ECSE 321

Introduction to Software Engineering

Deliverable 2

Design Specification and Initial Prototype

February 25, 2018

Group 11

Alexander Harris - 260688155

Abbas Yadollahi - 260680343

Filip Bernevec - 260689062

Yunus Can Cukran - 260669715

Gareth Peters - 260678626

System Architecture **3**

Systems & Subsystems **3**

Detailed System Design **4**

<<Persistence>> SQLiteJDBC 4

<<Entity>> Tree 5

<<Entity>> Location 5

<<Entity>> Municipality 5

<<Entity>> User 5

<<Entity>> Species 5

<<Entity>> SurveyReport 6

<<Entity>> SustainabilityCalculator 6

<<REST>> TreepleRestController 6

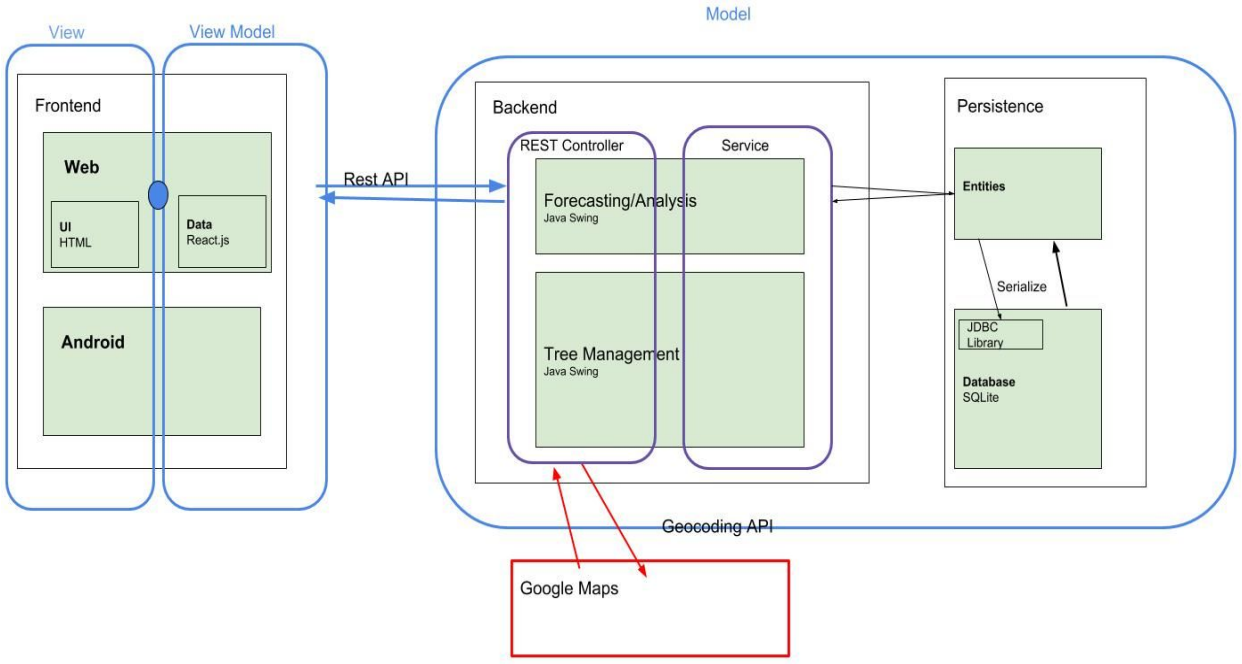
<<Controller>> TreepleService 6

Class Diagram 7

Sequence-Level Diagram **8**

Work Plan **10**

System Architecture



Systems & Subsystems

Backend

The backend consists of the two main services we provide: tree management and forecast. Both services communicate with the Frontend and Google Maps via a REST Controller. Between the controller and the front end, DTO representations of the business entities are used and communication is done by HTTP requests. Communication between the controller and Google Maps is done by HTTP requests, and the JSON objects that Google Maps provide are used.

