IOWA STATE UNIVERSITY

SDMAY18-23

Snow Plow Mobile Data Collection and Visualization

Presented by:

Alan Peine,
Colin Heinrichs,
Evan Warych,
Michael Entin,
Zach Wilson

Advisor:

Dr. Goce Trajcevski

Website:

http://sdmay18-23.sd.ece.iastate.edu

Accepted as a demo-paper at 19th IEEE International Conference on Mobile Data Management (IEEE MDM 2018)

Outline

- Introduction
- Project Plan
  - Problem Statement
  - Conceptual Sketch
  - Functional Requirements
  - Non-functional Requirements
- System Design
  - Project Outline
  - Data Loggers
  - Data Logger Conversion Service
  - Data Storage
  - Webapp Backend
  - Webapp Frontend
  - Testing plan
  - Demo
- Conclusion

Project Plan

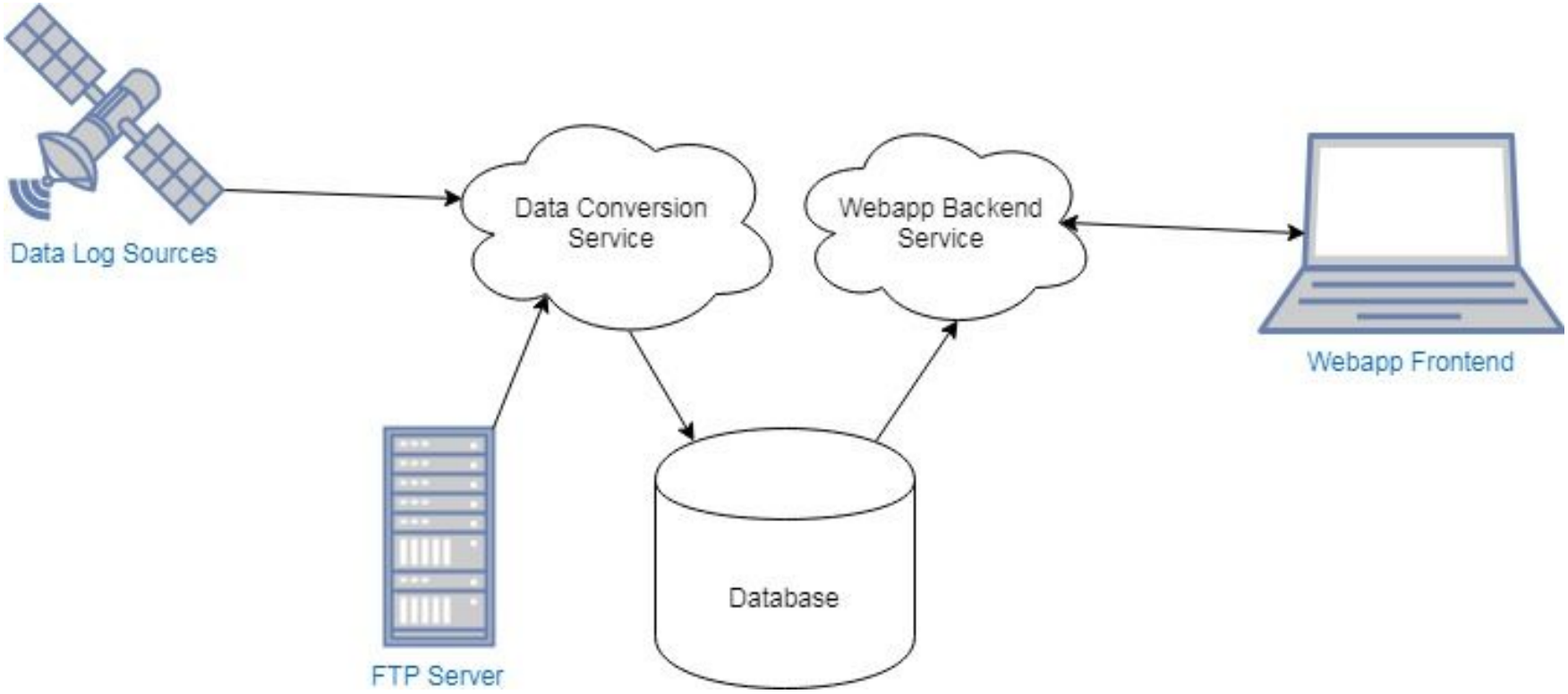
Problem Statement

Typical Data Logger Output:

```
