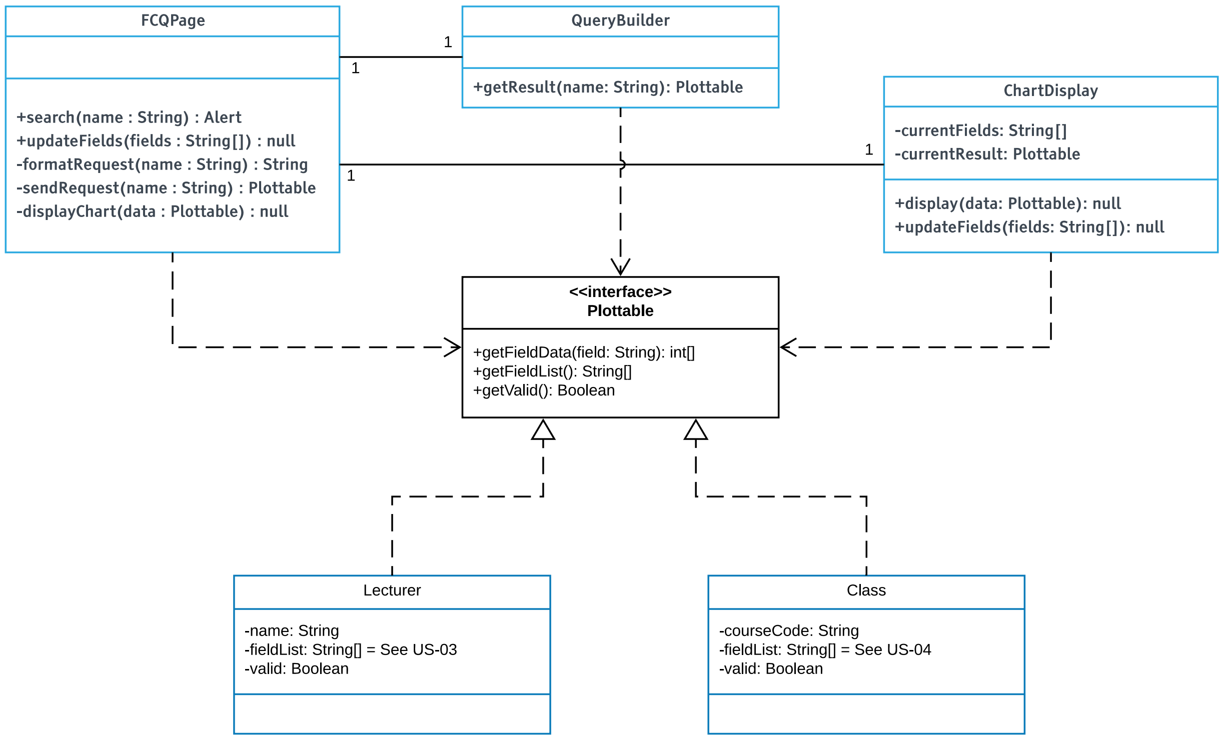
1. **Team:** Samuel Cuthbertson (scuthbert), Connor Hudson (technoboy10)
2. **Title:** fcq.fun - Team 7
3. **Project Summary:** A CU faculty course questionnaire viewer with improved readability, focused mainly on graphs. Limited to Boulder campus, initially limited to College of Engineering results.
4. **Features Implemented:**

ID	Description
US-01	As a user, I can search by lecturer name in the format (Lastname, Firstname) to see results of that professors FCQ results.
US-03	As a user, I want to see the following on a chart as results of a professor search: Instructor Overall, Instructor's Effectiveness, Availability, Instructor's Respect.
FR-01	Queries to the FCQ database should be limited to College of Engineering and Applied Science results.
FR-02	Queries to the FCQ database should bound "First Term" as Fall 2006 and "Last Term" as Summer 2017.
NFR-01	Platform Constraints: The page should display the same across multiple browsers.
NFR-03	Legal: All code will be protected under the GNU-General Purpose license, hosted as LICENSE in the GitHub repository.