Tree Management: Manages the cutting/marking/surveying of trees in the city.
Forecasting: Calculates benefits and different possible outcomes for city planning choices.

Persistence

Business entities, which are Java objects, are manipulated via Backend services. The entities are converted with the help of SQLite JDBC library.

Entities: The domain model. Java objects generated by UMPLE
Database: SQLite

Frontend

The frontend consists of a Web app and an Android App. All communications with the REST controller are done via HTTP GET and POST requests.

Web App
Android App

Google Maps

Used for displaying trees, and getting geocoding data for selected trees. Only communicates with the backend REST controller via HTTP requests. Returns JSON objects.

Design Choices:

On the highest level, A Model-View-Model View architecture is implemented. If we look within the backend and persistence components, we can see a layer structure where the services are separated by functionality. Within the Frontend, the Android app itself follows an MVC pattern, and the web app has tightly coupled view and viewmodel aspects. Google Maps and the backend themselves form a layer, which is a single point of communications. This control may help us optimize Google API calls and avoid passing Google Maps' daily 25,000 request limit.

Generally we have tried to limit communication surfaces to a few centralized interfaces between layers for security and reliability. We avoided unnecessary access by unnecessary components.

Detailed System Design

<<Persistence>> SQLiteJDBC

Saves and loads all necessary information in an SQL database from the backend. It is able to save all sustainability attributes of each tree, as well as tree location, address, municipality, land, species, status, height, and diameter. Trees may also be be deleted from

the database. Municipalities, lands, species, survey reports, and users may be saved, loaded or deleted from the database.

<<Entity>> Tree

The tree entity contains attributes height, diameter, address, date planted, land, municipality, status, ownership, a unique id, species, location, and list of survey reports. Ownership, status and land are all enums in this Tree class. The ownership enum has only two entries which are Private and Public. The Status of a tree can be either Planted, Diseased, MarkedForCutdown, or Cutdown. The types of Lands are parklands, residential lands, institutional land, and municipal land. This class has only getters and setters for each attribute.

<<Entity>> Location

The location entity has three attributes: longitude, latitude and a unique id. This entity is used to specify the location of a tree on the map. This class has only getters and setters for each attribute.

<<Entity>> Municipality

The municipality entity has attributes for its name, the total trees within the municipality, a list of locations to represent the borders, and a hashmap with key that is the name and value that is the municipality. A hashmap is used so that each municipality is unique. This class has only getters and setters for each attribute.

<<Entity>> User

The user class has attributes for username, password, a list of addresses, and a list of trees that the user owns. The user class also has an enum for the user role, where a user can be a residential user or a scientist. There can only be one of each user, so the user class has a static hashmap that takes as key the username and outputs the user. This class has only getters and setters for each attribute.

<<Entity>> Species

The species entity has three string attributes pertaining to the species' common name, species name, and its genus. The class also has a static hashmap that takes a key that is the common name and gives the Species object. This class has only getters and setters for each attribute.

<<Entity>> SurveyReport

The survey reporting class has three attributes: the reporting date of the survey, the name of the user who did the reporting, and a unique id. This class has only getters and setters for each attribute.

<<Entity>> SustainabilityCalculator

The sustainability calculator entity has only one attribute which is a Tree. There are several methods that are not just getters and setters. These different methods calculate the different sustainability attributes of the tree. Examples of these methods are CO2 reduction, energy conserved, and stormwater intercepted. This entity is subject to change if more sustainability attributes should be added. The extra methods included thus far are:

- getShadesize(Tree tree)
    - Description: Returns an int that represents the diameter of shade that the tree provides
- getCO2Reduced(Tree tree)
    - Description: Returns an int that represents the amount of CO2 being sunk into the tree.
- getEnergyConserved(Tree tree)
    - Description: Returns an int that is the amount of energy conserved by having a that tree in its specific location
- getStormwaterIntercepted(Tree tree)
    - Description: Returns an int that represents the amount of stormwater that the tree intercepts.
- getBioDiversityIndex(Tree tree)
    - Description: Returns the biodiversity index of the municipality that the tree resides in.