(2016-01-24 12:39:42.522667)  can0  19FA0400    [8]  CE 00 F2 FF FF FF FF FF
(2016-01-24 12:39:42.523155)  can0        387    [8]  01 00 00 00 00 00 00 00
(2016-01-24 12:39:42.523632)  can0        187    [8]  03 00 00 00 10 00 FD FF
(2016-01-24 12:39:42.524111)  can0        186    [8]  03 04 00 00 0C 00 F9 FF
(2016-01-24 12:39:42.524592)  can0        287    [8]  00 00 00 00 59 05 50 A0
```

- Human Translation
  - Excel
  - ~30,000 lines
  - 2 Hours
  - $150
- Mobile Network
  - No Compression
  - Redundant

Conceptual Sketch



Functional Requirements

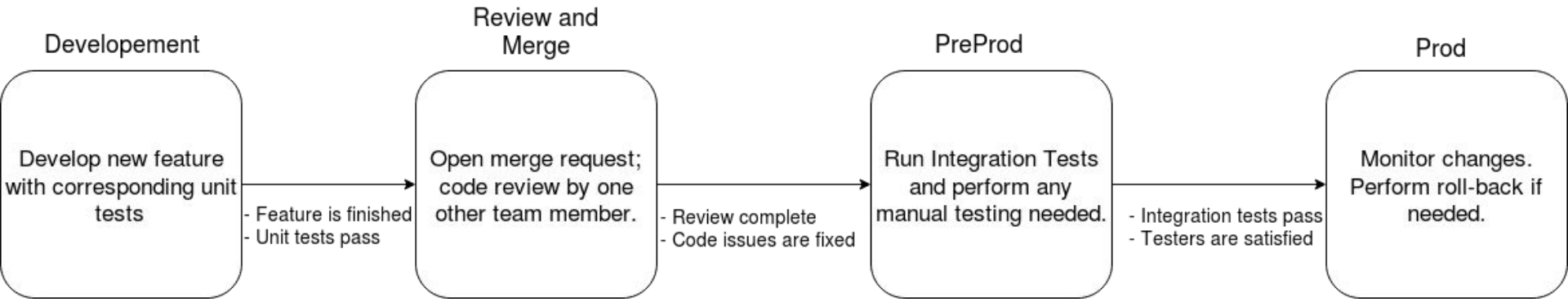
- Data Conversion
- Data Storage
- Data Access
- Data Visualization

Non-functional Requirements

- Configurability
- Performance Requirements
  - Responsiveness
  - System Throughput / Scalability

