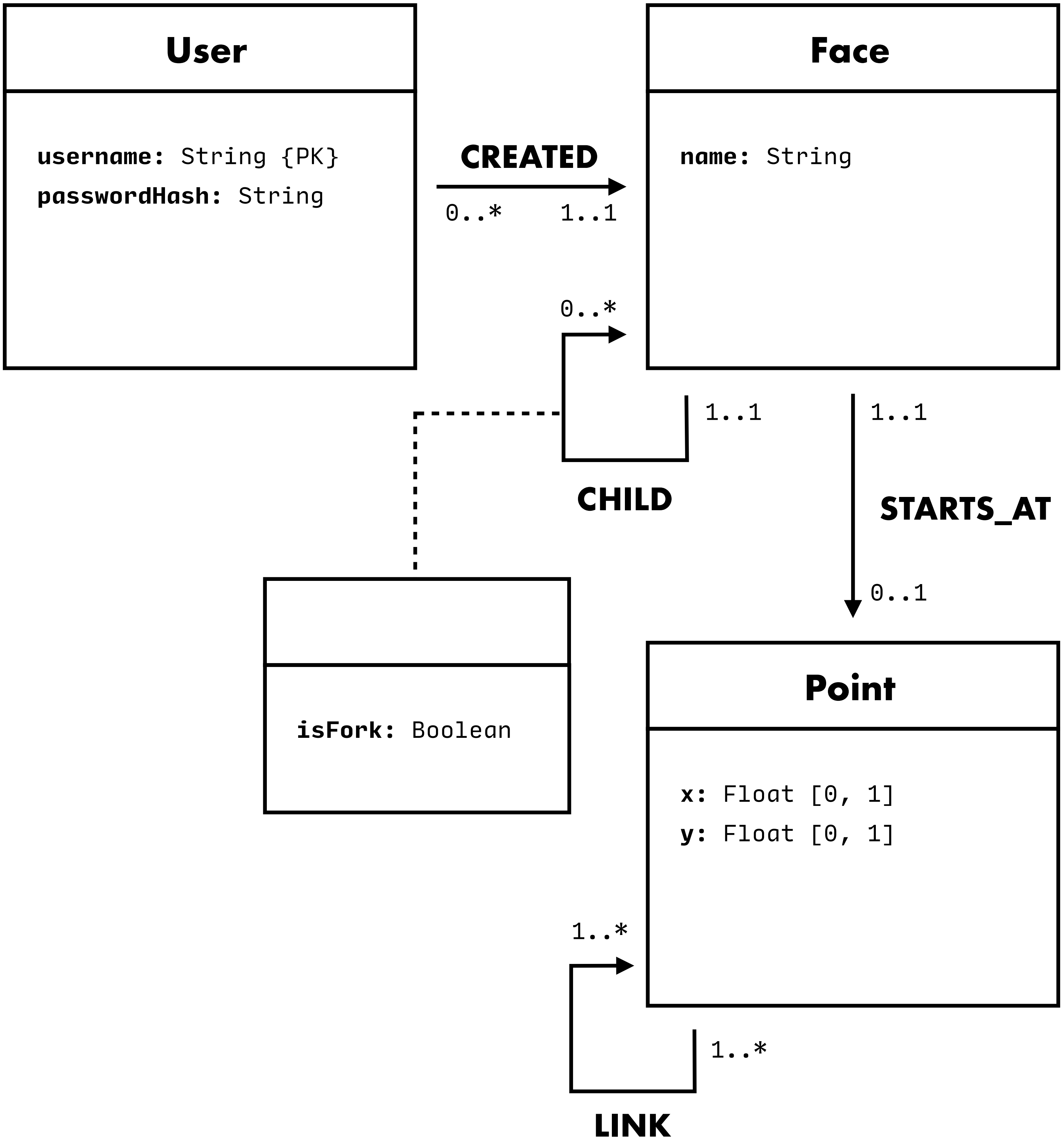
CS 3200 Project Report

README is at the end of this document.

