

BaaSBox

Open Source Backend as a Service

Otto Hylli

Overview (1/2)

- Developed by BaasBox an Italian startup company
- Project was declared started on 1st of July 2012 on the BaasBox blog
- Open source under Apache license V2.0
- Self hosted: requires a server
- Cloud hosted version coming

Overview (2/2)

- Provides general baas services: user management, database, push notifications
- Usage through JSON based REST API or iOS and Android SDKs
- Currently in beta features somewhat limited
- More features coming
- Not ready for production use, changes possible

Contents

- Features: user management, datastorage, notifications, assets, admin console
- How to deploy BaasBox
- iOS SDK and REST API example: DearDiary
- BaasBox under the hood
- Evaluation
- Conclusions

User management

- Users have roles:
- Admin: can use admin API
- Backoffice user: can view / edit users' data
- Registered user: regular user of the app
- API for creating an user (sign up)
- User contains custom objects that are visible to different people: user, friends, registered users, all
- Feature for resetting a forgotten password through a link in an email

Datastorage

- Consists of collections where documents of records are saved
- Admin can create collections
- Documents are schema-less JSON
- Document viewable and editable by default by its creator, admins and backoffice users
- Not documented / implemented yet: friends can see documents. API for setting permissions
- User can do CRUD operations, updating limited querying poorly documented

Push notifications

- Supports push notifications for iOS and Android through a single API
- Both notification systems has to be first configured on the server: For Android an API key an for iOS a certificate
- App has to register to BaasBox with the device's id / token before it can receive notifications
- API for sending messages to a specific user

Assets

- Assets are resources (images, files, JSON objects)
- Can be thought as a building block in the model of your application.
- Only admins can add and remove assets
- Everybody can retrieve them

Administration web console

- Admin users can perform administration tasks by using a web browser.
- Technically an one-page web app which makes REST calls with javascript to the admin APIs
- Dashboard offers an overview: number of users, collections and documents, system info (memory, OS, java, database)
- Settings: application, images, password reset, push notifications
- DB manager: database backups, database reset
- Can manage users, collections, documents and assets

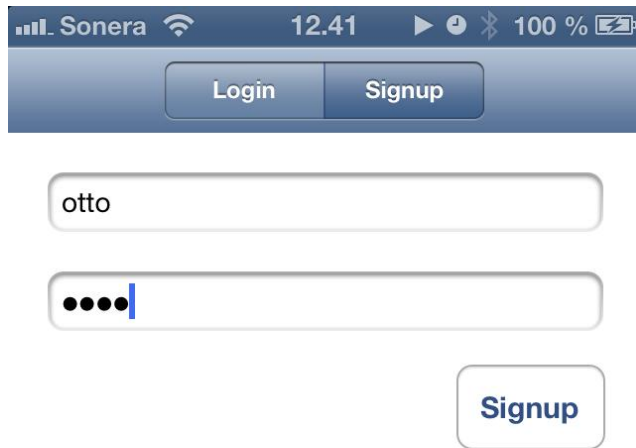
Deploying BaasBox

- Everything in one Java based web application -> only system requirement is a Java VM (version 6 or later)
- Easy to deploy to own laptop (for development), own server, PaaS (e.g. OpenShift) or to a virtual machine for example from Amazon
- Download the zip package containing BaasBox
- Extract the package to directory of your choice and run the start script
- BaasBox creates its database directory and starts listening for HTTP connections on port 9000 (default) -> ready for use

