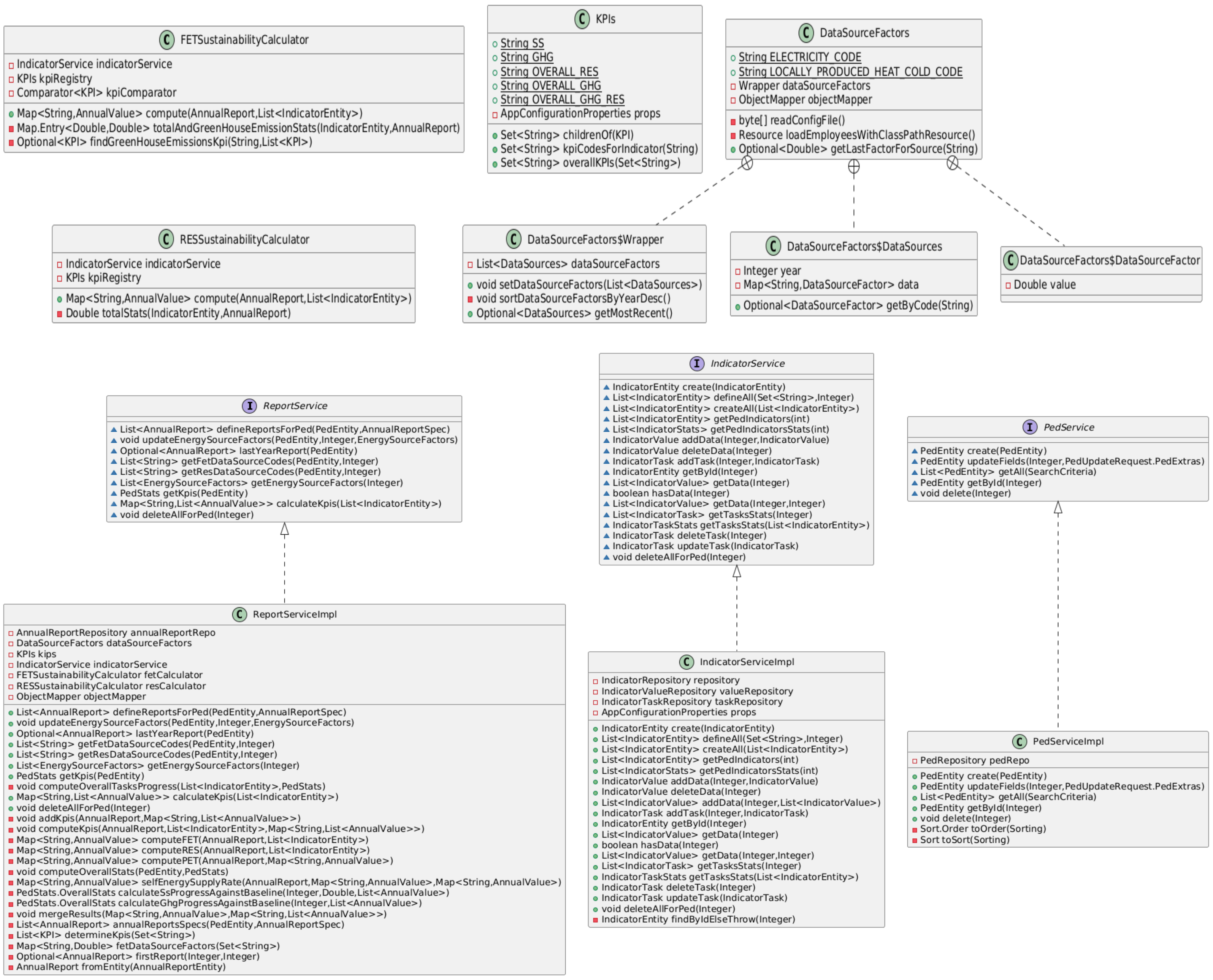
Domain Package

*models and exceptions are omitted for simplicity

Business logic context:

When a PED is defined via the web interface, the backend creates a PED entity together with all the Indicator Entities and their data-sources and identifies the KPIs needed for the chosen Indicators. The web interface presents reports (KPIs) for every year between baseline and target years and the reports are impacted by the gas emission factors for each source of data chosen for the indicators. The gas emission factors might be modified from year to year therefore the decision was to save them for each reporting year.