Testing Plan

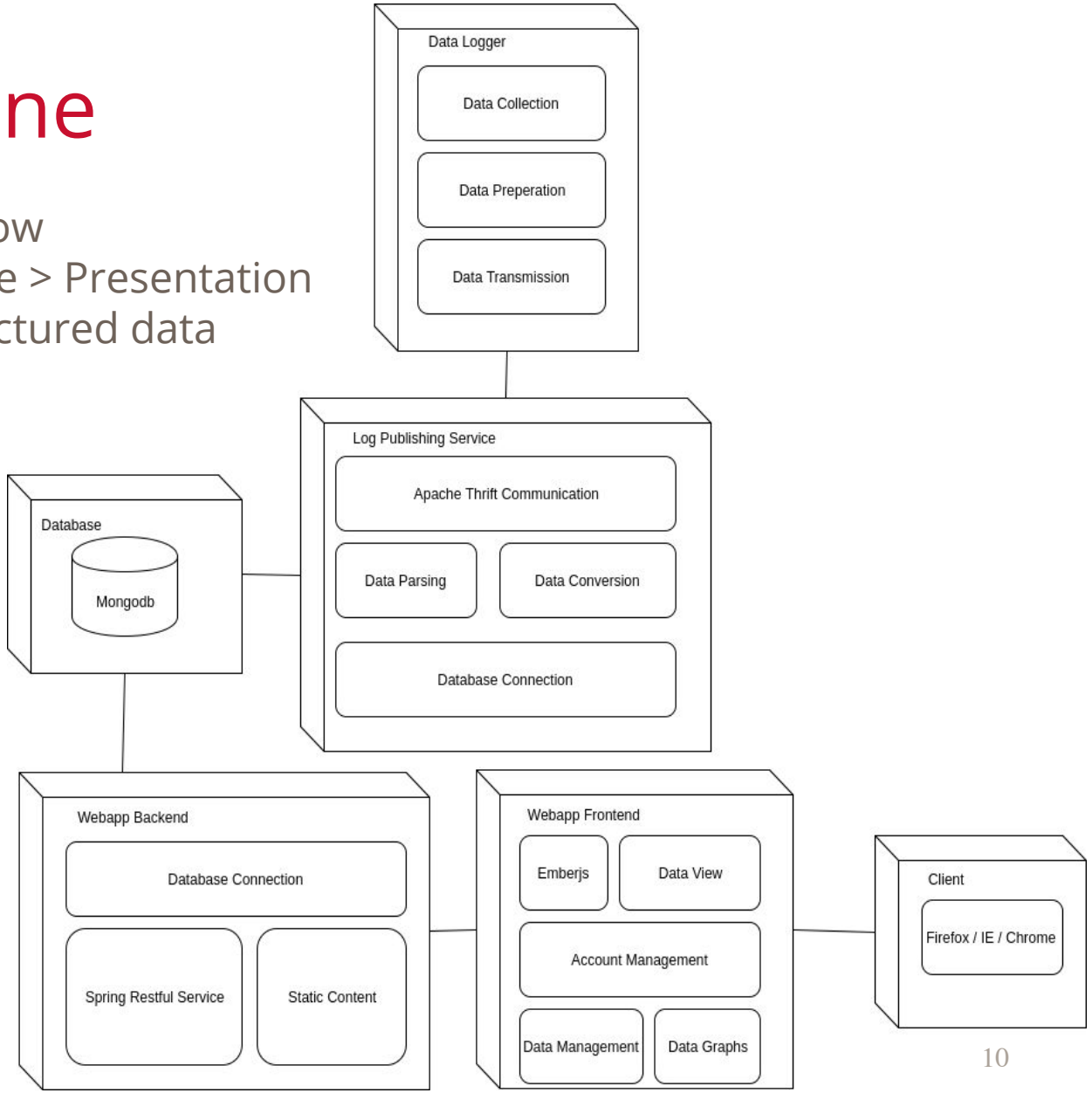
- Testing performed in stages
- Unit, Integration, and User tests
- Access to a data logger for additional testing
- Gitlab-ci for running regression tests