<<REST>> TreepleRestController

Handles all the conversions to DTOs and handles any REST requests. Acts as a communication from the server to the backend service controller. Contains 6 convertToDTOs for each entity except for SustainabilityCalculator, as well mapping functions for each service function.

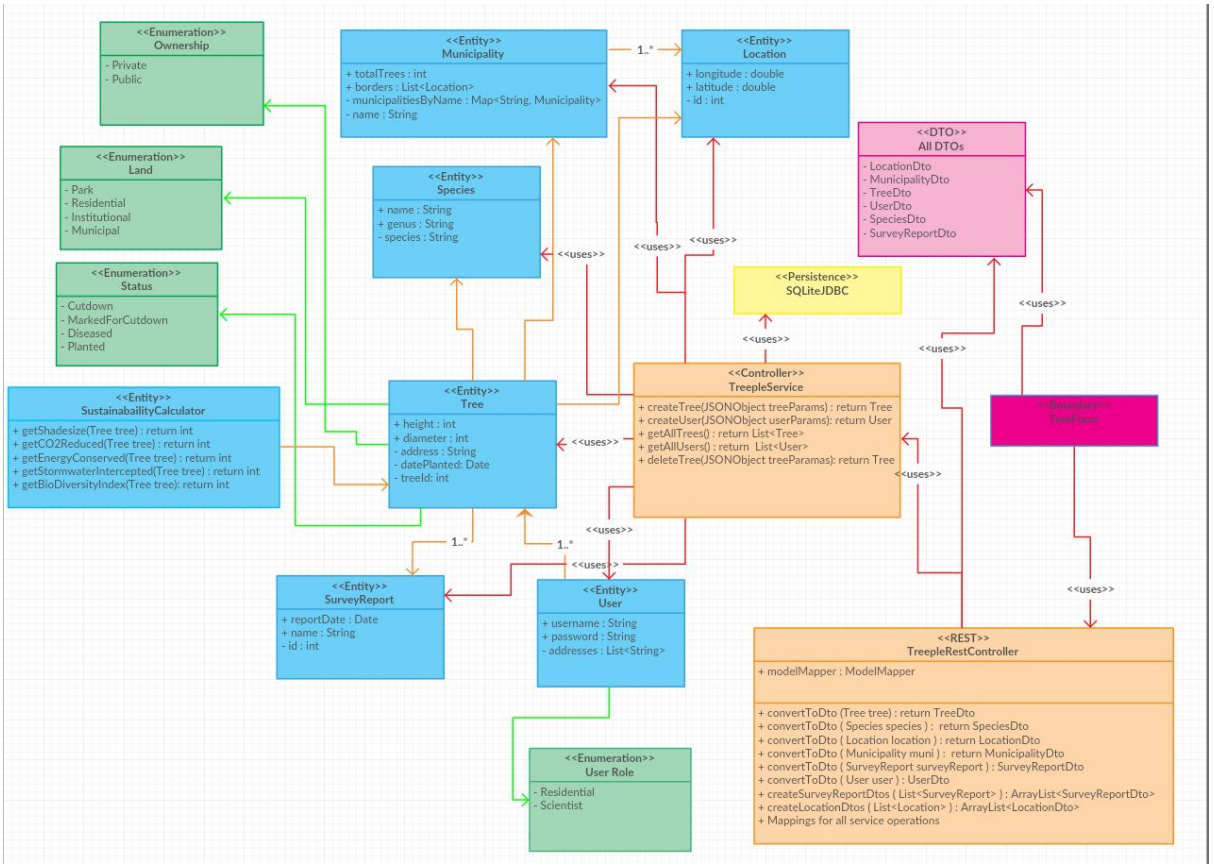
<<Controller>> TreepleService

The TreepleService class takes cares of any operations to be performed on any of the above entities aside from SustainabilityCalculator. This includes getting, adding, deleting or editing any objects in the database. The list of operations are as follows:

- createTree(JSONObject treeParams)

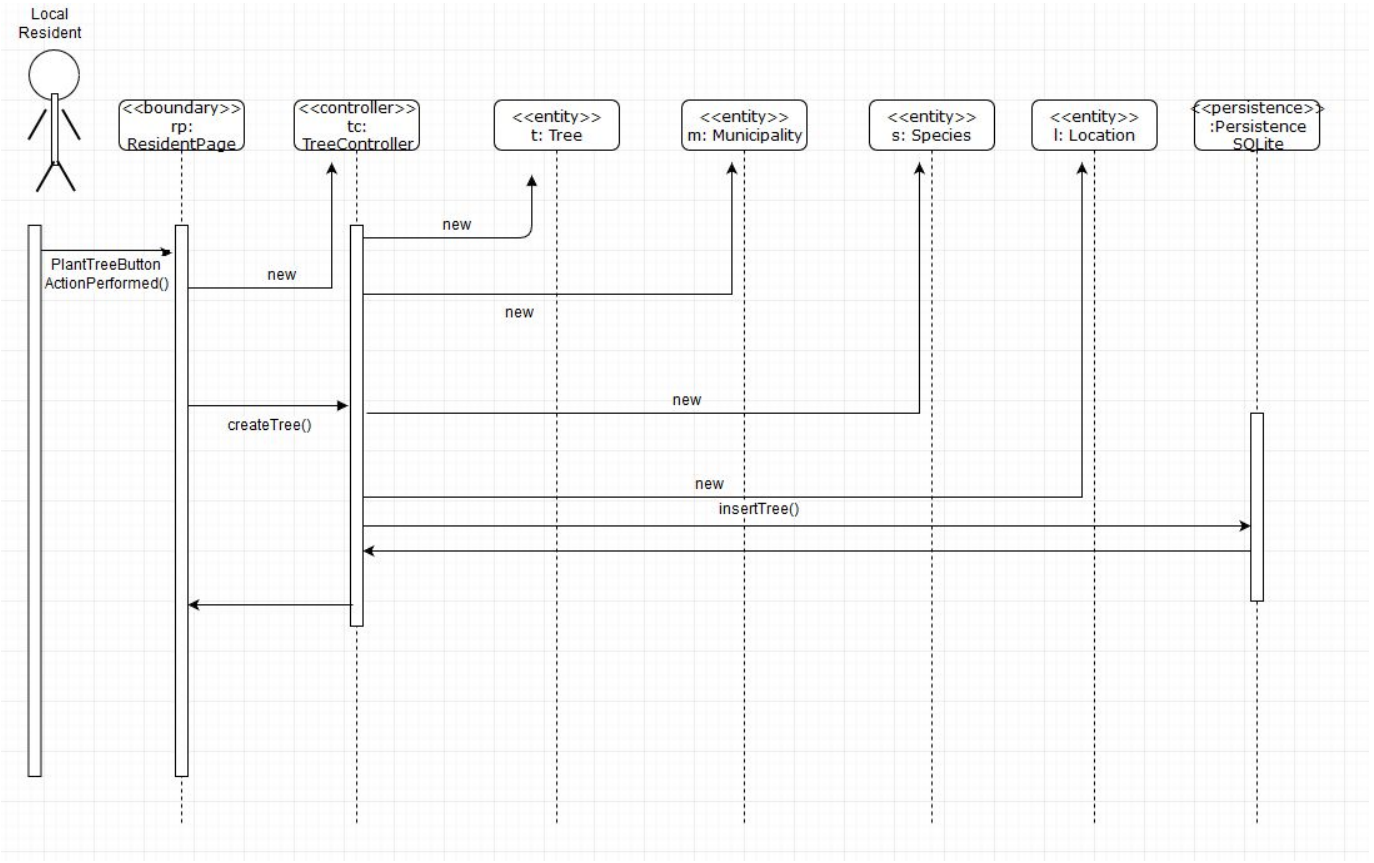
    - Creates a tree with a JSONObject and returns the tree object that was created
- createUser(JSONObject userParams)
    - Creates a user with a JSONObject and returns the user object that was created
- getAllTrees()
    - Returns a list of all the trees in the database
- getAllUsers()
    - Returns a list of all the users in the database
- deleteTree(JSONObject treeParams)
    - Deleted the tree of the type of JSONObject and returns the deleted tree object

