“Nodes”

Grupp 13

Tobias Edvardsson (tobedv@student.chalmers.se)
Marcus Flyckt (mflyckt@student.chalmers.se)

Table of contents

Overview

Nodes is a web application built to give users the ability to send temperature data from their temperature nodes to a simple API and display the results in an intuitive way.
Nodes features the ability to create an user, from that user you are able to create nodes and put the nodes in different collections and finally presenting the data connected to those collections in an intuitive and customizable way.

To distinguish where a temperature node is placed each node are created and named manually by the user. After the node is given a name, the node combined with the users username can be uniquely identified. This is used to report data for to specific node.

Collections are logical groupings of the nodes that are created by the user.

The idea is that the user creates one or several collections and binds the nodes to one of the collections. This give the user the ability to know where his or her nodes are placed on a larger scale.

To present the data the user can customize their front page by adding modules. A user can choose to attach one or several modules to each collection they have created. Each module presenting the data connected to that collection in a different way.

Summary of application functionality

- Simple user authentication
    - Register new user
    - Edit user settings
    - Generate api keys
- Create nodes
    - Report data to a specific node with the use of username and api key.
- Create collections
    - Put nodes in logical groupings
- Customizable frontpage
    - Attach modules to created collections
    - Easy to add more modules if desired, a modular solution

Technical design

Approach and software components:

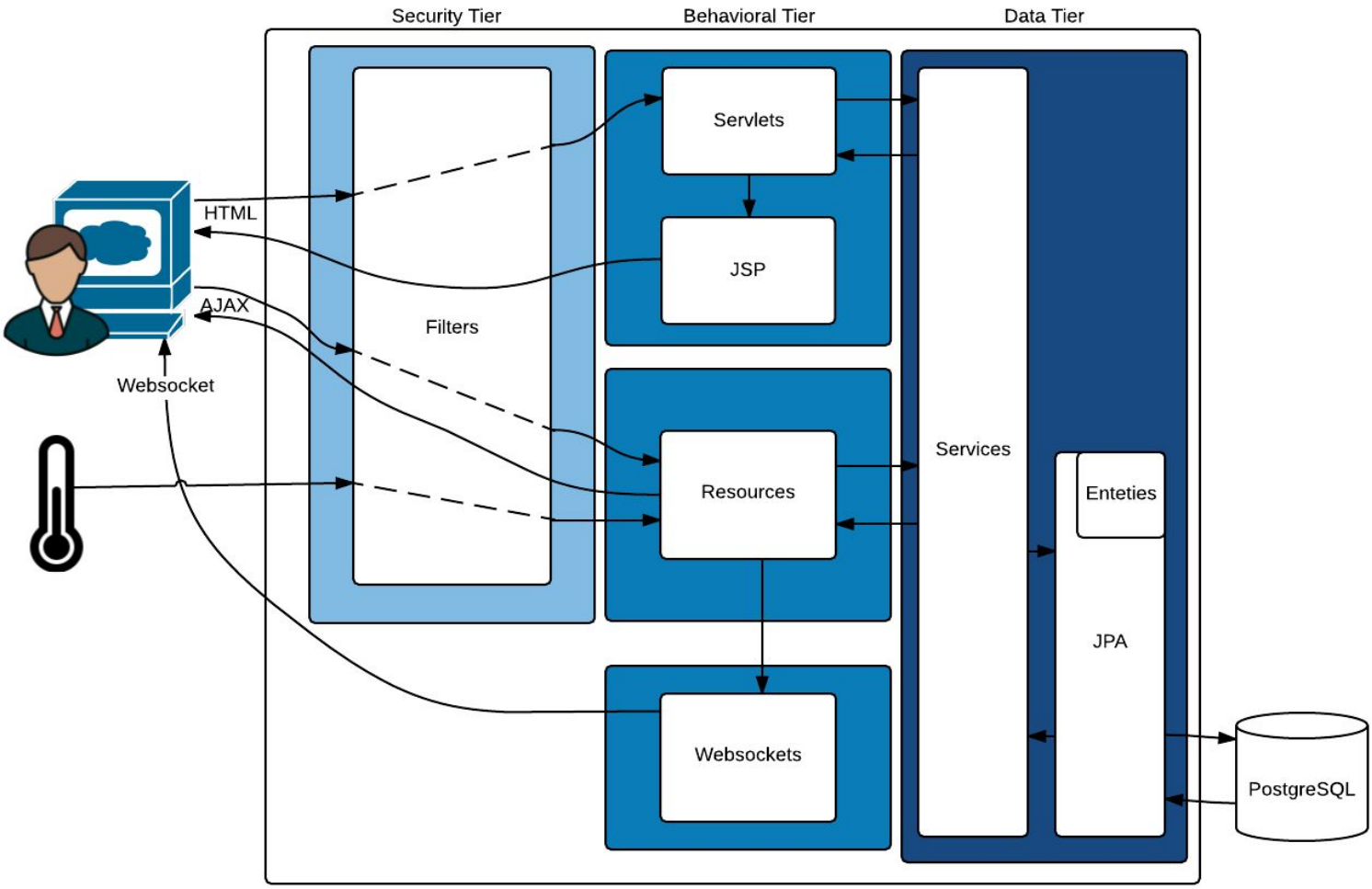
We decided after a bit of discussion to develop our application using the JavaEE ORM together with JSPs as visualisation. On top of that we also needed support for an API where our temperature nodes could report their data. Our goal was also to use websockets to visualize real time data.

