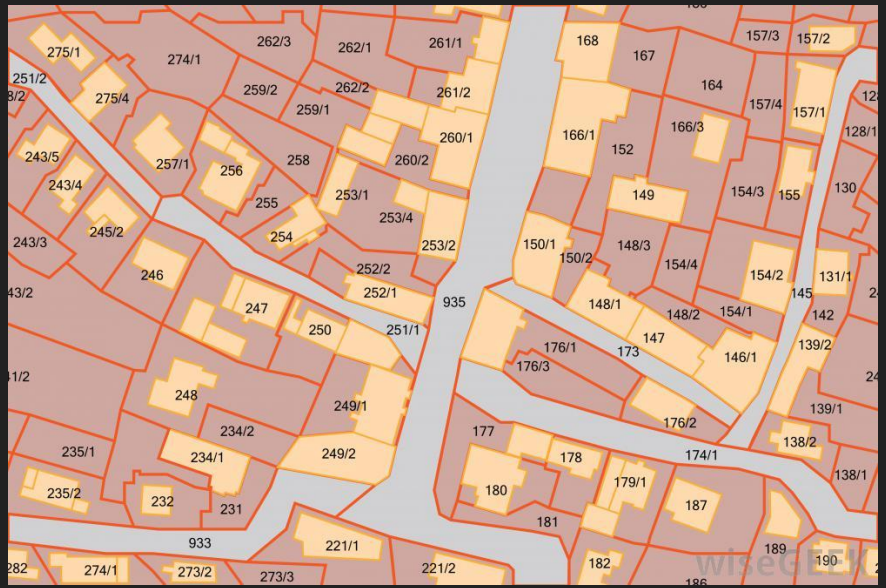
Cadastre

By Amanda Davis, Halle Fields, Nathan Fox, Devin Michaels, Brandon Purvis, Alex Voutsinas

What is Cadastre Mapping?

Users can look at a composite drone photo of their neighborhood, draw a polygon that delineates their property lines, and store it as a uniquely identified digital property to speed up property title issuance.



Requirements

Overview of Functional Requirements

- Polygon Drawing
  - The user must be able to draw polygons on the given map to specify property lines
- Collision Detection/Resolution
  - The program needs to recognize when two polygons are overlapping and find some way to deal with this issue
- Review/Approval Process for new Polygons
  - After polygons are drawn, they are not immediately stored as valid. They must meet polygon definition requirements and not overlap other valid polygons

Overview of Functional Requirements

- Customizable Snap
  - When a user creates a polygon and a point is within a set distance from an existing point, the points must snap together to form a single, shared point.
- Polygon Neighbor-Awareness
  - All polygons must be aware of neighboring polygons based on shared points/borders.
- Store polygons to external database
  - Polygons should be stored so the information is accessible later

