

[Roadmap](#)

[Jump to Submission](#)

BloQuery

BloQuery

Employ a **Mobile Backend as a Service** (MBaaS) to provide an easy-to-use cloud backend for your iOS app.

Purpose

BloQuery takes advantage of the free-to-use Parse platform in order to create an open question and answer platform. Still use **Quora**? That's so *yesterday*, BloQuery is the future...

Use Case

It's difficult to offer significant value with an isolated mobile application. Games comprise a majority of successful off-line experiences; for everything else there's the cloud. Facebook, Google, Twitter, Quora, Instagram, Evernote, and so many more successful applications are linked to an online universe of users and content recoverable from anywhere.

Mobile developers with great ideas yet limited resources struggle to build both the user-facing mobile experience *and* the full backend stack required to support it; that's where a service such as Parse comes into play.

Parse provides tools such as: a database in the cloud, user authentication and creation, push notifications, blob/file storage, JavaScript code to manipulate your data on the backend, and much more. With Parse, the heavy lifting required when building your own backend is already done. Better yet, for small developers it's absolutely *free*.

Pace

A Bloc iOS project should take about **80 hours** to complete. This project has **9 user stories**, so expect to spend about **8-9 hours** on each. Some user stories are easier than others - this is just an average.

In This Track	Complete Each User Story In
12-week	About 1 day
18-week	About 3 days
36-week	About 6 days

Consider breaking the user stories into smaller pieces to make them more manageable. Also, remember it's typical for the first few user stories to take more time, since you also have to build the foundation of the app.

If you finish early, you can add additional features to this project (see the "Extra Credit" section).

Questions

You'll see questions sprinkled throughout the rest of this checkpoint. Questions aren't writing assignments, and you don't have to discuss them with your mentor. However, writing is a great way to think through these important questions, and they can be great discussion topics with your mentor.

User Stories

The project's 9 user stories are:

- As a user, I want to be able to create a BloQuery **account**
- As a user, I want to be able to **view** the latest user-generated questions or *BloQueries*
- As a user, I want to be able to **answer** questions with text responses
- As a user, I want to be able to **view** existing answers to questions
- As a user, I want to be able to **post** new questions
- As a user, I want to be able to **customize** my profile with a profile image and description
- As a user, I want to be able to **view** other user's profiles
- As a user, I want to be able to **up vote** answers to questions
- As a user, I want to be able to sort answers by the number of **up votes** they've received

These stories are meant to be done in sequence, and become gradually more difficult.

Create a BloQuery Account

*As a user, I want to be able to create a BloQuery **account***

Design the Data Model

Parse offers a number of utilities and conveniences for creating users. Determine whether you can use Parse's **PFUser** out of the box or whether or not you need to subclass it by [reading the documentation](#).

Keep in mind our later user story, in which we want to allow the user the ability to **customize** their profile with an image and description.

Build a View and View Controller

We need a screen for our user to log in, or to create an account if they don't have one already.

You could build your own view controllers that create or fetch a **PFUser**, or you may find that the open source elements in **ParseUI** are worth adopting. There are more details on creating a signup flow in the [Parse documentation](#) and [a separate article on the topic](#).

Test Your Code

- Create a new BloQuery user. Verify that your new user shows up in [your app's dashboard](#).
- Try logging in with invalid credentials.
- Quit and relaunch the app. Log in as the user you've just created.

View the Latest User-Generated Questions

*As a user, I want to be able to **view** the latest user-generated questions or BloQueries*

Design the Data Model

Our questions will live in the cloud, in Parse, so they will be a subclass of a **PFObject**. Read how to subclass Parse objects in the [documentation](#).

Build a View and View Controller

Determine if you will use **ParseUI**'s built-in **PFQueryTableViewController** or if you will use a datasource-like pattern as you did with Blocstagram to feed a **UITableViewController**.

You may also want to create your own custom cell, like Blocstagram, to layout your question elements for each row of the table.

You will also want to build a view controller to house the individual question view that appears when a user taps on a question's row.

Test Your Code

- Write a unit test to create a few sample questions using your new PFObject subclass.
- Verify the questions show up on the cloud side in **your app's dashboard**.
- Run the app and verify that you see your questions showing up in your table view.
- Select a question by tapping on it to exercise your single question view.
- Enable airplane mode and try your application. Consider what you can do to improve the experience.

Answer Questions with Text Responses

*As a user, I want to be able to **answer** questions with text responses*

Design the Data Model

Our answers are all tied to questions, so consider where this object's code will be written. From where will you create a new answer? Is it on the question object or do you have a datasource-type object mediating the creation of answers to questions?

Build a View and View Controller

In our wireframe, we present a view controller that hides the other answers. It will strongly resemble the commenting UI you built for Blocstagram. Revisit that project as a reference for implementing a similar experience here.

Test Your Code

- On your test-created questions, begin adding answers.
- Verify the answers show up on the cloud side in **your app's dashboard** and have a relationship to the correct question.
- Quit and relaunch the app to verify that your answers appear against the appropriate questions.
- Try logging in as different users and posting answers to the same questions. Verify that those answers appear.