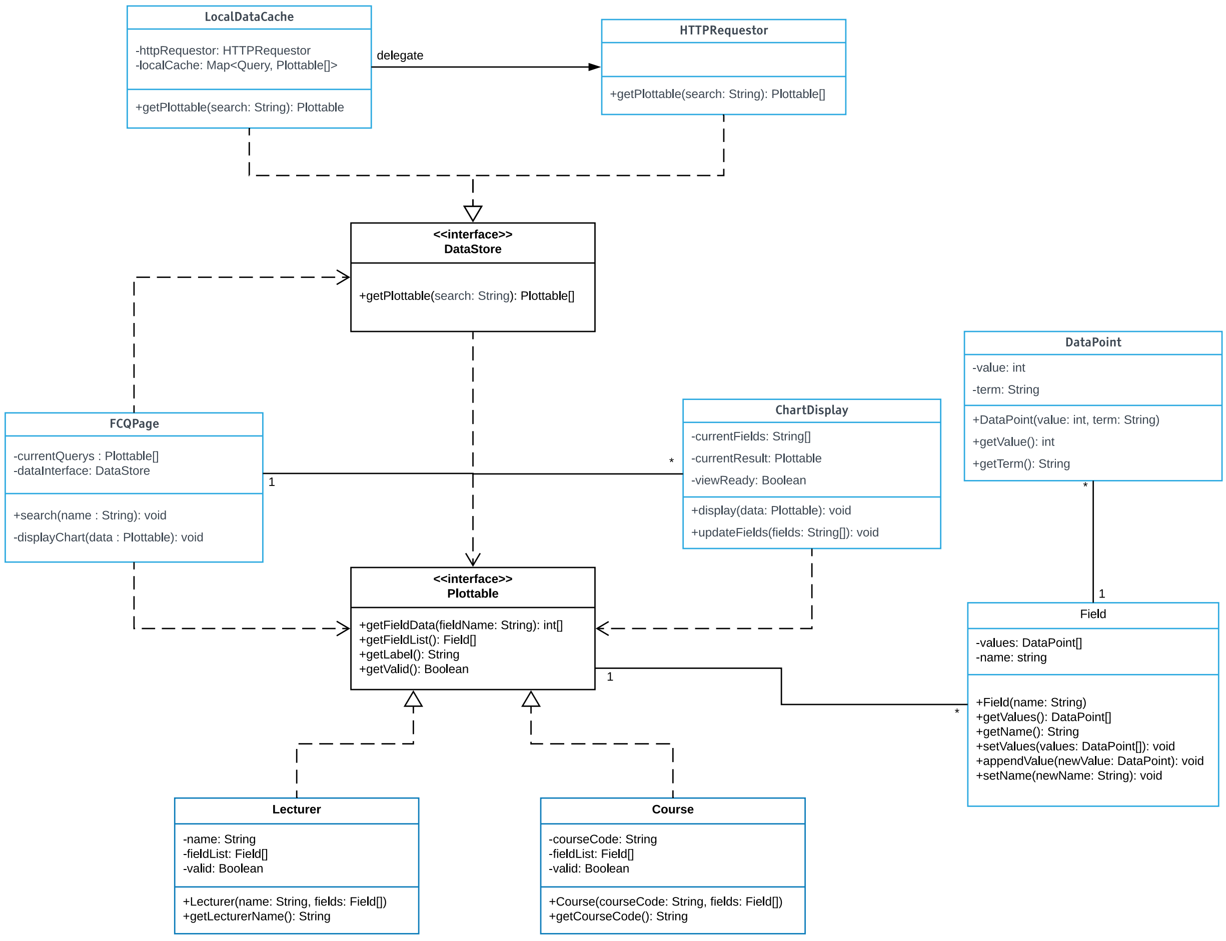
5. **Features Implemented:**

ID	Description
US-02	As a user, I can search by course code to see results of that course's FCQ results.
US-04	As a user, I want to see the the following on a chart as the results of a class search: Course Overall, Hours per Week, Challenge, How Much Learned.
NFR-02	Reliability: In the event of a failed database query, a user facing error alert should appear.