System Design

Project Outline

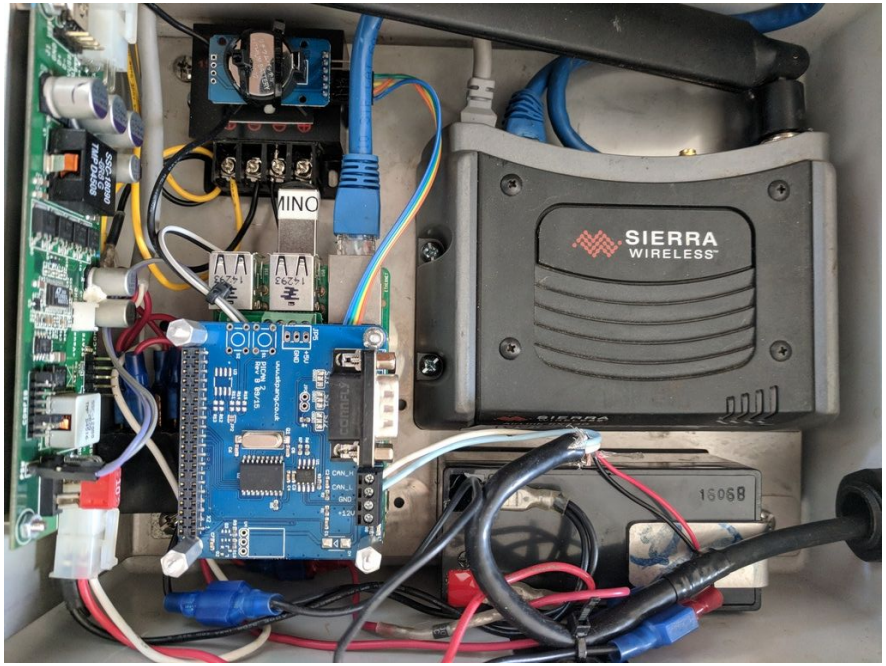
- Pipeline for data flow
- Collection > Storage > Presentation
- Conversion to Structured data



Data Loggers

- Raspberry Pi connected to CAN (J1939/CANOpen)
- Python client built to be:
  - Reliable
  - Efficient
  - Robust