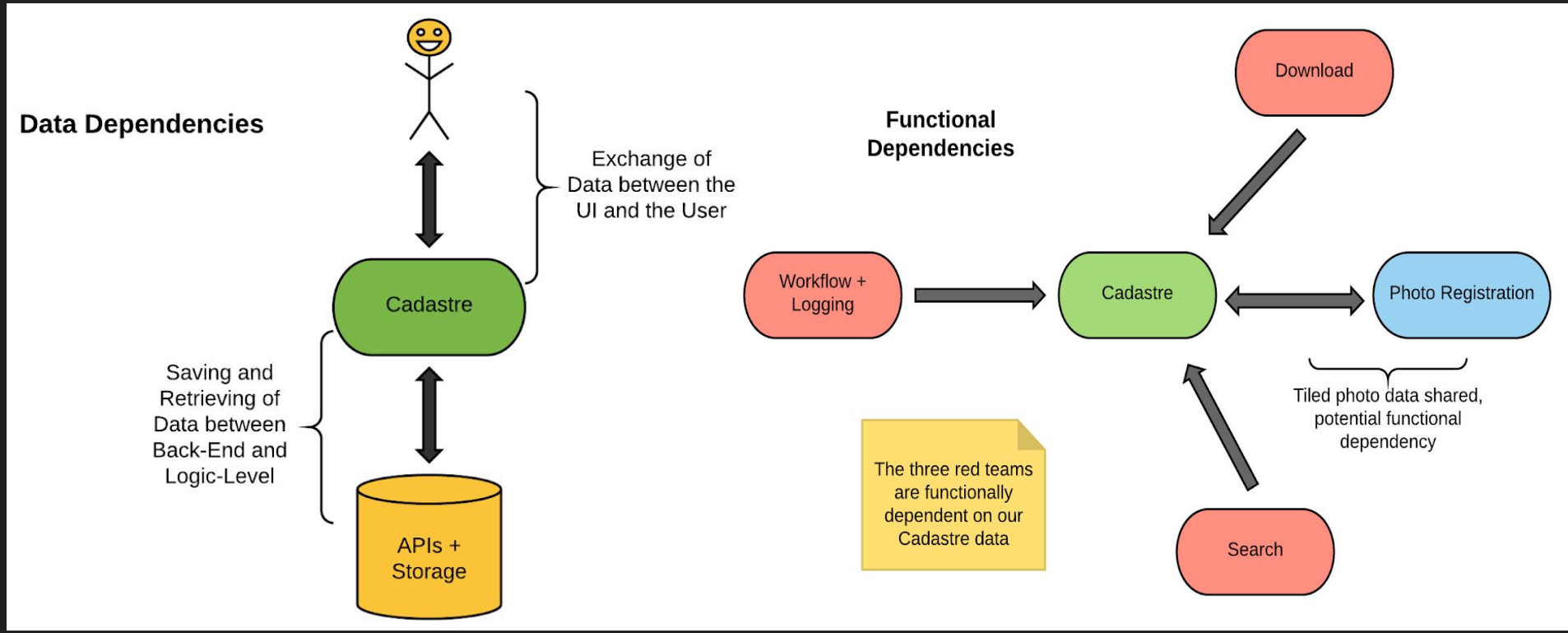
Overview of Non-Functional Requirements

- Slippy Tiled Images
  - Ability to load tiled images of community and drag around within the UI frame a la Google Maps
- Point and Click Polygon Drawing
  - Preferred method by which polygons are drawn. Clicks draw vertices and edges are automatically drawn between them.

Overview of Constraints

- Vector layers
  - Important that image detail of the cadastral maps is not lost after any kind of resizing occurs.
  - Using vector layers as opposed to raster images allows for greater accuracy when determining property borders.
- geoJSON
  - Much of the groundwork for the program has already been laid out by libraries. For Javascript, geoJSON will allow for the construction of polygons in a way that accounts for mapping concepts like latitude and longitude coordinates.
- Web Interface
  - In Columbia, internet cafes are the norm as opposed to personal computers.
  - Program must be accessible online as opposed to any finite locations.

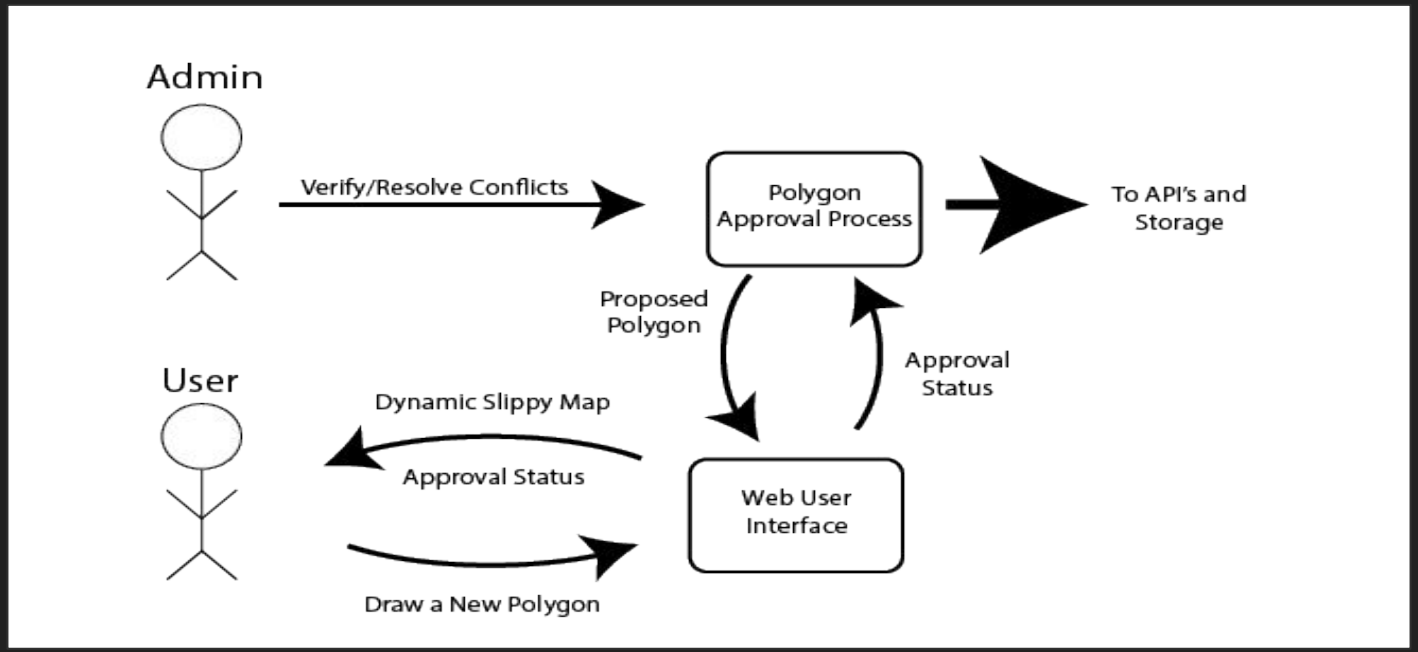
Overview of Data and Functional Dependencies



