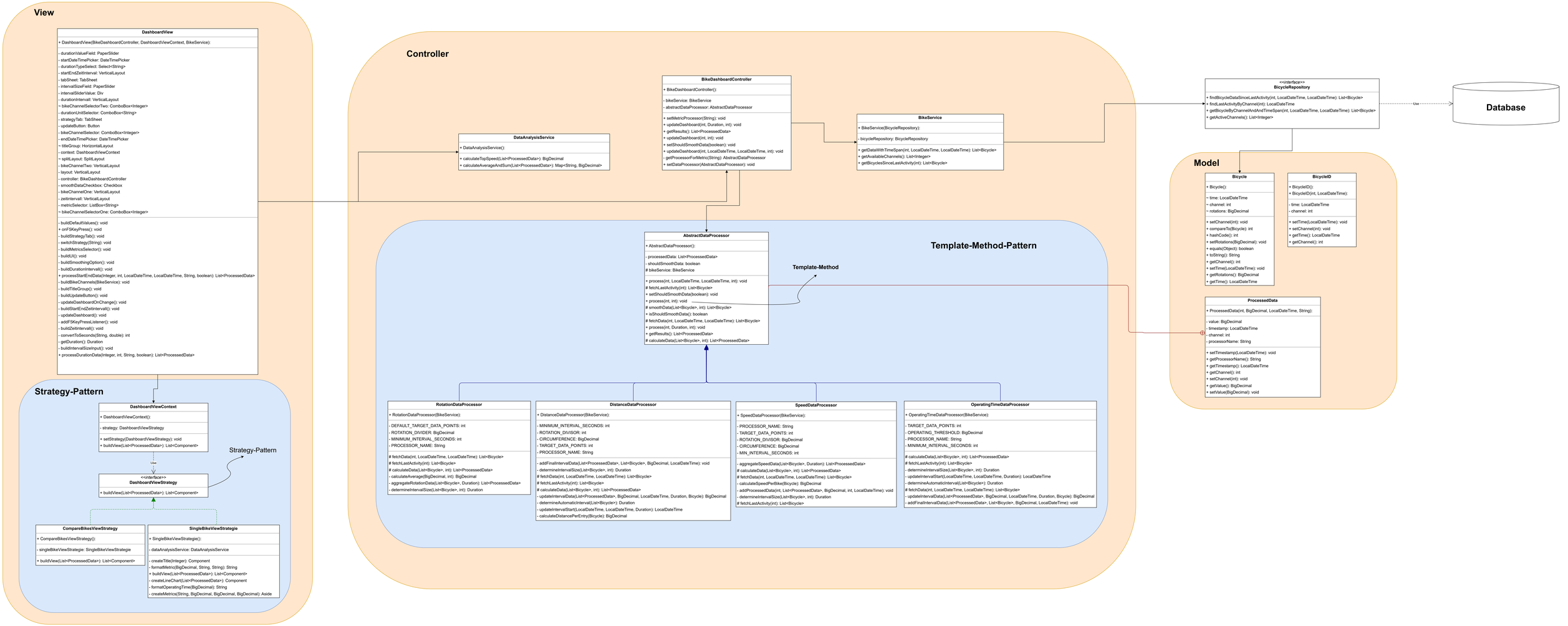
View
DashboardView
+ DashboardView(BikeDashboardController, DashboardViewContext, BikeService):
- durationValueField: PaperSlider
- startDateTimePicker: DateTimePicker
- durationTypeSelect: Select<String>
- startEndZeitInterval: VerticalLayout
- tabSheet: TabSheet
- intervalSizeField: PaperSlider
- intervallSliderValue: Div
- durationIntervall: VerticalLayout
- bikeChannelSelectorTwo: ComboBox<Integer>
- durationUnitSelector: ComboBox<String>
- strategyTab: TabSheet
- updateButton: Button
- bikeChannelSelector: ComboBox<Integer>
- endDateTimePicker: DateTimePicker
- titleGroup: HorizontalLayout
- context: DashboardViewContext
- splitLayout: SplitLayout
- bikeChannelTwo: VerticalLayout
- layout: VerticalLayout
- controller: BikeDashboardController
- smoothDataCheckbox: Checkbox
- bikeChannelOne: VerticalLayout
- zeitIntervall: VerticalLayout
- metricSelector: ListBox<String>
- bikeChannelSelectorOne: ComboBox<Integer>
- buildDefaultValues(): void
+ onF5KeyPress(): void
- buildStrategyTab(): void
- switchStrategy(String): void
- buildMetricsSelector(): void
- buildUI(): void
- buildSmoothingOption(): void
- buildDurationIntervall(): void
+ processStartEndData(Integer, int, LocalDateTime, LocalDateTime, String, boolean): List<ProcessedData>
- buildBikeChannels(BikeService): void
- buildTileGroup(): void
- buildUpdateButton(): void
- updateDashboardOnChange(): void
- buildStartEndZeitIntervall(): void
- updateDashboard(): void
- addF5KeyPressListener(): void
- buildZeitIntervall(): void
- convertToSeconds(String, double): int
- getDuration(): Duration
- buildIntervallSizeInput(): void
+ processDurationData(Integer, int, String, boolean): List<ProcessedData>
Strategy-Pattern
DashboardViewContext
+ DashboardViewContext():
- strategy: DashboardViewStrategy
+ setStrategy(DashboardViewStrategy): void
+ buildView(List<ProcessedData>): List<Component>
Use
Strategy-Pattern
<<interface>>
DashboardViewStrategy
+ buildView(List<ProcessedData>): List<Component>
CompareBikesViewStrategy
+ CompareBikesViewStrategy():
- singleBikeViewStrategie: SingleBikeViewStrategie
+ buildView(List<ProcessedData>): List<Component>
SingleBikeViewStrategie
+ SingleBikeViewStrategie():
- dataAnalysisService: DataAnalysisService
- createTitle(Integer): Component
- formatMetric(BigDecimal, String, String): String
+ buildView(List<ProcessedData>): List<Component>
- createLineChart(List<ProcessedData>): Component
- formatOperatingTime(BigDecimal): String
- createMetrics(String, BigDecimal, BigDecimal, BigDecimal): Aside
Controller
DataAnalysisService
+ DataAnalysisService():