Backend we created servlets to act as controllers for our services and entities, also to pass the correct data to the frontend application. We built an REST api to retrieve node information for some parts of the frontend GUI and to enable reporting in new data for the nodes. Our choice of database system was Postgres and it is located on a remote raspberry pi. Since we have experience in database design we felt the best approach was to first create a good relational scheme and to generate the entities from that. Check Appendix A for an ER diagram over the database.

The frontend is built using boostrap for easier layout and jQuery for most of the javascript. To display realtime data we are using websockets together with ajax calls. This is achieved by the backend sending a message when new data connected to either a collection or a node of the current user has arrived, this triggers the frontend javascript to fetch new data via ajax and redraw the graphs. Except for the temperature values that are being fetched via ajax calls static data such as username, is passed to the frontend via the servlets. We have also made use of the JSP taglib in our javascripts.

To simulate temperature reportings we also wrote a small python script which sends values based on a standard deviation to targeted nodes.

Tiers



Description of tiers and technical approach

Security Tier

All handling of sensitive information must pass through some kind of authentication filter. Our security tier contains two filters, one handling all requests regarding web pages and another handling the requests going to resources. The web pages filter checks that a valid session is in place and sends the client browser to the login page if that is not the case. All pages except for the login page is protected by this filter.

The resource filter makes sure that requests for resources information is authorized to read or post information. This regards both the nodes reporting data and browser ajax call retrieving data.

Behavioral Tier

This tier contains all code that makes our application what it is. Its divided into three modules, each with its own responsibility. The servlet/jsp module acts as controller and view respectively. Servlets receives browser requests, interacts with the data tier and uses JSP files to render a response back to the client.