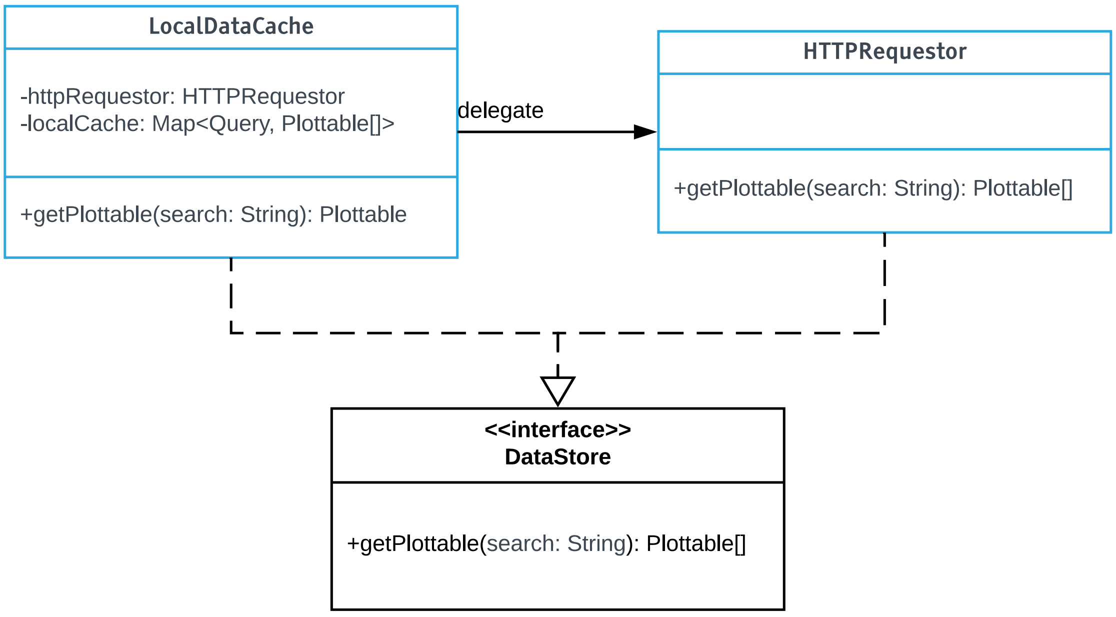
6. **Class Diagram from Part 2:**


FCQPage
+search(name : String) : Alert
+updateFields(fields : String[]) : null
-formatRequest(name : String) : String
-sendRequest(name : String) : Plottable
-displayChart(data : Plottable) : null
QueryBuilder
+getResult(name: String): Plottable
ChartDisplay
-currentFields: String[]
-currentResult: Plottable
+display(data: Plottable): null
+updateFields(fields: String[]): null
1
1
1
1
<<interface>>
Plottable
+getFieldData(field: String): int[]
+getFieldList(): String[]
+getValid(): Boolean
Lecturer
-name: String
-fieldList: String[] = See US-03
-valid: Boolean
Class
-courseCode: String
-fieldList: String[] = See US-04
-valid: Boolean