Visual Representation of Data Logger

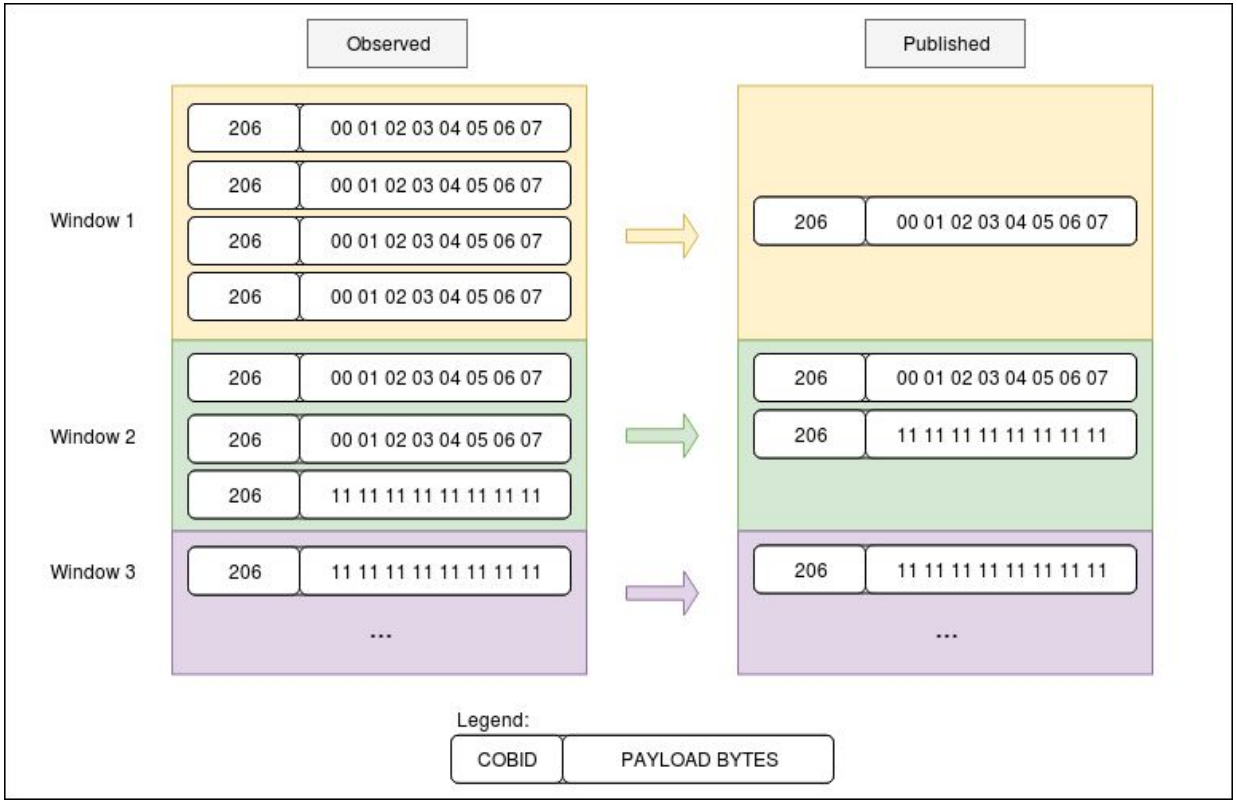


Data Loggers : Fault Tolerance

- Publish in configurable “microbatches”
- Retry on connection failure
- Store files on failed publication
  - Cron job to try again later
- Extensive exception handling for different operations