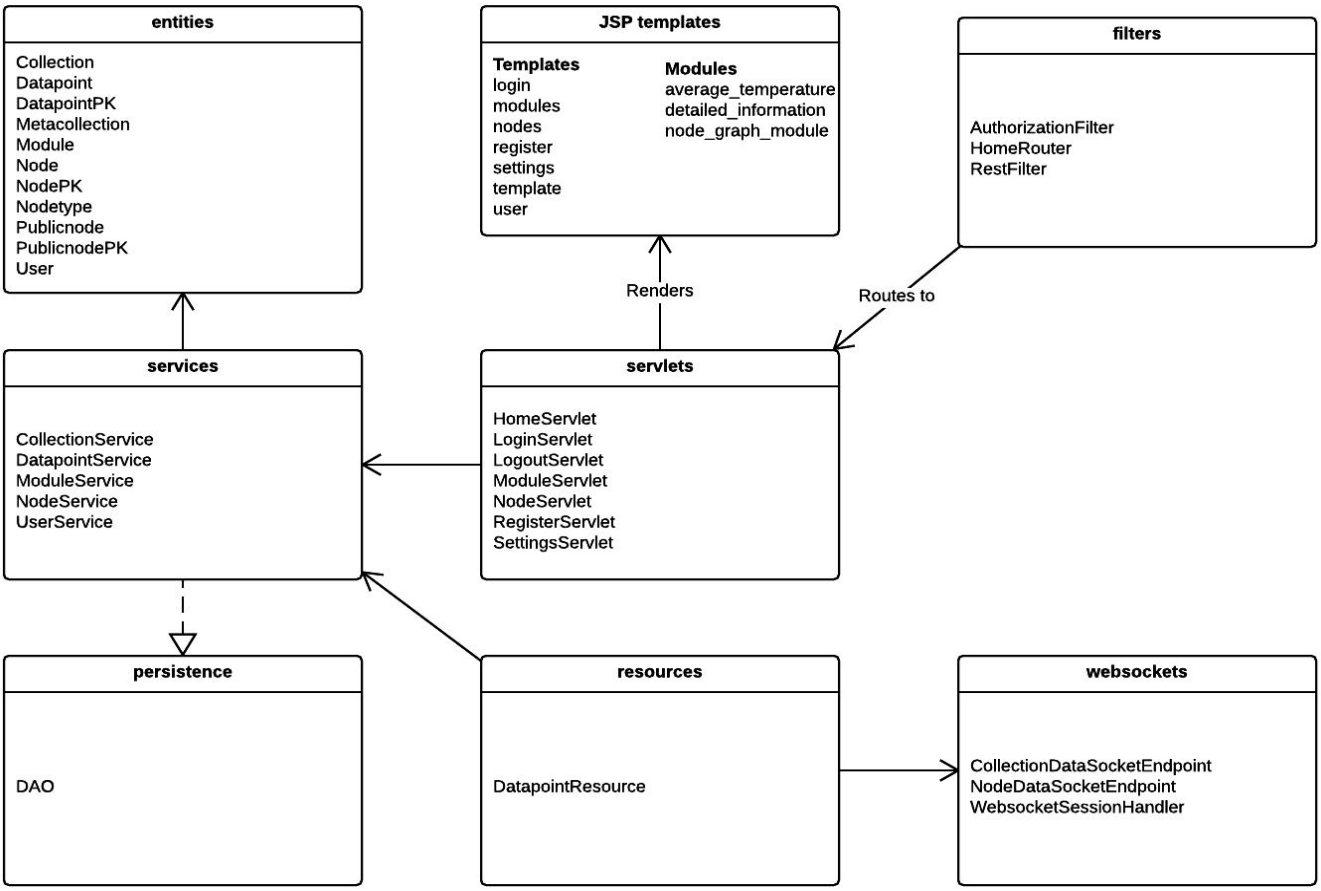
The resources module handles all requests directly regarding resources. It contains functionality for nodes to post new data and allows other clients to retrieve stored data. This module is tightly coupled with the websockets module.

Every time a node reports a new value to the resources module, the websockets module receives a call to send notifications to all websocket sessions subscribing to the node. Since these notifications does not contain any actual data, there is no need for the websockets module to be protected by the security tier.