Architecture Design

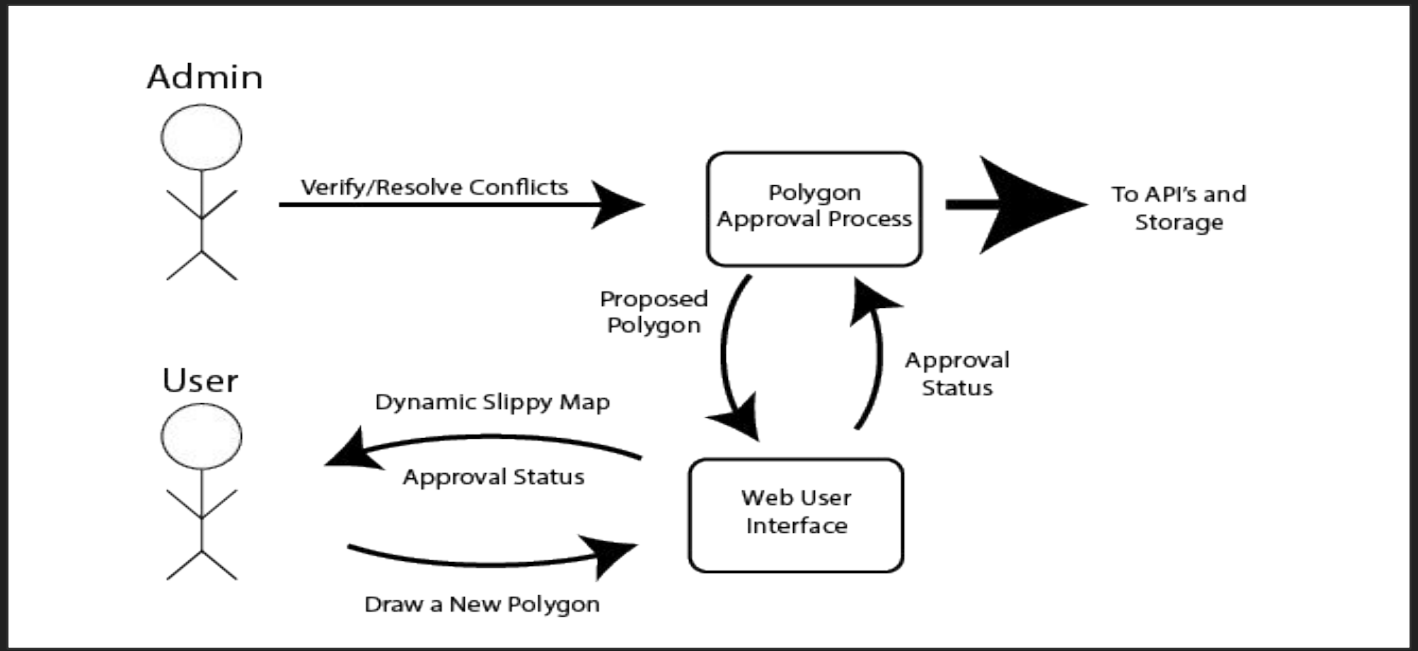
Data and Functional Dependencies

- User - A user in our system has the ability to draw new polygons and give them attributes.