For now, the mapping between the KPIs and Indicators and the default emission factors for every source of data are stored within the APP **/resources** folder. In order to provide the client with new defaults the corresponding files need to be updated and a new version delivered to the clients.


FETSustainabilityCalculator
IndicatorService indicatorService
KPIs kpiRegistry
Comparator<KPI> kpiComparator
Map<String,AnnualValue> compute(AnnualReport,List<IndicatorEntity>)
Map.Entry<Double,Double> totalAndGreenHouseEmissionStats(IndicatorEntity,AnnualReport)
Optional<KPI> findGreenHouseEmissionsKpi(String,List<KPI>)
KPIs
String SS
String GHG
String OVERALL_RES
String OVERALL_GHG
String OVERALL_GHG_RES
AppConfigurationProperties props
Set<String> childrenOf(KPI)
Set<String> kpiCodesForIndicator(String)
Set<String> overallKPIs(Set<String>)
DataSourceFactors
String ELECTRICITY_CODE
String LOCALLY_PRODUCED_HEAT_COLD_CODE
Wrapper dataSourceFactors
ObjectMapper objectMapper
byte[] readConfigFile()
Resource loadEmployeesWithClassPathResource()
Optional<Double> getLastFactorForSource(String)
RESSustainabilityCalculator
IndicatorService indicatorService
KPIs kpiRegistry
Map<String,AnnualValue> compute(AnnualReport,List<IndicatorEntity>)
Double totalStats(IndicatorEntity,AnnualReport)
DataSourceFactors$Wrapper
List<DataSources> dataSourceFactors
void setDataSourceFactors(List<DataSources>)
void sortDataSourceFactorsByYearDesc()
Optional<DataSources> getMostRecent()
DataSourceFactors$DataSources
Integer year
Map<String,DataSourceFactor> data
Optional<DataSourceFactor> getByCode(String)
DataSourceFactors$DataSourceFactor
Double value
IndicatorService
IndicatorEntity create(IndicatorEntity)
List<IndicatorEntity> defineAll(Set<String>,Integer)
List<IndicatorEntity> createAll(List<IndicatorEntity>)
List<IndicatorEntity> getPedIndicators(int)
List<IndicatorStats> getPedIndicatorsStats(int)
IndicatorValue addData(Integer,IndicatorValue)
IndicatorValue deleteData(Integer)
IndicatorTask addTask(Integer,IndicatorTask)
IndicatorEntity getById(Integer)
List<IndicatorValue> getData(Integer)
boolean hasData(Integer)
List<IndicatorValue> getData(Integer,Integer)
List<IndicatorTask> getTasksStats(Integer)
IndicatorTaskStats getTasksStats(List<IndicatorEntity>)
IndicatorTask deleteTask(Integer)
IndicatorTask updateTask(IndicatorTask)
void deleteAllForPed(Integer)
ReportService
List<AnnualReport> defineReportsForPed(PedEntity,AnnualReportSpec)
void updateEnergySourceFactors(PedEntity,Integer,EnergySourceFactors)
Optional<AnnualReport> lastYearReport(PedEntity)
List<String> getFetDataSourceCodes(PedEntity,Integer)
List<String> getResDataSourceCodes(PedEntity,Integer)
List<EnergySourceFactors> getEnergySourceFactors(Integer)
PedStats getKpis(PedEntity)
Map<String,List<AnnualValue>> calculateKpis(List<IndicatorEntity>)
void deleteAllForPed(Integer)
PedService
PedEntity create(PedEntity)
PedEntity updateFields(Integer,PedUpdateRequest.PedExtras)
List<PedEntity> getAll(SearchCriteria)
PedEntity getById(Integer)
void delete(Integer)