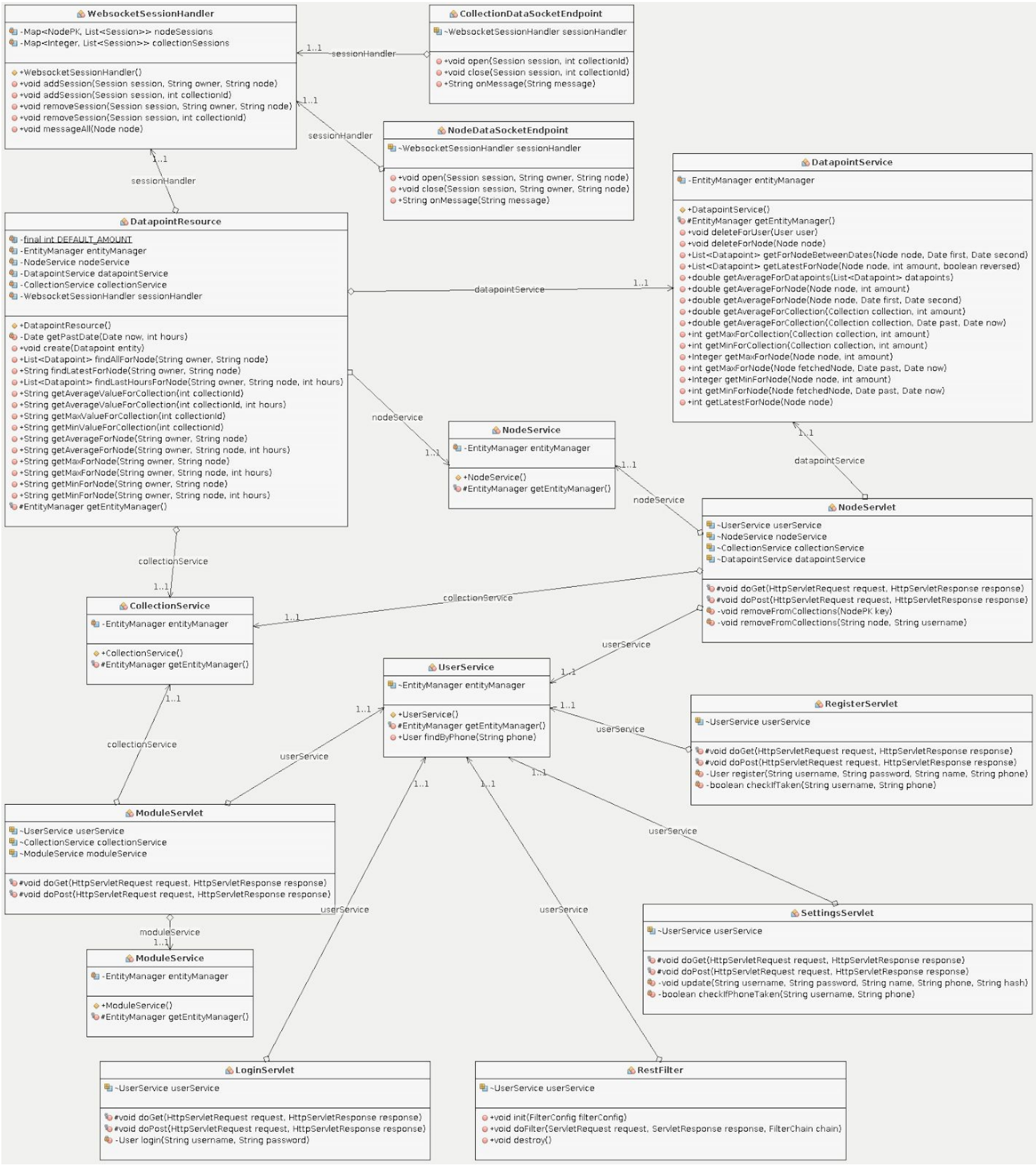
Data Tier

All persistence is handled by the data tier. This module acts as the model in our MVC architecture, and is responsible for storing and retrieving data. Services acts as an interface between the behavioral and data tier, and contains all logic regarding how to send data to and receive data from the Java EE Persistance API which in turn directly interacts with the database.