Class Diagram



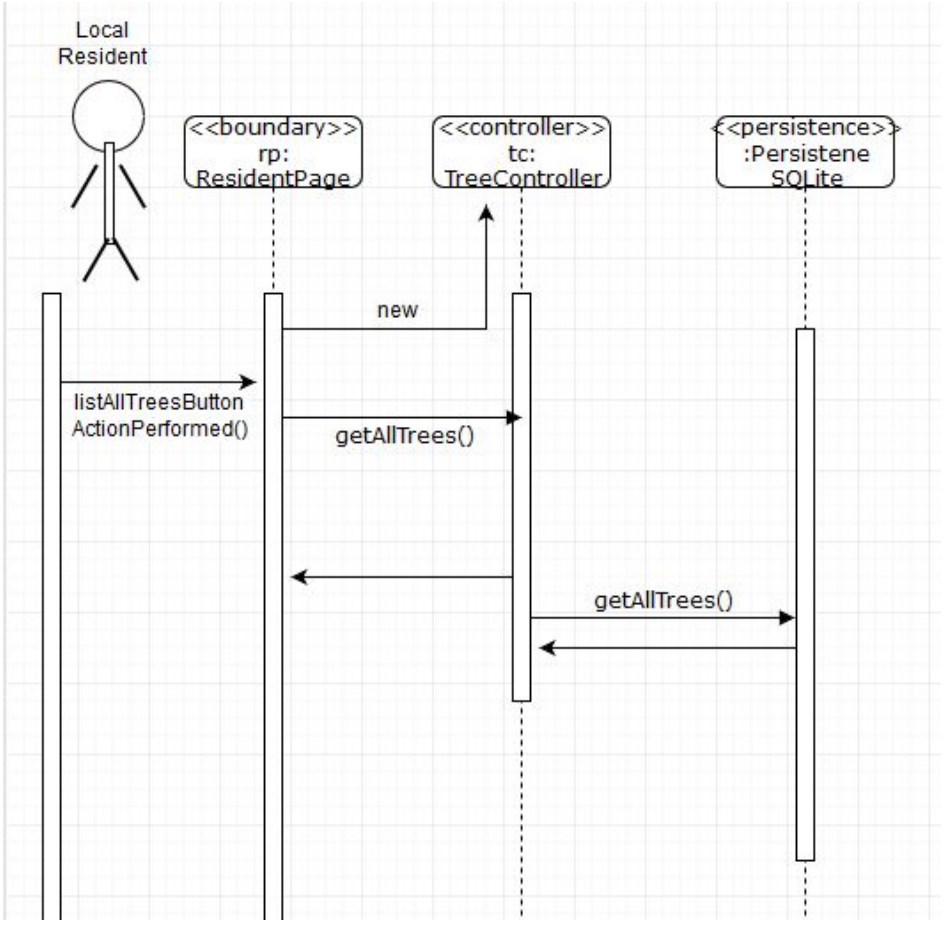
Sequence-Level Diagram

Sequence-Level Diagram: Plant Tree



From the diagram above, one of the architectural layers that can be seen is the MVC, where the View is the boundary class, the Model is the SQLite database, and the Controller is the controller class.

Sequence-Level Diagram: List All Trees



From the diagram above, one of the architectural layers that can be seen is the MVC, where the View is the boundary class, the Model is the SQLite database, and the Controller is the controller class.