ReportServiceImpl
AnnualReportRepository annualReportRepo
DataSourceFactors dataSourceFactors
KPIs kips
IndicatorService indicatorService
FETSustainabilityCalculator fetCalculator
RESSustainabilityCalculator resCalculator
ObjectMapper objectMapper
List<AnnualReport> defineReportsForPed(PedEntity,AnnualReportSpec)
void updateEnergySourceFactors(PedEntity,Integer,EnergySourceFactors)
Optional<AnnualReport> lastYearReport(PedEntity)
List<String> getFetDataSourceCodes(PedEntity,Integer)
List<String> getResDataSourceCodes(PedEntity,Integer)
List<EnergySourceFactors> getEnergySourceFactors(Integer)
PedStats getKpis(PedEntity)
void computeOverallTasksProgress(List<IndicatorEntity>,PedStats)
Map<String,List<AnnualValue>> calculateKpis(List<IndicatorEntity>)
void deleteAllForPed(Integer)
void addKpis(AnnualReport,Map<String,List<AnnualValue>>)
void computeKpis(AnnualReport,List<IndicatorEntity>,Map<String,List<AnnualValue>>)
Map<String,AnnualValue> computeFET(AnnualReport,List<IndicatorEntity>)
Map<String,AnnualValue> computeRES(AnnualReport,List<IndicatorEntity>)
Map<String,AnnualValue> computePET(AnnualReport,Map<String,AnnualValue>)
void computeOverallStats(PedEntity,PedStats)
Map<String,AnnualValue> selfEnergySupplyRate(AnnualReport,Map<String,AnnualValue>,Map<String,AnnualValue>)
PedStats.OverallStats calculateSsProgressAgainstBaseline(Integer,Double,List<AnnualValue>)
PedStats.OverallStats calculateGhgProgressAgainstBaseline(Integer,List<AnnualValue>)
void mergeResults(Map<String,AnnualValue>,Map<String,List<AnnualValue>>)
List<AnnualReport> annualReportsSpecs(PedEntity,AnnualReportSpec)
List<KPI> determineKpis(Set<String>)
Map<String,Double> fetDataSourceFactors(Set<String>)
Optional<AnnualReport> firstReport(Integer,Integer)
AnnualReport fromEntity(AnnualReportEntity)
IndicatorServiceImpl
IndicatorRepository repository
IndicatorValueRepository valueRepository
IndicatorTaskRepository taskRepository
AppConfigurationProperties props
IndicatorEntity create(IndicatorEntity)
List<IndicatorEntity> defineAll(Set<String>,Integer)
List<IndicatorEntity> createAll(List<IndicatorEntity>)
List<IndicatorEntity> getPedIndicators(int)
List<IndicatorStats> getPedIndicatorsStats(int)
IndicatorValue addData(Integer,IndicatorValue)
IndicatorValue deleteData(Integer)
List<IndicatorValue> addData(Integer,List<IndicatorValue>)
IndicatorTask addTask(Integer,IndicatorTask)
IndicatorEntity getById(Integer)
List<IndicatorValue> getData(Integer)
boolean hasData(Integer)
List<IndicatorValue> getData(Integer,Integer)
List<IndicatorTask> getTasksStats(Integer)
IndicatorTaskStats getTasksStats(List<IndicatorEntity>)
IndicatorTask deleteTask(Integer)
IndicatorTask updateTask(IndicatorTask)
void deleteAllForPed(Integer)
IndicatorEntity findByIdElseThrow(Integer)
PedServiceImpl
PedRepository pedRepo
PedEntity create(PedEntity)
PedEntity updateFields(Integer,PedUpdateRequest.PedExtras)
List<PedEntity> getAll(SearchCriteria)
PedEntity getById(Integer)
void delete(Integer)
Sort.Order toOrder(Sorting)
Sort toSort(Sorting)