Packet UML

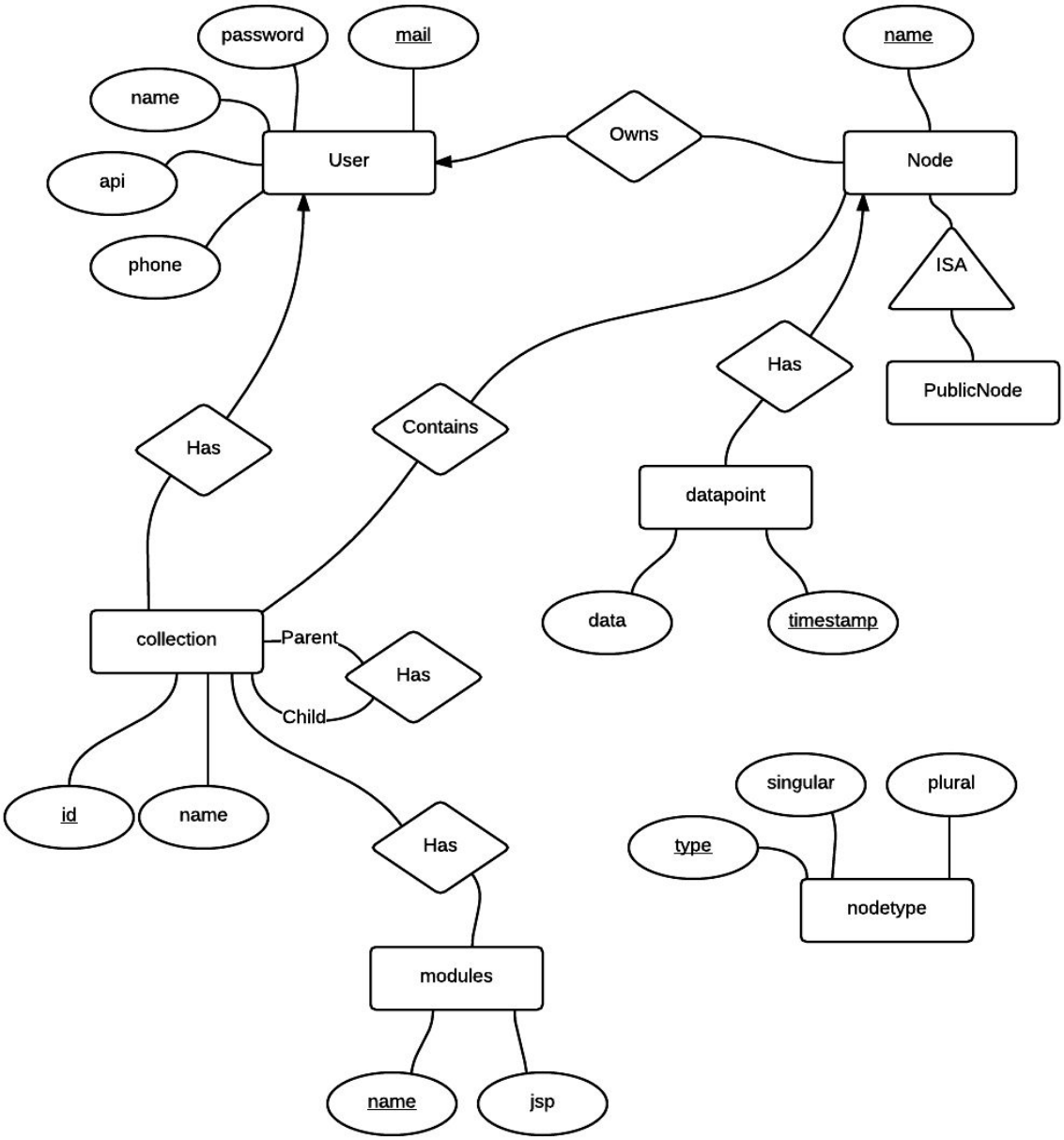

entities
Collection
Datapoint
DatapointPK
Metacollection
Module
Node
NodePK
Nodetype
Publicnode
PublicnodePK
User
JSP templates
Templates
login
modules
nodes
register
settings
template
user
Modules
average_temperature
detailed_information
node_graph_module
filters
AuthorizationFilter
HomeRouter
RestFilter
Renders
Routes to
services
CollectionService
DatapointService
ModuleService
NodeService
UserService
servlets
HomeServlet
LoginServlet
LogoutServlet
ModuleServlet
NodeServlet
RegisterServlet
SettingsServlet
persistence
DAO
resources
DatapointResource
websockets
CollectionDataSocketEndpoint
NodeDataSocketEndpoint
WebsocketSessionHandler

Class UML


WebsocketSessionHandler
-Map<NodePK, List<Session>> nodeSessions
-Map<Integer, List<Session>> collectionSessions
+WebsocketSessionHandler()
+void addSession(Session session, String owner, String node)
+void addSession(Session session, int collectionId)
+void removeSession(Session session, String owner, String node)
+void removeSession(Session session, int collectionId)
+void messageAll(Node node)
CollectionDataSocketEndpoint
~WebsocketSessionHandler sessionHandler
+void open(Session session, int collectionId)
+void close(Session session, int collectionId)
+String onMessage(String message)
NodeDataSocketEndpoint
~WebsocketSessionHandler sessionHandler
+void open(Session session, String owner, String node)
+void close(Session session, String owner, String node)
+String onMessage(String message)
DatapointService
-EntityManager entityManager
+DatapointService()
#EntityManager getEntityManager()
+void deleteForUser(User user)
+void deleteForNode(Node node)
+List<Datapoint> getForNodeBetweenDates(Node node, Date first, Date second)
+List<Datapoint> getLatestForNode(Node node, int amount, boolean reversed)
+double getAverageForDatapoints(List<Datapoint> datapoints)
+double getAverageForNode(Node node, int amount)
+double getAverageForNode(Node node, Date first, Date second)
+double getAverageForCollection(Collection collection, int amount)
+double getAverageForCollection(Collection collection, Date past, Date now)
+int getMaxForCollection(Collection collection, int amount)
+int getMinForCollection(Collection collection, int amount)
+Integer getMaxForNode(Node node, int amount)
+int getMaxForNode(Node fetchedNode, Date past, Date now)
+Integer getMinForNode(Node node, int amount)
+int getMinForNode(Node fetchedNode, Date past, Date now)
+int getLatestForNode(Node node)
DatapointResource
-final int DEFAULT_AMOUNT
-EntityManager entityManager
-NodeService nodeService
-DatapointService datapointService
-CollectionService collectionService
-WebsocketSessionHandler sessionHandler
+DatapointResource()
-Date getPastDate(Date now, int hours)
+void create(Datapoint entity)
+List<Datapoint> findAllForNode(String owner, String node)