+ calculateTopSpeed(List<ProcessedData>): BigDecimal
+ calculateAverageAndSum(List<ProcessedData>): Map<String, BigDecimal>
BikeDashboardController
+ BikeDashboardController():
- bikeService: BikeService
- abstractDataProcessor: AbstractDataProcessor
+ setMetricProcessor(String): void
+ updateDashboard(int, Duration, int): void
+ getResults(): List<ProcessedData>
+ updateDashboard(int, int): void
+ setShouldSmoothData(boolean): void
+ updateDashboard(int, LocalDateTime, LocalDateTime, int): void
- getProcessorForMetric(String): AbstractDataProcessor
+ setDataProcessor(AbstractDataProcessor): void
BikeService
+ BikeService(BicycleRepository):
- bicycleRepository: BicycleRepository
+ getDataWithTimeSpan(int, LocalDateTime, LocalDateTime): List<Bicycle>
+ getAvailableChannels(): List<Integer>
+ getBicyclesSinceLastActivity(int): List<Bicycle>
Template-Method-Pattern
AbstractDataProcessor
+ AbstractDataProcessor():
- processedData: List<ProcessedData>
- shouldSmoothData: boolean
bikeService: BikeService
+ process(int, LocalDateTime, LocalDateTime, int): void
# fetchLastActivity(int): List<Bicycle>
+ setShouldSmoothData(boolean): void
+ process(int, int): void
# smoothData(List<Bicycle>, int): List<Bicycle>
+ isShouldSmoothData(): boolean
# fetchData(int, LocalDateTime, LocalDateTime): List<Bicycle>
+ process(int, Duration, int): void
+ getResults(): List<ProcessedData>
# calculateData(List<Bicycle>, int): List<ProcessedData>
Template-Method
RotationDataProcessor
+ RotationDataProcessor(BikeService):
- DEFAULT_TARGET_DATA_POINTS: int
- ROTATION_DIVIDER: BigDecimal
- MINIMUM_INTERVAL_SECONDS: int
- PROCESSOR_NAME: String
fetchData(int, LocalDateTime, LocalDateTime): List<Bicycle>
fetchLastActivity(int): List<Bicycle>
calculateData(List<Bicycle>, int): List<ProcessedData>
- calculateAverage(BigDecimal, int): BigDecimal
- aggregateRotationData(List<Bicycle>, Duration): List<ProcessedData>
- determineIntervalSize(List<Bicycle>, int): Duration
DistanceDataProcessor
+ DistanceDataProcessor(BikeService):
- MINIMUM_INTERVAL_SECONDS: int
- ROTATION_DIVISOR: int
- CIRCUMFERENCE: BigDecimal
- TARGET_DATA_POINTS: int
- PROCESSOR_NAME: String
- addFinalIntervalData(List<ProcessedData>, List<Bicycle>, BigDecimal, LocalDateTime): void
- determineIntervalSize(List<Bicycle>, int): Duration
fetchData(int, LocalDateTime, LocalDateTime): List<Bicycle>
fetchLastActivity(int): List<Bicycle>
calculateData(List<Bicycle>, int): List<ProcessedData>
- updateIntervalData(List<ProcessedData>, BigDecimal, LocalDateTime, Duration, Bicycle): BigDecimal
- determineAutomaticInterval(List<Bicycle>): Duration