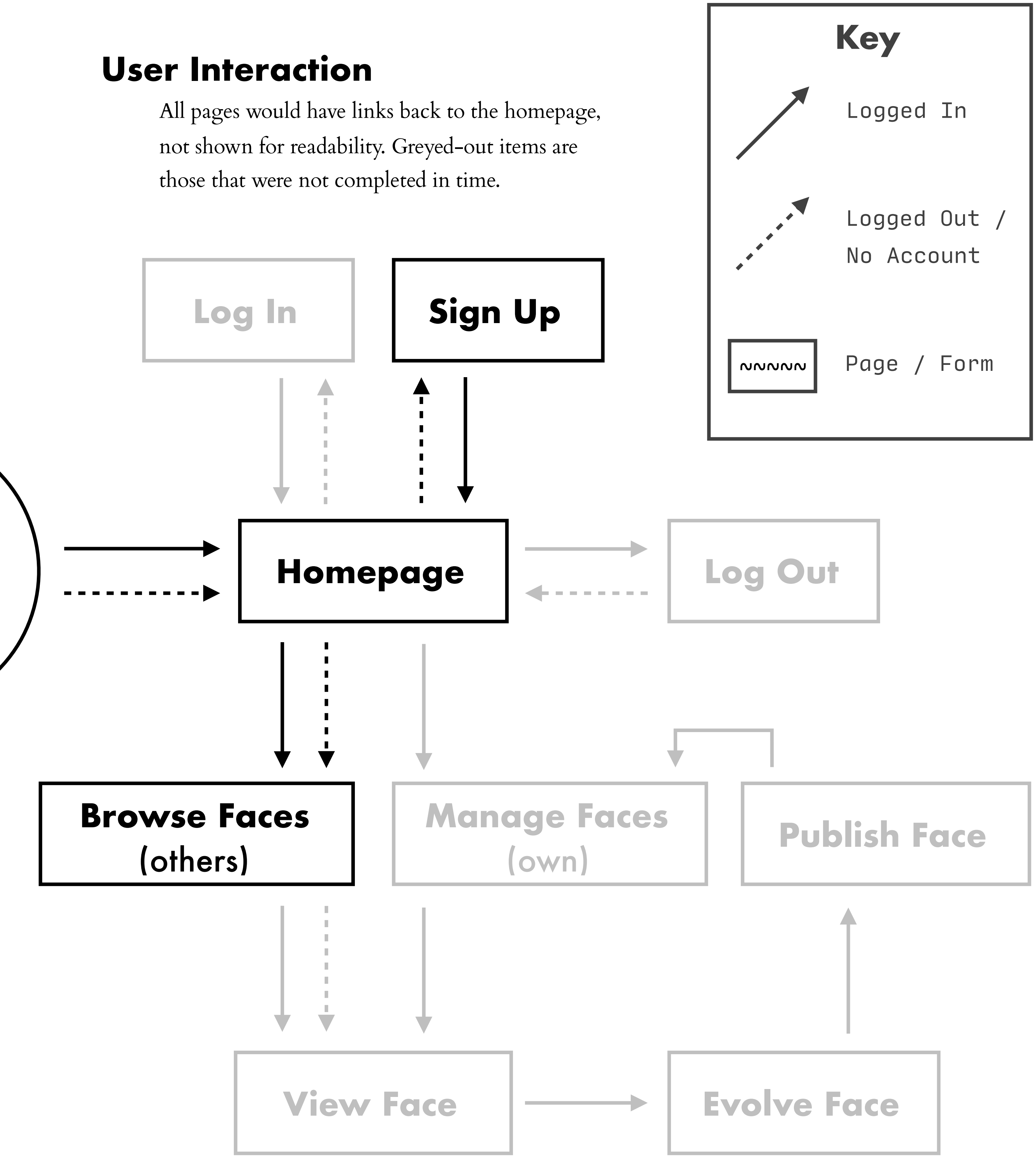
Team Members	Jack Davis, Logan Moore
Blackboard Group	DavisMoore
Description	A web game à la picbreeder (http://picbreeder.org), in which users select from a randomised human face structure (stored as a series of points in 2 or 3 dimensions). Slight modifications to each new "generation" of faces compounds to yield major changes over the course of many generations / selection cycles.
Storage	neo4j (NoSQL Graph Database)
Software & Libraries	Haskell, Hasbolt (Haskell framework for neo4j), Scotty (Haskell web framework)
Requirements	Cross-platform, requires Haskell Platform and a locally running neo4j instance.
Domain Interest	We are interested in this domain because it combines interesting image generation, web development, 2/3D vector transformations, graph operations, and an interactive product. Picbreeder also has interesting applications in the realm of AI.