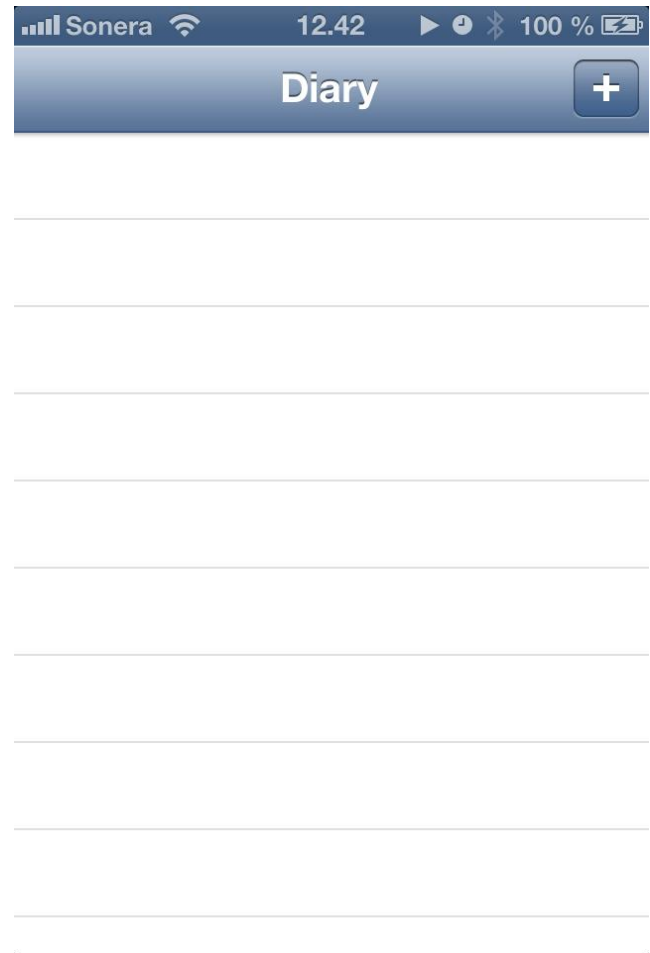
iPhone SDK example: DearDiary

- An iPhone app for making notes from "building your first app with BaasBox" tutorial
- App has a list of notes or posts, user can add and edit them, post has a title and a body.
- At the beginning you get an app which saves notes to the iPhone's memory
- The tutorial shows how to make this app use BaasBox for storing notes.

DearDiary: sign up and posts list

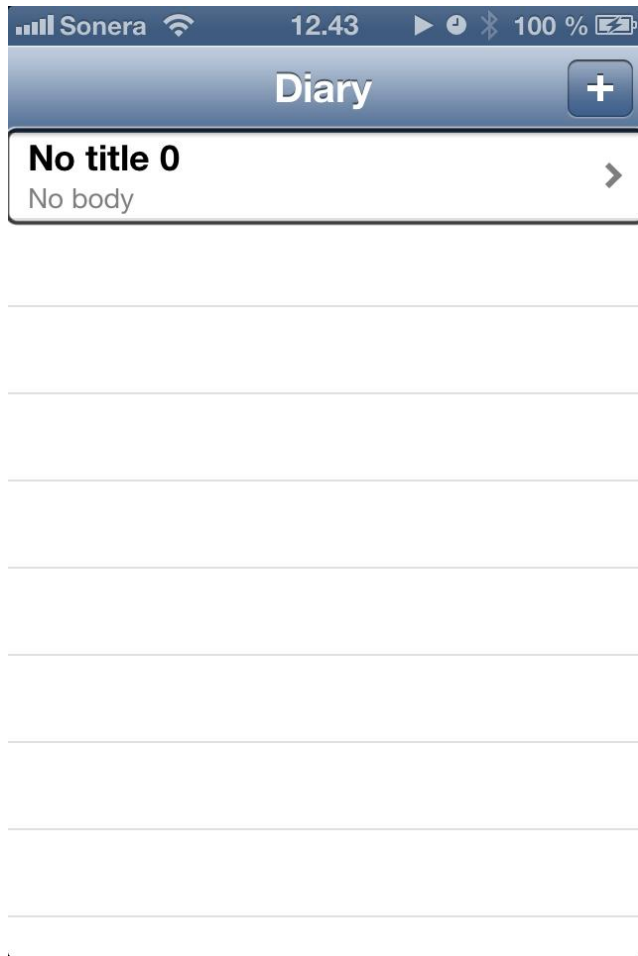


Mobile app interface showing the login/signup screen. The status bar at the top displays "Sonera", signal strength, Wi-Fi, time "12.41", and battery "100%". Below the status bar are two buttons: "Login" and "Signup". There are two input fields: the first contains the text "otto", and the second contains four dots and a vertical cursor. A "Signup" button is located below the second input field.



Mobile app interface showing the diary list screen. The status bar at the top displays "Sonera", signal strength, Wi-Fi, time "12.42", and battery "100%". Below the status bar is a header bar with the text "Diary" and a "+" button. The main area contains a list of horizontal lines representing diary entries.

DearDiary: adding a post and editing it

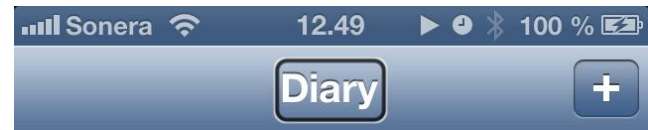


DearDiary: saving changes



baasbox

Is an open source baas.



baasbox

Is an open source baas.