Work Plan

KEY DEADLINES

- February 11th : Deliverable 1
- February 25th : Deliverable 2
- March 18th : Deliverable 3
- March 29th : Deliverable 4
- April 8th : Deliverable 5
- April 11th : Deliverable 6

WORKING HOURS

Team Members	**Week 1**	**Week 2**	**Week 3**	**Week 4**	**Week 5**	**Week 6**	**Week 7**	**Week 8**	**Week 9**	**Week 10**
Filip Bernevec	6	4	5							
Abbas Yadollahi	6	5	10							
Alexander Harris	6	5	7							
Gareth Peters	6	4	7							
Yunus Cukran	6	5	5							

TASK ALLOCATIONS

WEEK 1: February 5th – 11th		
Team Members	**Leadership Roles**	**Main Task**
Filip Bernevec	Project Manager	Requirements-level activity diagram
Abbas Yadollahi	Back-end developer	Domain level statechart + workplan

Alexander Harris	Android developer	Use case diagram + use case specifications
Gareth Peters	Web developer	Functional and non-functional system requirements
Yunus Cukran	Back-end developer	Domain model in Umple and class diagram

WEEK 2: February 12th – 18th		
Team Members	**Leadership Roles**	**Main Task**
Filip Bernevec	Project Manager	Sequence diagrams + Work Plan
Abbas Yadollahi	Back-end developer	Backend Business Logic + Database
Alexander Harris	Android developer	Android Front-end
Gareth Peters	Web developer	Detailed Design of proposed solution + class diagram
Yunus Cukran	Back-end developer	Architecture of proposed solution + block diagram + Persistence Business entities

WEEK 3: February 19th – 25th		
Team Members	**Leadership Roles**	**Main Task**
Filip Bernevec	Project Manager	Sequence diagrams + Work Plan
Abbas Yadollahi	Back-end developer	Backend Business Logic + Database
Alexander Harris	Android developer	Android Front-end
Gareth Peters	Web developer	Detailed Design of proposed solution + class diagram
Yunus Cukran	Back-end developer	Architecture of proposed solution + block diagram

WORK PLAN

RED = Task completed

YELLOW = Modification to last work plan

Week 1 (February 5th – 11th)

Key Deadline: Deliverable 1 Feb 11th

- Determine the functional and non-functional requirements.
- Determine the use cases with diagram and specifications.
- Design the activity diagram for entire scenario.
- Design the domain model in Umple with class diagrams
- Design the statechart for class Tree.

Week 2 (February 12th – 18th)

- Develop a system architecture including block diagrams.
- Describe a detailed solution including class diagrams.
- Develop a prototype including "Plant Tree" and "Cut Down Tree" use cases for the Java Spring backend and the Web and Android front-end.

Week 3 (February 19th – 25th)

Key Deadline Deliverable 2 Feb 25th

- Develop a prototype implementation containing the "List All Trees" use case in the backend and frontend.
- Create an implementation-level sequence diagram from "Plant Tree" and "List All Trees" use cases covering all architectural layers.
- Implementation for Android frontend pushed to next week.

Week 4 (February 26th – March 4th)

- Develop a prototype implementation for the "Update Tree Status" and "Generate Forecast" use cases in the backend and frontend.

Week 5 (March 5th – 11th)

- Develop test cases for unit testing, system testing and component testing.
- Start testing the software prototype.

Week 6 (March 12th – 18th)

Key Deadline: Deliverable 3 Mar 18th

- Provide a description of release pipeline

Week 7 (March 19th – 25th)

- Finalization of the Android and Web front-end.
- Implementation of extra features and testing extra features.

Week 8 (March 26th – April 1st)

Key Deadline: Deliverable 4 Mar 29th

- Create a presentation for the TreePLE application.

Week 9 (April 2nd – 8th)

Key Deadline: Deliverable 5 Apr 8th

- Review source code of full implementation.

Week 10 (April 9th – 11th)

Key Deadline: Deliverable 6 Apr 11th

- Submit source code and commit history.