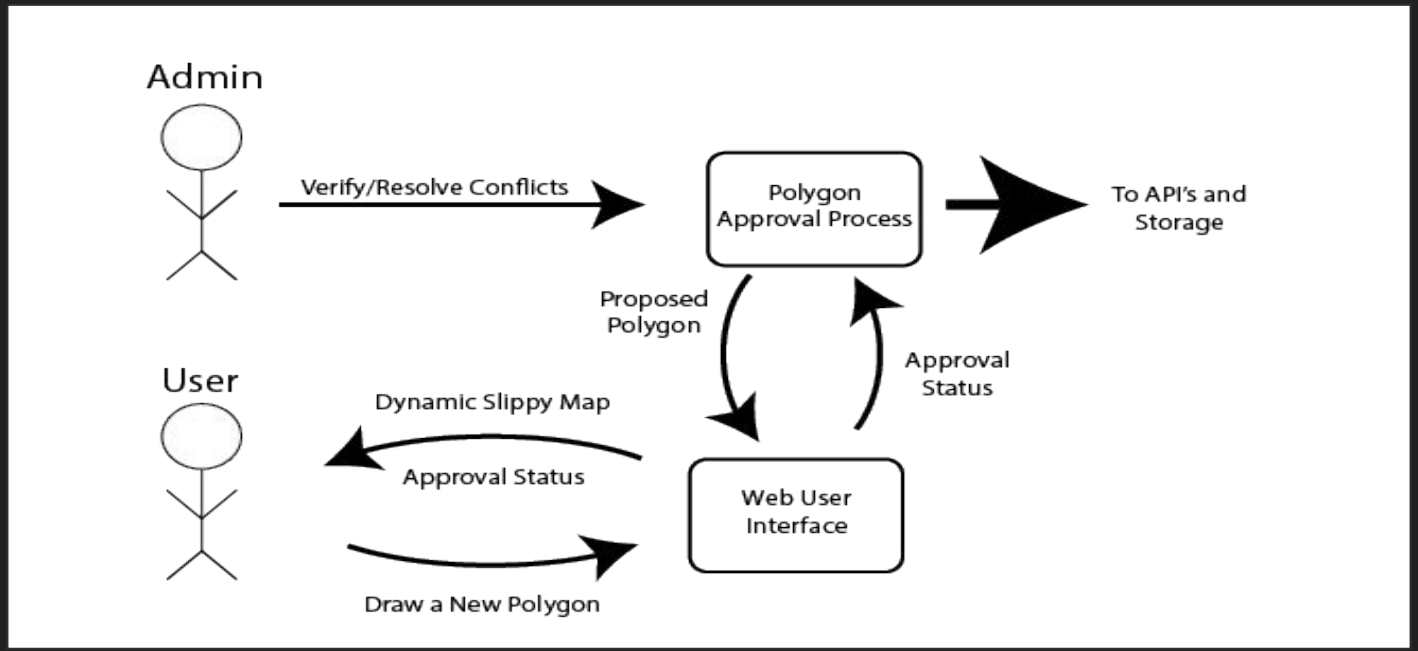
Data and Functional Dependencies

- System Administrator - A system administrator can verify the validity of newly added polygons and resolve conflicts in polygon overlapping.



