Ivan Antipin

Email: ivan.antipin@gmail.com
Mobile: +(45)50173238

Education

- **Moscow Aviation Institute** Moscow
  Master in computer science and mathematics *Sep. 1998 – Feb. 2005*

Employment

- **Nordea Bank** Copenhagen, Denmark
  Expert Developer, Team Lead *Feb 20015 - Present*
    - Leaded 3 different teams. Once two teams at the same time.
    - Implemented fault-tolerant, distributed trade system based on Kafka,Mongo,gRPC. Implemented three gRPC interfaces: historical, publish, and subscribe.
    - Implemented backend and frontend of fault-tolerant, distributed limits monitoring system using Kotlin/Kafka/React/Event sourcing pattern. Introduced OpenAPI as a standard API layer between backend and frontend to decouple development. These techniques lately spread in the department.
    - Implemented alternative to default Avro, java model generation for simplified Cassandra, and mongo persistence.
    - Reimplemented legacy risk system and reduced processing time by order of magnitude.
    - Started internal cloud initiative in the department to suit applications for the real cloud. Designed and implemented build and deploy scripts to dockerize applications. Wrote scripts to propagate distributed configuration and secrets.
    - Worked in the "hack team" for making POCs of technologies in the department. Run demos to share knowledge.
    - Proposed and implemented a couple of initiatives to improve cross-team productivity:
        - Monolith git repository to share code across teams
        - use gRPC specification for cross-team / systems communication instead of REST.

  Tech stack: Java, Kotlin, gRPC, Kafka, Protobuf, Avro, Docker, Cassandra, Redis, Mongo, Oracle, Hashicorp Consul, Vault, c-sharp, ActivePivot, React, OpenAPI

- **Deutsche Bank** Moscow, Russia
  Senior Software Engineer *Apr 2012 - Nov 2014*
    - Implemented in-memory rule matching engine using bit index.
    - Implemented risk metrics storage based on Oracle Coherence.

  Tech stack: Java, Oracle, Oracle Coherence

- **Luxoft** Moscow, Russia
  Senior Software Engineer *March 2011 - March 2012*

  Achievements:
    - Participated in the implementation of a low latency pricing system in java. It could process a few hundred thousand records in a second with just 100x microseconds latency in a single thread. To achieve low latency, we used:
        - multicast
        - cache-friendly techniques (primitive value types)
        - minimum number of system calls to prevent context switches
        - single thread
        - minimized copying of data (java direct buffers)

Tech stack: Java, NIO, Multicast

- **Deutsche Bank** — Moscow, Russia
  Senior Software Engineer — *Apr 2005 - Nov 2010*

  Achievements:

  - Implemented pricing system.
  - Implemented an order management system based on a state machine with an internal matching engine.
  - Implemented a declarative testing framework for the order management system

  Tech stack: Java, JMS, Oracle

Side activities

- Developed trading strategy backtesting/analytical framework in kotlin, java, python.
- Made AI research for trading strategies using python, pandas, numpy, scikit-learn libs.

Skills

- **Languages**: Java, Kotlin, Scala, Python, c-sharp, SQL, JavaScript
- **Technologies**: Docker, Kafka, Mongo, gRPC, Avro, Protobuf, Cassandra, Hashicorp Consul, Vault
- **Other**: Gradle, Unix command line, CS knowledge, git, basic ML knowledge, pandas, scikit-learn, numpy

My principles to gain optimal productivity at work

- Complexity and entropy of any system tend to increase during the lifetime (sometimes exponentially) and this is critical to keep it under control by continuous refactoring and state removal.
- Need to minimize people sidetracking as context switches are costly (same analogy correct for computer). We need to avoid too many microtasks not related to each other for the very same reason.
- Cross teams' communication has to be covered by API contracts as much as possible to minimize human interaction during systems integration.
- OOP stateful and sophisticated by design so need to be used carefully and in a limited scope, mainly for creating frameworks that are rarely needed.
- Testing: we need to shape your application the way it would be easily testable; otherwise, developers will try to avoid test writing.