View Existing Answers to Questions

*As a user, I want to be able to **view** existing answers to questions*

Design the Data Model

If you've configured your answer and question relationship correctly in the previous use case you should be able to extract a list of answers tied to a particular question.

How can you ensure these answers are up-to-date with what answers have been submitted since a user navigated to a particular question screen?

Build a View and View Controller

Add a **UITableViewController** to display the answers related to the selected question. You will have to consider, as above, whether you use a **PFQueryTableViewController** or your own datasource and delegate for a **UITableViewController**.

Test Your Code

- View the test-created questions by tapping on each question row.
- Verify the answers show up beneath your question with the correct layout.
- Run the app and verify that you see the questions and their answers.
- As an added bonus, from a separate account on a separate device, submit an answer for a question you are viewing and verify the first screen updates correctly.

Post New Questions

*As a user, I want to be able to **post** new questions*

Design the Data Model

You will need to consider where your question creation logic happens. Have you already created a centralized object creation mechanism or are you using Parse to do the heavy lifting for you in **PFObj**ects?

The sample question you display for a user should be localized (or at least localizable). You may wish to change the sample question without having to do a rebuild. Consider using the **PFConfig** object for storing the sample question.

For more information on localization, consult [Apple's documentation](#) or [this tutorial on Ray Wenderlich's site](#).

Build a View and View Controller

In our wireframe, we present a view controller that doesn't occupy the whole screen and looks like a **UIAlert**View with a text field and buttons. Create this effect by using a new **UIWindow** (see the [Apple View Programming Guide](#)).

As an alternative, consider using a third party library like **SDCA**lertView to accomplish this. In this case, your code might look like this:

```
SDCAviewController *alert = [SDCAviewController alertControllerWithTitle:NSString(@"New Question", @"New Question"
                                                                    message:NSString(@"Type a new question."), @"Type a new question."];

[alert addAction:[SDCAviewController actionWithTitle:@"Post" style:SDCAviewControllerStyleDefault handler:...]];

UITextField *textField = // create a text field...
    // configure your text field's layout (constraints or frames)...

[alert.contentView addSubview:textField];

[alert presentViewController:nil];
```

Test Your Code

- Tap the new question button in the UI, and enter your question.
- Verify the questions show up on the cloud side in **your app's dashboard**.
- Run the app and verify that you see your questions in your table view.

Customize User Profile with an Image and Description

*As a user, I want to be able to **customize** my profile with a profile image and description*

Design the Data Model

To store images in Parse you'll need to check out their **PFFile** class and consider subclassing that. Parse has a **tutorial for saving images to the backend** which you may find useful.

You'll access the photo library again to allow the user to upload their own photo and perhaps additionally providing some stock profile images from which users can choose.

Build a View and View Controller

The wireframes don't provide a specific profile editing UI. At the very least, you'll need to build a profile editing view controller, which will provide a way to select a profile image and add and edit a description.

You'll also need to decide from where the user will access this profile editing and enhancement functionality. Consider the need to constrain the image to certain dimensions and think about what you might be able to re-use from the Blocstagram project to accomplish this end.

Take a look at the **PFImageView** from the open source ParseUI collection.

Test Your Code

- Create a new BloQuery user. Navigate to your new profile editing screen, add a profile image and description, and verify the information has been saved in **your app's dashboard**.
- Log in as an existing BloQuery user. Navigate to your new profile editing screen, add a profile image and description, and verify the information has been saved in **your app's dashboard**.

View Other Users' Profiles

*As a user, I want to be able to **view** other user's profiles*

Design the Data Model

We'll be creating a read only version of the information you've just modeled in the previous user story, so we won't add any real model facets in this story.

Build a View and View Controller

Consider whether you can create a subclass of the controller you've written in the previous user story or whether you need to create a new one altogether.

Test Your Code

- Tap on an existing user from a question or answer they have contributed and inspect their user profile screen.
- Via unit tests, create a number of users with varying lengths of text in the description field and view them in the app. One test might look like this:


```
- (void) testViewProfileLabels {
    User *shortDescription = [User userWithName:@"John Francis 'Jack' Donaghy" description:@"Vice President of East C
    User *longDescription = [User userWithName:@"Scrooge McDuck" description:@"Scrooge is an elderly Scottish anthrop

    ViewProfileViewController *shortProfileVC = [[ViewProfileViewController alloc] initWithUser:shortDescription];
    ViewProfileViewController *longProfileVC = [[ViewProfileViewController alloc] initWithUser:longDescription];

    // Force the views to load
    [shortProfileVC view];
    [longProfileVC view];

    XCTAssertEqualObjects(shortDescription, shortProfileVC.descriptionLabel.text);
    XCTAssertEqualObjects(longDescription, longProfileVC.descriptionLabel.text);

    CGRect expectedLongProfileRect = CGRectMake(44, 100, 200, 600); // Just for example
    XCTAssertTrue(CGRectEqualToRect(expectedLongProfileRect, longProfileVC.descriptionLabel.frame));
}
```

Up Vote Answers to Questions

*As a user, I want to be able to **up vote** answers to questions*

Design the Data Model

You will need to add a **votes** property on your answer objects.