+String findLatestForNode(String owner, String node)
+List<Datapoint> findLastHoursForNode(String owner, String node, int hours)
+String getAverageValueForCollection(int collectionId)
+String getAverageValueForCollection(int collectionId, int hours)
+String getMaxValueForCollection(int collectionId)
+String getMinValueForCollection(int collectionId)
+String getAverageForNode(String owner, String node)
+String getAverageForNode(String owner, String node, int hours)
+String getMaxForNode(String owner, String node)
+String getMaxForNode(String owner, String node, int hours)
+String getMinForNode(String owner, String node)
+String getMinForNode(String owner, String node, int hours)
#EntityManager getEntityManager()
NodeService
-EntityManager entityManager
+NodeService()
#EntityManager getEntityManager()
NodeServlet
~UserService userService
~NodeService nodeService
~CollectionService collectionService
~DatapointService datapointService
#void doGet(HttpServletRequest request, HttpServletResponse response)
#void doPost(HttpServletRequest request, HttpServletResponse response)
-void removeFromCollections(NodePK key)
-void removeFromCollections(String node, String username)
CollectionService
-EntityManager entityManager
+CollectionService()
#EntityManager getEntityManager()
UserService
~EntityManager entityManager
+UserService()
#EntityManager getEntityManager()
+User findByPhone(String phone)
RegisterServlet
~UserService userService
#void doGet(HttpServletRequest request, HttpServletResponse response)
#void doPost(HttpServletRequest request, HttpServletResponse response)
-User register(String username, String password, String name, String phone)
-boolean checkIfTaken(String username, String phone)
ModuleServlet
~UserService userService
~CollectionService collectionService
~ModuleService moduleService
#void doGet(HttpServletRequest request, HttpServletResponse response)
#void doPost(HttpServletRequest request, HttpServletResponse response)
ModuleService
-EntityManager entityManager
+ModuleService()
#EntityManager getEntityManager()
SettingsServlet
~UserService userService
#void doGet(HttpServletRequest request, HttpServletResponse response)
#void doPost(HttpServletRequest request, HttpServletResponse response)
-void update(String username, String password, String name, String phone, String hash)
-boolean checkIfPhoneTaken(String username, String phone)
LoginServlet
~UserService userService
#void doGet(HttpServletRequest request, HttpServletResponse response)
#void doPost(HttpServletRequest request, HttpServletResponse response)
-User login(String username, String password)
RestFilter
~UserService userService
+void init(FilterConfig filterConfig)
+void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
+void destroy()
sessionHandler
1..1
datapointService
nodeService
collectionService
userService
moduleService