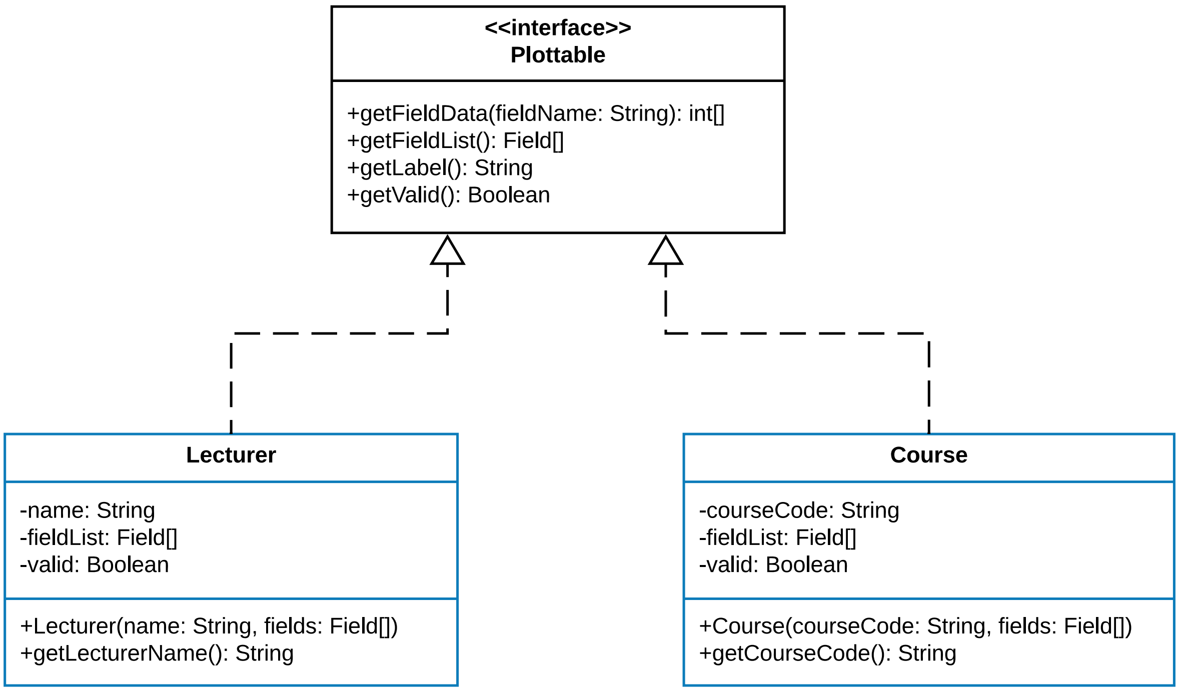
7. **Current Class Diagram:**


LocalDataCache
-httpRequestor: HTTPRequestor
-localCache: Map<Query, Plottable[]>
+getPlottable(search: String): Plottable
delegate
HTTPRequestor
+getPlottable(search: String): Plottable[]
<<interface>>
DataStore
+getPlottable(search: String): Plottable[]
DataPoint
-value: int
-term: String
+DataPoint(value: int, term: String)
+getValue(): int
+getTerm(): String
FCQPage
-currentQuerys : Plottable[]
-dataInterface: DataStore
+search(name : String): void
-displayChart(data : Plottable): void
1
*
ChartDisplay
-currentFields: String[]
-currentResult: Plottable
-viewReady: Boolean
+display(data: Plottable): void
+updateFields(fields: String[]): void
*
1
Field
-values: DataPoint[]
-name: string
+Field(name: String)
+getValues(): DataPoint[]
+getName(): String
+setValues(values: DataPoint[]): void
+appendValue(newValue: DataPoint): void
+setName(newName: String): void
<<interface>>
Plottable
+getFieldData(fieldName: String): int[]
+getFieldList(): Field[]
+getLabel(): String
+getValid(): Boolean
1
*
Lecturer
-name: String
-fieldList: Field[]
-valid: Boolean
+Lecturer(name: String, fields: Field[])
+getLecturerName(): String
Course
-courseCode: String
-fieldList: Field[]
-valid: Boolean
+Course(courseCode: String, fields: Field[])
+getCourseCode(): String

8. **Implemented Design Patterns - Proxy:**


LocalDataCache
-httpRequestor: HTTPRequestor
-localCache: Map<Query, Plottable[]>
+getPlottable(search: String): Plottable
delegate
HTTPRequestor
+getPlottable(search: String): Plottable[]
<<interface>>
DataStore
+getPlottable(search: String): Plottable[]

9. **Implemented Design Patterns - Strategy:**



10. **Takeaways:**

    Our primary takeaway from this project is that design patterns are incredibly useful, but difficult to understand without prior knowledge of what/why they're applied. When strategy or decorator used effectively, it's incredibly slick. However, without any prior knowledge of what various design patterns are or why they're being applied, writing and debugging code which requires patterns is extremely difficult. We ran into this problem using Angular 4, which forces best practices through the use of a heavily object oriented framework and requiring specific return objects to be `Observable` and async safe. This led to significant changes to our part 2 design. However, once we came around to doing things the way Angular wants us to do them, implementing design patterns in typescript became relatively easy.