- updateIntervalStart(LocalDateTime, LocalDateTime, Duration): LocalDateTime
- calculateDistancePerEntry(Bicycle): BigDecimal
SpeedDataProcessor
+ SpeedDataProcessor(BikeService):
- PROCESSOR_NAME: String
- TARGET_DATA_POINTS: int
- ROTATION_DIVISOR: BigDecimal
- CIRCUMFERENCE: BigDecimal
- MIN_INTERVAL_SECONDS: int
- aggregateSpeedData(List<Bicycle>, Duration): List<ProcessedData>
calculateData(List<Bicycle>, int): List<ProcessedData>
fetchData(int, LocalDateTime, LocalDateTime): List<Bicycle>
- calculateSpeedPerBike(Bicycle): BigDecimal
- addProcessedData(int, List<ProcessedData>, BigDecimal, int, LocalDateTime): void
- determineIntervalSize(List<Bicycle>, int): Duration
fetchLastActivity(int): List<Bicycle>
OperatingTimeDataProcessor
+ OperatingTimeDataProcessor(BikeService):
- TARGET_DATA_POINTS: int
- OPERATING_THRESHOLD: BigDecimal
- PROCESSOR_NAME: String
- MINIMUM_INTERVAL_SECONDS: int
calculateData(List<Bicycle>, int): List<ProcessedData>
fetchLastActivity(int): List<Bicycle>
- determineIntervalSize(List<Bicycle>, int): Duration
- updateIntervalStart(LocalDateTime, LocalDateTime, Duration): LocalDateTime
- determineAutomaticInterval(List<Bicycle>): Duration
fetchData(int, LocalDateTime, LocalDateTime): List<Bicycle>
- updateIntervalData(List<ProcessedData>, BigDecimal, LocalDateTime, Duration, Bicycle): BigDecimal
- addFinalIntervalData(List<ProcessedData>, List<Bicycle>, BigDecimal, LocalDateTime): void
<<interface>>
BicycleRepository
+ findBicycleDataSinceLastActivity(int, LocalDateTime, LocalDateTime): List<Bicycle>
+ findLastActivityByChannel(int): LocalDateTime
+ getBicycleByChannelAndAndTimeSpan(int, LocalDateTime, LocalDateTime): List<Bicycle>
+ getActiveChannels(): List<Integer>
Use
Database
Model
Bicycle
+ Bicycle():
~ time: LocalDateTime
~ channel: int
~ rotations: BigDecimal
+ setChannel(int): void
+ compareTo(Bicycle): int
+ hashCode(): int
+ setRotations(BigDecimal): void
+ equals(Object): boolean
+ toString(): String
+ getChannel(): int
+ setTime(LocalDateTime): void
+ getRotations(): BigDecimal
+ getTime(): LocalDateTime
BicycleID
+ BicycleID():
+ BicycleID(int, LocalDateTime):
- time: LocalDateTime
- channel: int
+ setTime(LocalDateTime): void
+ setChannel(int): void
+ getTime(): LocalDateTime
+ getChannel(): int
ProcessedData
+ ProcessedData(int, BigDecimal, LocalDateTime, String):
- value: BigDecimal
- timestamp: LocalDateTime
- channel: int
- processorName: String
+ setTimestamp(LocalDateTime): void
+ getProcessorName(): String
+ getTimestamp(): LocalDateTime
+ getChannel(): int
+ setChannel(int): void
+ getValue(): BigDecimal
+ setValue(BigDecimal): void