Appendix A

SQL ER DIAGRAM & SCHEMA



SQL SCHEMA (For Postgres):

```
CREATE TABLE users(
 mail VARCHAR(50) PRIMARY KEY,
 password VARCHAR(50),
 name VARCHAR(50),
 api VARCHAR(50),
 phone VARCHAR(50) UNIQUE
);

CREATE TABLE nodes (
 name VARCHAR(50),
 owner VARCHAR(50) REFERENCES users(mail),
 PRIMARY KEY(name, owner)
);

CREATE TABLE nodetypes(
 type VARCHAR(50),
 singular VARCHAR(50),
 plural VARCHAR(50),
 PRIMARY KEY (type)
);

CREATE TABLE publicnode(
 node VARCHAR(50),
 owner VARCHAR(50),
 FOREIGN KEY (node, owner) REFERENCES nodes(name,owner),
 PRIMARY KEY (node, owner)
);

CREATE TABLE datapoints (
 node VARCHAR(50),
 owner VARCHAR(50),
 time timestamp,
 data INTEGER not null,
 PRIMARY KEY(node, owner, time),
FOREIGN KEY(node, owner) REFERENCES nodes(name, owner)
);

CREATE TABLE collections(
 id SERIAL primary key,
 owner VARCHAR(50) references users(mail),
 name VARCHAR(50)
);

CREATE TABLE metacollections(
```

```
 parent INTEGER references collections(id) primary key,
 child INTEGER references collections(id)
);

CREATE TABLE collectionnodes(
 node VARCHAR(50),
 owner VARCHAR(50),
 collection integer references collections(id),
 FOREIGN KEY (node, owner) REFERENCES nodes(name,owner),
 PRIMARY KEY (node, owner, collection)
);

CREATE TABLE modules(
        name VARCHAR(50) PRIMARY KEY,
        jsp VARCHAR(50)
);

CREATE TABLE modulecollections(
        module VARCHAR(50) REFERENCES modules(name),
        collection INTEGER REFERENCES collections(id),
        PRIMARY KEY(module, collection)
);
```