Data Loggers : Log Filtering

- Configured based on COBid of log
  - FilterAll, FilterNone, FilterUnchanged

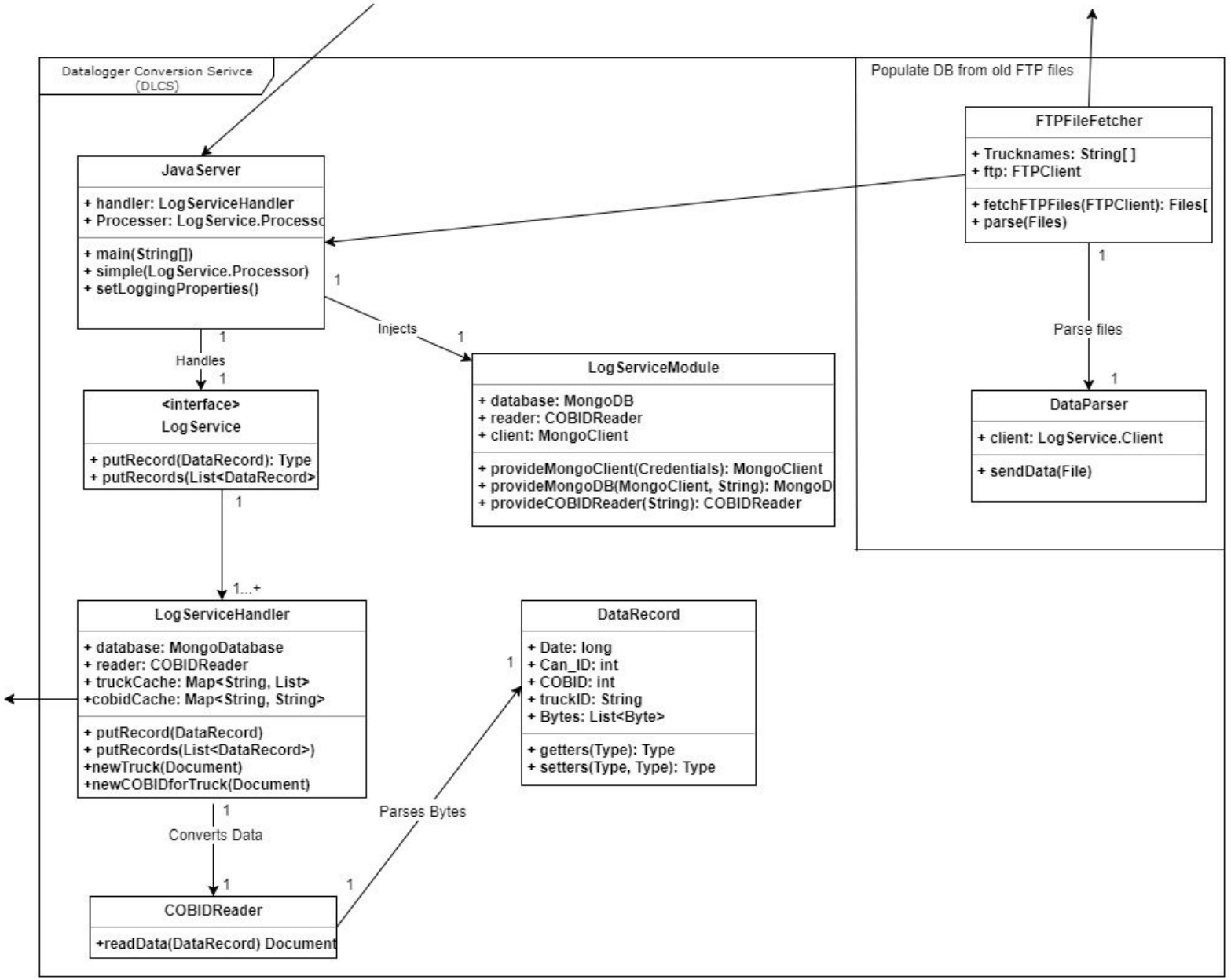


Data Log Conversion Service

- AWS EC2 RedHat Linux
- Java
- Easily Configured
- Responsibilities
  1. Communication/Injection
  2. Data Conversion
  3. Backwards Compatibility

```
],
"286" : [
  {
    "desc" : "TorqueCurrent",
    "dataLength" : 2,
    "type" : "decimal",
    "divisionAdjust" : 10
  },
  {
    "desc" : "FilteredDCCurrent",
    "dataLength" : 2,
    "type" : "decimal",
    "divisionAdjust" : 10
  },
  {
    "desc" : "FilteredDCBusVoltage",
    "dataLength" : 2,
    "type" : "decimal",
    "divisionAdjust" : 100
  },
  {
    "desc" : "DiginAll",
    "dataLength" : 2,
    "type" : "unsignedInt",
    "divisionAdjust" : 1
  }
],
```

DLCS Design


Datalogger Conversion Serivce (DLCS)
Populate DB from old FTP files
JavaServer
+ handler: LogServiceHandler
+ Processer: LogService.Processc
+ main(String[])
+ simple(LogService.Processor)
+ setLoggingProperties()
FTPFileFetcher
+ Trucknames: String[ ]
+ ftp: FTPClient
+ fetchFTPFiles(FTPClient): Files[
+ parse(Files)
Handles
Injects
Parse files
<interface>
LogService
+ putRecord(DataRecord): Type
+ putRecords(List<DataRecord>
LogServiceModule
+ database: MongoDB
+ reader: COBIDReader
+ client: MongoClient
+ provideMongoClient(Credentials): MongoClient
+ provideMongoDB(MongoClient, String): MongoD
+ provideCOBIDReader(String): COBIDReader
DataParser
+ client: LogService.Client
+ sendData(File)
LogServiceHandler
+ database: MongoDatabase
+ reader: COBIDReader
+ truckCache: Map<String, List>
+cobidCache: Map<String, String>
+ putRecord(DataRecord)
+ putRecords(List<DataRecord>)
+newTruck(Document)
+newCOBIDforTruck(Document)
DataRecord
+ Date: long
+ Can_ID: int
+ COBID: int
+ truckID: String
+ Bytes: List<Byte>
+ getters(Type): Type
+ setters(Type, Type): Type
Converts Data
Parses Bytes
COBIDReader
+readData(DataRecord) Document

