Give your thoughts on the technical details needed to build a code versioning system like github? How would you incorporate code reviews, different developer teams, and code rollbacks?

Firstly, let’s clarify that git and github are different things. Git is the code versioning system. Github is a website view on top of the git code versioning system.  
  
1) Git – to create git we need to implement many data structures and algorithms like trees and binary searches and hash maps etc. to do things like rollback to a given commit, track commits, determine changesets between versions and so on.  
  
2) Github – to create github we need to implement an object oriented MVC based website on top of the git versioning system using standard things like user role management, auth and so on.

There will be models and views for managing things like **developer teams** stored in some relational database.

A **code rollback** is nothing but moving the code back to a different commit in the underlying git versioning system. A user would request to do so in a view and a model in our MVC stack would go and do so.

The **code review** view is but a presentation of a given commit with added JavaScript to create text boxes for adding comments when clicking on a line of displayed code and so on.

If someone asked you to build a city building simulator (i.e. Sim City), what would your technical writeup be? You are given a grant of ten billion dollars (dollar amount isn't important). Focus on the data structures used to store all of your city data as it grows (buildings, roads, budget, population etc. ...).

*Project*: City Building Simulator

*Proposal*: Using a distributed online simulation system for city development

*Budget*: 10 billion dollars

Abstract: So far all the city building simulators have been desktop based and become slow to operate as a city grows and single desktop capacity is reached; but using ajax and online simulation we can work on different parts of the city without slowing down the simulator, keeping the large data sets on distributed server systems and only loading the data for the particular part of the city that we are editing; when in a zoomed out view of the city we can only load the outer objects such as zoning information and ignore the more detailed nested objects such as public internet modem installations on roads

Having the simulator online and created via RESTful APIs would also allow a quick launch of an MVP and then continuously adding new objects and features into the city simulation environment and enhancing simulator capabilities  
  
*Architecture overview:*

**JSON** is the obvious choice for data here because we have objects which can dynamically grow and become nested – for example the road network could start adding light poles, construction sites, electrical grid, internet modem points and so on; **Distributed NoSQL database** like **Cassandra** would be appropriate for storing this type of data

We can develop an object-oriented model for each new object type in the city corresponding also to the given objects in JSON; a city can have hundreds of thousands of properties to store if we add them all up for all the objects and have different types of administrators for each segment so we could namespace these objects into broader categories by city departments such as city council, construction, police, fire, electric, businesses, hospitals, water management, transportation and so on.   
  
**Models** for zoning, buildings, roads, rail tracks etc. would go under the construction **namespace**, while buses, railway management, taxis etc. would go under transportation namespace. Things like licenses, budget and population would go to city council namespace.  
  
Obviously, **abstract objects** like Vehicle would exist for simulating various vehicle types in the object-oriented world. Design patterns can be used such as there would be a singleton for city council as there won’t be multiple of those.

If the project continues to grow each namespace could become its own **repository managed** by different teams as **microservices** provided for the city simulator which would be made available online to **authenticated users**. Guest users could also play with a limited set of models in a **sandbox** environment. Because of the organization in an **object-oriented** model fashion, it would be easy to limit access to certain models depending on user roles.

Further, the project could provide the sims in sim city geolocation services to help with the traffic management. Sim city used to get a bit congested**. Data collected** from the **geolocation** could help measure commute times for the sims and be useful for **city planning**.  
  
**A key-value store** like **Redis** could be used for collecting this type of geolocation data from the Sims. **Web-sockets** could be used to reduce server calls when simulating the sims in the city and moving them around. Or the sims could simply live on the client side using **browser cache and local storage**. **Progressive web app** concepts can also be used to speed up development for the locality being actively worked on by the user without constraining his machine with the rest of the city.

*Models overview:*

Namespaces: city council, construction, police, fire, electric, businesses, hospitals, water management, transportation

Models and interfaces – parks, roads, income statement, balance sheet, advisors, zones interface implemented by industrial/commercial/residential zones etc.  
  
[UML Diagrams and model details representing relationships between various models would go here for design review and feedback from product and engineering teams]

*Views overview:*

[Proposed simulator views go here] – Ex:

City budget management view – would allow the city to look at tax collection, balance sheet, income statement, cash flow

City advisors view – advisory board for things that need to happen in the city

City terrain view

City zoning view

City underground view – things under the ground such as heavy material blockades; sewer lines

Active construction sites view

Water management view

*Services overview:*

External services like weather, stocks etc. could be hooked into the simulator as well through implementation of existing RESTful API