Consider whether this should be a simple count, or something more complex, like an array of users who have upvoted. Weigh the pros and cons of each approach and choose one.

Build a View and View Controller

Per the wireframes, you will need to add a button for allowing a user to vote for a particular answer. Clicking it again should un-vote for the answer.

Test Your Code

- Navigate to a particular question's screen and vote for an answer. Verify that the vote count for that answer shows up in **your app's dashboard** and is reflected in the app's UI.
- Quit and relaunch the app. Verify that the answer's vote count reflects your vote.
- Tap the button again and rescind your vote for that particular answer. Verify that the vote total updates correctly both in your dashboard and in the app's user interface.

Sort Answers by the Number of Up Votes

*As a user, I want to be able to sort answers by the number of **up votes** they've received*

Design the Data Model

There shouldn't be any major model changes for this user story.

Build a View and View Controller

The visible work on this story will center mainly around your **query to the Parse data store**. If you are using a **PFQueryTableViewController** or a subclass you can specify a query and sort order in your subclass by implementing a **– (PFQuery *)queryForTable;** method.

When the user votes for an answer and the new vote total triggers a change in the ordering of the answers you should animate the reordering. See the Apple documentation, particularly [the section on managing the reordering of rows](#) and `– (void)tableView:(UITableView *)tableView moveRowAtIndexPath:(NSIndexPath *)sourceIndexPath toIndexPath:(NSIndexPath *)destinationIndexPath;` for an example of how to do this. `

Test Your Code

- Navigate to a particular question's screen and verify that the answers are sorted by votes.
- Vote for an answer to raise it above the answer preceding it. Verify that the UI reflects the new order of answers for that question.
- Quit and relaunch the app. Verify that the sort ordering of the answers remain the same.

Extra Credit User Stories

Some ideas for extra credit:

- add the ability for people to answer questions with pictures
- allow people to subscribe to notifications for when someone answers a particular question [Parse Push Notification Guide](#)
- add support for [Gravatar](#) profile images
- add photo filters (possibly from Blocstagram) to your profile photo selection process
- add the ability to follow users
- customize your UI for iPad

Additional Resources



- [Parse](#)
 - [Documentation](#)
 - [Quick Start Guide](#)
 - [Getting Started Tutorial](#)
 - [Relational Database Tutorial](#)
 - [3rd-Party Parse Chat Room Tutorial](#)
- [Quora in the iOS App Store](#)


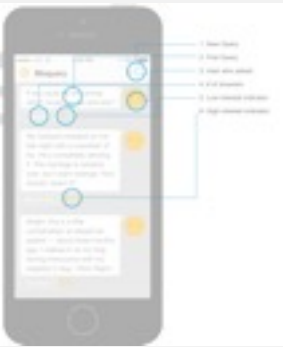




Wireframe Examples

These mockups are meant to guide your design, not dictate it. They represent *one* particular way to interface with an application possessing BloQuery's features. Feel free to express your own creative ideas or attempt to match the ones below.

Professional iOS developers are expected to implement designs generated by their peers or project managers. If you become an independent developer, both design and implementation may become your responsibility.




[Download the full resolution wireframes](#). Here's a description of what's included:

Preview	Name	Description
	Answers view	See a question and its answers.
	Answers view with details	Descriptions of the function of each view on the answers view.




	Questions view	See a list of questions.
	Questions view with details	Descriptions of the function of each view on the questions view.
	New Query view	Post a new question.
	New Query view with details	Descriptions of the function of each view on the new query view.
	Submit answer view	Answer an existing question.
	Submit answer view with details	Descriptions of the function of each view on the submit answer view.

Roadmap

Jump to Submission

 Your assignment	 Ask a question	 Submit your work
---	---	--

COURSES

- >_ Full Stack Web Development
- </> Frontend Web Development
-  UX Design
-  Android Development
-  iOS Development

ABOUT BLOC

[Our Team | Jobs](#)

[Bloc Veterans Program](#)

[Employer Sponsored](#)

[FAQ](#)

[Blog](#)

[Engineering Blog](#)

[Refer-a-Friend](#)

[Privacy Policy](#)

[Terms of Service](#)

MADE BY BLOC

[Tech Talks & Resources](#)

[Programming Bootcamp Comparison](#)

[Jottly: A Beginner's Guide to HTML, CSS, Skeleton and Animate.css](#)

[Swiftris: Build Your First iOS Game with Swift](#)

[Webflow Tutorial: Design Responsive Sites with Webflow](#)

[Ruby Warrior](#)

[Bloc's Diversity Scholarship](#)

SIGN UP FOR OUR MAILING LIST

Send

✉ hello@bloc.io

☎ Considering enrolling? (404) 480-2562

☎ Partnership / Corporate Inquiries? (650) 741-5682