Data Storage

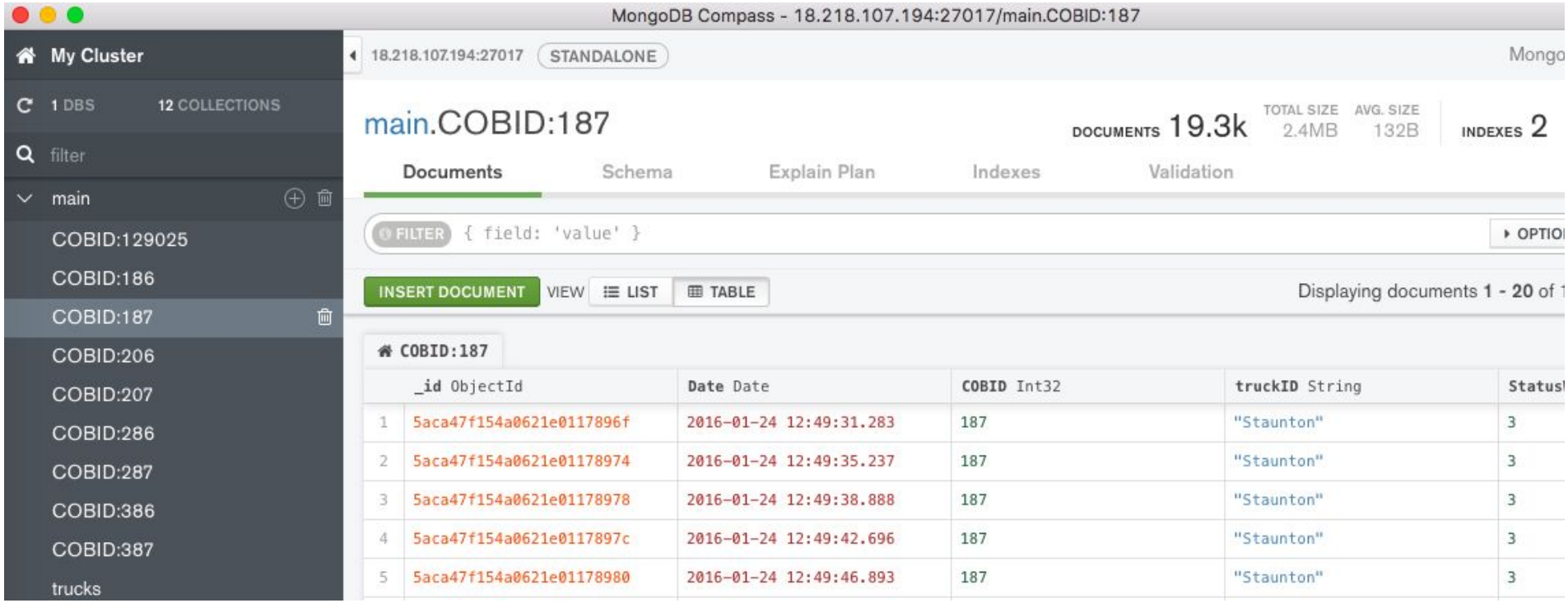
- MongoDB Database
  - Easy communication and data handling
- Scheme based on plow components

StatusWord Int32	**ActualSpeed** Int32	**MotorTemp** Int32	**DriveTemp** Int32
3	0	16	-3
3	0	16	-3
3	0	16	-3

COBID Int32	**truckID** String	**Latitude** Double	**Longitude** Double
129025	"Staunton"	39.19897	-79.0029728
129025	"Staunton"	40.19765	-79.0029728
129025	"Staunton"	38.1974414	-98.0029728
129025	"Staunton"	38.1974414	-79.0029728
129025	"Staunton"	38.1974414	-79.0029728