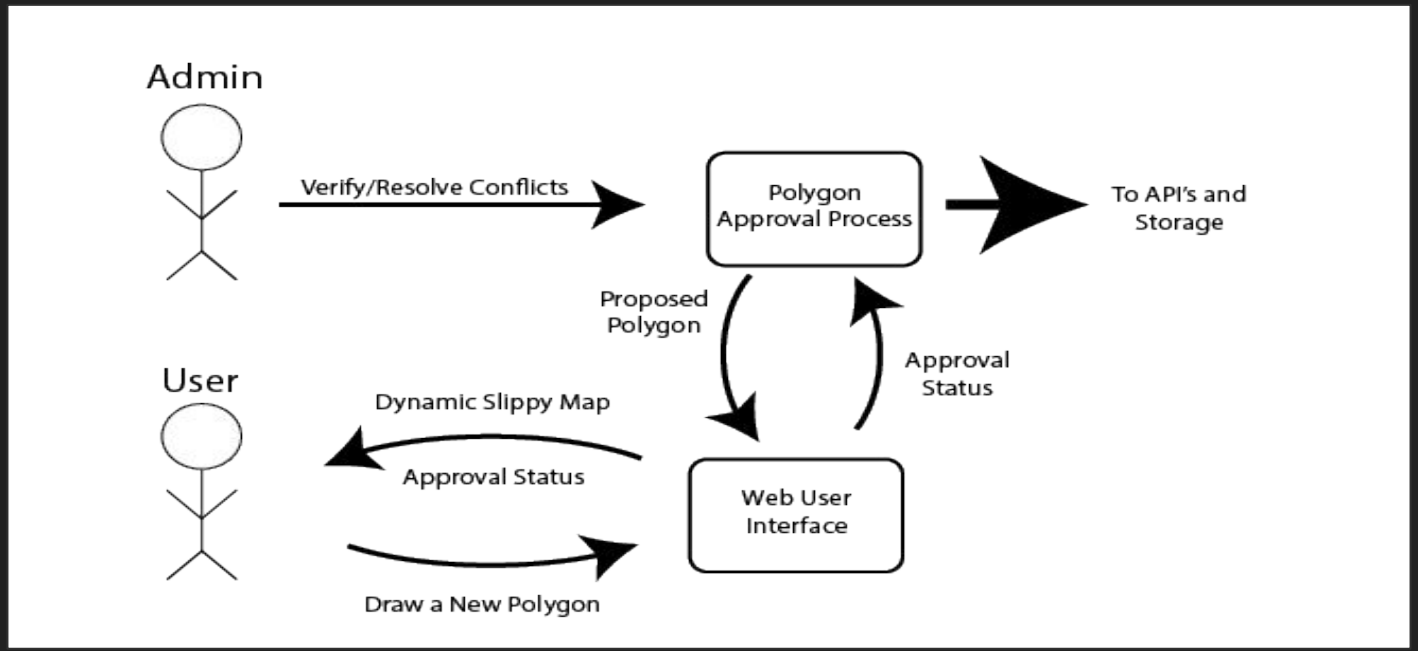
Data and Functional Dependencies

- Application Server- Application server hosts the server-side code that runs the back-end of our web-based UI. In our case, we will be using Amazon Web Services(AWS) to host our layer.



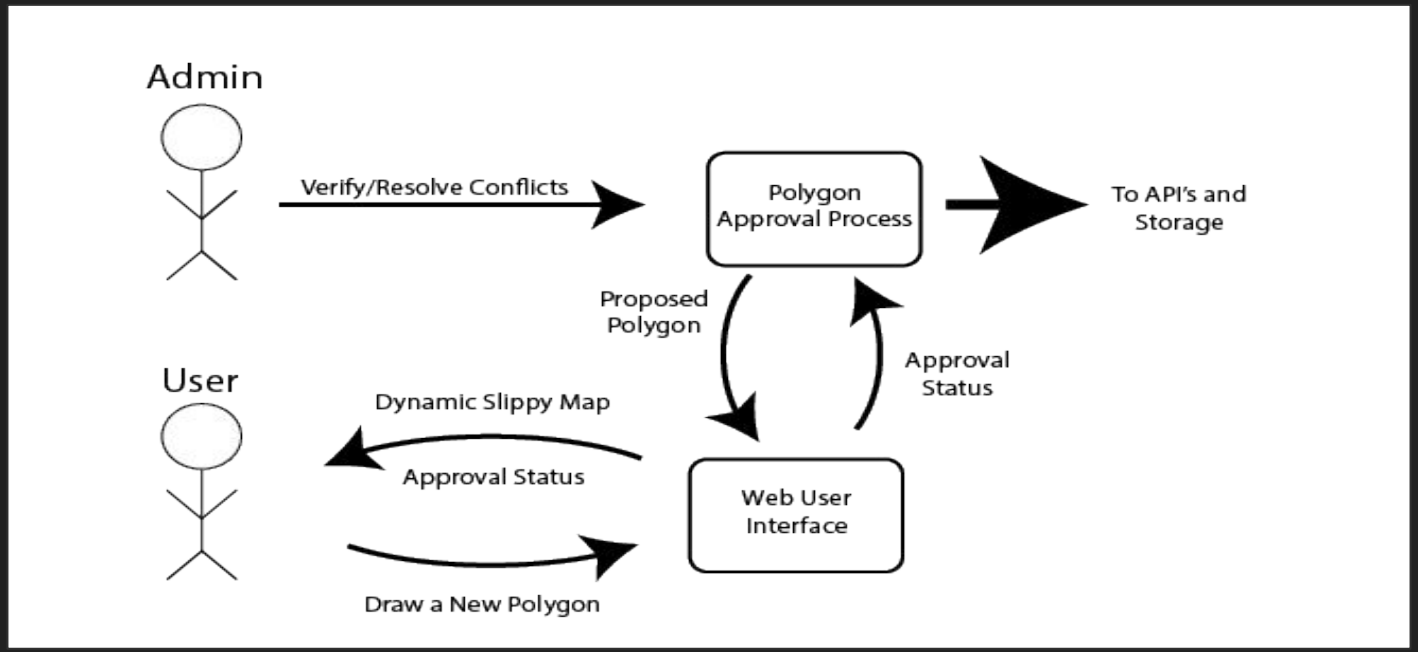
Data and Functional Dependencies

- Web Server - The web server hosts our client-side code which contains all of the UI side, which is also going to be hosted by an AWS server.