ReportService:

- handles the annual reports for a PED
- manages emission factors updates for a reporting year

IndicatorService:

- CRUD operations on indicators
- CRUD operations on data
- CRUD operations on tasks & calculate some statistics based on the Open/Closed tasks and the planned vs actual budget

PedService: - CRUD operations on Positive Energy Districts

FETSustainabilityCalculator:

- calculcates FET KPIs based on the FET Indicators data

RESSustainabilityCalculator:

- calculates RES KPIs based on the REST Indicators data

DataSourceFactors:

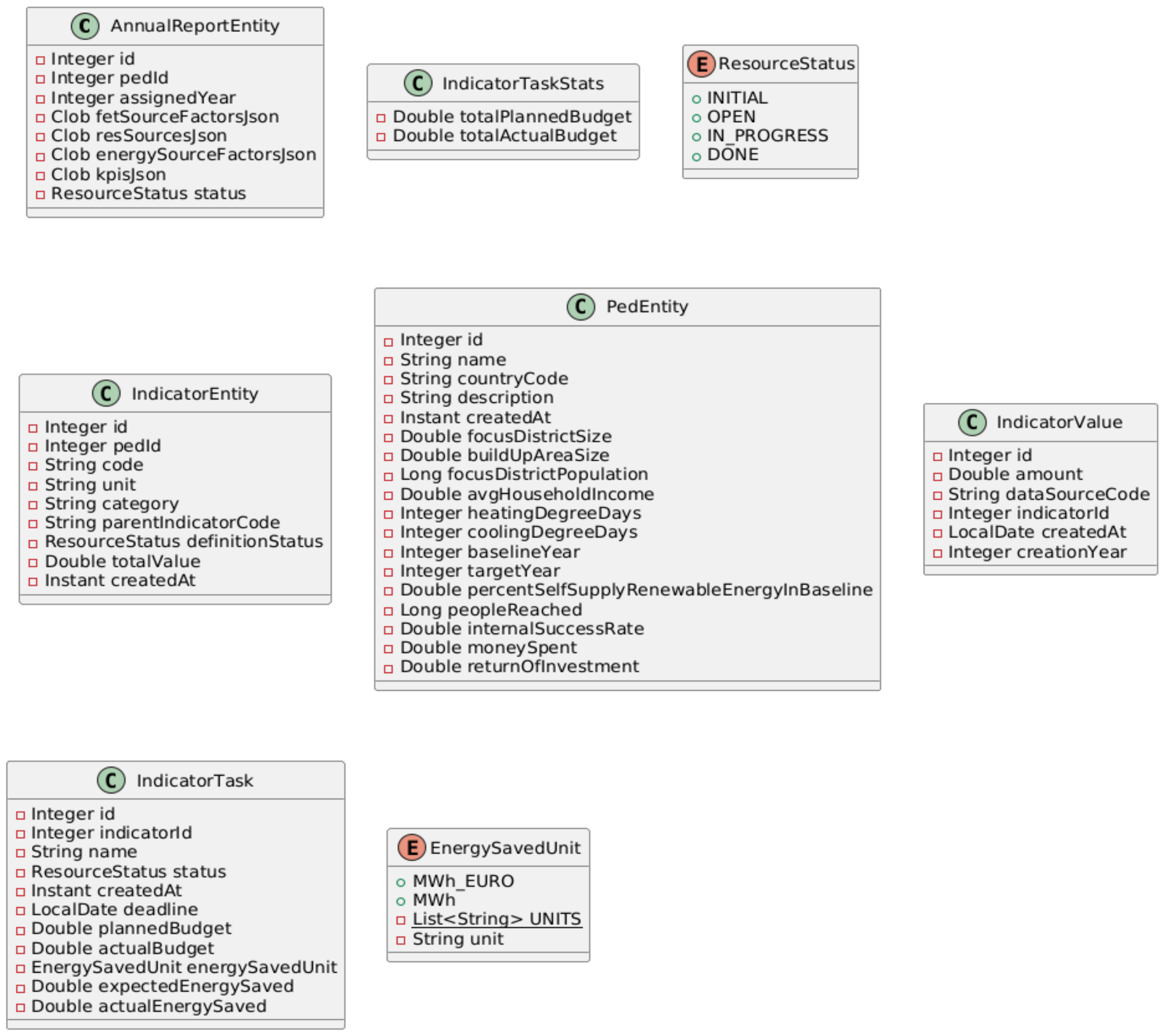
- is an abstraction over the emission factors defined in the /resources folder to easily access the values at runtime

KPIs:

- is an abstraction over the mapping between the KPIs and the Indicators defined in the /resources folder to easily access the values at runtime

Persistence Package - contains the entities used by application:

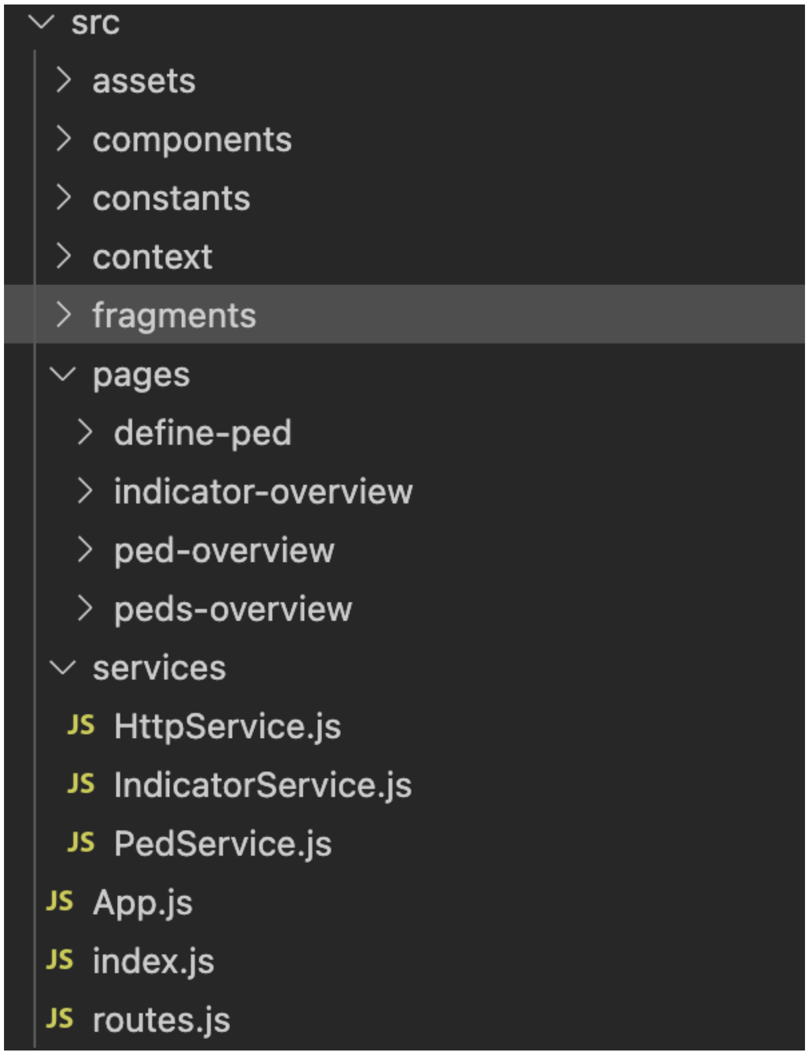
- Positive Energy District – which defines the project to be implemented between a baseline and a target year
- Sustainability Indicator – the indicators to be filled with data during the project implementation
    - The consumed or produced energy are modeled into IndicatorTask model
    - The tasks to be done in order to reduce the consumed/ increase the produced energy are modeled into IndicatorTask
- Annual Report – contains information about the KPIs to be calculated for each year between the baseline and target year together with all energy related factors for each source of data (i.e. renewable resources, energy consumption etc)
    - For the moment they are dynamically calculated for each year between baseline and target years because the user has possibility to add data for past years



Entities model

2. Frontend Application

PED Monitor frontend application is an Web Application built with ReactJS and Material UI frameworks and some more open-source frameworks as React ChartJS 2 & ChromaJS.
The structure of the application is the following:



```
src
  assets
  components
  constants
  context
  fragments
  pages
    define-ped
    indicator-overview
    ped-overview
    peds-overview
  services
    HttpService.js
    IndicatorService.js
    PedService.js
  App.js
  index.js
  routes.js
```

- /assets folders contains images and theme definition (a custom theme based on MUI was used)
- /components folder contains basic reusable elements styled according to the theme
- /fragments folder contains reusable fragments which make use of components and defines some logic on top
- /pages folder contains all the pages and the corresponding functionality/logic
- /services handles the interaction with the backend application

3. Build System

The project uses the Maven build system and contains the parent POM and 2 modules (frontend and backend).

What is doing Maven?

- parent pom.xml - declares the 2 modules to build **frontend** and **backend**.
- install the missing libraries (node) and dependencies
- build the frontend using *frontend-maven-plugin* and the frontend *static/public* directory of Spring Boot
- Both backend and frontend are packed together into a JAR file, the HTML & JS resources being served by the embedded SpringBoot Tomcat application server. Therefore, the final fat JAR that can be started/deployed in a container OR locally
- The 2 modules can be started separately during the development. The React app will use the port 3000, the Java application will use the port 8080.

Create Deliverables

- Build the APP and create a ZIP file
  - build the fat JAR by running the Maven **package** command on the parent POM file
  - Download JRE v21 from https://adoptium.net/en-GB/temurin/releases/?os=windows&arch=x64&package=jre
  - rename the downloaded JRE directory to **'jre'**
  - Create a new directory called **'ped_monitor' and ** copy the **'jre'** directory, the fat JAR and all the files from the **executables** folder
- Host the ZIP file somewhere so it can be downloaded by the **installer.cmd** (i.e. Google Drive)
- On a Windows host run the **installer.cmd**