Data model before BaasBox

```
@interface SMPost : NSObject
    @property (copy) NSString *postTitle;
    @property (copy) NSString *postBody;
@end
```

Data model after BaasBox

```
@interface SMPost : BAAObject
```

```
@property (copy) NSString *postTitle;
```

```
@property (copy) NSString *postBody;
```

```
@end
```


SMPost implementation (1/2)

```
- (instancetype)
initWithDictionary:(NSDictionary *)dictionary
{
    self = [super initWithDictionary:dictionary];
    if (self) {
        _postTitle = dictionary[@"postTitle"];
        _postBody = dictionary[@"postBody"];
    }
    return self;
}
```

SMPost implementation (2/2)

```
- (NSString *)collectionName {  
    return @"document/posts";  
}
```

Getting posts from BaasBox

```
BAAClient *client = [BAAClient sharedInstance];
if (client.isAuthenticated) {
    [SMPost getObjectsWithCompletion:
     ^(NSArray *objects, NSError *error) {
        _posts = [objects mutableCopy];
        [self.tableView reloadData];
    }];
}
```

REST API example: DearDiary from command line

- A command line version of DearDiary to demonstrate how the REST API works
- Written in Python uses the Requests HTTP client library for making HTTP requests

Authenticating to BaasBox using REST

```
def getToken( username, password ):
    headers = { "content-type":
                "application/x-www-form-urlencoded" }
    payload = "username" + "=" + username
    payload += "&password" + "=" + password
    payload += "&appcode=deardiary"
    url = "http://130.230.142.84:8080/login"
    response = requests.post(url, data = payload,
                             headers = headers)
    if response.status_code != 200:
        return None
    return response.json()["data"]["X-BB-SESSION"]
```

Authentication HTTP response content

```
{
  "data": {
    "X-BB-SESSION": "8a3c49a2-99e5-4825-8534
                    -e286f575dbd2",
    "user": { "@fieldTypes": "roles=e",
              "name": "otto",
              "roles": [ { "name": "registereduser" } ]
            }
  },
  "result": "ok", "http_code": 200
}
```

Adding a post using REST

```
def addPost( title, body, token ):
    headers = { "X-BB-SESSION": token,
                "content-type": "application/json" }
    url = "http://130.230.142.84:8080/document/posts"
    payload = '{ "postTitle": "' + title + '", '
    payload += '"postBody": "' + body + '" }'
    response = requests.post( url, data = payload,
                              headers = headers )
    if response.status_code != 200:
        print "Error: posting failed."
```

BaaSBox under the hood

- BaasBox is built on top of the Play! Framework
- Play is a lightweight web application framework for Java and Scala
- Database is an embedded OrientDB which is a document oriented graph database
- OrientDB is a fully-featured db: documents with schema / without, supports SQL queries, can work embedded in a program or as a separate server.

BaaSBox architecture

- Not documented much, mainly based on a look at the code
- BaasBox is a traditional RESTful web application
- URLs are mapped on to methods in controller classes that process the HTTP requests and produce the responses
- Controllers use service classes in talking to the database
- Services use data access objects to manipulate data in the db

Example: routes configuration

#Storage Actions

POST /document/:collection

com.baasbox.controllers.Document.createDocument(collection: String)

GET /document/:collection

com.baasbox.controllers.Document.getDocuments(collection: String)

GET /document/%23:rid

com.baasbox.controllers.Document.getDocumentByRid(rid: String)

PUT /document/:collection/%23:rid

com.baasbox.controllers.Document.updateDocument(collection: String, rid: String, isUUID: Boolean ?= false)

DELETE /document/:collection/%23:rid

com.baasbox.controllers.Document.deleteDocument(collection: String, rid: String, isUUID: Boolean ?= false)

Controller method example

```
public static Result getDocuments(String collectionName){
    List<ODocument> result;
    String ret="{}";
    try {
        Context ctx=Http.Context.current.get();
        QueryParams criteria = (QueryParams) ctx.args.get(IQueryParametersKeys.QUERY_PARAMETERS);
        result = DocumentService.getDocuments(collectionName,criteria);
    } catch (InvalidCollectionException e) {
        return notFound(collectionName + " is not a valid collection name");
    } catch (Exception e){
        return internalServerError(e.getMessage());
    }
    try{
        ret=prepareResponseToJson(result);
    }catch (IOException e){
        return internalServerError(ExceptionUtils.getFullStackTrace(e));
    }
    return ok(ret);
}
```

Evaluation (1/2)

- + Self hosted: you control everything
- - self hosted: you have to worry about everything
- + Open source: you can modify it to your needs
- + Documentation: quite clear and understandable
- - Documentation: good overview missing, some conflicts, somethings poorly documented

Evaluation (2/2)

- + the all in one design: easy to install
- - the all in one design: scalability?
- - Features limited: e.g. no real time data synchronization
- - Code and architecture poorly or not at all documented -> how do you get outside developers aboard
- - Quite new and untested in real use: may yet fail.
- - Business model of the company not clear.

Conclusions

- A basic BaaS: no fancy features (yet)
- Looks quite promising.
- Not ready for production use.
- Too early for final conclusions.