Data and Functional Dependencies

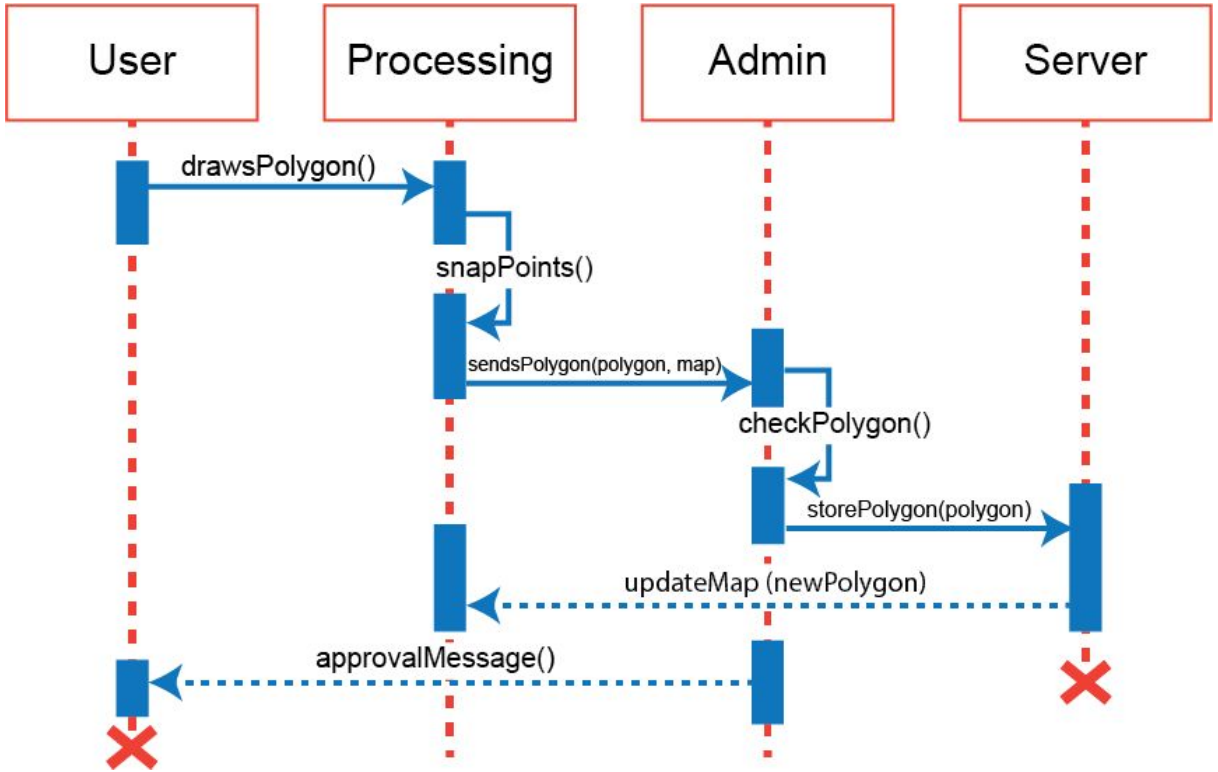
- Database System - interfacing with a database system as provided to us by the Back-End team of this project.