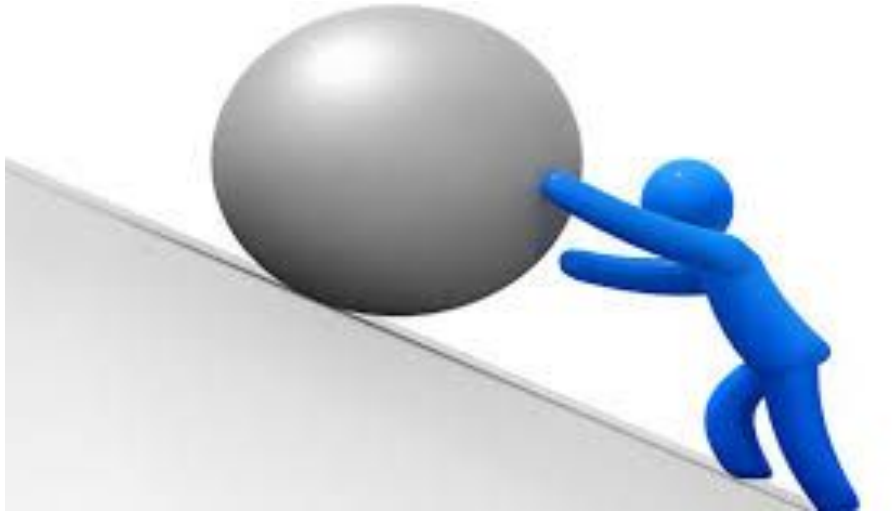
Data Storage

- Overview of Database Structure
- Indexing Improved Queries



Data Challenges Faced

- GPS Conversion
- Configuring MongoDB on an AWS instance
- Documentation and Version Compatibility



Webapp Backend

- REST API
- Java
- Spring Framework

Webapp Backend

- MongoConfig
- MongoInteraction
- TruckController
- GPSController

REST Controller & Endpoints

```
@RestController
public class TruckController extends MongoInteraction {

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


    @RequestMapping("/trucks")
    public @ResponseBody
    String indexTrucks() {

        DBCollection data = getCollection("trucks");

        return getAll(data);
    }

    @RequestMapping("/types")
    public @ResponseBody
    String indexTypes() {
        DBCollection data = getCollection("types");

        return getAll(data);
    }
```

Webapp Frontend

- Single Page Web Application
- EmberJS
- ChartJS
- Google Maps API

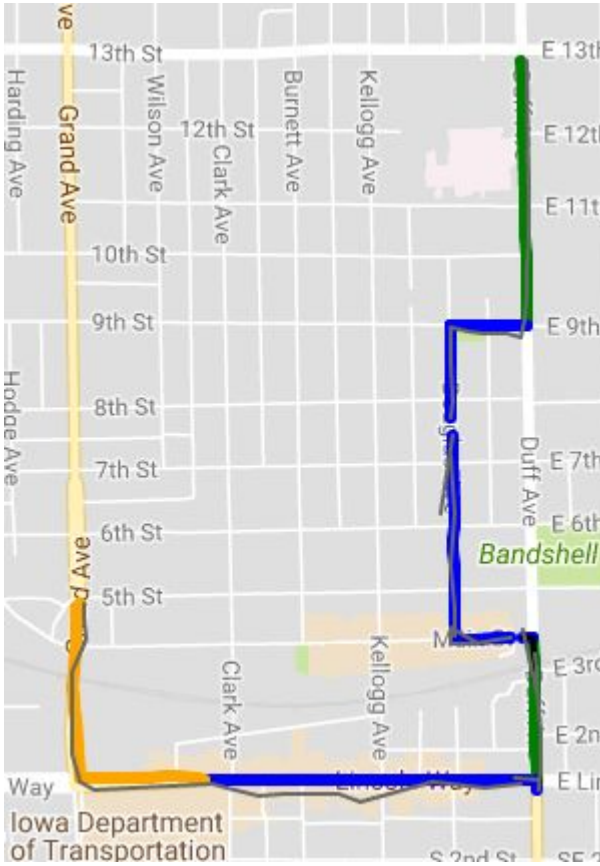


Chart.js

Demo

- Helpful Links
  - Home Page
  - Staunton FilteredDCBusVoltage
  - Staunton Location

Summary

- Simplify incoming information
- Create a web application
- 5 design aspects being used

Thank you for your time.

Any questions?