2) htmlString.replace(/(<p(.\*)>)((\s)|(&nbsp;))\*(</p>) g, “”);

Matches:

<p> </p>

<p></p>

<p>&nbsp;</p>

<p>&nbsp; </p>

<p> &nbsp;&nbsp;&nbsp; p>

<p class="body"> &nbsp;</p>

does not match:

<p> h</p>

<p>& n b s p ;</p>

I included the “(.\*)” in the opening p tag to allow for classes, IDs etc.

3)

I would have each team member fork the main repository and create a separate branch on there own fork for each feature that they are working on. When a feature is complete, they can submit a pull request to the “development” branch of the main repository. This allows the developer and his/her teammates to easily view a diff that shows exactly what changes went into making the new feature. It also allows a developer to easily switch gears if he/she needs to stop working on one feature and work on another, this way a pull request can be created that doesn’t include code from any other features in progress at the time.

Once the feature is finished and assuming the code style is in line with any formal/informal company polices. I would have a code review where the developer first explains the code, what it does, and the reasoning involved in any trade offs that might have made in performance, readability etc. Next I would open up for constructive criticism, for small changes this could be done by commenting individual lines in the pull request on github, while for bigger conceptual changes it would be best to explain in person or via email. Once the Lead developer is satisfied with the code, it would be merged into the development branch where it can be tested in a production environment via a staging server to ensure there are no bugs that were missed while testing locally. Finally, when this it complete the changes would be merged into the master of the main repository and any specific deployment procedure would follow.

4)

My app would be object oriented and built from the ground up, creating each of the classes listed below and then writing tests for the class and its methods in isolation before moving on to the next step. This way if and when errors occur along the way, it will be easier to locate them since I can be more confident that the class I have built in a previous step is performing as it should.

I would start with a class that I will call a “Square”, the Square would have a attribute color that would store an integer in the range 0-5, corresponding to the 6 colors of a rubix cube. This call would also include an instance method for changing it color property.

Next would be a class called “side”, this would hold a 2 dimensional array of “Square” objects. It would be 3X3 to represent the 9 squares on the sides of a rubix cube. The side class would have a method for changing it associated squares called “swap\_squares”, its inputs would indicate a Boolean indicating whether a row, or a column of squares is to be changed, an integer from 0-2 indicating which row or columns is being changed, and an ordered array of 3 integers for the 3 new colors codes that the new squares will be changed to. This method would change the underlying color attribute of the squares based on the inputs. This method is likely to be long, so it would likely be abstracted out in such a way that control flow in the main method would call 1 of 2 helper methods called “swap\_row” and “swap\_col”

The “Cube” would be the most challenging part of the application, this would hold an ordered array of 6 “Side” objects. It would also have 6 convenience methods that all call the same master method called “perform\_shift”, each would call the master with parameters that correspond to twisting the left, middle, or right front face of the cube, or the top, middle or bottom layers of the cube.

perform\_shift would use a spatial algorithm that relates the 6 sides of the Cube to each other and constructs a set of transformations that must occur to alter the cube’s underlying data, and subsequently it appearance to the user. This would be achieved by delegating responsibility to the previously mentioned “swap\_squares” method of each of the Side objects that are altered during the shift.

Atomicity would be important in persisting changes to the database. Model Level validations would be run first (ex, 9 squares of each color exist on the cube), followed by a single database transaction that would execute all the changes together. This way if something goes wrong the data is not corrupted, the Cube can remain in it previous state and error handling can take place.

After each move is made, the GUI would be re-rendered, and a check would if the Cube is solved in a method called “solved?” or “”is\_solved”. This would run a loop that calls an instance method of the Side objects called “matched?” or “is\_matched” which compares the colors of all squares and returns false the first time one square is not the same color as the first one was. The loop inside “is\_matched” and the one inside the “is\_solved” would terminate early for performance sake, returning false as soon as one color, or one side in not matched, there is no point in checking the rest.

Shuffling the cube in real life entails making an arbitrary number of transformations to the cube until the user is satisfied that the Cube looks “random enough”. This would result in a lot frivolous computations and hitting the database. Instead I would just design the program so that when the cube is instantiated, colors are determined by randomly choosing from an array containing 9 instances of the numbers 0-5. Each time a number is picked, a color is assigned and that instance of the number is removed from the array. Once this is done a Model validation can be run and then the data persisted to the database.

5)

Due to the high number of reads, there will be a big emphasis on middleware and caching. Since there are few writes, you don’t need to worry too much about busting/expiring the cache as the data will not change often. Use of SSL might need workers to preserve CPU.

Possible approaches to dealing with video

(Depending on concurrency, size of the videos etc.):

1. A single server that receives the videos and transcodes them. The disadvantage is you this might take up a lot of processing power.
2. A separate server that grabs the raw videos from the first and does the transcoding on its own. The disadvantage for this one is you need a separate server and you have to move the raw files between servers.