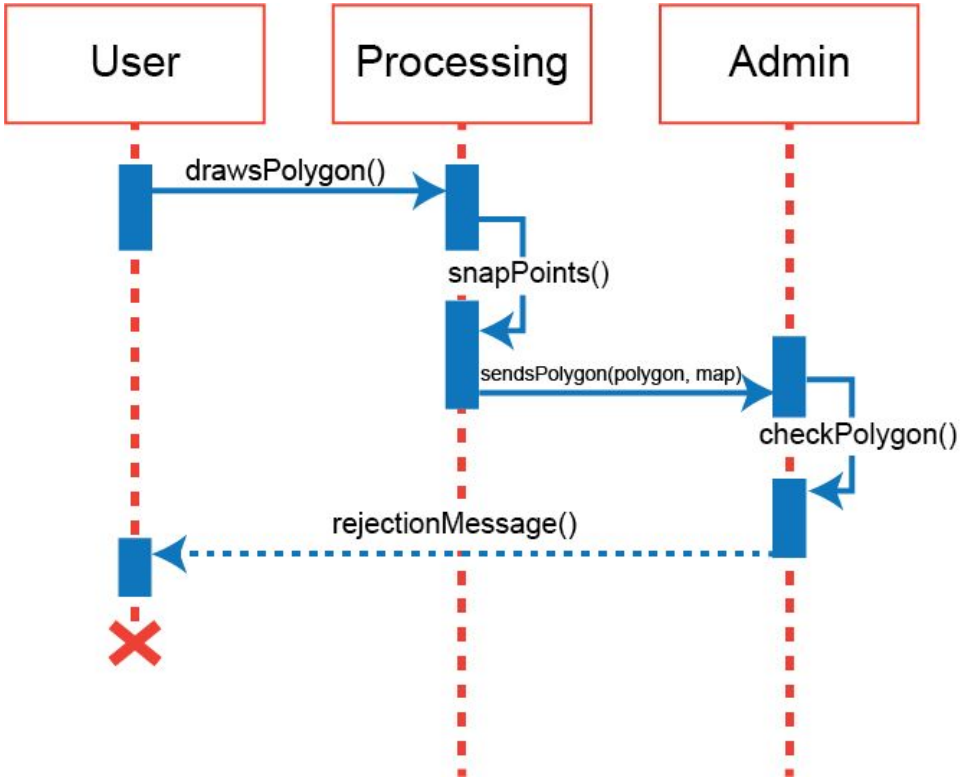
Framework Used

- GeoJSON Entity Framework
  - to work with user-submitted data to form GeoJSON packages of information to send to related groups in the project teams
- Bootstrap/CSS Framework for HTML development
  - Will be used to format our UI portions
- Leaflet
  - Leaflet Snap
  - Leaflet Draw
- Turf.js
- OpenStreetMap