UML Diagram


User
username: String {PK}
passwordHash: String
CREATED
0..*
1..1
Face
name: String
0..*
1..1
CHILD
1..1
STARTS_AT
0..1
isFork: Boolean
Point
x: Float [0, 1]
y: Float [0, 1]
1..*
1..*
LINK

User Interaction

All pages would have links back to the homepage, not shown for readability. Greyed-out items are those that were not completed in time.



Technical Specifications

Storage	neo4j (NoSQL Graph Database)
Software & Libraries	Haskell, Hasbolt (Haskell framework for neo4j), Scotty (Haskell web framework)
Requirements	Cross-platform, requires Haskell Platform and a locally running neo4j instance.

README

Installation Requirements

- Haskell Platform (https://www.haskell.org/downloads#platform)
- neo4j (https://neo4j.com/download/community-edition/)
  - Run locally (will run on localhost:7474)
  - Set up user with username & password of dataface, or change \`Main.hs\` to match your existing username & password
  - To get a sample face to render, drag `init/init_rootFace.cypher` into the text box in the neo4j web client (localhost:7474) and run it.

Installation

```
git clone https://github.com/ThoseGrapefruits/dataface.git
cd dataface
stack build
stack exec dataface-exe
```

This will take a while to compile all required libraries.

Lessons Learned

Technical Expertise Gained

- Familiarity with a new language, Haskell, which has pushed us both to learn a lot about a very interesting language. It is very much a designed language, with a very mathematical and precise approach. This proved very challenging to adjust to, but was a learning experience nonetheless.
- Knowledge of the strengths and weaknesses of graph databases, and where they fit in with relational and other nonrelational databases. Graph databases seem to be an interesting compromise between relational and nonrelational databases.

Group Work Insights

- We collaborated via Git, which is effective at making all changes very visible and easy to track. However, it can be hard to work on a fairly small project simultaneously over any channel, as there are so many overlapping changes that can cause problems. We used a lot of in-person communication to define what we were working on to overcome this issue.
- We spent a fair amount of time in the beginning designing what we wanted to do, and I think this was good for aligning on our goals and determining what we needed to do to get there. However, as with many projects, there was a major hump to get over getting the project started. This was amplified by the fact that both the language and the database system were previously unfamiliar to us. So, it took a long time to get to the stage that we could actually start making changes and fully understand what was going on.

Alternative Approaches

- As mentioned in the last section, we spent a fair amount of time determining where we wanted to go with the project and how we wanted to do that. This had a positive impact on our approach. One thing we questioned frequently was whether we really wanted to go down a road with a language and a database technology that were completely foreign to us. In the end, we traded having a more complete and extensive final product for learning a lot about new things, and I think that was the right decision.

Future Work

Planned Use

The planned use of our database is as the backing for our user web application, to support the user experience of viewing faces, evolving faces, and analysing evolution trees.

Potential Functionality

User Login

This is the piece that we are the closest to this as the project comes to a close. We can set login cookies when users sign up right now, but there is work to be done on both the API and frontend to support user login. This would be central to the face evolution, which is the next piece of future work. It would be the groundwork for providing users separate viewing and management of the faces they created/published, and would allow the possibility of having user interaction like face popularity.

Face Evolution

The core of our ideal endgame is being able to actually evolve faces through user interaction. Currently, there is a lot in-between here and there. Cloning loosely structured graph data while maintaining its structure has proven fairly difficult through pure neo4j queries. There is also an entire page on the frontend and a few API endpoints that would have to be built out.