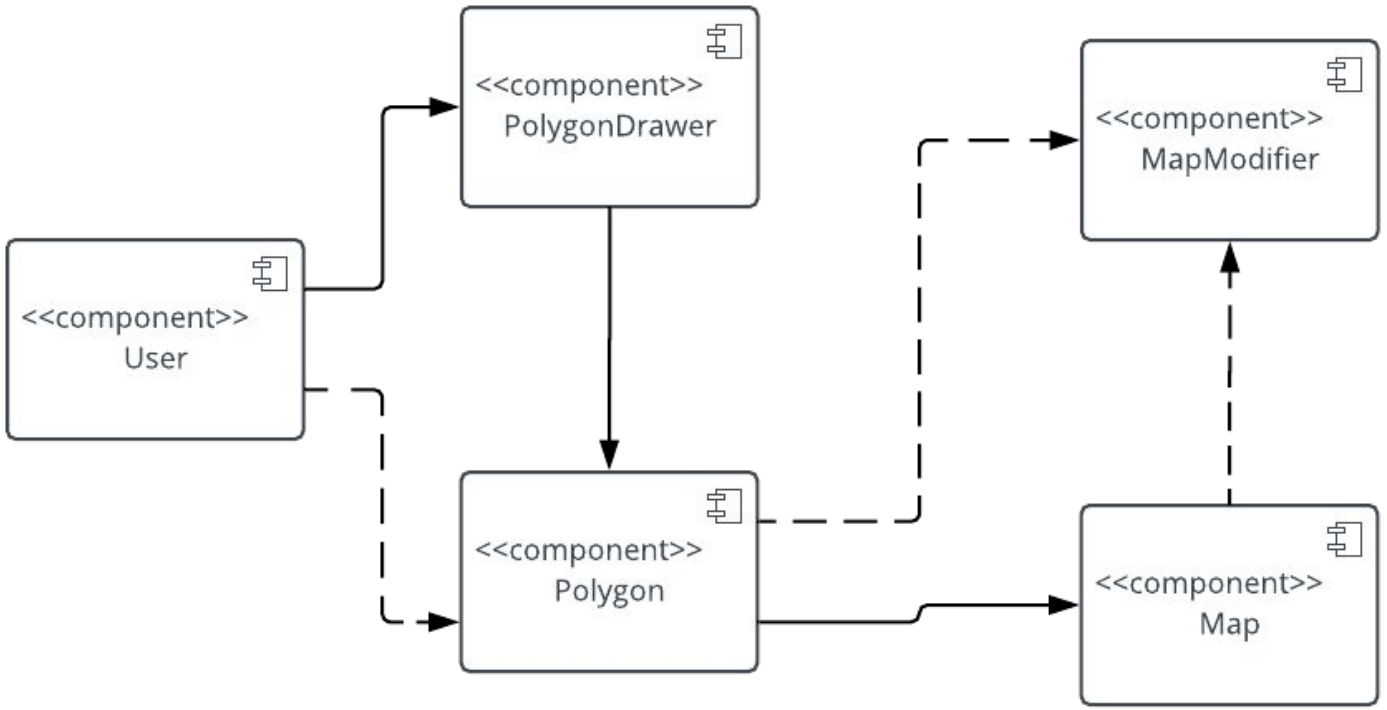
Accepted Polygon Sequence Diagram



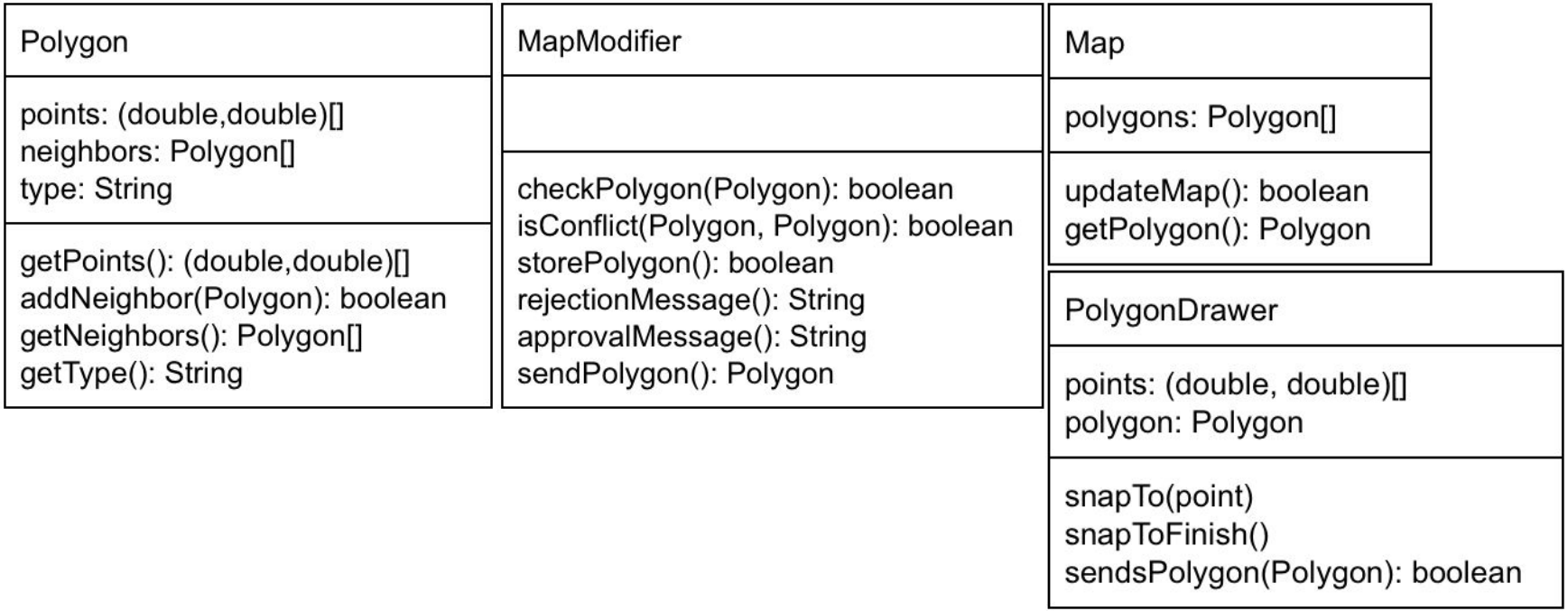
Rejected Polygon Sequence Diagram



Component Level Diagram



Class Diagrams



Implementation

Requirements Implemented

- Polygon Drawing
- Collision Detection/Resolution
- Review/Approval Process for new Polygons
  - In exportable custom geoJSON form, can send to Database team’s API
- Store polygons to external database
  - No actual database set up but the code is ready to accept one. Temporary database was used.
- Customizable Snap

Requirements Partially Implemented

- Polygon Neighbor-Awareness
  - Attempted not completely implemented, still needs more debugging

Demo

Acknowledgements

Our incredible teammates with extended web development experience - we couldn’t have done this without your knowledge

Julio Perez and Chris Mader, our professors

Tim Norris, our Project Sponsor